

En la primera parte del tema se ha repasado toda la sintaxis de la sentencia SELECT. Una vez conocidas todas las cláusulas, es momento de sacar toda la potencia de SQL accediendo a bases de datos relacionales, haciendo unas sentencias "avanzadas", de un nivel de complejidad mayor que hasta ahora.

Los tipos de sentencias avanzadas que veremos serán:

- Combinaciones de tablas, en las que entrará en juego más de una tabla. Veremos las distintas posibilidades que tendremos.
- Subconsultas, donde tendremos una sentencia SQL dentro de otra.
- Consultas de unión, donde uniremos más de una sentencia SQL.

Vimos en la sentencia básica que en la cláusula FROM poníamos la mesa **o mesas** de donde cogerían los datos, pero en todos los ejemplos posteriores sólo entraba en juego una única mesa.

Es el momento de estudiar las diferentes posibilidades que tendremos cuando ponemos más de una tabla.

- La primera es el **producto cartesiano**, que no utilizaremos nunca, pero debemos saber en qué consiste para poder evitarlo.
- La segunda será la más utilizada, la **combinación** (que a veces llamaremos **reunión**).
- La tercera es una variante de la anterior, la **combinación externa**, muy útil en algunos casos.

2.2.1 Producto cartesiano

La manera más sencilla es poner las tablas separadas por comas, pero seguramente el resultado no es lo que esperábamos.

Por ejemplo podemos hacer la siguiente sentencia:

```
SELECT COMARQUES.nom_c, nombre
FROM COMARCAS, POBLACIONES;
```

Nota

Obsérvese que hemos puesto el nombre de la tabla frente al campo **nom_c**, porque las dos tablas tienen un campo con ese nombre. Esta operación se denomina **calificación**. Si no califica con el nombre de la tabla delante, habría ambigüedad, no sabría a qué campo se refiere, si el de una mesa o el de la otra. Cuando los nombres de los campos son diferentes y por lo tanto no coinciden en las dos tablas, no hay que calificar el campo (como por ejemplo con el campo **nombre**).

Si ejecutamos la sentencia, veremos que tendremos un número de filas inesperadamente alto, **18.428 filas !!!** Y si analizamos las filas veremos el porqué: se ha combinado cada comarca con todos los pueblos (sean de la comarca o no).

The screenshot shows a database query window titled "Query - geo sobre geo@80.35.84.29:5432 *". The "Editor SQL" tab is active, displaying the query: `SELECT COMARQUES.nom_c, nom FROM COMARQUES, POBLACIONES;`. Below the editor, the "Subfinestra de sortida" (Output Subwindow) is open, showing the results of the query. The results are displayed in a table with two columns: **nom_c character varying(50)** and **nom character varying(50)**. The table contains 12 rows, numbered 1 to 12, each showing a comarca name and the word "Ademuz". At the bottom of the window, the status bar indicates "OK.", "Unix", "Ln 2, Col 27, Ch 55", "18428 regis", and "11354 ms".

	nom_c character varying(50)	nom character varying(50)
1	Safor	Ademuz
2	Horta Sud	Ademuz
3	Camp de Morvedr	Ademuz
4	Foia de Bunyol	Ademuz
5	Alacantí	Ademuz
6	Alt Maestrat	Ademuz
7	Plana Baixa	Ademuz
8	Horta Nord	Ademuz
9	Ports	Ademuz
10	Racó	Ademuz
11	Plana d'Utiel	Ademuz
12	Vinalopó Mitjà	Ademuz

Esta operación se denomina **producto cartesiano** (**cross join**), y se caracteriza en que cada una de las filas de una tabla se combina con todas las filas de la otra tabla. El número de filas resultante será, pues, el resultado de multiplicar el número de filas de una tabla por el número de filas de la otra tabla (en nuestro caso $34 \times 542 = 18428$).

Raramente queremos hacer un producto cartesiano. Lo normal será combinar un poquito mejor las tablas. En nuestro ejemplo seguramente nos interesará mucho más combinar *cada comarca con sus poblaciones*.

Combinación de dos tablas: Sintaxis

Normalmente el producto cartesiano no nos interesará. Más bien queremos combinar las tablas de modo que dos campos, un campo de cada mesa, coincidan. Y lo más habitual, si tenemos la Base de Datos bien diseñada, será que los campos coincidentes sean una clave externa con la clave principal a la que apunta. Así, en el ejemplo que utilizábamos en el punto anterior, lo que sí nos será útil es combinar cada comarca con sus poblaciones. Y justamente tenemos un campo en la tabla POBLACIONES, **nom_c**, que es clave externa y apunta a la clave principal de **COMARCAS**.

Esta operación la llamaremos **COMBINACIÓN INTERNA** o sencillamente **COMBINACIÓN**, ya veces también se llama **REUNIÓN**. Su sintaxis es la siguiente:

```
SELECT ...  
FROM tabla1 INNER JOIN Tabla2 ON condición
```

y donde la condición de la reunión consistirá en comparar un campo de cada mesa. Los dos campos deberán ser del mismo tipo, pero no será necesario que tengan el mismo nombre. Las filas que saldrán al resultado serán las que cumplirán la condición.

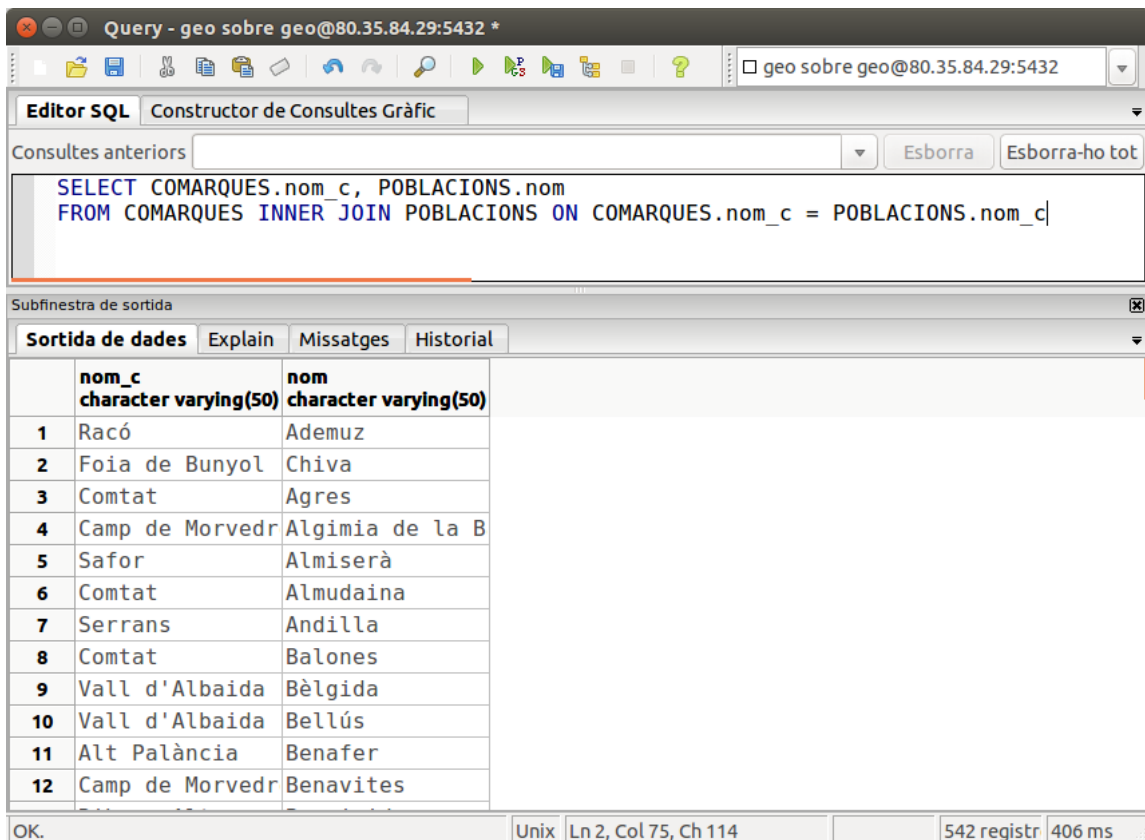
Aunque los operadores que se pueden utilizar son todos los de comparación, en la práctica SIEMPRE utilizaremos el de igualar. Por lo tanto podemos refinar mejor la combinación de 2 mesas

```
SELECT ...  
FROM tabla1 INNER JOIN Tabla2 ON taula1.camp1 = taula2.camp2
```

El ejemplo de las comarcas y sus poblaciones quedará así

```
SELECT COMARQUES.nom_c, POBLACIONES.nom  
FROM COMARCAS INNER JOIN POBLACIONES DONDE COMARQUES.nom_c = POBLACIONES.nom_c
```

Y este sería el resultado



The screenshot shows a database query editor window titled "Query - geo sobre geo@80.35.84.29:5432 *". The "Editor SQL" tab is active, displaying the following query:

```
SELECT COMARQUES.nom_c, POBLACIONES.nom  
FROM COMARCAS INNER JOIN POBLACIONES ON COMARQUES.nom_c = POBLACIONES.nom_c
```

Below the editor, the "Subfinestra de sortida" (Output Subwindow) is visible, showing the results of the query. The results are displayed in a table with two columns: **nom_c** (character varying(50)) and **nom** (character varying(50)). The table contains 12 rows of data:

	nom_c character varying(50)	nom character varying(50)
1	Racó	Ademuz
2	Foia de Bunyol	Chiva
3	Comtat	Agres
4	Camp de Morvedr	Algimia de la B
5	Safor	Almiserà
6	Comtat	Almudaina
7	Serrans	Andilla
8	Comtat	Balones
9	Vall d'Albaida	Bèlgida
10	Vall d'Albaida	Bellús
11	Alt Palància	Benafer
12	Camp de Morvedr	Benavites

At the bottom of the window, the status bar indicates "OK", "Unix", "Ln 2, Col 75, Ch 114", "542 registr", and "406 ms".

que como vemos vuelve 542 filas (tantas como pueblos)

Alternativamente, podríamos poner la misma reunión de otra forma:

```
SELECT COMARQUES.nom_c, POBLACIONES.nom  
FROM COMARCAS, POBLACIONES  
WHERE COMARQUES.nom_c = POBLACIONES.nom_c
```

donde estrictamente lo que estamos haciendo es, del producto cartesiano de las 2 tablas, seleccionar únicamente cuando coincide el **nom_c** (es decir la comarca con sus poblaciones), y por tanto el resultado sería el mismo. Quizás sería más eficiente utilizar el primer modo, pero a veces la comodidad nos hará utilizar la segunda (sobre todo cuando se tengan que combinar muchas tablas).

