Tema 6: SQL - Consulta básica (parte I-2)

12. Funciones de agregado

Las **funciones de agrega** t, o funciones de dominio agregado, son aquellas que sacan un resultado a partir de los valores de un determinado campo en un conjunto de filas. Así tendremos una función para **sumar** los valores de una columna, o **contar** -los, o sacar la **media** , o el **máximo** , ...

Actuarán sobre un conjunto de filas determinado, que en principio se supone que es toda la tabla (todas las filas de la tabla). En la siguiente pregunta, veremos que el conjunto de filas sobre el que se calcula una función de agregado, lo podremos cambiar con la cláusula **GROUP BY** .

sintaxis

- **COUNT (* | <expresión>)** : cuenta el número de filas; si se pone una columna o expresión, no se contarán los valores nulos.
- **SUM (<expresión>)** : devuelve la suma de la columna o expresión especificada. Ignorar los valores nulos.
- **AVG (<expresión>)** : calcula la media aritmética de la columna o expresión especificada. Ignorar los valores nulos.
- **VAR_SAMP (<expresión>)** : calcula la varianza de una muestra a partir de la columna o expresión especificada.
- **STDDEV (<expresión>)** : desviación típica de una muestra.
- **MAX (<expresión>)** : calcula el máximo.
- **MIN (<expresión>)** : calcula el mínimo.

Por ejemplo, si queremos saber el número de Institutos:

```
SELECT COUNT (*) AS "Número de Institutos"
    FROM INSTITUTOS;
```

Nota

Es interesante la utilización de alias, para que no aparezcan cabeceras como ***count***

ejemplos

1. Contar el número total de pueblos.

```
SELECT Count (*)
    FROM POBLACIONES;
```

2. Contar el número de poblaciones de la **Plana Alta** .

```
SELECT Count (*)
    FROM POBLACIONES
    WHERE nom_c = 'Plana Alta';
```

3. Calcular la media de habitantes de los pueblos de la **Plana Alta** y **Plana Baja** .

```
SELECT AVG (población)
    FROM POBLACIONES
    WHERE nom_c = 'Plana Alta' OR nom_c = 'Plana Baja'
```

4. Calcular la media de densidad de las poblaciones. La densidad se calcula como el número de habitantes dividido por la extensión.

```
SELECT AVG (población / extension)
    FROM POBLACIONES;
```

5. Calcular la altura máxima y mínima de todos los pueblos.

```
SELECT MAX (altura), MIN (altura)
    FROM POBLACIONES
```

Ejercicios apartado 12

06:15 Contar el número de **clientes** que tienen el **código postal nulo** .

6.16 Contar el número de veces que el artículo **L76104** entra en las líneas de factura, y el número total de unidades vendidas de este artículo. Sólo os hace falta la tabla LINIA_FAC.

6.17 Sacar la **media** del **stock** de los artículos.

6.17b Modificar el anterior para tener en cuenta los valores nulos, como si fueran 0 . Os vendrá bien la función COALESCE que convierte los nulos del primer parámetro al valor dado como segundo parámetro (si es diferente de nulo, deja igual el valor). Por lo tanto la debe utilizar de esta manera: COALESCE (stock, 0)

06:18 Contar **cuántas facturas** tiene el cliente **375**

19.06 Calcular el **descuento máximo** , el **mínimo** y el descuento **medio** de las **facturas** .

13. La cláusula GROUP BY

Agrupa todas las filas con valores iguales de una o de unas columnas

sintaxis

```
SELECT <columnas>
    FROM <tablas>
    GROUP BY <columnas>
```

Por cada fila con valores iguales de las columnas de la cláusula **GROUP BY** , saca sólo una, es decir, las agrupa.

Las ***funciones de agregado*** cogen todo su sentido y potencia combinadas con el **GROUP BY** : volverán un valor para cada grupo. Así, por ejemplo, esta sentencia sacará cuantas poblaciones hay en cada comarca:

```
SELECT COUNT (*)
    FROM POBLACIONES
    GROUP BY nom_c
```

Si queremos excluir filas para que no entren en las agrupaciones, lo haremos por medio de la cláusula **WHERE** . En este ejemplo nos aprovechamos del código de municipio, que en el caso de los municipios de la provincia de Castellón es 12000 y pico.

```
SELECT COUNT (*)
    FROM POBLACIONES
    WHERE cod_m> = 12000 and cod_m <13000
    GROUP BY nom_c
```

Cuando tenemos agrupaciones de filas, bien porque utilizamos la cláusula GROUP BY, bien porque entre las columnas que se eligen en el SELECT hay alguna función de agregado, o ambas cosas a la vez, se pueden cometer errores con una relativa facilidad . Cuando hay una agrupación **todas las columnas que seleccionamos con el SELECT deberán estar en el GROUP BY, o bien estar dentro de una función de agregado** . De lo contrario nos dará error. Por ejemplo, esta consulta funciona bien, y de hecho es mejor consulta que las de antes, ya que nos dice la comarca y el numero de pueblos que tiene cada una:

```
SELECT nom_c, COUNT (*)
    FROM POBLACIONES
    GROUP BY nom_c
```

Pero esta no:

```
SELECT nom_c, COUNT (*), cod_m
    FROM POBLACIONES
    GROUP BY nom_c
```

Es sintácticamente incorrecta, y aparte no tiene sentido: si agrupamos todos los pueblos de la misma comarca ¿cómo debemos poder sacar después el código del municipio de cada pueblo ?.

Como es relativamente fácil cometer este error, tendremos que identificar este error, para poder solucionarlo.

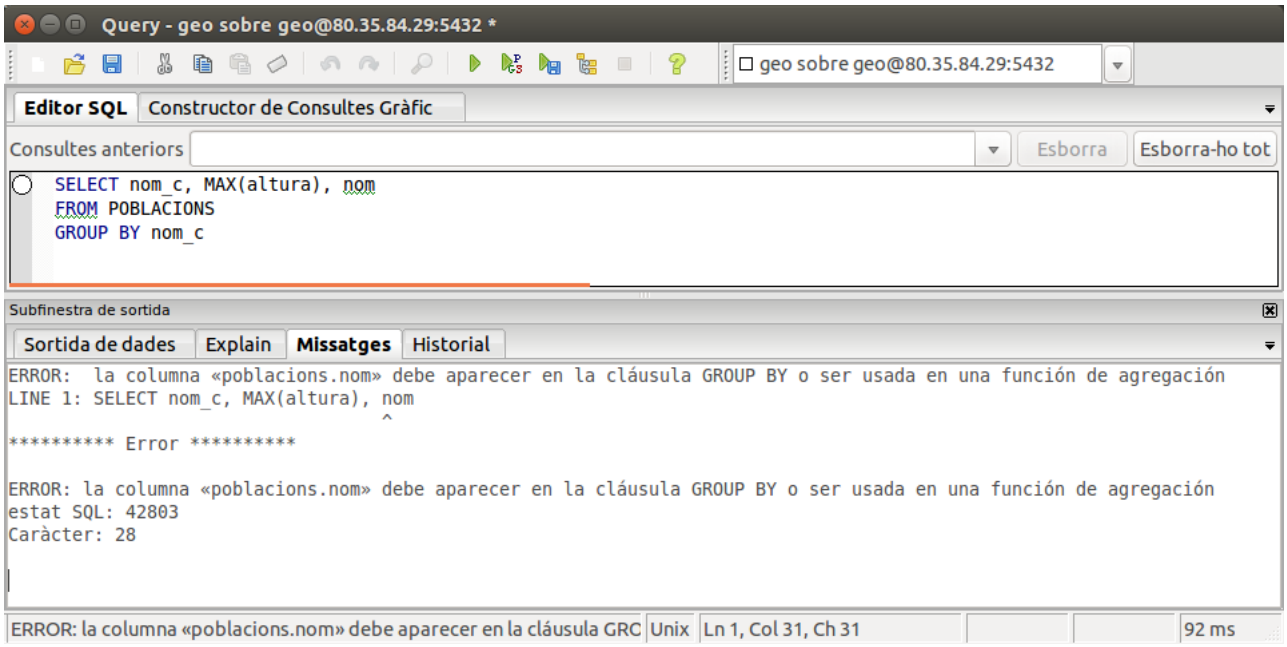
Por ejemplo, esta consulta nos da la población más alta de cada comarca:

```
SELECT nom_c, MAX (altura)
    FROM POBLACIONES
    GROUP BY nom_c
```

