

El lenguaje SQL-Consultas Select

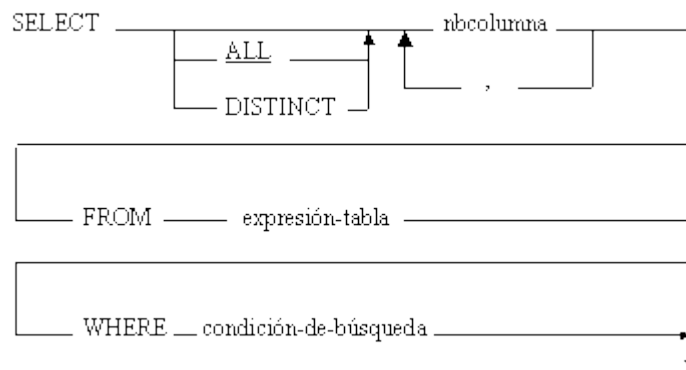
1. El lenguaje SQL.

- El lenguaje SQL (*Structured Query Language*), es un lenguaje de cuarta generación (4GL) surgido de un proyecto de investigación de IBM para el **acceso a bases de datos relacionales**. Por tanto, no se trata de un lenguaje de programación.
- IBM se basó en el modelo relacional propuesto por *E. F. Codd* en 1970. En sus inicios, SQL se conoció con el nombre de SEQUEL (*Structured English QUery Language*).
- Inicialmente, SEQUEL fue implementado por el SGBD experimental *System R*, también desarrollado por IBM en 1977.
- La empresa *Oracle* introdujo por primera vez el lenguaje SQL en un SGBD comercial, en 1979, y actualmente se ha convertido en un estándar de lenguaje de acceso a bases de datos, y la mayoría de los sistemas de bases de datos lo soportan, desde sistemas para ordenadores personales hasta los sistemas orientados a grandes servidores.
- En 1986 el lenguaje SQL fue estandarizado por primera vez por la ANSI, conociéndose dicho estándar como el *SQL-86* o *SQL1*. Al año siguiente dicho estándar fue también adoptado por la ISO.
- Una lista con los distintos estándares del lenguaje SQL definidos a lo largo de los años puede verse en http://es.wikipedia.org/wiki/Celda_activa.
- A partir del estándar cada sistema ha desarrollado su propio SQL que puede variar de un sistema a otro, pero con cambios que no suponen ninguna complicación para alguien que conozca un SQL concreto.
- Como su nombre indica, SQL permite realizar consultas a la base de datos. Pero el nombre se queda corto, ya que SQL realiza, además, funciones de definición, control y gestión de la base de datos. Las sentencias SQL se clasifican según su finalidad, dando origen a tres sublenguajes:
 - A. DDL (*Data Description Language*): lenguaje de definición de datos, incluye órdenes para definir, modificar o borrar las tablas en las que se almacenan los datos (es el que más puede variar de un sistema a otro).
 - B. DCL (*Data Control Language*): lenguaje de control de datos, contiene elementos útiles para trabajar en un entorno multiusuario, en el que es importante la protección de los datos, la seguridad de las tablas y el establecimiento de restricciones en el acceso, así como elementos para coordinar la compartición de datos por parte de usuarios concurrentes, asegurando que no interfieren unos con otros.

C. DML (*Data Manipulation Language*): lenguaje de manipulación de datos, nos permite recuperar los datos almacenados en la base de datos y también incluye órdenes para permitir al usuario actualizar la base de datos añadiendo nuevos datos, suprimiendo datos antiguos o modificando datos previamente almacenados.