Las dos maneras de poner la combinación de dos tablas son las más habituales, y que funcionan en cualquier Sistema Gestor de Bases de Datos.

Sin embargo en PostgreSQL (y en otros SGBD como Oracle) hay más maneras de hacer una combinación. No las veremos tan a fondo para que las anteriores nos bastan y sobran:

- **INNER JOIN con USING** : En caso de que los campos a reunir de las dos tablas se digan exactamente igual, podemos sustituir la condición puesta en **ON** para la expresión **USING** , con el campo de la reunión entre paréntesis

```
SELECT COMARQUES.nom_c, POBLACIONES.nom
FROM COMARCAS INNER JOIN POBLACIONES USING (nom_c)
```

- **NATURAL JOIN** : También para el caso anterior, en el que el campo en las dos tablas se llama igual, podemos hacerlo de forma aún más abreviada. Hará una reunión, igualando todos los campos que se digan igual de las dos tablas. Debemos tener cuidado, por si hay algún otro campo en las dos tablas que se diga igual.

```
SELECT COMARQUES.nom_c, POBLACIONES.nom
FROM COMARCAS NATURAL JOIN POBLACIONES
```

ejemplos

1. Sacar el nombres de las Poblaciones y los nombres de los Institutos que hay en ellas.

Tendremos que combinar las tablas para la clave externa de INSTITUTOS a POBLACIONES (es decir, la que apunta de cod_m en INSTITUTOS hasta la clave principal de COMARCAS, que es justamente cod_m).

```
SELECT POBLACIONES.nom, INSTITUTS.nom
FROM POBLACIONES INNER JOIN INSTITUTOS ON POBLACIONES.cod_m = INSTITUTS.cod_m;
```

Utilizando la otra sintaxis, que ponemos la condición en el **WHERE** , nos quedaría:

```
SELECT POBLACIONES.nom, INSTITUTS.nom
FROM POBLACIONES, INSTITUTOS
WHERE POBLACIONES.cod_m = INSTITUTS.cod_m;
```

Como en este caso el campo que debemos igualar tiene el mismo nombre en ambas tablas, utilizando la sintaxis del **USING** nos quedará más fácil:

```
SELECT POBLACIONES.nom, INSTITUTS.nom
FROM POBLACIONES INNER JOIN INSTITUTOS USING (cod_m);
```

En cambio, debemos tener mucho cuidado con la sintaxis del **NATURAL JOIN** , porque itentará igualar todos los campos que se llaman igual, y en este caso tenemos dos campos coincidentes: cod_m y nombre. cod_m es lo que queremos, pero nombre nos fastidiará, y evidentemente no coincide nunca el nombre del Instituto y el de la población, y por tanto no devolverá ninguna fila.

```
SELECT POBLACIONES.nom, INSTITUTS.nom
FROM POBLACIONES NATURAL JOIN INSTITUTOS;
```

2. Sacar los nombres de las comarcas y la provincia, con el número de poblaciones que tiene cada comarca.

Nos hacen falta dos tablas, COMARCAS para poder sacar el nombre de la comarca y la provincia, y POBLACIONES para poder contar los pueblos de cada comarca. Las tendremos que combinar, agrupar por comarca (y provincia también, porque queremos que aparezca el nombre de la provincia) y contar las poblaciones. A la hora de contar podemos contar filas (COUNT (*)), pero tal vez sea mejor contar algún campo de la tabla POBLACIONES, por ejemplo cod_m, que es la clave principal (recordemos que los valores nulos no se contarán, y cod_m por fue clave principal no puede ser nulo).

```
SELECT COMARQUES.nom_c, provincia, COUNT (cod_m) AS Cuántos
FROM COMARCAS INNER JOIN POBLACIONES DONDE COMARQUES.nom_c = POBLACIONES.nom_c
GROUP BY COMARQUES.nom_c, provincia;
```

Tres o más tablas

Si tenemos más de 2 mesas, tendremos que proceder de la misma manera, ya que si dejamos de combinar alguna mesa, tendremos el producto cartesiano. Como en la inmensa mayoría de casos, la reunión la haremos por las claves externas que tenemos definidas. Únicamente tendremos que cuidar los paréntesis, para marcar primero una condición de combinación y después la otra. En un ejemplo lo veremos perfectamente ilustrado.

Intentamos sacar el nombre de una comarca y la provincia, el nombre de sus pueblos y el nombre de los institutos de estos pueblos. Nos hacen falta las tablas COMARCAS (para poder sacar el nombre de la comarca y provincia), POBLACIONES (para sacar el nombre de la población) y INSTITUTOS (para el nombre de estos). Ordenaremos por nombre de comarca, y dentro de éste por población, para una mejor lectura del resultado

```
SELECT COMARQUES.nom_c, provincia, POBLACIONES.nom, INSTITUTS.nom
FROM (COMARCAS INNER JOIN POBLACIONES DONDE COMARQUES.nom_c = POBLACIONES.nom_c)
INNER JOIN INSTITUTOS ON POBLACIONES.cod_m = INSTITUTS.cod_m
ORDER BY 1,3;
```

Podríamos poner la consulta de la forma alternativa, en la que las condiciones de reunión en el WHERE. Obviamente estas condiciones deben ir unidas por el operador AND.

```
SELECT COMARQUES.nom_c, provincia, POBLACIONES.nom, INSTITUTS.nom
FROM COMARCAS, POBLACIONES, INSTITUTOS
WHERE COMARQUES.nom_c = POBLACIONES.nom_c AND POBLACIONES.cod_m = INSTITUTS.cod_m
ORDER BY 1,3;
```

Vamos a plantear otro ejemplo. Se trata de sacar el nombre y la provincia de las comarcas, con el número de Institutos que hay en ellas. En principio podríamos pensar que las únicas tablas que nos hacen falta son COMARCAS (para sacar el nombre y provincia de la comarca) y INSTITUTOS (para poder contar los INSTITUTOS). Si intentamos hacer esta consulta, **NO** obtendremos el resultado deseado.

```
SELECT nom_c, provincia, COUNT (código)
FROM COMARCAS, INSTITUTOS
GROUP BY nom_c, provincia
```

Evidentemente habrá un producto cartesiano, ya que no hemos combinado las mesas, y nos saldrá para cada comarca 375 institutos, que es el número total de institutos, ya que se ha combinado cada comarca con todos los institutos.

Pero entonces, por qué campo combinamos? Si intentamos unir las claves principales, **nom_c** con **código** (el código de Instituto) no pueden combinar bien por razones evidentes. Nos tenemos que fijar en el diseño de la Base de Datos. Observaremos que el problema es que no hay una clave externa entre INSTITUTOS y COMARCAS. Pero también nos da la solución: **tendremos que poner también la tabla POBLACIONES** aunque no queramos visualizar ningún campo de esta tabla, ya que si están relacionadas las tablas INSTITUTOS y COMARCAS es a través de esta tabla. Por lo tanto la consulta correcta será:

```
SELECT COMARQUES.nom_c, provincia, COUNT (código)
FROM (COMARCAS INNER JOIN POBLACIONES DONDE COMARQUES.nom_c = POBLACIONES.nom_c)
INNER JOIN INSTITUTOS ON POBLACIONES.cod_m = INSTITUTS.cod_m
GROUP BY COMARQUES.nom_c, provincia
```

La forma alternativa parece más corta. Está claro que si son 3 mesas, deberán haber 2 condiciones de combinación unidas por AND.

```
SELECT COMARQUES.nom_c, provincia, COUNT (código)
FROM COMARCAS, POBLACIONES, INSTITUTOS
WHERE COMARQUES.nom_c = POBLACIONES.nom_c AND POBLACIONES.cod_m = INSTITUTS.cod_m
GROUP BY COMARQUES.nom_c, provincia;
```

De forma general, si tenemos **n** tablas en una consulta, nos harán falta **n-1** condiciones de combinación unidas por AND. Por ejemplo, si en una consulta entran 5 mesas, para no tener ningún producto cartesiano nos harán falta 4 condiciones unidas por AND.

Una mesa más de una vez.

Vamos a plantear otro ejemplo interesante: sacar el nombre de las poblaciones, con el nombre de la capital de comarca. Lamentablemente con los datos que tenemos en la Base de Datos de ejemplo no podremos probarlo, así que vamos a hacer una suposición, una Base de Datos ligeramente modificada para ilustrar este ejemplo.

Supongamos que nuestra mesa de POBLACIONES fuera ligeramente diferente, y que incorporara un campo nuevo con el código del municipio que es capital de comarca de la población:

```
POBLACIONES
(
```

```

cod_m numérico (5,0) CONSTRAINT cp_pobl PRIMARY KEY,
nombre character Varying (50) NOT NULL,
población numérico (6,0),
extensión numérico (6,2),
altura numérico (4,0),
longitud character Varying (50),
latitud character Varying (50),
lengua character (1),
nom_c character Varying (50),
cod_capital numérico (5,0) CONSTRAINT ce_capital REFERENCES POBLACIONES (cod_m)
)

```

Para poder sacar al mismo tiempo el nombre de las poblaciones y el nombre de su capital de comarca no nos basta con poner la mesa POBLACIONES una vez: sólo sacaríamos el nombre de la población y nos quedaríamos con el código de municipio de la capital . La solución será reunirse con la mesa POBLACIONES, poniéndola una segunda vez para tener dos instancias de la mesa, una instancia para el población normal y otra para la capital. Pero como distinguiremos entre las dos instancias? Pues poniendo un nombre a cada una. En general podemos poner un nombre en la sentencia a cualquier mesa que aparezca, poniendo este nombre a continuación de la tabla (opcionalmente podríamos poner AS medio):

```

SELECT ...
FROM T

```

En el resto de la consulta deberemos utilizar este nombre. El ejemplo quedará de la siguiente manera:

```

SELECT P1.nom AS "Nombre población", P2.nom as "Nombre capital"
FROM POBLACIONES P1 INNER JOIN POBLACIONES P2 ON P1.cod_capital = P2.cod_m

```

o de la forma alternativa:

```

SELECT P1.nom AS "Nombre población", P2.nom as "Nombre capital"
FROM POBLACIONES P1, POBLACIONES P2
WHERE P1.cod_capital = P2.cod_m

```

Nota

Recuerde que estas instrucciones no las podemos probar, porque no tenemos el campo **cod_capital** .