Pero si intentamos sacar también el nombre de la población más alta de cada comarca:

```
SELECT nom_c, MAX (altura), nombre
    FROM POBLACIONES
    GROUP BY nom_c
```

nos daría el siguiente error:



Debemos aprender a identificar este error, y solucionarlo. En este caso el solucionaremos levante el campo **Nombre** . La sentencia para poder sacar el nombre de la población más alta de cada comarca es complicada, y aunque no podemos hacerla.

ejemplos

1. Contar el número de institutos de cada población.

```
SELECT cod_m, COUNT (*)
    FROM INSTITUTOS
    GROUP BY cod_m
```

2. Contar el número de comarcas de cada provincia.

```
SELECT provincia, COUNT (*)
    FROM COMARCAS
    GROUP BY provincia
```

3. Calcular la altura máxima, mínima y la altura media de cada comarca.

```
SELECT nom_c, MAX (altura), MIN (altura), AVG (altura)
    FROM POBLACIONES
    GROUP BY nom_c
```

4. Contar el número de institutos de cada población y cada código postal.

```
SELECT cod_m, codpostal, COUNT (*)
    FROM INSTITUTOS
    GROUP BY cod_m, codpostal
```

Ejercicios apartado 13

6.20 Contar el número de pueblos de cada provincia (es suficiente sacar el código de la provincia y el número de pueblos).

6:21 Contar el número de facturas de cada vendedor a cada cliente.

22.6 Contar el número de facturas de cada trimestre. Para poder sacar el trimestre y agrupar por él (nos vale el número de trimestre, que va del 1 al 4), podemos utilizar la función **to_char (fecha, 'Q')** .

06.23 Calcular el total de cada factura, sin aplicar descuentos ni IVA. Sólo nos hará falta la mesa **LINIES_FAC** , y consistirá en agrupar por cada **num_f** para calcular la suma del **precio** multiplicado por la **cantidad** .

14. La cláusula HAVING

Esta cláusula acompaña normalmente a la de **GROUP BY** , y servirá pera poder elegir algunos grupos que cumplan una determinada condición.

Puede ir sin el GROUP BY, pero entonces su funcionamiento es como el WHERE, y por lo tanto no vale la pena. Es decir, en la práctica siempre que encontramos un HAVING estará también GROUP BY y servirá para seleccionar algunos grupos, los que cumplan la condición del HAVING.

También podríamos decir que el HAVING es el GROUP BY, lo que el WHERE es el SELECT

sintaxis

```
SELECT <columnas>
    FROM <tablas>
    [GROUP BY <columnas>]
    HAVING <condición>
```

Únicamente comentaremos el caso en que acompaña al GROUP BY. Y como hemos dicho, lo que hace es filtrar los grupos: los grupos resultantes del GROUP BY, sólo saldrán los que cumplan la condición.

Esta condición contendrá alguna función de agregado o contendrá columnas incluidas en el GROUP BY. Tenga en cuenta que es lógico, ya que sirve para elegir grupos una vez hechos, y entonces ya no se podrá ir a un elemento del grupo.

Por ejemplo, esta sentencia servirá para sacar las comarcas donde hay más de 20 pueblos, y el número que hay:

```
SELECT nom_c, COUNT (*)
    FROM POBLACIONES
    GROUP BY nom_c
    HAVING COUNT (*)> 20;
```

ejemplos

1. Sacar aquellas poblaciones que tienen más de un de Centros Integrados de Formación Profesional. La manera de saber que es un Centro Integrado es porque su nombre empieza por CIPFP. De momento sólo sacaremos el código de la población, y así sólo nos hace falta la tabla INSTITUTOS. Más adelante aprenderemos a coger los datos de más de una mesa, y entonces sacaremos también el nombre de la población

   ```
   SELECT cod_m, COUNT (*)
       FROM INSTITUTOS
       WHERE nombre LIKE 'CIPFP%'
       GROUP BY cod_m
       HAVING COUNT (*)> 1
   ```

   Lo que hacemos en esta sentencia es, de la tabla INSTITUTOS seleccionar únicamente los Centros Integrados (utilizando el operador LIKE para que empiezan por CIPFP) ii después agrupar por el código de municipio. Una vez hechos los grupos, eliminaremos los grupos que no cumplen la condición de HAVING, es decir, los que tienen 0 o 1 Centro Integrado. Y de estos grupos seleccionados sacaremos el código del municipio y el número de Centros Integrados (que siempre será igual o mayor que 2).

2. Calcular el número de habitantes máximo, el mínimo y el número de habitantes medio de las poblaciones de las comarcas con más de 20 pueblos.

   ```
   SELECT nom_c, COUNT (cod_m) AS "Número de pueblos", Max (población) AS Máximo, Min (población) AS
       Mínimo, Avg (población) AS Media
       FROM POBLACIONES
       GROUP BY nom_c
       HAVING COUNT (cod_m)> 20;
   ```

3. Sacar la altura media, total de población y población media, de aquellas comarcas que tienen una altura media superior a 800 metros.

```
SELECT nom_c, AVG (altura) AS "Altura media", SUM (población) AS "Total población", Avg
    (población) AS "Población media"
    FROM POBLACIONES
    GROUP BY nom_c
    HAVING AVG (altura)> 800;
```

Ejercicios apartado 14

24.6 Calcular la media de cantidades demandadas de aquellos artículos que se han pedido más de dos veces. Obsérvese que la tabla que nos hace falta es LINIA_FAC, y que la condición (en el HAVING) es sobre el número de veces que entra el artículo en una línea de factura, pero el resultado que se debe mostrar es la media de la cantidad.

06:25 Sacar los pueblos que tienen entre 3 y 7 clientes. Sacar sólo el código del pueblo y este número

06:26 Sacar las categorías que tienen más de un artículo "caro" (de más de 100 €). Obsérvese que también nos saldrá la categoría NULL, es decir, aparecerá como una categoría aquellos artículos que no están catalogados.