2. Características del lenguaje.

- Una sentencia SQL es como una frase (escrita en inglés) en la cual especificamos **qué** es lo que queremos obtener y de **dónde** obtenerlo.
- Todas las sentencias empiezan con un verbo (palabra reservada que indica la acción a realizar), seguido del resto de cláusulas, unas obligatorias y otras opcionales, que completan la frase.
- La sintaxis de las sentencias SQL suelen representarse mediante un diagrama sintáctico como el que se muestra a continuación:



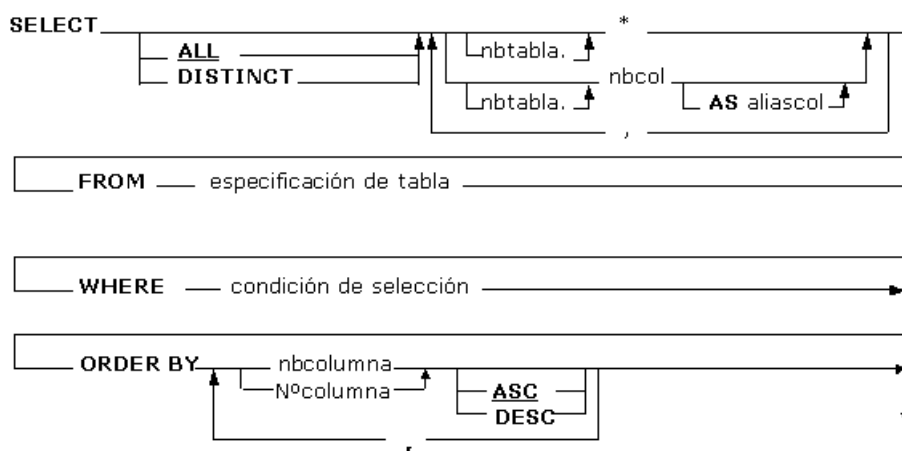
- Las palabras que aparecen en mayúsculas son palabras reservadas del lenguaje, por ejemplo, en el diagrama de la figura anterior aparecen las palabras reservadas *SELECT*, *ALL*, *DISTINCT*, *FROM* y *WHERE*. Por su parte, las palabras en minúsculas son variables que el usuario deberá sustituir por un dato concreto. En el diagrama tenemos *nbcolumna*, *expresión-tabla* y *condición-de-búsqueda*.
- Una sentencia válida se construye siguiendo la línea a través del diagrama hasta el punto que marca el final. Las líneas se siguen de izquierda a derecha y de arriba abajo. Cuando se quiere alterar el orden normal se indica con una flecha.
- Cuando una palabra opcional aparece subrayada indica que ése es el valor por defecto.

Ejemplo 1: Algunas sentencias SQL válidas.

```
SELECT ALL col1, col2, col3 FROM mitabla;
SELECT col1, col2, col3 FROM mitabla;
SELECT DISTINCT col1 FROM mitabla;
SELECT col1, col2 FROM mitabla WHERE col2 = 0;
```

3. Las consultas simples.

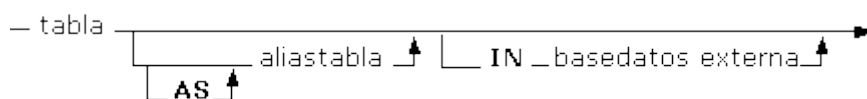
- La sentencia más usada en SQL es **SELECT**. Dicha sentencia se utiliza para realizar consultas a una BD. Es posible realizar consultas simples (a una sola tabla) o consultas multitabla, en este apartado estudiaremos las primeras.
- La sentencia SELECT forma parte del DML (*lenguaje de manipulación de datos*).
- El formato completo de la sentencia SELECT se muestra en el siguiente diagrama sintáctico:



- La cláusula **FROM** se usa para indicar **en qué tabla** se tiene que buscar la información. En el caso de consultas simples el resultado se obtiene de una única tabla. La sintaxis de la cláusula es:

FROM especificación de tabla

- La especificación de una tabla tiene el siguiente formato:



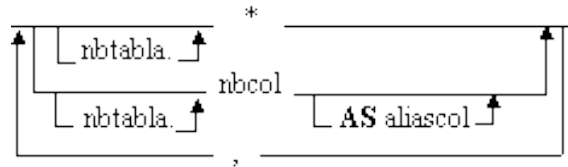
- En ocasiones será necesario asignar un sobrenombre o alias a una tabla, para lo cual escribiremos dicho alias justamente detrás del nombre de la tabla, pudiéndose incluir entre el nombre de la tabla y su alias la palabra reservada **AS**. Los alias se usarán en las consultas multitabla. El alias de una tabla es sólo válido en la consulta en la que se define. Además, cuando a una tabla se le asigna un alias siempre deberá ser referenciada usando dicho alias. El uso de la palabra AS es opcional.

Ejemplo 2: Utilización de un alias de tabla. Ambas sentencias son equivalentes.

```

SELECT ... FROM oficina AS ofi;
SELECT... FROM oficina ofi;
  
```

- La lista de columnas que queremos que aparezcan en el resultado se conoce como **lista de selección** y se especifica delante de la cláusula FROM siguiendo el siguiente formato:



- Se utiliza el asterisco * en la lista de selección para indicar que se pretenden seleccionar todas las columnas de la tabla.
- Se puede combinar el * con el nombre de una tabla (ej. oficina.*), pero esto se utiliza más cuando el origen de la consulta son dos tablas.

Ejemplo 3: Uso del * para referenciar a todas las columnas de la tabla.

```
SELECT * FROM oficina;
SELECT oficina.*, numemp FROM oficina , empleado where oficina=codoficina;
```

- Las columnas se pueden especificar mediante su nombre simple (nbc col) o su nombre cualificado (nbtbla.nbc ol, el nombre de la columna precedido del nombre de la tabla que contiene la columna y separados por un punto).
- El nombre cualificado se puede emplear siempre que queramos y es obligatorio en algunos casos que veremos más adelante.

Ejemplo 4: Especificación de la lista de selección.

```
SELECT nombre, oficina, fcontrato FROM ofiventas;
SELECT producto.idfab, producto.precio FROM producto; -- innecesario
```

Ejercicio 1: Implementa las siguientes consultas en SQL:

- Seleccionar el nombre de todos los empleados.
- Seleccionar el cargo y el número de oficina de todos empleados.
- Seleccionar la fecha del contrato y el nombre de los empleados que no superaron su cuota de ventas.
- Seleccionar los códigos de las oficinas cuyas ventas superaron los 500000 euros y que además sean del norte.

IMPORTANTE: MySql funciona con CASE INSENSITIVE por defecto esto quiere decir que no distingue entre mayúsculas y minúsculas. Para poder distinguirlas en los tipos Char y Varchar tiene la opción **Binary** para estos datos. Tenemos también las siguientes funciones de cadena:

poner la cadena a minúsculas --> lower() ó lcase();
poner la cadena a mayúsculas --> upper() ó ucase();

- Cuando se visualiza el resultado de una consulta las columnas toman el nombre que tienen en la tabla, si queremos cambiar ese nombre lo podemos hacer definiendo un **alias de columna** mediante la cláusula **AS** (o mejor sin ella). Si el alias contiene espacios en blanco hay que entrecomillar.

Ejemplo 5: Utilización de un alias de columna.

```
SELECT idproducto AS producto, descripcion FROM producto;  
SELECT idproducto producto, descripcion FROM producto;  
SELECT idproducto 'producto en stock', descripcion FROM producto;
```

Ejercicio 2: (alias de columna) Realiza las siguientes consultas:

- a) Seleccionar el nombre de todos los empleados, haciendo que aparezca como título de la columna “Nombre de empleado”.
 - b) Seleccionar el número de oficina, con encabezado “Número de oficina”, la ciudad, con encabezado “Ciudad de destino” y la región, de todas las oficinas cuyo objetivo de venta sea menor de 400000.
- Además de las columnas que provienen directamente de la tabla origen, una consulta SQL puede incluir **columnas calculadas**, cuyos valores se calculan a partir de los valores de los datos almacenados.
 - Para solicitar una columna calculada especificaremos en la lista de selección una expresión en vez de un nombre de columna. La expresión puede contener sumas, restas, multiplicaciones, divisiones, concatenación (&), paréntesis y también funciones predefinidas (SUM, MAX, MIN, AVG, MONTH, YEAR,...).