Clave externa formada por más de un campo

Por último vamos a considerar el caso de que la clave externa esté formada por más de un campo. Lo basaremos en el ejemplo de los **Bancos** , donde la mesa CUENTA CORRIENTE depende en identificación de SUCURSAL. Como la clave principal de SUCURSAL está formada por 2 campos, la clave externa de CUENTA CORRIENTE, que apunta a la primera también estará formada por 2 campos. Si queremos sacar el número de cuenta corriente, el nombre de la sucursal de donde es la cuenta, y el saldo, nos harán falta las dos tablas. Esta sería la manera de combinarlas:

```

SELECT C.n_ent, C.n_suc, n_cc, S.nom, C.saldo
FROM SUCURSAL S INNER JOIN COMPTE_CORRENT C ON S.n_ent = C.n_ent AND S.n_suc = C.n_suc

```

o de la forma alternativa:

```

SELECT C.n_ent, C.n_suc, n_cc, S.nom, C.saldo
FROM SUCURSAL S, COMPTE_CORRENT C
WHERE S.n_ent = C.n_ent AND S.n_suc = C.n_suc

```

En ambos casos se ha optado por poner nombre a las tablas (S y C respectivamente) por comodidad, para que no quedara tan larga la consulta.



Ejercicios apartado 2.2.2

6.5 1 Sacar el nombre de los clientes con el número de factura y la fecha (individuales, sin agrupar nada) que tiene cada cliente. Sacar el resultado ordenado por cliente, y dentro de éste por fecha de la factura

6.5 2 Sacar el nombre del socio, con el código y la descripción de cada artículo que ha pedido. Ordenar por nombre del socio y código del artículo.

6.5 3 Modificar el anterior para que no se repitan los resultados.

6.5 4 Sacar el nombre de los clientes con la cantidad de facturas que tienen, ordenadas por este número de mayor a menor

6.5 5 Sacar el número de factura, fecha, código de cliente, total de la factura (con el alias IMPORTE) y total de la factura aplicando descuentos de artículo (con alias DESCOMPTE_1). Tendremos el problema de que el valor NULL es especial, y al operar con cualquier otro valor dará NULL. En este caso claramente la debemos considerar como un descuento 0. Puede utilizar una función que sustituye los valores nulos encontrados en el primer parámetro, el segundo parámetro de este modo: **COALESCE (dto, 0)**. Ordenar por número de factura.

6.5 6 Modificar el anterior para aplicar también el descuento de la factura (con el alias DESCOMPTE_2)

6.5 7 Sacar el código y nombre de aquellos vendedores que supervisan alguien (constan como cabeza). Sacar también el número de supervisados de cada uno de estos supervisores.

6.5 8 Sacar el código y descripción de los artículos junto con el número de veces que se ha vendido, el total de unidades vendidas y la media de unidades vendidas por factura.

6.5 9 Sacar el código y la descripción de las categorías, con la cantidad de artículos vendidos de cada categoría, de aquellas categorías de las que se han vendido más de 100 unidades. Ordenar por este número de forma decedent.

2.2.3 Combinación externa

En ocasiones nos interesará hacer una combinación diferente. Como casi siempre nos basaremos en un ejemplo. Cuando en un ejemplo del punto anterior sacando los nombre de las poblaciones con el nombre de los institutos, no podían salir las poblaciones que no tienen institutos. Ahora nos plantearemos la posibilidad de sacar todas las poblaciones, incluso aquellas que no tienen institutos, pero de aquellas que sí lo tengan sacar también el nombre de los institutos. Esta operación se llama **COMBINACIÓN EXTERNA**.

sintaxis

Tendremos dos posibilidades: sacar todas las de la izquierda o sacar todas las de la derecha.

Para sacar TODAS las filas de la tabla de la izquierda, y aquellas que estén relacionadas de la de la derecha:

```
SELECT ...  
FROM tabla1 LEFT [OUTER] JOIN Tabla2 ON condición
```

Así sacaremos TODAS las filas de tabla1, y aquellas que estén relacionadas de Tabla2.

Para hacerlo al revés, es decir, todas las filas de la tabla de la derecha y aquellas filas que estén relacionadas de la izquierda:

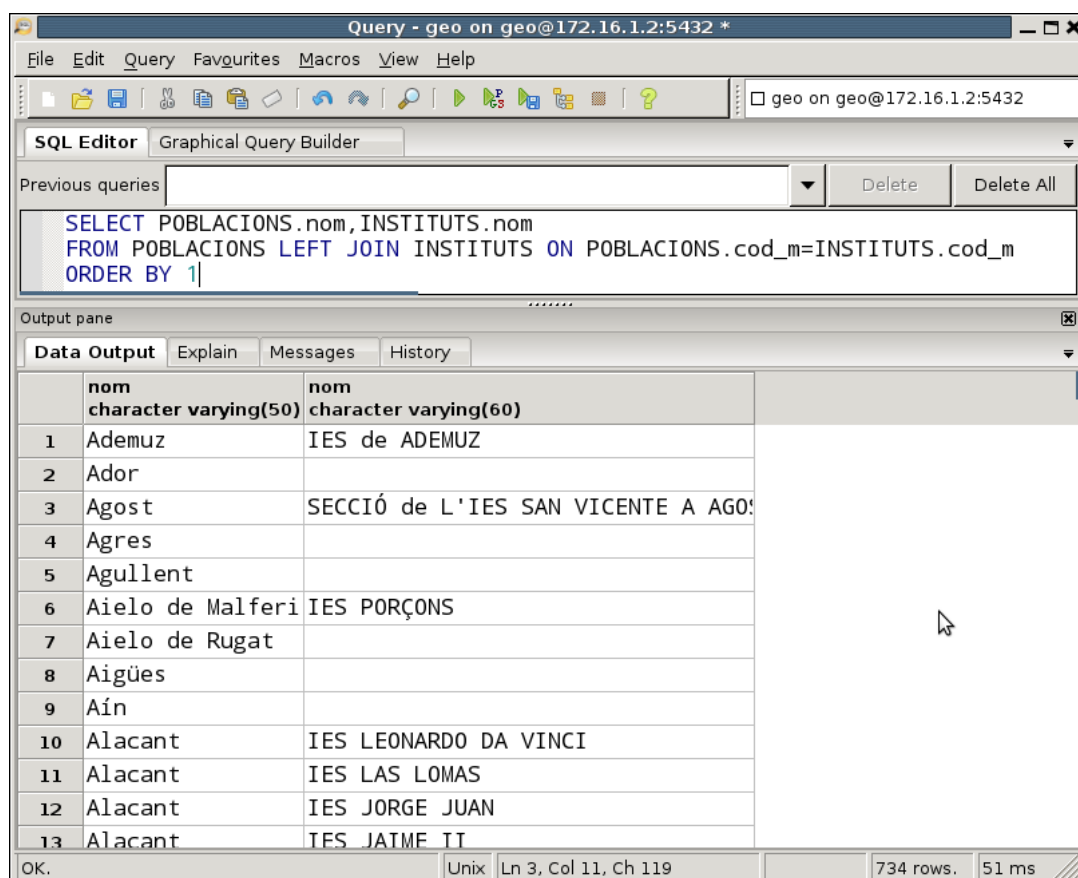
```
SELECT ...  
FROM tabla1 RIGHT [OUTER] JOIN Tabla2 ON condición
```

De esta manera sacaremos TODAS las filas de Tabla2, y aquellas que estén relacionadas de tabla1.

En nuestro ejemplo:

```
SELECT POBLACIONES.nom, INSTITUTS.nom  
FROM POBLACIONES LEFT JOIN INSTITUTS ON POBLACIONES.cod_m = INSTITUTS.cod_m  
ORDER BY 1
```

donde hemos ordenado por el nombre de la población para una mejor lectura, y nos dará el siguiente resultado:



The screenshot shows a database query tool window titled "Query - geo on geo@172.16.1.2:5432 *". The SQL Editor contains the following query:

```
SELECT POBLACIONES.nom, INSTITUTS.nom  
FROM POBLACIONES LEFT JOIN INSTITUTS ON POBLACIONES.cod_m=INSTITUTS.cod_m  
ORDER BY 1
```

The Output pane shows the results of the query in a table with two columns: "nom" (character varying(50)) and "nom" (character varying(60)). The results are as follows:

	nom character varying(50)	nom character varying(60)
1	Ademuz	IES de ADEMUZ
2	Ador	
3	Agost	SECCIÓ de L'IES SAN VICENTE A AGOS
4	Agres	
5	Agullent	
6	Aielo de Malferi	IES PORÇONS
7	Aielo de Rugat	
8	Aigües	
9	Aín	
10	Alacant	IES LEONARDO DA VINCI
11	Alacant	IES LAS LOMAS
12	Alacant	IES JORGE JUAN
13	Alacant	IES JATME II

The status bar at the bottom indicates "OK.", "Unix", "Ln 3, Col 11, Ch 119", "734 rows.", and "51 ms".

Podemos observar que nos saca incluso los pueblos que no tienen institutos, y que en el campo nombre del instituto tienen el valor NULL.