15. La cláusula ORDER BY

Ordenar las filas del resultado respecto al orden especificado

sintaxis

```
SELECT <columnas>
FROM <tablas>
ORDER BY campo1 [ASC | DESC] {, camp2 [ASC | DESC]}
```

Aquí tenemos un ejemplo:

```
SELECT nom_c, provincia
    FROM COMARCAS
    ORDER BY nom_c
```

Se ordenarán las filas por el campo, en el orden marcado ascendente o descendente (por defecto, ascendente). Si hubiera un segundo campo de ordenación, este entraría en juego en caso de valores iguales del primero. Este segundo podrá ser en orden ascendente o descendente, independientemente del orden del primer campo.

```
SELECT nom_c, provincia
    FROM COMARCAS
    ORDER BY provincia DESC, nom_c
```

Se podrá ordenar por cualquier campo de la tabla, esté indexada por este campo o no. La ventaja de estar indexada respecto a un campo es la rapidez en la ordenación. Así, si tenemos una mesa grande y ordenamos por un determinado campo, se perderá tiempo en esta ordenación. Si continuamente estamos ordenando por este campo, perderemos este tiempo muchas veces. Entonces sería conveniente crear un índice. Pero debemos recordar que la creación de un índice ocupa espacio. Por lo tanto no es buena solución indexar por todos los campos. Únicamente, en todo caso, por aquellos que más se ordene. Veremos la creación de índice en la tercera parte de este tema.

Y se puede especificar en la ordenación, una expresión que coja uno o más de un campo con operadores y funciones. Se puede poner un campo o una expresión que no esté en la lista de campos o expresiones que se quieren visualizar (junto al SELECT), aunque normalmente sí lo visualizaremos, para poder comprobar que realmente está ordenado por el que se ha especificado. Por ejemplo, si queremos ordenar por la densidad de habitantes de las poblaciones, que se calcula dividiendo el número de habitantes por la extensión:

```
SELECT nombre, poblacion, extension
    FROM POBLACIONES
    ORDER BY población / extension DESC
```

Obsérvese que estamos ordenando por un campo (**población / extension**) que no estamos visualizando, aunque sería mucho más ilustrativo mostrarlo

Opcionalmente, en el momento de especificar el campo o la expresión por la que queremos ordenar, si ésta aparece en la lista de columnas a visualizar, podremos poner sencillamente el número de orden de la columna. Así, por ejemplo, podemos hacer lo siguiente:

```
SELECT nombre, poblacion, extensión, población / extension
    FROM POBLACIONES
    ORDER BY 4 DESC
```

Dónde estamos indicando que ordene de forma descendente por la cuarta columna que va a visualizarse, es decir, para **población / extension**

Debemos observar que la cláusula ORDER BY es la última, y que en caso de haber cláusula GROUP BY, se intentará ordenar después de haber agrupado. Por lo tanto en caso de que la sentencia contenga un GROUP BY o se haya utilizado alguna función de agregado (que implica hacer grupos), sólo podremos poner en el ORDER BY campos que estén en el GROUP BY o que forman parte de una función de agregado. El razonamiento es el mismo que el hecho en la cláusula GROUP BY o HAVING y el error en caso de no respetar esto sería el matiex que el visto en ese apartado.

ejemplos

1. Sacar toda la información de las poblaciones ordenadas por el nombre de la población.

```
SELECT *
    FROM POBLACIONES
    ORDER BY nombre
```

2. Sacar toda la información de las poblaciones, ordenadas por el nombre de la comarca, y dentro de ésta por la altura (de forma descendente).

```
SELECT *
    FROM POBLACIONES
    ORDER BY nom_c, altura DESC
```

3. Sacar las comarcas con el número de pueblos y total de habitantes, de aquellas que tienen más de 10 pueblos, ordenadas por el número de pueblos, y dentro de este por el total de población de forma descendente.

```
SELECT nom_c, COUNT (*), SUM (población)
    FROM POBLACIONES
    GROUP BY nom_c
    HAVING COUNT (*)> 10
    order by COUNT (*), SUM (población) DESC
```

```
SELECT nom_c, COUNT (*), SUM (población)
    FROM POBLACIONES
    GROUP BY nom_c
    HAVING COUNT (*)> 10
    order by 2, 3 DESC
```

Ejercicios apartado 15

06:27 Sacar todos los clientes ordenados por código de población, y dentro de estos por código postal.

06:28 Sacar todos los artículos ordenados por la categoría, dentro de este por el stock, y dentro de éste por precio (de forma descendente)

06:29 Sacar los códigos de vendedor con el número de facturas vendidas en el segundo semestre de 2015, ordenadas por este número de forma descendente

16. El predicado DISTINCT

Por defecto, si no especificamos el contrario, saldrán *todas* las filas de la tabla o tablas que cumplan las condiciones. Así, por ejemplo, si de la tabla de institutos quisiéramos sacar únicamente el código de la población, nos saldría por ejemplo 12040 (el código de Castellón) en tantas filas como institutos de Castellón tengamos (concretamente 14). Es el mismo resultado que si hubiéramos puesto el predicado **ALL** frente a las columnas, ya que este es el predicado por defecto:

```
SELECT ALL cod_m
    FROM INSTITUTOS
```

Este resultado no es el correcto, si queremos consultas como "Poblaciones donde hay institutos". Sería mejor si saliera 12040 sólo una vez. Esto lo conseguiremos con el predicado DISTINCT.

En definitiva lo que hará el predicado DISTINCT será sacar las filas diferentes del resultado que pedimos. Si sólo pedimos un campo, sacará los valores distintos de este campo. Si pedimos más de un campo, sacará los valores diferentes para el conjunto de los campos (es decir las filas diferentes, que en un campo pueden coincidir, pero no en el conjunto de todos los campos)

sintaxis

```
SELECT DISTINCT <columnas>
    FROM <tablas>
```

Así, en el ejemplo anterior:

```
SELECT DISTINCT cod_m
    FROM INSTITUTOS
```

Hay una variante del DISTINCT que suparta PostgreSQL, pero que no soportan otros SGBD más sencillos, como Access. Es poner el DISTINCT dentro de una función de agregado, como por ejemplo COUNT. El resultado es que contará (o la función implicada: sumará calculará la media, ...) los valores diferentes del campo que vaya a continuación.

Así por ejemplo, si queremos contar en cuántas poblaciones diferentes tenemos institutos, la consulta es tan sencilla como esta:

```
SELECT COUNT (DISTINCT cod_m)
    FROM INSTITUTOS
```

ejemplos

1. Sacar las diferentes provincias.

```
SELECT DISTINCT provincia
    FROM COMARCAS
```

2. Sacar los distintos distritos (códigos postales) de Castellón (código de municipio 12040) donde hay institutos

```
SELECT DISTINCT codpostal
    FROM INSTITUTOS
    WHERE cod_m = 12040
```

3. Sacar las distintas comarcas y lenguas que se hablan en ellas

```
SELECT DISTINCT nom_c, lengua
    FROM POBLACIONES
    ORDER BY 1
```

Ejercicios apartado 16

06:30 Sacar los vendedores que han vendido algo el mes de enero de 2015.

06:31 Sacar los diferentes fines de vendedores (evitar que aparezca el valor nulo)

06:32 Contar en cuantas poblaciones tenemos clientes

17. La cláusula LIMIT .. OFFSET

Por medio de la cláusula **LIMIT - OFFSET** podremos hacer que no aparezcan todas las filas que devuelve la sentencia, sino varias.