Ejemplo 6: Obtención de columnas calculadas.

```
SELECT idproducto, (existencias * precio) valoracion FROM producto;  
SELECT nombre, MONTH(fcontrato), YEAR(fcontrato) FROM empleado;  
SELECT oficina, 'tiene ventas de ', ventas FROM oficina;  
SELECT SUM(cant) FROM lineaspedido;  
SELECT ciudad, region, (ventas-objetivo) superavit FROM oficina  
WHERE (ventas-objetivo) >20000;
```

En esta última select, **el uso del alias en la cláusula Where no funciona:**

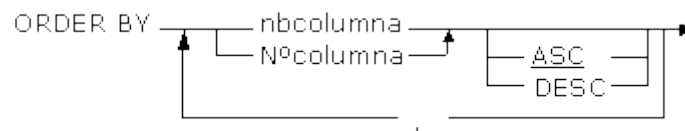
```
SELECT ciudad, region, (ventas-objetivo) "superavit" FROM oficina  
WHERE "superavit" >20000;      -- ERROR
```

En las cláusula HAVING sí puede usarse un alias sin espacios y sin comillas de una columna.

```
SELECT ciudad, sum(ventas) suma FROM oficina
GROUP BY 1 HAVING suma > 20000;
```

Ejercicio 3: (columnas calculadas) Realiza las siguientes consultas:

- a) Seleccionar el nombre de cada empleado y la mitad de su cuota. El encabezado de la mitad de su cuota será “*Mitad de la cuota*”. Sólo aparecerán los datos de los empleados para los cuales el doble de sus ventas sea mayor que 650000 y menor que 800000.
 - b) Seleccionar el nombre de cada empleado y el número de años que le restan para jubilarse, encabezando estos valores con el cargo “*Años restantes de trabajo*”. Sólo se incluirán en el resultado de la consulta a aquellos empleados mayores de 40.
 - c) Seleccionar el nombre de la ciudad, el código del director de la misma y la mitad de la diferencia entre ventas y objetivo, de todas las oficinas para las que el 30 % de las ventas sea mayor que el 15 % del objetivo. El formato de la salida será:
Ciudad oficina: ciudad – Código director: dir – Mitad diferencia: resultado
- La cláusula **ORDER BY** se utiliza para ordenar las filas del resultado de una consulta. Su formato es el siguiente:



- Podemos indicar la columna por la que queremos ordenar el resultado utilizando su nombre de columna o bien utilizando su número de orden que ocupa en la lista de selección.

Ejemplo 7: Ordenación del resultado.

```
SELECT nombre, codoficina, fcontrato FROM empleado ORDER BY codoficina;
SELECT nombre, codoficina, fcontrato FROM empleado ORDER BY 2;
SELECT nombre, numemp FROM empleado ORDER BY 1;
SELECT nombre, numemp, fcontrato FROM empleado ORDER BY fcontrato;
SELECT nombre, numemp, ventas FROM empleado ORDER BY ventas;
```

- Si se desea realizar una ordenación en orden descendente debemos utilizar la cláusula **DESC**.

Ejemplo 8: Ordenación descendente del resultado.

```
SELECT nombre, fcontrato FROM empleado ORDER BY fcontrato DESC;
SELECT nombre, numemp,ventas FROM empleado ORDER BY ventas DESC;
```

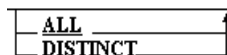
- También es posible ordenar el resultado por varias columnas, en este caso se indican las columnas separadas por comas. Se ordenan las filas por la primera columna de ordenación, para un mismo valor de la primera columna se ordenan por la segunda columna, y así sucesivamente.