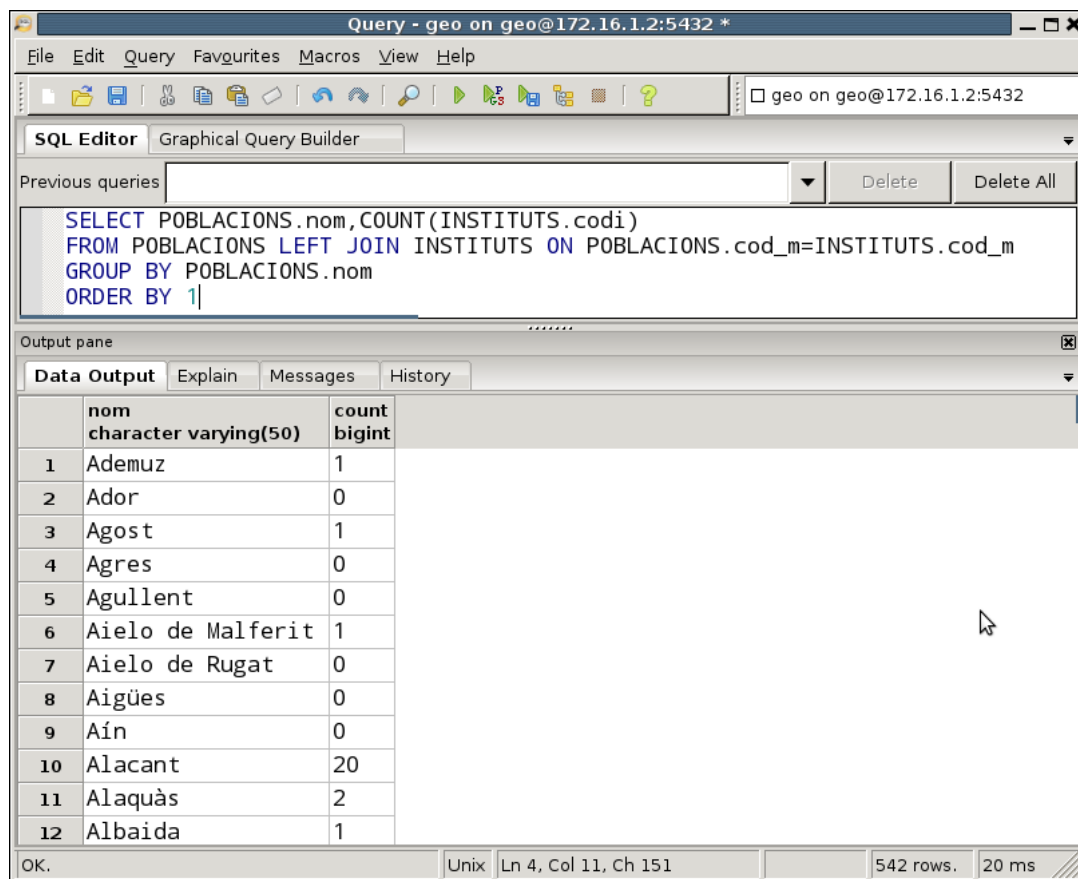
Hacemos una variante interesante. Vamos a sacar los pueblos con el número de institutos que tiene cada uno. Nos hará falta la mesa POBLACIONES para poder sacar el nombre de la población y la mesa INSTITUTOS para sacar el número de institutos y agruparemos por el nombre de la población. Las dos mesas las tenemos que combinar (para evitar el producto cartesiano). Si hacemos una combinación normal (interna), *los que no tienen institutos no entran* . Pero si hacemos una **combinación externa sí entrarán** .

Sólo nos queda contar por un campo que en el caso de los que no tienen institutos tenga el valor nulo, es decir, por un campo de la tabla INSTITUTOS, y lo que mejor se nos acopla es alguno que forma parte de la clave principal , ya que como no puede coger un valor nulo en la tabla INSTITUTOS, la única posibilidad de que coja el valor nulo en la combinación externa es que la población no tenga instituto, y entonces en el momento de contar nos dará el valor 0.

Esta será la consulta, donde hemos vuelto a ordenar por el nombre de la población

```
SELECT POBLACIONES.nom, COUNT (INSTITUTS.codi)
FROM POBLACIONES LEFT JOIN INSTITUTS ON POBLACIONES.cod_m = INSTITUTS.cod_m
GROUP BY POBLACIONES.nom
ORDER BY 1
```

Y este será el resultado



The screenshot shows a SQL query editor window titled "Query - geo on geo@172.16.1.2:5432 *". The query is:
`SELECT POBLACIONES.nom, COUNT(INSTITUTS.codi)`
`FROM POBLACIONES LEFT JOIN INSTITUTS ON POBLACIONES.cod_m=INSTITUTS.cod_m`
`GROUP BY POBLACIONES.nom`
`ORDER BY 1`
The "Output pane" shows the "Data Output" tab with the following results:

	nom character varying(50)	count bigint
1	Ademuz	1
2	Ador	0
3	Agost	1
4	Agres	0
5	Agullent	0
6	Aielo de Malferit	1
7	Aielo de Rugat	0
8	Aigües	0
9	Aín	0
10	Alacant	20
11	Alaquàs	2
12	Albaida	1

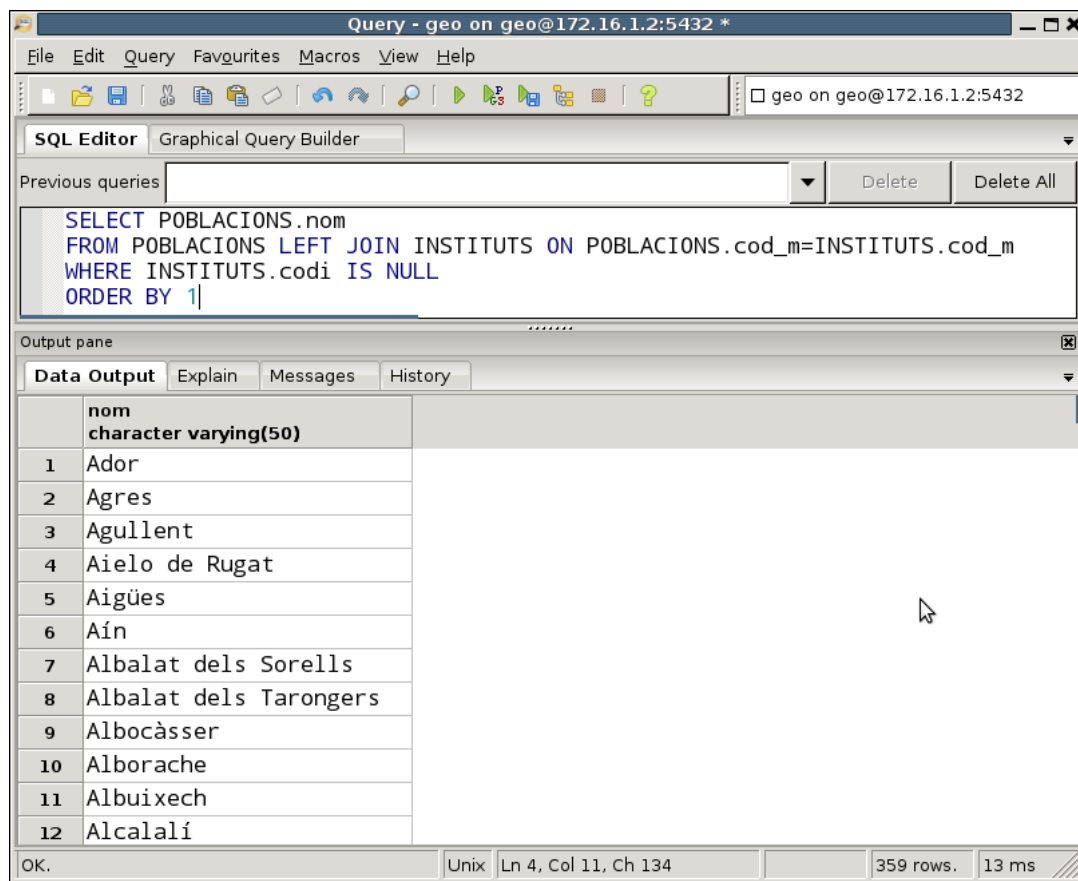
At the bottom of the window, it says "OK.", "Unix", "Ln 4, Col 11, Ch 151", "542 rows.", and "20 ms".

Otra variante también interesante es hacer una consulta similar para sacar los pueblos que no tienen instituto. Tendremos que hacer una combinación externa, y en la condición poner justamente que uno de los campos de la tabla INSTITUTOS sea nulo (por ejemplo, la clave principal):

Esta será la consulta, donde hemos vuelto a ordenar por el nombre de la población:

```
SELECT POBLACIONES.nom
FROM POBLACIONES LEFT JOIN INSTITUTS ON POBLACIONES.cod_m = INSTITUTS.cod_m
WHERE INSTITUTS.codi IS NULL
ORDER BY 1
```

Y este será el resultado:



ejemplos

1. Sacar todas las comarcas con el número de pueblos que tiene cada una, incluso aquellas comarcas que no tengan ningún pueblo.

Este ejemplo es poco ilustrativo, porque no tenemos en principio ninguna Camarca que no tenga pueblos. De todas formas, la manera sería haciendo un LEFT JOIN entre COMARCAS y POBLACIONES, para luego agrupar por comarca y contar las poblaciones. Observe como también podemos utilizar la sintaxis del **USING** en el LEFT JOIN.

```
SELECT COMARQUES.nom_c, COUNT (cod_m)
FROM COMARCAS LEFT JOIN POBLACIONES USING (nom_c)
GROUP BY COMARQUES.nom_c
ORDER BY 1;
```



Ejercicios Apartado 2.2.3

6.6 0 Sacar el código y el nombre de los clientes que no tienen ninguna factura.

6.6 1 Sacar el código, descripción y total de unidades vendidas de todos los artículos, incluso de aquellos que no se ha vendido nada.

Nota

Para dejarlo más bonito, como la suma de valores nulos no es 0 sino nulo, para que nos aparezca el valor 0 podemos utilizar la función COALESCE (*valor* , 0), que si el valor es nulo devuelve un 0.

6.62 Sacar el nombre de todos los pueblos y el número de clientes en caso de que las haya. Ordenar por número de clientes de forma descendente.

6.6 3 Sacar el código y la descripción de las categorías, con el número de artículos de cada categoría y el número total de unidades vendidas de cada categoría, de aquellas categorías de las que tenemos más de 15 artículos, y ordenado por número de artículos de forma descendente. Esta sentencia ya es bastante complicada. Concretamente deberá tener en cuenta que:

- Querremos sacar el número de artículos de cada categoría, pero quizás algunos artículos no se han vendido, y por tanto no aparecerán en la tabla LINIA_FAC.
- Y también tenemos el problema de que, como nos hace falta la mesa LINIA_FAC, un artículo vendido en más de una factura aparecerá más de una vez. Si contamos de forma normal, el contaríamos más de una vez cada artículo. Por lo tanto queremos contar los diferentes artículos de cada categoría.

Una subconsulta es una consulta dentro de otra consulta. Esta subconsulta puede tener todos los elementos que hemos visto hasta ahora.

El lugar donde poner una subconsulta dentro de la consulta principal puede ser en la cláusula WHERE o en la cláusula HAVING (formando parte de una condición) o en el FROM, y debe ir entre paréntesis. Incluso se puede poner en el mismo SELECT, es decir, en las columnas que van tras el SELECT.