- **LIMIT número** : especificaremos cuántas filas queremos que se vuelven
- **OFFSET número** : especificaremos a partir de qué posición queremos que se vuelven las filas

Si no ponemos la parte del OFFSET, aparecerán las primeras, y si especificamos OFFSET, se saltarán las primeras, tantas como se indica en OFFSET. Para que realmente tenga sentido esta cláusula, es conveninet ordenar la información, ya que al decir las primeras ya se está asumiendo que serán las primeras respecto algún orden. Así, por ejemplo nos podremos plantear sacar cosas como las 10 poblaciones más altas, o las más habitadas.

El orden implícito que acabamos de comentar deberá hacerse por medio de la cláusula ORDER BY. Así, si queremos sacar los clientes más jóvenes tendremos que ordenar por la fecha de nacimiento en orden descendente, para luego sacar los primenrs. Por lo tanto es muy difícil ver una cláusula LIMIT si no tenemos una cláusula ORDER BY.

> **Nota**
>
> En el SQL de Access no existe la cláusula LIMIT. Para hacer cosas similar dispone del predicado **TOP** , que se pone inmediatamente después del SELECT, pero siempre sacará las primeras, no tiene posibilidad de OFFSET.

sintaxis

```
SELECT <columnas>
    FROM <tablas>
    [LIMIT n ] [OFFSET m ]
```

El número ***n*** debe ser un entero, y se seleccionarán únicamente las ***n*** primeras filas (las de arriba). En caso de puesto OFFSET se saltarán ***m*** filas. En caso de no poner **LIMIT** , se saltarán ***m*** filas y se sacarán todas las demás hasta el final.

Por ejemplo, si queremos sacar las 10 poblaciones más altas, tendremos que coger las 10 primeras, ordenando por altura en forma descendente:

```
SELECT nombre, altura
    FROM POBLACIONES
    ORDER BY altura DESC
    LIMIT 10
```

El ejemplo anterior parece correcto, pero no funciona del todo bien porque con los datos que tenemos, hay 3 poblaciones que no tienen altura, y el valor nulo lo pone al final de todos los otros valores, en orden ascendente, y por tanto al principio en orden descendente.

Lo podemos arreglar sencillamente saltando las 3 primeras (que son las del nulo)

```
SELECT nombre, altura
    FROM POBLACIONES
    ORDER BY altura DESC
    LIMIT 10 OFFSET 3
```

Aunque parece mejor quitar las de altura nula, así no estamos obligados a saber cuántas poblaciones con altura nula hay

```
SELECT nombre, altura
    FROM POBLACIONES
    WHERE altura IS NOT NULL
    ORDER BY altura DESC
    LIMIT 10
```

Si quiera sacar todas las poblaciones y alturas, excepto las que tienen nulo, y sabemos que estas son 3, podemos poner OFFSET sin LIMIT, para saltar las 3 primeras, y sacar todas hasta el final

```
SELECT nombre, altura
    FROM POBLACIONES
```

```
    ORDER BY altura DESC
    OFFSET 3
```

ejemplos

1. Sacar las 5 poblaciones más pobladas

```
SELECT nombre, poblacion
    FROM POBLACIONES
    ORDER BY población DESC
    LIMIT 5
```

2. Sacar las 4 comarcas con más pueblos.

```
SELECT nom_c, COUNT (*)
    FROM POBLACIONES
    GROUP BY nom_c
    ORDER BY 2 DESC
    LIMIT 4
```

3. Sacar las 10 poblaciones con más institutos, saltándonos las 3 primeras.

```
SELECT cod_m, COUNT (*)
    FROM INSTITUTOS
    GROUP BY cod_m
    ORDER BY 2 DESC
    LIMIT 10 OFFSET 3
```

Ejercicios apartado 17

06:33 Sacar toda la información de los dos artículos más caros.

06:34 Sacar el código de las tres ciudades con más clientes

06:35 Sacar el vendedor que ha vendido menos facturas

18. Consulta de creación de tablas

Aparte de poder consultar información de una o más de una mesa, la sentencia SELECT puede servir para crear una nueva tabla, con estructura y datos (las que vienen de la propia sentencia SELECT). Eso sí, no podremos definir de esta manera ni clave principal, ni claves externas, ni ninguna otra restricción de las conocidas.

Además, esta característica escapa del estándar ANSI SQL, por lo que no le daremos excesiva importancia.

sintaxis

```
SELECT <columnas> INTO nova_taula
    FROM <tablas>
```

La sentencia puede llevar cualquier cláusula o predicado de los vistos hasta ahora, y el resultado que dé esta sentencia, se guardará en una nueva tabla, con el nombre especificado.

El nombre de los campos de la nueva tabla serán los especificados en el apartado <columnas>. Por lo tanto es especialmente recomendable la utilización de alias, ya que si ponemos serán los nombres de los campos de la nueva tabla.

Los tipos de datos de los campos serán los heredados de la consulta SELECT.

En caso de existir ya una tabla con el nombre especificado nos avisará de este hecho, dándonos la posibilidad de borrar la tabla anterior y crear la nueva o cancelar.

> **Nota**
>
> Es muy recomendable, como de otras sentencias de manipulación de datos que veremos más adelante, ejecutar primero la sentencia sin el **INTO** , para no crear la mesa todavía. Cuando estemos seguros de que el resultado es lo que deseamos, añadimos el INTO, y la mesa se creará además garantías.

ejemplos

1. Crear una copia de la tabla comarcas llamada **COMARQUES_COPIA**.

   ```
   SELECT * INTO COMARQUES_COPIA
       FROM COMARCAS
   ```

   Para no "ensuciar" la Base de Datos, podemos borrarla después de haber visto su creación con la sentencia

   ```
   DROP TABLE COMARQUES_COPIA
   ```

2. Crear una tabla llamada **RESUM_COMARQUES** que contenga el nombre de la comarca, el número de pueblos, el total de población y la altura media

   ```
   SELECT nom_c, COUNT (*) AS num_pobles, SUM (población) AS poblacion, SUM (extension) AS extension,
       AVG (altura) AS altura_mitjana INTO RESUM_COMARQUES
       FROM POBLACIONES
       GROUP BY nom_c
   ```

   Al igual que en la anterior, después de haber visto su creación y contenido, podemos borrarla con la sentencia

   ```
   DROP TABLE RESUM_COMARQUES
   ```

Ejercicios apartado 18

06:36 Crear una tabla llamada **ARTICLE_36** , que sea una copia de la tabla ARTÍCULO, pero sustituyendo los valores nulos de **stock** y **stock_min** por ceros.

06:37 Utilizar la tabla anterior para sacar el stock máximo, el mínimo y el promedio de stocks. Obsérvese que si utilizamos la tabla ARTÍCULO, los resultados no serían los mismos (excepto el máximo), sobre todo la media, ya que los valores nulos no entrarían en los cálculos de esta media.

19. Orden con que se ejecuta una sentencia SQL

Como hemos visto, y como veremos en la Parte II de este tema, la sentencia **SELECT** es muy completa y muy potente. Puede hacer muchas cosas.

Quizás nos convenga saber en qué orden se ejecutan las cláusulas de que se compone, porque esto nos puede prevenir de posibles errores en el momento de construir una sentencia de una cierta envergadura. La orden de ejecución es el siguiente:

- Primero se toman los datos desde la mesa o las mesas especificadas en el **FROM** . No podremos tratar información que no tengamos en este origen de datos.
- Después se eliminan las filas que no cumplen la condición del **WHERE** , en caso de que tengamos esta cláusula.
- Después las filas resultantes se agrupan por el o los campos especificados en el **GROUP BY** , en caso de que tengamos esta cláusula.
- Una vez hechos los grupos, se eliminan los que no cumplan la condición del **HAVING** , en caso de que tengamos esta cláusula.
- Después se selecciona la información especificada en las columnas, que en caso de haber alguna función de agregado actuará sobre los grupos que quedan (si teníamos cláusula GROUP BY) o sobre el total del origen de datos.
- Posteriormente se ordena por los campos especificados en el **ORDER BY** , en caso de que tengamos esta cláusula.
- Después aplica los predicado **DISTINCT** en caso de tenerlo especificado.
- Por último se cogen tantas filas como indica la cláusula **LIMIT** , desplazadas tantas como indique **OFFSET** , si es que tenemos esta cláusula especificada ..
- Si tenemos cláusula **INTO** se procederá a crear una nueva tabla con el resultado anterior (la veremos más adelante)

Tener claro este orden nos puede clarificar algo, y poder evitar algunos errores. El error de la siguiente sentencia ya se había explicado en la pregunta 13.

```
SELECT nom_c, COUNT (*), cod_m
    FROM POBLACIONES
    GROUP BY nom_c
```

nos dará el siguiente error:

Pero si analizamos el orden en que se ejecutan es lógico: cuando llegamos a mostrar los campos (entre ellos **cod_m**) los grupos ya se han hecho, y para valores iguales de **nom_c** . En este momento no puedo sacar algo individual de cada grupo como es el código de municipio, porque ya se ha agrupado. En este momento sólo se puede intentar sacar el nombre de la comarca (ya que tiene el mismo valor para todo el grupo, es el campo por el que hemos agrupado), o alguna función de agregado, que calcula sobre el grupo. Y de eso nos intenta avisar PostgreSQL.

Para solucionarlo podemos incluir el **cod_m** en el **GROUP BY** , y entonces haremos un grupo para cada comarca y población diferente, pero seguramente eso no nos valdrá de nada en este ejemplo, porque cada grupo sólo contendrá un elemento (un municipio), aunque en otros ejemplos sí puede tener sentido. O si no era eso lo que pretendíamos, sencillamente quitamos el campo cod_m de la sentencia, y nos funcionará bien.

Otro ejemplo ilustrativo (que ya lo pusimos muy parecido en la pregunta 13) puede ser el siguiente: podríamos intentar sacar la altura máxima de todos los pueblos, y la población que tiene esta altura. Podríamos estar tentados de hacerlo de esta manera:

```
SELECT MAX (altura), nombre
    FROM POBLACIONES;
```

Nos dará el mismo error que antes, ya que como tenemos una función de agregado intentará hacer grupos, y como no tenemos cláusula GROUP BY toda la mesa será un grupo. Y podrá calcular el máximo sin problemas, pero no podrá sacar algo individual del grupo, como es el nombre. Y en este caso no se puede solucionar incluyendo el nombre en el GROUP BY, porque entonces haremos un grupo para cada población. De momento, antes de ver las subconsultas, sólo podemos resolver este ejemplo ordenando por la altura de forma descendente, y hacer **LIMIT 1** .

Podemos observar que si tenemos cláusula **GROUP BY** , a partir de este momento todos los campos que ponemos deben estar en el GROUP BY o en una función de agregado, tanto en la condición del **HAVING** , como en las **columnas** como en el **ORDER BY** . En cambio no habrá para la cláusula **WHERE** , ya que ésta se realiza antes de que el **GROUP BY**

Ejercicios de todo el tema, con los resultados

En la BD **factura** , conectando como usuario **factura** :

6.1 Sacar toda la información de los pueblos (nómbrala **Ex_6_1.sql**).

	cod_pob numeric(5,0)	nom character varying(50)	cod_pro numeric(2,0)
1	8009	CAÑADA SECA	46
2	8018	CAÑADAS DE DON C	3
3	8029	CAÑAES (LES)	46
4	8194	CABALLUSA (LA)	3
5	8258	CABANES	12
6	8303	CABES BORT	46
7	8362	CABEZUELA (LA)	46
8	8394	CABO CERVERA-PLA	3
9	8406	CABO LAS HUERTAS	3
10	8407	CABO ROIG	3

Un total de **1.663 filas**

6.2 Sacar el código postal, el nombre y la dirección, por este orden, de todos los vendedores (nómbrala **Ex_6_2.sql**).

	cp numeric(5,0)	nom character varying(100)	adreca character varying(100)
1	12001	GUILLEN VILLALONGA, NATALIA	SANT JOSEP, 110
2	50054	VALERO GARCIA, TERESA	CLAPISA, 147-15
3	12015	POY OMELLA, PALOMA	SANCHIS TARAZONA, 103-1
4	12020	RUBERT CANO, DIEGO GUILLERMO	BENICARLO RESIDENCIAL, 154
5	12060	AGOST TIRADO, JORGE VICTOR	PASAJE PEÑAGOLOSA, 21-19
6	12067	DANIEL MIRALLES, SERGIO	CORTS VALENCIANES, 177-14
7	3903	CORBATO CARUANA, JOSE JUSTO	HISTORIADOR BETI, 200
8	12005	PEREZ CEBRIA, IGNACIO DIEGO	TOMBATOSSALS, 176-8
9	12030	ROCA FAURA, ANTONIO DIEGO	POU DEN CALVO,31
10	3380	VIDAL DIEZ, JOSE	PLAZA ESPAÑA, 33-19

6.3 Sacar el código de artículo, la descripción, precio y precio incrementado en un 5%, de todos los artículos.

	cod_a character varying(10)	descrip character varying(50)	preu numeric(6,2)	?column? numeric
1	L92212	Sofito 60 W	1.03	1.0815
2	P1044	Cruzamiento Plas	1.48	1.5540
3	L76010	Base Enchufe T.t	1.67	1.7535
4	L76011	Toma Altavoz 1 M	1.85	1.9425
5	L76014	Toma Telefono 2	2.18	2.2890
6	L76041	Cortacircuitos 1	4.70	4.9350
7	L76046	Cortacircuitos 1	5.39	5.6595
8	L76104	Pulsador Pus Leg	1.65	1.7325
9	L76114	Pulsador Legrand	1.97	2.0685
10	L76121	Pulsador Legrand	1.36	1.4280
11	L76126	Rioza 3 F. Legra	2.19	2.2995

Un total de **812 filas**

6.4 Sacar la información de los clientes con el siguiente formato (debe ir todo en una columna):

Damborenea Corbato, Alicia. CALLE MADRID, 83 (12425)

Tenga en cuenta que está todo en una columna, y por lo tanto deberá concatenar de la forma adecuada. Fíjese también que en en el nombre sólo las iniciales están en mayúsculas

	?column? text
1	Damborenea Corbato, Alicia. CALLE MADRID, 83 (12425)
2	Gual Sales, Maria. FELIPE II, 49 (3707)
3	Mateu Marti, Maria Dolores. CALLE PADRE LUIS MADRE LLOP, 30 (3607)
4	Beltran Meneu, Cristina. GABRIEL SOLE, 10-6 (46186)
5	Cancelas Mora, Maria. PINTOR ZARIQENA, 117-4 (46501)
6	Tichell Monlleo, Maria Angeles. CORAZON DE JESUS, 171 (46426)
7	Gallen Huerta, Olga. RICARDO CARRERAS, 75 (12094)
8	Navarro Barbero, Maria Lledo. BURRIANA, 19-15 (12406)
9	Iglesias Navarro, Ignacio. ESCULTOR VICIANA, 30-15 (3615)
10	Castello Damborenea, Enrique Javier. GRUPO SAN VICENTE, 138-11 (12257)
11	Rodriguez Ales, Jordi Sabina. EL PLA, 100 8 (12042)