- Si va en el **FROM**, la subconsulta será el origen de los datos, y por tanto se ejecutará antes y proporcionará los datos para la consulta principal.
- Si va en el **WHERE** o **HAVING** formará parte de una condición, y así podremos comparar en la consulta principal un campo con el que vuelva la subconsulta, por ejemplo. Aparte de las comparaciones normales que ya hemos visto en el WHERE o HAVING, podremos poner algunos operadores y predicados especiales, como veremos más adelante.
- Si va en el mismo **SELECT** normalmente será para sacar un resultado global que no afecta al resto de la consulta

Sintaxis en el FROM

```
SELECT ...  
FROM ( Subconsulta ) AS Nom_Subconsulta
```

Vamos a poner un ejemplo para entenderlo. Queremos sacar la media de pueblos por comarca. Lo podemos hacer de la siguiente manera: primero contamos cuántos pueblos hay en cada comarca, y una vez calculado esto sacamos la media.

```
SELECT AVG (cuantos)  
FROM (SELECT COUNT (*) AS cuantos  
      FROM POBLACIONES  
      GROUP BY nom_c) AS S;
```

Obsérvese que es necesario ponerle un alias en la columna de la subconsulta (cuantos) para poder hacer referencia a ella en la consulta principal. Y por otra parte, en PostgreSQL las subconsultas que el FROM deben tener obligatoriamente un alias. Si no pusimos ... **AS S** (o cualquier otro nombre) nos daría error.

Como veis, ya tiene un nivel de complejidad más que aceptable. Siempre se ejecuta primero la subconsulta y con los datos que proporciona, se ejecuta la consulta principal. Por el grado de complejidad es muy recomendable ir de dentro hacia fuera, es decir, pensar bien la subconsulta, incluso ejecutarla para ver si saca el que necesitamos (en el ejemplo ver si saca el número de poblaciones de cada una de las 34 comarcas), y cuando estemos seguros de que funciona bien crear la consulta principal.

Sintaxis en el WHERE o HAVING

```
SELECT ...  
FROM Tabla  
WHERE campo operador ( Subconsulta )
```

Podemos observar que lo que haremos será comparar algún campo de la tabla (o una expresión con alguna función) con el resultado que viene de la subconsulta.

Antes de ver qué podemos poner como operador o como campo o incluso ver unos predicados que podremos utilizar, pondremos un ejemplo, para clarificar las cosas. Intentaremos sacar las comarcas con una altura superior a la media. Calcular la media de las alturas es fácil, y será la subconsulta. Lo que haremos será comparar la altura de cada población con esta media.

```
SELECT *  
FROM POBLACIONES  
WHERE altura > (SELECT AVG (altura)  
               FROM POBLACIONES)
```

Si ejecuta la consulta, verá que la altura de todos los pueblos es superior a 300.44 que es la altura media (aproximadamente, porque lo calcula con mucha precisión)

No hay ningún problema en poner dos veces la misma mesa. Los campos se refieren a la mesa más cercana. Y funciona perfectamente porque la subconsulta nos devuelve un único valor, el promedio de alturas, y en la consulta principal se compara cada altura con este valor. Posteriormente veremos cómo solucionar el problema de que la subconsulta devuelva más de un valor.

Los operadores de la condición pueden ser de 3 tipos:

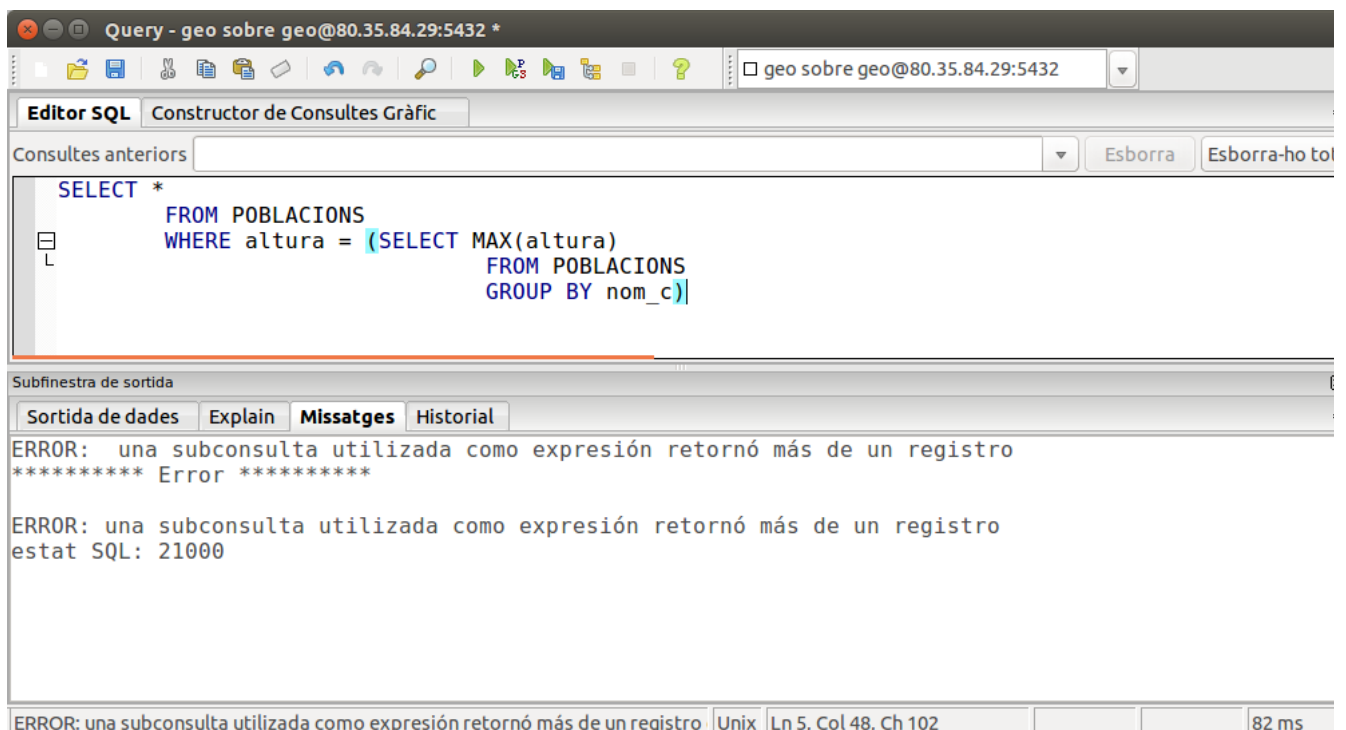
- **De comparación** . Es como el ejemplo de arriba, pero con cualquier operador de comparación. Se compara el campo (o la expresión) con el resultado de la subconsulta. Si la subconsulta sólo devuelve un valor, no hay más problema, pero si vuelve más de un valor (más de una fila) de momento sería incorrecto (no se puede comparar un campo con varios valores). Pongamos otro ejemplo para ilustrar. Sacar la población más alta se podría hacer de esta manera.

```
SELECT *
FROM POBLACIONES
WHERE altura = (SELECT MAX (altura)
                FROM POBLACIONES)
```

No hay problema para que la subconsulta devuelva un valor. Pero vamos a complicarse la vamos a ver las poblaciones más altas de cada comarca. Podríamos intentarlo de esta manera:

```
SELECT *
FROM POBLACIONES
WHERE altura = (SELECT MAX (altura)
                FROM POBLACIONES
                GROUP BY nom_c)
```

Pero nos daría el siguiente error:



Y es que la subconsulta devuelve 34 valores (uno por cada comarca), y de esta manera no se puede comparar el valor de la izquierda del igual con los 34 valores de la derecha. Para solucionar el problema de cuándo vuelve más de un valor podemos utilizar los predicados **ALL** , **ANY** , **SOME** .

- Si utilizamos **ALL** el resultado será cierto si la comparación es cierta con **TODOS** los valores que devuelve la subconsulta.
- Si utilizamos **ANY** o **SOME** (que son sinónimos) el resultado será cierto si la comparación es cierta con **ALGUNO** valor de la subconsulta .

En nuestro ejemplo, seguramente nos convendría **ANY**

```
SELECT *
FROM POBLACIONES
WHERE altura = ANY (SELECT MAX (altura)
                   FROM POBLACIONES
                   GROUP BY nom_c)
```

Esta consulta no funcionará bien del todo, ya que seleccionará todas las poblaciones que coinciden con alguna de las alturas máximas, sean de su comarca o no. Así por ejemplo, la altura máxima de la comarca de la **Plana Alta** es la **Serratella**, con 781 metros, que efectivamente aparece en el listado. Pero también aparece **Castellón** , que tiene una altura de 30 metros. Y aparece porque la altura máxima de la comarca **Ribera Baja** da la casualidad que es 30 metros (Almussafes). Ya dependeremos a hacer bien esta consulta en los ejemplos posteriores, pero para el hecho de comparar con muchos valores nos va bien.

- **El operador IN** . No será problema de que la subconsulta devuelva un valor o muchos. La condición será cierta si el valor del campo (o de la expresión) está entre la lista de valores que devuelve la subconsulta. También pueden utilizar NOT IN, y la condición será cierta cuando el valor del campo no está en la lista. Por ejemplo, otra manera de sacar las poblaciones que no tienen instituto, que la vista en las combinaciones externas. En la subconsulta sacamos los códigos de municipio de la mesa INSTITUTOS, y por tanto son los pueblos que tienen instituto, y en la consulta principal queremos los que no están en esta lista

```
SELECT *
FROM POBLACIONES
WHERE cod_m NOT IN (SELECT cod_m
                    FROM INSTITUTOS)
```

- **El operador EXISTS** . Es seguramente el más incómodo. No se compara un campo (o expresión) con la subconsulta, sino únicamente se pone [NOT] EXISTS (*subconsulta*) . La condición será cierta si la subconsulta devuelve **alguna fila** , y no será cierta si no vuelve ninguna fila. Intentamos hacer el mismo ejemplo de antes, el de los pueblos sin instituto. Tenemos que conseguir que la subconsulta no tenga ninguna fila en el caso de los que no tienen instituto. De palabra lo podemos decir así: queremos los pueblos para los que no existe ninguna fila en INSTITUTOS con el mismo código de municipio. Ahora ya se puede intuir por donde van los tiros:

```
SELECT *
FROM POBLACIONES
WHERE NOT EXISTS (SELECT *
                  FROM INSTITUTOS
                  WHERE cod_m = POBLACIONES.cod_m)
```

mirad como si en la subconsulta ponemos un campo (en el ejemplo cod_m), si el campo es de la tabla (o tablas) de la subconsulta, se referirá a él, por eso si queremos hacer referencia a un campo de la tabla o tablas de la consulta principal tenemos que poner el nombre de la tabla delante.