Un total de **49 filas**

6.5 Sacar el num_f, fecha y cod_ven de las facturas con las siguientes cabeceras respectivamente: **Número Factura** , **fecha** y **Código Vendedor** (llámelo **Ex_6_5**)

	Número Factura numeric(5,0)	data date	Codi Venedor numeric(5,0)
1	6535	2015-01-01	205
2	6538	2015-01-11	5
3	6539	2015-01-14	305
4	6542	2015-01-18	205
5	6547	2015-01-26	205
6	6549	2015-02-01	155
7	6550	2015-02-03	405
8	6552	2015-02-10	305
9	6553	2015-02-10	5
10	6554	2015-02-12	455
11	6557	2015-02-15	205

Un total de **105 filas**

6.6 Dar alias en los campos que lo necesitan de la tabla ARTÍCULO (llámelo **Ex_6_6**)

	Codi Article character varying(10)	descripció character varying(50)	preu numeric(6,2)	stock numeric(4,0)	Stock mínim numeric(4,0)	Codi Categoria character varying(15
1	L92212	Sofito 60 W	1.03	2	1	
2	P1044	Cruzamiento Plastimetal	1.48			
3	L76010	Base Enchufe T.t. 1 M. Legrand Se	1.67	14	2	Legrand
4	L76011	Toma Altavoz 1 M. Legrand Serie M	1.85		12	Legrand
5	L76014	Toma Telefono 2 M. Legrand Serie	2.18	13	8	Legrand
6	L76041	Cortacircuitos 10 A. 1 M. Legrand	4.70	2	1	Legrand
7	L76046	Cortacircuitos 16a. 1 M. Legrand	5.39	5	5	Legrand
8	L76104	Pulsador Pus Legrang Mosaic	1.65	8	2	Legrand
9	L76114	Pulsador Legrand Mosaic Campana P	1.97		10	Legrand
10	L76121	Pulsador Legrand Mosaic Luz Plano	1.36	14	9	Legrand
11	L76126	Pieza 3 E. Legrand Serie Mosaic	2.19	7	6	Legrand

Un total de **812 filas**

6.7 Sacar los **clientes** de la **ciudad** con código **12309** .

	cod_cli numeric(	nom character varying(100)	adreca character varying(100)	cp numeric(5,0)	cod_pob numeric(5,0)
1	309	GISBERT MIRALLES, BEATRIZ LAURA	CALLE ASUNCION FRANCH, 79	12111	12309
2	318	PITARCH MONSONIS, MARIA CARMEN	SALVADOR, 136	12708	12309
3	345	LOPEZ BOTELLA, MAURO	AVENIDA DEL PUERTO, 20-1	12439	12309
4	348	PALAU MARTINEZ, JORGE	RAVAL DE SANT JOSEP, 97-2	12401	12309
5	351	RINCON VERNIA, DAVID	BORRIOL, 74		12309
6	354	MURIA VINAIZA, JOSE	CIUDADELA, 90-18	12990	12309
7	357	HUGUET PERIS, JUAN ANGEL	CALLE MESTRE RODRIGI, 7	12930	12309
8	366	BADENES CEPRIA, ANDRES RICARDO	MAESTRO CABALLEROS, 30-4	12397	12309
9	381	GUILLOT BELDA, FRANCISCO JOSE	PLAZA MAYOR, 200	12059	12309

6.8 Sacar todas las **facturas** del mes de **marzo** de **2015** .

	num_f numeric(5,0)	data date	cod_cli numeric	cod_ven numeric(5,0)	iva numeric(2,0)	dte numeric(2,0)
1	6572	2015-03-09	384	5	21	10
2	6574	2015-03-11	363	405	0	0
3	6577	2015-03-25	36	155	0	50
4	6579	2015-03-28	342	355	21	50
5	6581	2015-03-31	315	5	0	10

6.9 Sacar todos los artículos de la **categoría BjcOlimpia** con un **stock** entre **2** y **7** unidades.

	cod_a character var	descrip character varying(50)	preu numeric(6,2)	stock numeric(4,0)	stock_min numeric(4,0)	cod_cat character varying(15)
1	B10009B	Placa 3 E. Bjc Serie Olimpia	5.57	2	1	BjcOlimpia
2	B10010B	Interruptor Bjc Serie Olimpia	7.13	2	1	BjcOlimpia
3	B10023B	Conmutador Bjc Serie Olimpia	3.12	6	1	BjcOlimpia
4	B10028B	Cruzamiento Bjc Serie Olimpia	4.38	2	1	BjcOlimpia
5	B14005L	Base Normal Bjc Serie Olimpia	5.21	5	4	BjcOlimpia
6	B14006	Base T.t. Lateral Bjc Serie Olimp	2.79	7	7	BjcOlimpia
7	B14006L	Base Con T.t. Bjc Serie Olimpia	5.54	2	1	BjcOlimpia
8	B14007	Cortacircuitos Bjc Serie Olimpia	5.34	2	2	BjcOlimpia

06:10 Sacar todos los **clientes** que **no** tienen introducido el **código postal** .

	cod_cli numeric(5,0)	nom character varying(100)	adreca character varying(100)	cp numeric(5,0)	cod_pob numeric(5,0)
1	303	MIRAVET SALA, MARIA MERCEDES	URBANIZACION EL BALCO, 84-11		53596
2	315	VILLALONGA SANCHIS, MILAGROS	RONDA ESCALANTE, 71-11		53596
3	330	MARTI MOLTO, CONCHITA	SAN ABDON, 152		53596
4	339	CHALER SORIANO, MANUEL DIEGO	SAN ROBERTO, 27		53596
5	351	RINCON VERNIA, DAVID	BORRIOL, 74		12309
6	375	LOPEZ RINCON, LUIS MIGUEL	PADRE MELIA, 49		7766
7	384	LOPEZ GUITART, XAVIER	CALLE PUIG RODA, 162		7766
8	390	AZNAR MONFERRER, ADRIAN	EBANISANTA HERVAS, 138		32093

06:11 Sacar todos los **artículos** con el **stock** introducido pero que **no** tienen introducido el **stock mínimo** .

	cod_a character v	descrip character varying(50)	preu numeric(6,2)	stock numeric(4,0)	stock_min numeric(4,0)	cod_cat character varying(15)
1	L85547	Tecla Base Tt Lateral Legrang Dip	1.74	14		Legrand
2	T10026	Alimentador Tegui Europa	0.44	13		
3	TF16	Manguito Tubo Fercondur 21	0.91	26		
4	B10200B	Cruzamiento Bjc Olimpia Con Visor	0.88	29		BjcOlimpia
5	cm2X1.5	Pirepol N 1 X 1	0.75	36		
6	CN1X2.5	Paralelo 3 X 0.75	0.19	52		
7	CP2X1	Ojo De Buey Orientable Dorado R6	0.28	14		
8	VI25E	Caja Empalme Vilaplana 200 X 130	0.72	24		

06:12 Sacar todos los **clientes** , el **primer apellido** de los cuales es **VILLALONGA** .