Sintaxis en el SELECT

```
SELECT ... ( Subconsulta )
FROM Tabla
```

Vamos a poner también un ejemplo para entenderlo. Vamos a calcular la diferencia de la altura de cada población con la media. La media es un resultado global independiente del resto de la consulta, que en este caso es muy sencilla porque debemos coger información simple de las poblaciones. La subconsulta también es muy sencilla, porque sólo tenemos que calcular la media de alturas).

```
SELECT nombre, altura, altura - (SELECT AVG (altura) FROM POBLACIONES)
FROM POBLACIONES
```

ejemplos

1. **Sacar la altura media de comarca más grande y la más pequeña.**

Nos hace falta previamente la altura media de cada comarca, y esto será la subconsulta. No olvidemos poner un alias en el campo de la subconsulta, para poder hacer referencia en la consulta principal. Y no olvidemos tampoco que las subconsultas en el FROM deben tener alias.

```
SELECT MAX (media), MIN (media)
FROM (SELECT AVG (altura) AS media
      FROM POBLACIONES
      GROUP BY nom_c) AS S;
```

2. **Sacar toda la información de las poblaciones que tienen más de 5 institutos.**

Podemos pensar en una subconsulta donde estén los códigos de municipio de las poblaciones que tienen más de 5 institutos (se consulta en la teja INSTITUTOS agrupando por codi_m y contando el número de filas para que sea mayor que 5).

```
SELECT *
FROM POBLACIONES
WHERE cod_m IN (SELECT cod_m
                FROM INSTITUTOS
                GROUP BY cod_m
                HAVING count (*) > 5)
```

3. **Sacar toda la información de la población más alta y la más baja.**

Nos planteamos 2 subconsultas, la que saca la altura máxima y la que saca la altura mínima (en una única subconsulta nos devolvería valores en 2 columnas, y estaría más complicado). Sencillamente será sacar toda la información de las poblaciones que tienen una altura igual al que vuelve una subconsulta o al que vuelve la otra.

```
SELECT *
  FROM POBLACIONES
 WHERE altura = (SELECT MAX (altura)
                  FROM POBLACIONES)
        OR altura = (SELECT MIN (altura)
                      FROM POBLACIONES);
```

4. Sacar la población más alta de cada comarca.

La dificultad está en que debe ser la máxima de las alturas de su comarca. Por tanto, en la subconsulta debemos hacer referencia a la comarca en cuestión. Como siempre estamos tratando la mesa POBLACIONES, tanto en la consulta como en la subconsulta, tendremos que poner un nombre a la de la consulta principal, para poder hacer referencia a ella desde la subconsulta.

```
SELECT *
  FROM POBLACIONES T1
 WHERE altura = (SELECT MAX (altura)
                  FROM POBLACIONES
                  WHERE nom_c = T1.nom_c);
```

En el resultado obtenemos 35 poblaciones, cuando sólo hay 34 comarcas. La razón es que al Alcoyano, hay 2 poblaciones con la altura máxima (816 metros), por lo tanto el resultado es correcto

5. Obtener el nombre de la comarca y la provincia de las comarcas que tienen una altura media más alta que la media de todas las poblaciones.

La subconsulta será bastante sencilla: el promedio de alturas de las poblaciones. En la consulta principal deberemos agrupar por cada comarca y calcular la media de alturas de las poblaciones, y compararla con la media que nos viene de la subconsulta. Como pide también la provincia, nos hace falta también la tabla COMARCAS, y por tanto tendremos que reunir con POBLACIONES; en esta ocasión lo hemos hecho poniendo el aseo en el WHERE.

```
SELECT COMARQUES.nom_c, provincia, AVG (altura)
  FROM COMARCAS, POBLACIONES
 WHERE COMARQUES.nom_c = POBLACIONES.nom_c
 GROUP BY COMARQUES.nom_c, provincia
 HAVING AVG (altura) > (SELECT AVG (altura)
                       FROM POBLACIONES)
```

6. Sacar el nombre de las comarcas, el número de municipios y el porcentaje que supone respecto al total de municipios.

Toda la información la podemos sacar de una reunión entre las mesas COMARCAS y POBLACIONES (para contar cuántos pueblos hay en cada comarca), pero para poder calcular el porcentaje necesitamos el número total de poblaciones, que podemos calcular con una sencilla subconsulta. El lugar más cómodo es en el SELECT. La reunión la hemos hecho en esta ocasión con el USING.

```
SELECT COMARQUES.nom_c, provincia, COUNT (cod_m), COUNT (cod_m) * 100.0 / (SELECT COUNT (*)
  FROM POBLACIONES)
  FROM COMARCAS INNER JOIN POBLACIONES USING (nom_c)
 GROUP BY 1,2;
```



Ejercicios apartado 2.3

6.64 Sacar el número máximo de facturas hechas a un cliente

6.65 Sacar el importe que supone la factura más cara y el importe que supone la más barata (sin considerar ni descuentos ni IVA)

6.66 Sacar el número de facturas más alto que se ha vendido por vendedor en cada trimestre (no sacaremos quién es el vendedor, que sería aún más complicado). Para poder agrupar por trimestre, nos hará falta la función `to_char (fecha, 'Q')`, que saca el número de trimestre. El paso previo es calcular el número de facturas de cada vendedor y en cada trimestre. Después, con la información anterior, queremos calcular el máximo de cada trimestre.

6.67 Sacar los artículos más caros que la media. Saquéis ordenados por la categoría, y luego por código de artículo.

6.68 Modificar el anterior para sacar los artículos más caros que la media de su categoría. Saquéis ordenados por categoría

6.69 Sacar los pueblos donde tenemos clientes pero no tenemos vendedores. Debe ser por medio de subconsultas (en plural). Ordene por código del pueblo.

2.4 Consultas de operaciones de conjuntos

Agruparemos en este apartado las consultas que tratan conjuntos de filas para realizar operaciones de algebra de conjuntos: **unión** , **intersección** y **diferencia** de conjuntos

Totero estas consultas juntan los resultados de dos o más consultas.

Sintaxis de la UNIÓN

```
[TABLE] consulta1
UNION [ALL]
[TABLE] consulta2 ...
```

Cada una de las consultas puede ser una tabla (poniendo la palabra **TABLE** delante) o el nombre de una consulta ya guardada, aunque lo más habitual será poner directamente la **sentencia SQL** .

Los requisitos son que las dos (o más) consultas devuelven el mismo número de campos, y que sean sino del mismo tipo, sí de tipo compatibles

Al igual que en la unión de conjuntos, el resultado serán todas las filas de las dos (o más) consultas individuales, pero sin repetir filas, es decir, si de las dos consultas obtienen filas iguales, éstas sólo saldrán una vez. Lo anterior se puede evitar si ponemos el predicado **ALL**, y entonces sí que saldrán las filas repetidas.

Los nombres de los campos vendrán dados por la primera consulta.

Si queremos ordenar por algún campo, lo tendremos que poner al final de la última consulta, pero refiriéndose en todo caso a los campos de la primera consulta (lo podemos evitar poniendo el número de orden en el **ORDER BY**)

ejemplo

1. Queremos ver en un único resultado tanto el nombre de las comarcas como el nombre de las poblaciones, siempre con el nombre de la provincia junto

```
SELECT nom_c, provincia
  FROM COMARCAS
UNION
SELECT nombre, provincia
  FROM COMARCAS INNER JOIN POBLACIONES USING (nom_c)
ORDER BY nom_c;
```

Como curiosidad, saldrán 575 filas, pero si pusiéramos **UNION ALL** nos saldrían 576. Esto es porque la comarca de la ciudad de Valencia se llama Valencia y está en la provincia de Valencia. Por lo tanto es una fila que aparecerá tanto en la primera como en la segunda consulta. Si hacemos **UNION** no se repetirá, pero si hacemos **UNION ALL** sí se repetirá.

Sintaxis de la INTERSECCIÓN

Es idéntica a la unión, pero poniendo la palabra **INTERSECT** , y servirá para sacar únicamente las filas que están en las dos consultas.

```
[TABLE] consulta1
INTERSECT [ALL]
[TABLE] consulta2 ...
```

Igual que antes, cada una de las consultas puede ser una tabla (poniendo la palabra **TABLE** delante), y tenemos el requisito de que las dos (o más) consultas devuelven el mismo número de campos, y de tipo compatibles.

En principio no saldrán filas repetidas, a menos que pongamos **ALL**

ejemplo

1. Como en el ejemplo de la unión habíamos visto que la fila Valencia Valencia salía en las 2 consultas, vamos a comprobar que aparece en la intersección:

```
SELECT nom_c, provincia
  FROM COMARCAS
INTERSECT
SELECT nombre, provincia
  FROM COMARCAS INNER JOIN POBLACIONES USING (nom_c)
ORDER BY nom_c;
```

Sintaxis de la DIFERENCIA

Es idéntica a las anteriores, pero poniendo la palabra **EXCEPT** , y servirá para sacar las filas que están en la primera consulta pero que no están en la segunda.

```
[TABLE] consulta1  
EXCEPT [ALL]  
[TABLE] consulta2 ...
```

Igual que antes, cada una de las consultas puede ser una tabla (poniendo la palabra **TABLE** delante), y tenemos el requisito de que las dos (o más) consultas devuelven el mismo número de campos, y de tipo compatibles.

En principio no saldrán filas repetidas, a menos que pongamos **ALL**

ejemplo

1. Aprovechamos el mismo ejemplo de antes para comprobar que con EXCEPT no saldrá la comarca Valencia, ya que hay una fila idéntica en la segunda consulta:

```
SELECT nom_c, provincia  
FROM COMARCAS  
EXCEPT  
SELECT nombre, provincia  
FROM COMARCAS INNER JOIN POBLACIONES USING (nom_c)  
ORDER BY nom_c;
```



Ejercicios apartado 2.4

- 6.70** Sacar el nombre de todos los clientes y vendedores implicados en alguna venta del primer trimestre de 2015.
- 6.71** Sacar por medio de sentencias de operaciones de conjuntos los pueblos donde tenemos algún vendedor o algún cliente. No queremos resultados repetidos, y lo queremos ordenado por el nombre del pueblo.
- 6.71b** Modificar el anterior para sacar los pueblos donde tenemos al mismo tiempo vendedores y clientes
- 6.71c** Modificar el anterior para sacar los pueblos donde tenemos vendedores pero no tenemos clientes

Ejercicios de todo el tema, con los resultados

06:51 Sacar el nombre de los clientes con el número de factura y la fecha (individuales, sin agrupar nada) que tiene cada cliente. Saca el resultado ordenado por cliente, y dentro de éste por fecha de la factura

	nom character varying(100)	num_f numeric(5,0)	data date
1	ADELL GALMES, MERCEDES ROSARIO	6675	2015-09-04
2	ADELL VILLALONGA, LUIS JOSE	6586	2015-04-10
3	ADELL VILLALONGA, LUIS JOSE	6688	2015-09-21
4	ADELL VILLALONGA, LUIS JOSE	6736	2015-12-05
5	AMO MONTSN, RAMON FERNANDO	6550	2015-02-03
6	AMO MONTSN, RAMON FERNANDO	6678	2015-09-07
7	BADENES CEPRIA, ANDRES RICARDO	6584	2015-04-07
8	BELTRAN MENEU, CRISTINA	6604	2015-05-04
9	BELTRAN MENEU, CRISTINA	6633	2015-06-30
10	BELTRAN MUNYOZ, JAIME VICENTE	6674	2015-09-02
11	BELTRAN MUNYOZ, JAIME VICENTE	6670	2015-09-07

Un total de **105** filas

6.5 2 Sacar el nombre del socio, con el código y la descripción de cada artículo que ha pedido. Ordenar por nombre del socio y código del artículo.

	nom character varying(100)	cod_a character varying(10)	descrip character varying(50)
1	ADELL GALMES, MERCEDES ROSARIO	L85457	Marco Legrand 2
2	ADELL GALMES, MERCEDES ROSARIO	L85685	Tecla Base Legra
3	ADELL GALMES, MERCEDES ROSARIO	LAPC	Reactancia 40 W
4	ADELL GALMES, MERCEDES ROSARIO	LAR40125	Bloque Emergenci
5	ADELL GALMES, MERCEDES ROSARIO	TC23	Tubo Fercondur
6	ADELL GALMES, MERCEDES ROSARIO	TNF21	Tubo Sapa 11
7	ADELL GALMES, MERCEDES ROSARIO	ZN5108B	Base Plastimetal
8	ADELL VILLALONGA, LUIS JOSE	CN1X10	Paralelo 2 X 1
9	ADELL VILLALONGA, LUIS JOSE	IM3P15V	Interruptor Magn
10	ADELL VILLALONGA, LUIS JOSE	IMFN25L	Cartucho Fusible
11	ADELL VILLALONGA, LUIS JOSE	IMFN25L	Cartucho Fusible
12	ADELL VILLALONGA, LUIS JOSE	L58068	Base Enchufe T.t

Un total de **541** filas

6.5 3 Modificar el anterior para que no se repitan los resultados.

	nom character varying(100)	cod_a character varying(10)	descrip character varying(50)
1	ADELL GALMES, MERCEDES ROSARIO	L85457	Marco Legrand 2
2	ADELL GALMES, MERCEDES ROSARIO	L85685	Tecla Base Legra
3	ADELL GALMES, MERCEDES ROSARIO	LAPC	Reactancia 40 W
4	ADELL GALMES, MERCEDES ROSARIO	LAR40125	Bloque Emergenci
5	ADELL GALMES, MERCEDES ROSARIO	TC23	Tubo Fercondur
6	ADELL GALMES, MERCEDES ROSARIO	TNF21	Tubo Sapa 11
7	ADELL GALMES, MERCEDES ROSARIO	ZN5108B	Base Plastimetal
8	ADELL VILLALONGA, LUIS JOSE	CN1X10	Paralelo 2 X 1
9	ADELL VILLALONGA, LUIS JOSE	IM3P15V	Interruptor Magn
10	ADELL VILLALONGA, LUIS JOSE	IMFN25L	Cartucho Fusible
11	ADELL VILLALONGA, LUIS JOSE	L58068	Base Enchufe T.t
12	ADELL VILLALONGA, LUIS JOSE	L76179	Placa 1 F Legra

Un total de **532** filas

Observa como ahora no se repite la fila 10 y 11, y antes sí

6.5 4 Sacar el nombre de los clientes con la cantidad de facturas que tienen, ordenadas por este número de mayor a menor

	nom character varying(100)	Número de factures bigint
1	LOPEZ RINCON, LUIS MIGUEL	5
2	BOTELLA CATALA, JUAN	5
3	PINEL HUERTA, VICENTE	4
4	HERRERA SALA, ANA	4
5	MIGUEL ARCHILES, OSCAR RAMON	4
6	VILLALONGA RAMIREZ, DIEGO SERGIO	4
7	NAVARRO BARBERO, MARIA LLEDO	4
8	HUGUET PERIS, JUAN ANGEL	4
9	CANCELAS MORA, MARIA	3
10	GALLEN HUERTA, OLGA	3
11	LOPEZ GUITART, XAVIER	3

Un total de 40 filas

6.5 5 Sacar el número de factura, fecha, código de cliente, total de la factura (con el alias IMPORTE) y total de la factura aplicando descuentos de artículo (con alias DESCOMPTE_1), como en la consulta 6:33 , pero sin el límite de las 10 líneas de factura. Ordenar por número de factura.

	num_f numeric(5,0)	data date	cod_cli numeric(5,0)	import numeric	descompte_1 numeric
1	6535	2015-01-01	306	11.78	8.83500000
2	6538	2015-01-11	345	50.30	37.17000000
3	6539	2015-01-14	357	234.59	234.59000000
4	6542	2015-01-18	375	209.82	209.82000000
5	6547	2015-01-26	72	220.32	176.25600000
6	6549	2015-02-01	315	972.43	972.43000000
7	6550	2015-02-03	261	669.87	502.40250000
8	6552	2015-02-10	78	102.07	93.14700000
9	6553	2015-02-10	387	501.11	501.11000000
10	6554	2015-02-12	306	0.90	0.67500000
11	6557	2015-02-15	318	49.76	39.80800000

Un total de 105 filas

6.5 6 Modificar el anterior para aplicar también el descuento de la factura (con el alias DESCOMPTE_2)

	num_f numeric(5,0)	data date	cod_cli numeric(5,0)	import numeric	descompte_1 numeric	descompte_2 numeric
1	6535	2015-01-01	306	11.78	8.83500000	7.95150000
2	6538	2015-01-11	345	50.30	37.17000000	33.45300000
3	6539	2015-01-14	357	234.59	234.59000000	175.94250000
4	6542	2015-01-18	375	209.82	209.82000000	188.83800000
5	6547	2015-01-26	72	220.32	176.25600000	176.25600000
6	6549	2015-02-01	315	972.43	972.43000000	777.94400000
7	6550	2015-02-03	261	669.87	502.40250000	401.92200000
8	6552	2015-02-10	78	102.07	93.14700000	93.14700000
9	6553	2015-02-10	387	501.11	501.11000000	400.88800000
10	6554	2015-02-12	306	0.90	0.67500000	0.60750000
11	6557	2015-02-15	318	49.76	39.80800000	19.90400000

Un total de 105 filas

6.5 7 Sacar el código y nombre de aquellos vendedores que supervisan alguien (constan como cabeza). Sacar también el número de supervisados de cada uno de estos supervisores.

	cod_ven numeric(5,0)	nom character varying(100)	count bigint
1	5	GUILLÉN VILLALONG	2
2	105	POY OMELLA, PALOM	3
3	255	DANIEL MIRALLES,	2
4	405	ROCA FAURA, ANTON	2

6.5 8 Sacar el código y descripción de los artículos junto con el número de veces que se ha vendido, el total de unidades vendidas y la media de unidades vendidas por factura. Ordenar por número total de unidades vendidas de forma descendente, y dentro de ésta por código de artículo de forma ascendente.

	cod_a character varying(10)	descrip character varying(50)	count bigint	sum numeric	avg numeric
1	L76104	Pulsador Pus Leg	4	36	9.0000
2	L16500	Cartucho Fusible	3	26	8.6666
3	LA2760EC	Reactancia Vapor	3	26	8.6666
4	L76138	Pieza 4 E. Legra	3	25	8.3333
5	L85393	Toma Desplazada	3	24	8.0000
6	B10010B	Interruptor Bjc	2	23	11.500
7	L85510	Marco Legrand	2	23	11.500
8	N5088	Cortacircuitos N	2	23	11.500
9	ZN5104B	Base Enchufe Tt	3	23	7.6666
10	L55812	Interruptor 2 M.	3	22	7.3333
11	L85685	Tubo Base Legra	3	22	7.3333

Un total de 399 filas

6.5 9 Sacar el código y la descripción de las categorías, con la cantidad de artículos vendidos de cada categoría, de aquellas categorías de las que se han vendido más de 100 unidades. Ordenar por este número de forma descendente.

	cod_cat character varying(15)	descripcio character varying(50)	sum numeric
1	Legrand	Components marca	887
2	Niessen	Components Niese	201
3	IntMagn	Interruptor Magn	182
4	Simon	Components marca	148
5	Ticino	Components marca	117

6.6 0 Sacar el código y el nombre de los clientes que no tienen ninguna factura.

	cod_cli numeric(5,0)	nom character varying(100)
1	3	DAMBORENEA CORBATO, ALICIA
2	15	GUAL SALES, MARIA
3	120	CASTELLO DAMBORENEA, ENRIQUE JAVIER
4	282	MORA RIBES, ENRIQUE MIGUEL
5	330	MARTI MOLTO, CONCHITA
6	372	LOPEZ LLORENS, SANCHEZ MARCOS
7	378	GUIMERA AGOST, LUIS
8	381	GUILLOT BELDA, FRANCISCO JOSE
9	390	AZNAR MONFERRER, ADRIAN

6.6 1 Sacar el código, descripción y total de unidades vendidas de todos los artículos, incluso de aquellos que no se ha vendido nada.