	cod_cli numeric(5,0)	nom character varying(100)	adreca character varying(100)	cp numeric(5,0)	cod_pob numeric(5,0)
1	315	VILLALONGA SANCHIS, MILAGROS	RONDA ESCALANTE, 71-11		53596
2	363	VILLALONGA RAMIREZ, DIEGO SERGIO	CALLE BARTOLOME REUS, 98-7	12332	32093

6.13.a Modificar el anterior para sacar todos los que son **VILLALONGA** de **primer** o de **segundo** apellido.

	cod_cli numeric(5,0)	nom character varying(100)	adreca character varying(100)	cp numeric(5,0)	cod_pob numeric(5,0)
1	264	ADELL VILLALONGA, LUIS JOSE	MANUEL BECERRA, 61	12712	28097
2	315	VILLALONGA SANCHIS, MILAGROS	RONDA ESCALANTE, 71-11		53596
3	363	VILLALONGA RAMIREZ, DIEGO SERGIO	CALLE BARTOLOME REUS, 98-7	12332	32093

6.13.b Modificar el anterior para sacar todos los que **no** son **VILLALONGA** ni de primer ni de segundo apellido.

	cod_cli numeric(5,0)	nom character varying(100)	adreca character varying(100)	cp numeric(5,0)	cod_pob numeric(5,0)
1	3	DAMBORENEA CORBATO, ALICIA	CALLE MADRID, 83	12425	17859
2	15	GUAL SALES, MARIA	FELIPE II, 49	3707	39063
3	18	MATEU MARTI, MARIA DOLORES	CALLE PADRE LUIS MADRE LLOP,	3607	29149
4	21	BELTRAN MENEU, CRISTINA	GABRIEL SOLE, 10-6	46186	31982
5	30	CANCELAS MORA, MARIA	PINTOR ZARIQENA, 117-4	46501	7625
6	36	TICHELL MONLLEO, MARIA ANGELES	CORAZON DE JESUS, 171	46426	33246
7	69	GALLEN HUERTA, OLGA	RICARDO CARRERAS, 75	12094	48192
8	72	NAVARRO BARBERO, MARIA LLEDO	BURRIANA, 19-15	12406	48037
9	78	IGLESIAS NAVARRO, IGNACIO	ESCULTOR VICIANA, 30-15	3615	45004
10	120	CASTELLO DAMBORENEA, ENRIQUE JAVIER	GRUPO SAN VICENTE, 138-11	12257	46332
11	174	RODRIGUEZ ALOS, JORDI SABINA	EL PLA, 100 8	12042	2814

Un total de **46 filas**

06:14 Sacar los **artículos** " **Pulsador** " (la descripción contiene esta palabra), el **precio** de los que oscila entre **2 y 4 €** y de los que tenemos un **stock** estrictamente **mayor** que el **stock mínimo** .

	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	S3144236	Pulsador Campana Marron Dec. Simo	3.62	4	2	Simon
2	THC29	Pulsador Para Placa Felmax	2.56	5	2	
3	B14005	Pulsador Luz Con Visor Bjc Serie	2.46	2	1	
4	B944B	Pulsador Niesen Lisa Campana B	2.59	8	4	Niessen

06:15 Contar el número de **clientes** que tienen el **código postal nulo** .

	count bigint
1	8

6.16 Contar el número de veces que el artículo **L76104** entra en las líneas de factura, y el número total de unidades vendidas de este artículo. Sólo os hace falta la tabla LINIA_FAC.

	count bigint	sum numeric
1	4	36

6.17 Sacar la **media** del **stock** de los artículos.

	avg numeric
1	11.2979797979797980

6.17b Modificar el anterior para tener en cuenta los valores nulos, como si fueran 0. Os vendrá bien la función COALESCE que convierte los nulos del primer parámetro al valor dado como segundo parámetro (si es diferente de nulo, deja igual el valor). Por lo tanto la debe utilizar de esta manera: COALESCE (stock, 0)

	avg numeric
1	11.0197044334975369

06:18 Contar **cuántas facturas** tiene el cliente **375**

	count bigint
1	5

19.06 Calcular el **descuento máximo** , el **mínimo** y el descuento **medio** de las **facturas** .

	max numeric	min numeric	avg numeric
1	50	0	16.7619047619047619

6.20 Contar el número de pueblos de cada provincia (es suficiente sacar el código de la provincia y el número de pueblos).

	cod_pro numeric(2,0)	count bigint
1	12	530
2	46	604
3	3	529

6:21 Contar el número de facturas de cada vendedor a cada cliente.

	cod_ven numeric(5,0	cod_cli numeric(5,0	count bigint
1	355	357	1
2	55	213	1
3	155	252	1
4	155	363	1
5	305	357	1
6	105	213	1
7	5	384	1
8	205	306	1
9	155	321	1
10	405	363	1
11	305	369	1

Un total de **96 filas**

De estas 96 filas, relativamente poque tienen un valor distinto de 1 en el número de facturas: la fila 29 (455, 30, 2) o la fila 34 (5, 342, 3)

22.6 Contar el número de facturas de cada trimestre. Para poder sacar el trimestre y agrupar por él (nos vale el número de trimestre, que va del 1 al 4), podemos utilizar la función **to_char (fecha, 'Q')** .

	to_char text	count bigint
1	2	25
2	4	25
3	3	33
4	1	22

No aparece ordenado, y quiere decir que en el trimestre 2 hay 25 facturas, en el trimestre 4 hay 35, en el trimestre 3 hay 33 y en el trimestre 1 hay 22

06.23 Calcular el total de cada factura, sin aplicar descuentos ni IVA. Sólo nos hará falta la mesa **LINIES_FAC** , y consistirá en agrupar por cada **num_f** para calcular la suma del **precio** multiplicado por la **cantidad** .

	num_f numeric(5,0)	sum numeric
1	6565	322.58
2	6750	495.68
3	6678	142.21
4	6629	579.93
5	6643	163.57
6	6664	474.13
7	6697	5.40
8	6647	64.92
9	6680	721.24
10	6690	1261.00
11	6693	456.14

Un total de **105 filas**

24.6 Calcular la media de cantidades demandadas de aquellos artículos que se han pedido más de dos veces. Obsérvese que la tabla que nos hace falta es LINIA_FAC, y que la condición (en el HAVING) es sobre el número de veces que entra el artículo en una línea de factura, pero el resultado que se debe mostrar es la media de la cantidad.

	cod_a character vary	count bigint	avg numeric
1	B10006B	3	3.3333333333333333
2	L85393	3	8.0000000000000000
3	L85685	3	7.3333333333333333
4	RF32	3	7.0000000000000000
5	TC125	3	6.0000000000000000
6	L76104	4	9.0000000000000000
7	im4P10L	3	6.6666666666666667
8	ME200	3	4.6666666666666667
9	T1023	3	5.6666666666666667
10	S3162031	3	4.6666666666666667
11	LA2760EC	3	8.6666666666666667
12	L03753	3	5.0000000000000000
13	VI833E	3	6.6666666666666667
14	L34000	3	6.3333333333333333
15	T4512	3	7.3333333333333333
16	L16500	3	8.6666666666666667
17	L55812	3	7.3333333333333333
18	L16555	3	5.3333333333333333
19	VI745E	3	7.0000000000000000
20	L76138	3	8.3333333333333333
21	ZN5104B	3	7.6666666666666667
22	L34044	3	6.0000000000000000