Nota

Para dejarlo más bonito, como la suma de valores nulos no es 0 sino nulo, para que nos aparezca el valor 0 podemos utilizar la función COALESCE (valor , 0), que si el valor es nulo devuelve un 0.

	cod_a character	descrip character varying(50)	sum numeri
1	1967346	Prolongador Doble 2.5 M	8
2	2309987	Prolongador Doble 5.0 M	4
3	3204209	Prolongador Cuaduple 2.5 M	0
4	3987348	Prolongador Cuaduple 20.0 M	4
5	4283200	Prolongador Doble 10.0 M	6
6	4565467	Prolongador Sencillo 10.0 M	0
7	6579878	Prolongador Sencillo 5.0 M	0
8	7897999	Prolongador Sencillo 2.5 M	0
9	8340098	Prolongador Cuaduple 5.0 M	8
10	8394800	Prolongador Cuaduple 10.0 M	2
11	AA3755	Tubo Fluorescente 15 W	0

Un total de 812 filas

6.62 Sacar el nombre de todos los pueblos y el número de clientes en caso de que las haya. Ordenar por número de clientes de forma descendente.

	cod_pob numeric(5,0)	nom character varying(50)	count bigint
1	53596	VILLARREAL	10
2	12309	CASTELLON	9
3	7766	BURRIANA	7
4	32093	NULES	4
5	48037	TORO (EL)	1
6	2050	ALTERO DE MOMPOY	1
7	5495	BASANOVA (LA)	1
8	37953	PUEBLA DE SAN MIGUE	1
9	33246	PAIPORTA	1
10	32101	NURTAL-CDA. FARANDO	1
11	29149	MIÑANA	1
12	7625	BUGARRA	1
13	46332	SONEJA	1
14	31982	NOVELE	1
15	11024	CARRASCA (LA)	1
16	2814	ARAYA	1
17	39063	RAIGUERO (EL)	1
18	28097	MAS DEL SECO	1
19	1651	ALGAR (EL)	1
20	49180	TURIS	1
21	45004	SAX	1
22	48192	TORRE-BELTRAN	1
23	17859	ENRAMONA	1
24	2539	ANNA	0
25	54092	VTSTABELIA DEL MAES	0

Un total de **1663** filas

6.6 3 Sacar el código y la descripción de las categorías, con el número de artículos de cada categoría y el número total de unidades vendidas de cada categoría, de aquellas categorías de las que tenemos más de 15 artículos, y ordenado por número de artículos de forma descendente. Esta sentencia ya es bastante complicada. Concretamente deberá tener en cuenta que:

- Querremos sacar el número de artículos de cada categoría, pero quizás algunos artículos no se han vendido, y por tanto no aparecerán en la tabla LINIA_FAC.
- Y también tenemos el problema de que, como nos hace falta la mesa LINIA_FAC, un artículo vendido en más de una factura aparecerá más de una vez. Si contamos de forma normal, el contaríamos más de una vez cada artículo. Por lo tanto queremos contar los diferentes artículos de cada categoría.

	cod_cat character varying(15)	descripcio character varying(50)	count bigint	sum numeric
1	Legrand	Components marca Le	189	887
2	IntMagn	Interruptor Magneto	55	182
3	Niessen	Components Niesen S	43	201
4	Ticino	Components marca Ti	34	117
5	Simon	Components marca Si	27	148

6.64 Sacar el número máximo de facturas hechas a un cliente

	Número màxim de factures bigint
1	5

6.65 Sacar el importe que supone la factura más cara y el importe que supone la más barata (sin considerar ni descuentos ni IVA)

	Màxim iport numeric	Mínim import numeric
1	1542.00	0.51

6.66 Sacar el número de facturas más alto que se ha vendido por vendedor en cada trimestre (no sacaremos quién es el vendedor, que sería aún más complicado). Para poder agrupar por trimestre, nos hará falta la función **to_char (fecha, 'Q')**, que saca el número de trimestre. El paso previo es calcular el número de facturas de cada vendedor y en cada trimestre. Después, con la información anterior, queremos calcular el máximo de cada trimestre.

	trim text	max bigint
1	1	5
2	2	6
3	3	7
4	4	4

6.67 Sacar los artículos más caros que la media. Saquéis ordenados por la categoría, y luego por código de artículo.

	cod_a character varying(10)	descrip character varying(50)	preu numeric(6,2)	stock numeric(4,0)	stock_min numeric(4,0)	cod_cat character varying(15)
1	B10007B	Placa 2 E. Bjc	68.29	1	1	BjcOlimpia
2	F05/16	Interruptor Dif	27.65	1	1	IntDif
3	F05/17	Interruptor Dif	20.73	1	1	IntDif
4	F05/50	Interruptor Dif	25.24	1	1	IntDif
5	HICH36	Interruptor Mag	28.58	1	1	IntMagn
6	ID22530	Interruptor Mag	47.60	1	1	IntMagn
7	ID24030	Interruptor Mag	50.39	1	1	IntMagn
8	ID42530	Interruptor Mag	87.36	1	1	IntMagn
9	ID425300	Interruptor Mag	73.83	1	1	IntMagn
10	ID44030	Interruptor Mag	94.18	1	1	IntMagn
11	ID440300	Interruptor Mag	79.63	1	1	IntMagn
12	ID46330	Interruptor Mag	204.34	1	1	IntMagn
13	ID463300	Interruptor Mag	105.06	1	1	IntMagn
14	IM2F15	Interruptor Mag	16.41	1	1	IntMagn
15	IM2F25	Interruptor Mag	16.41	1	1	IntMagn
16	IM2F30	Interruptor Mag	17.20	1	1	IntMagn

Un total de **164** filas

6.68 Modificar el anterior para sacar los artículos más caros que la media de su categoría. Saquéis ordenados por categoría

	cod_a character varying(10)	descrip character varying(50)	preu numeric(6,2)	stock numeric(4,0)	stock_min numeric(4,0)	cod_cat character varying(15)
1	B10007B	Placa 2 E. Bjc	68.29	1	1	BjcOlimpia
2	F05/16	Interruptor Dif	27.65	1	1	IntDif
3	F05/17	Interruptor Dif	20.73	1	1	IntDif
4	F05/50	Interruptor Dif	25.24	1	1	IntDif
5	ID22530	Interruptor Mag	47.60	1	1	IntMagn
6	ID24030	Interruptor Mag	50.39	1	1	IntMagn
7	ID42530	Interruptor Mag	87.36	1	1	IntMagn
8	ID425300	Interruptor Mag	73.83	1	1	IntMagn
9	ID44030	Interruptor Mag	94.18	1	1	IntMagn
10	ID440300	Interruptor Mag	79.63	1	1	IntMagn
11	ID46330	Interruptor Mag	204.34	1	1	IntMagn
12	ID463300	Interruptor Mag	105.06	1	1	IntMagn
13	IM3P50U	Interruptor Mag	51.39	1	1	IntMagn
14	im4P10L	Interruptor Mag	32.60	1	1	IntMagn
15	im4P10V	Interruptor Mag	39.15	1	1	IntMagn
16	im4P25L	Interruptor Mag	32.60	1	1	IntMagn

Un total de **75** filas

Es un resultado muy similar al anterior, pero obsérvese que ahora no están los productos de las filas 5, 14, 15, ...

6.69 Sacar los pueblos donde tenemos clientes pero no tenemos vendedores. Debe ser por medio de subconsultas (en plural). Ordene por código del pueblo.

	cod_pob numeric(5,0)	nom character varying(50)	cod_pro numeric(2,0)
1	1651	ALGAR (EL)	3
2	2050	ALTERO DE MOMPO	46
3	2814	ARAYA	12
4	5495	BASANOVA (LA)	12
5	7625	BUGARRA	46
6	7766	BURRIANA	12
7	11024	CARRASCA (LA)	46
8	17859	ENRAMONA	12
9	28097	MAS DEL SECO	12
10	29149	MIÑANA	3
11	31982	NOVELE	46
12	32093	NULES	12
13	32101	NURTAL - CDA. FAR	46
14	33246	PAIPORTA	46
15	37953	PUEBLA DE SAN M	46
16	39063	RAIGUERO (EL)	3
17	45004	SAX	3
18	46332	SONEJA	12
19	48037	TORO (EL)	12
20	48192	TORRE-BELTRAN	12
21	49180	TURIS	46
22	53596	VILLARREAL	12

6.70 Sacar el nombre de todos los clientes y vendedores implicados en alguna venta del primer trimestre de 2015. Intentar sacar en una segunda columna el texto **Vendedor** para los vendedores, y **Cliente** para los clientes. Ordenado por el nombre.

	nom character varying(100)	?column? text
1	AGOST TIRADO, JORGE VICTOR	Vendedor
2	AMO MONTSN, RAMON FERNANDO	Client
3	CANCELAS MORA, MARIA	Client
4	CORBATO CARUANA, JOSE JUSTO	Vendedor
5	DANIEL MIRALLES, SERGIO	Vendedor
6	GUILLEN VILLALONGA, NATALIA	Vendedor
7	HUGUET PERIS, JUAN ANGEL	Client
8	IGLESIAS NAVARRO, IGNACIO	Client
9	LOPEZ BOTELLA, MAURO	Client
10	LOPEZ GUITART, XAVIER	Client
11	LOPEZ RINCON, LUIS MIGUEL	Client
12	MATEU MARTI, MARIA DOLORES	Client
13	NAVARRO BARBERO, MARIA LLEDO	Client
14	PEREZ CEBRIA, IGNACIO DIEGO	Vendedor
15	PINEL HUERTA, VICENTE	Client
16	PITARCH MONSONIS, MARIA CARMEN	Client
17	POY OMELLA, PALOMA	Vendedor
18	ROCA FAURA, ANTONIO DIEGO	Vendedor
19	RUBERT CANO, DIEGO GUILLERMO	Vendedor
20	SAMPEDRO SIMO, MARIA MERCEDES	Client
21	TICHELL MONLLEO, MARIA ANGELES	Client
22	TUR MARTIN, MANUEL FRANCISCO	Client
23	VIDAL DIEZ, JOSE	Vendedor
24	VILLALONGA RAMIREZ, DIEGO SERGIO	Client
25	VILLALONGA SANCHIS, MILAGROS	Client

6.71 Sacar por medio de sentencias de operaciones de conjuntos los pueblos donde tenemos algún vendedor o algún cliente. No queremos resultados repetidos, y lo queremos ordenado por el nombre del pueblo.

	nom character varying(50)
1	ALGAR (EL)
2	ALTERO DE MOMPO
3	ARAYA
4	BASANOVA (LA)
5	BENICARLO
6	BUGARRA
7	BURRIANA
8	CARRASCA (LA)
9	CASTELLON
10	ENRAMONA
11	LUCENA DEL CID

Un total de **31** filas

6.71b Modificar el anterior para sacar los pueblos donde tenemos al mismo tiempo vendedores y clientes

	nom character varying(50)
1	CASTELLON

6.71c Modificar el anterior para sacar los pueblos donde tenemos vendedores pero no tenemos clientes

	nom character varying(50)
1	BENICARLO
2	LUCENA DEL CID
3	PILAR DE LA HORADADA
4	SAN JUAN DE ALICANTE
5	USERAS
6	VALENCIA
7	VISTABELLA
8	VIVER