06:25 Sacar los pueblos que tienen entre 3 y 7 clientes. Sacar sólo el código del pueblo y este número

	cod_pob numeric(5,0)	count bigint
1	32093	4
2	7766	7

06:26 Sacar las categorías que tienen más de un artículo "caro" (de más de 100 €). Obsérvese que también nos saldrá la categoría NULL, es decir, aparecerá como una categoría aquellos artículos que no están catalogados.

	cod_cat character varying(15)	count bigint
1		8
2	Ticino	3
3	IntMagn	2

06:27 Sacar todos los clientes ordenados por código de población, y dentro de estos por código postal.

	cod_cli numeric(5,0)	nom character varying(100)	adreca character varying(100)	cp numeric(5,0)	cod_pob numeric(5,0)
1	216	DAMBORENEA VALLS, DIEGO RAFAEL	AVENIDA ALMASSORA, 51	3008	1651
2	294	BELTRAN MUNYOZ, JAIME VICENTE	DEL ANGEL, 79-2	46390	2050
3	174	RODRIGUEZ ALOS, JORDI SABINA	EL PLA, 100-8	12042	2814
4	252	CEPRIA LORENTE, CARLOS MIGUEL	VINAROZ, 137	12974	5495
5	30	CANCELAS MORA, MARIA	PINTOR ZARIQENA, 117-4	46501	7625
6	387	TUR MARTIN, MANUEL FRANCISCO	CALLE PEDRO VIRUELA, 108-8	12008	7766
7	342	PINEL HUERTA, VICENTE	FRANCISCO SEMPERE, 37-10	12112	7766
8	306	SAMPEDRO SIMO, MARIA MERCEDES	FINELLO, 161	12217	7766
9	369	BOTELLA CATALA, JUAN	MONCADA, 70	12572	7766
10	336	MIGUEL ARCHILES, OSCAR RAMON	HISANT BERNARDO MUNDINA, 132 5	12652	7766

Un total de **49 filas**

06:28 Sacar todos los artículos ordenados por la categoría, dentro de este por el stock, y dentro de éste por precio (de forma descendente)

	cod_a character vary	descrip character varying(50)	preu numeric(6,2)	stock numeric(	stock_min numeric(4,(	cod_cat character varying
1	B10007B	Placa 2 E. Bjc Serie Olimpia	68.29	1	1	BjcOlimpia
2	B14007L	Regulador Bjc Olimpia	8.10	1	1	BjcOlimpia
3	B10010B	Interruptor Bjc Serie Olimpia	7.13	2	1	BjcOlimpia
4	B10009B	Placa 3 E. Bjc Serie Olimpia	5.57	2	1	BjcOlimpia
5	B14006L	Base Con T.t. Bjc Serie Olimpia	5.54	2	1	BjcOlimpia
6	B14007	Cortacircuitos Bjc Serie Olimpia	5.34	2	2	BjcOlimpia
7	B10028B	Cruzamiento Bjc Serie Olimpia	4.38	2	1	BjcOlimpia
8	B14005L	Base Normal Bjc Serie Olimpia	5.21	5	4	BjcOlimpia
9	B10023B	Conmutador Bjc Serie Olimpia	3.12	6	1	BjcOlimpia
10	B14006	Base T.t. Lateral Bjc Serie Olimp	2.79	7	7	BjcOlimpia
11	B10202B	Doble Conmutador Bjc Olimpia	2.33	11	2	BjcOlimpia

Un total de **812 filas**

06:29 Sacar los códigos de vendedor con el número de facturas vendidas en el segundo semestre de 2015, ordenadas por este número de forma descendente

	cod_ven numeric(5,0)	count bigint
1	105	7
2	355	7
3	405	7
4	205	6
5	5	6
6	255	6
7	155	5
8	305	4
9	55	4
10	455	4
11		2

06:30 Sacar los vendedores que han vendido algo el mes de enero de 2015.

	cod_ven numeric(5,0)
1	205
2	5
3	305

06:31 Sacar los diferentes fines de vendedores (evitar que aparezca el valor nulo)

	cod_cap numeric(5,0)
1	5
2	405
3	105
4	255

06:32 Contar en cuantas poblaciones tenemos clientes

	count bigint
1	23

06:33 Sacar toda la información de los dos artículos más caros.

	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	L18500	Bornas Clic 2 X	532.60	1	1	Legrand
2	CENTRA1	Cortacircuito B	315.09	1	1	

06:34 Sacar el código de las tres ciudades con más clientes

	cod_pob numeric(5,0)	count bigint
1	53596	10
2	12309	9
3	7766	7

06:35 Sacar el vendedor que ha vendido menos facturas

	cod_ven numeric(5,0)	count bigint
1	55	5

06:36 Crear una tabla llamada **ARTICLE_36** , que sea una copia de la tabla ARTÍCULO, pero sustituyendo los valores nulos de **stock** y **stock_min** por ceros.

El resultado debe ser la creación de la tabla. Si consulte su contenido debe ser el siguiente:

	cod_a character v	descrip character varying(50)	preu numeric(6,	stock numeric	stock_min numeric	cod_cat character vary
1	1967346	Prolongador Doble 2.5 M	4.51	1	1	
2	2309987	Prolongador Doble 5.0 M	7.36	4	4	
3	3204209	Prolongador Cuaduple 2.5 M	5.11	1	1	
4	3987348	Prolongador Cuaduple 20.0 M	12.17	2	2	
5	4283200	Prolongador Doble 10.0 M	9.32	3	3	
6	4565467	Prolongador Sencillo 10.0 M	7.21	4	1	
7	6579878	Prolongador Sencillo 5.0 M	4.21	1	1	
8	7897999	Prolongador Sencillo 2.5 M	3.16	2	1	
9	8340098	Prolongador Cuaduple 5.0 M	8.11	2	1	
10	8394800	Prolongador Cuaduple 10.0 M	10.37	2	2	
11	A03755	Tubo Fluorescente 15 W	50.63	1	1	
12	A58068	Tubo Fluorescente 18w Tld	6.37	4	1	
13	B10000B	Cortacicuito Bjc Ibiza Blaco	0.88	11	7	
14	B10001B	Marco Bjc 1 E Blanco	1.38	1	1	
15	B10005B	Marco Bjc 2 E Ibiza Blanco	3.93	1	1	
16	B10006B	Marco Bjc 3 E Ibiza Blanco	4.44	6	3	
17	B10007B	Placa 2 E. Bjc Serie Olimpia	68.29	1	1	BjcOlimpia
18	B10009B	Placa 3 E. Bjc Serie Olimpia	5.57	2	1	BjcOlimpia
19	B10010B	Interruptor Bjc Serie Olimpia	7.13	2	1	BjcOlimpia
20	B10022B	Interruptor Con Visor Bjc Serie O	2.70	0	6	
21	B10023B	Conmutador Bjc Serie Olimpia	3.12	6	1	BjcOlimpia

Un total de **812 filas**

06:37 Utilizar la tabla anterior para sacar el stock máximo, el mínimo y el promedio de stocks. Obsérvese que si utilizamos la tabla ARTÍCULO, los resultados no serían los mismos (excepto el máximo), sobre todo la media, ya que los valores nulos no entrarían en los cálculos de esta media.

	max numeric	min numeric	avg numeric
1	448	0	11.019