Ejemplo 9: Ordenación por varias columnas.

```
SELECT region, ciudad, ventas FROM oficina ORDER BY region, ciudad;
SELECT region, ciudad, (ventas - objetivo) AS superavit FROM oficina
ORDER BY region, 3 DESC;
```

Ejercicio 4: (ORDER BY) Realiza las siguientes consultas:

- Seleccionar la primera, tercera y última columna de la tabla empleados. Los registros aparecerán ordenados descendientemente por la tercera columna de la lista de selección.
 - Seleccionar el nombre y la edad de todos los empleados cuya edad esté comprendida estrictamente entre 45 y 65. El encabezado de la primera columna será “*Empleados mayores*”. Los nombres aparecerán ordenados por la edad.
 - Seleccionar la región y la ciudad de todas las oficinas que no estén en el norte ni tengan menos de 300000 euros de objetivo de ventas. El resultado aparecerá ordenado ascendientemente por región, y descendientemente por ciudad.
 - Seleccionar el nombre de empleado y el 25 % de sus ventas. Este último dato aparecerá con el encabezado “*Beneficios de empleado*”. No se incluirán en el resultado de la consulta a aquellos empleados cuyo beneficio no sea superior a 80000. El resultado aparecerá ordenado por beneficios, ascendientemente.
- Existen algunas cláusulas que permiten especificar qué filas del resultado de la consulta serán visualizadas. Dichas cláusulas se incluyen en el siguiente diagrama:



- Al incluir la cláusula **DISTINCT** en la SELECT se eliminan del resultado las repeticiones de filas. Si por el contrario queremos que aparezcan todas las filas, incluidas las duplicadas, podemos incluir la cláusula ALL o nada, ya que ALL es el valor que SQL asume por defecto.
- **Limit:** Indica el número máximo de filas a mostrar. Se sitúa como última cláusula.

Ej: *SELECT director FROM oficina LIMIT 7;*

Ejemplo 10: Utilización de las cláusulas ALL y DISTINCT.

```
SELECT DISTINCT director FROM oficina;  
SELECT director FROM oficina;  
SELECT ALL director FROM oficina;
```

Ejercicio 5: (ALL, DISTINCT, LIMIT) Realiza las siguientes consultas sobre la tabla *pedido*:

- Seleccionar los años en los que se realizó algún pedido (sin repeticiones). Los años aparecerán ordenados cronológicamente, de mayor a menor.
 - Seleccionar el número de pedido y su fecha, de los 7 pedidos más antiguos.
 - Seleccionar el número de pedido y su fecha, de los 5 pedidos más actuales.
 - Seleccionar el número de pedido y su fecha, para los tres primeros pedidos realizados en 1997.
 - Seleccionar el número de pedido y el importe de las 3 líneas de pedidos de menor importe.
 - Seleccionar el número de pedido y la fecha de los 10 pedidos más actuales.
- La cláusula **WHERE** permite seleccionar únicamente las filas que cumplan una condición. La condición de selección puede ser cualquier expresión válida o combinación de expresiones utilizando los operadores NOT, AND y OR.

Ejemplo 11: Uso básico de la cláusula WHERE.

```
SELECT nombre FROM empleado WHERE (codoficina = 12) AND (ventas > 3000);  
SELECT numemp, nombre FROM empleado WHERE fcontrato < '01/01/1988';  
SELECT numemp, nombre FROM empleado WHERE YEAR(fcontrato) < 1988;  
SELECT oficina FROM oficina WHERE (ventas) < (objetivo * 0.8);
```

Ejercicio 6: (WHERE) Realiza las siguientes consultas sobre la tabla *oficina*:

- Seleccionar el nombre de las regiones de las oficinas (sin repeticiones) cuyas ventas superen los 600000 euros.
 - Seleccionar el número de oficina y el nombre de sus ciudades cuyas ventas sean mayores de 600000 o menores de 200000, que además cumplan que hayan vendido menos del 80 % del objetivo previsto.
 - Seleccionar el nombre de la ciudad y de la región de todas las oficinas. Las filas aparecerán ordenadas por nombre de ciudad, y sólo se mostrarán aquellas filas cuyo objetivo de ventas sea mayor de 300000.
- El *test de rango* (**BETWEEN** o **NOT BETWEEN**) puede usarse para comprobar si un valor está o no comprendido entre dos valores.

Ejemplo 12: Uso del test de rango en la cláusula WHERE. Los valores 100 y 500 también se incluyen.

SELECT nombre FROM empleado WHERE ventas BETWEEN 100 AND 500;

- El *test de pertenencia a conjunto (IN)* puede usarse para comprobar si un valor está incluido en una lista de valores.

Ejemplo 13: Uso del test de pertenencia a conjunto.

SELECT numemp, oficina FROM empleado WHERE codoficina IN (12,14,16);

- Una condición de selección puede dar como resultado el valor TRUE, FALSE o NULL. Cuando una columna que interviene en una condición de selección contiene el valor nulo, el resultado de la condición no es verdadero ni falso, sino nulo, sea cual sea el test que se haya utilizado.
- Si queremos preguntar si una columna contiene el valor nulo debemos utilizar un test especial, *el test de valor nulo*, cuyo formato es el siguiente:

nbcolumna — IS — NOT — NULL —>

Ejemplo 14: Uso del test de valor nulo.

SELECT oficina, ciudad FROM oficina WHERE director IS NULL;
SELECT numemp, nombre FROM empleado WHERE codoficina IS NOT NULL;

- Por otra parte, es posible utilizar comodines para formar los valores a comparar en una condición. Para ello usaremos el test de correspondencia con patrón (**LIKE**). Su formato es el siguiente:

nbcolumna — NOT — LIKE — patron —>

- Los comodines más utilizados para formar patrones son los siguientes:

_ → representa un carácter cualquiera
 % → representa cero o más caracteres

Las comparaciones entre cadenas se hace sin diferenciación entre mayúsculas y minúsculas (CASE INSENSITIVE). Investigar cómo solucionar ese problema.

Ejemplo 15: Uso del test de correspondencia con patrón.

SELECT numemp, nombre FROM empleado WHERE nombre LIKE 'Luis%';
SELECT numemp, nombre FROM empleado WHERE nombre LIKE '%Luis%';
SELECT numemp, nombre FROM empleado WHERE nombre LIKE '__a%';

Ejercicio 7: (BETWEEN, IN, LIKE) Realiza las siguientes consultas sobre la tabla *empleado*:

- a) Seleccionar los nombres de los empleados cuya cuota de venta esté comprendida entre 250000 y 300000, y cuyas ventas no estén comprendidas entre 50000 y 250000 ni entre 300000 y 500000 (usa BETWEEN siempre que sea posible).
- b) Seleccionar el nombre y las fechas de los contratos de los empleados cuyos contratos se formalizasen entre 1989 y 2000, o bien su oficina sea la 21, la 13 o la 22. (usa BETWEEN e IN).
- c) Seleccionar el nombre y la edad de los empleados cuyo primer apellido no comience ni por G ni por V.
- d) Seleccionar el nombre de los empleados cuyo cargo no sea director de ningún ámbito('director ventas', 'director compras'.....).

Ejercicio 8: (IS NULL) Realiza las siguientes consultas sobre *empleado* y *oficina*:

- a) Seleccionar los nombres de los empleados que no tengan jefe o no pertenezcan a ninguna oficina.
- b) Selecciona los nombres de empleados que no tengan asignada cuota de venta.
- c) Seleccionar el código y el nombre de la ciudad de todas las oficinas que no tengan ni objetivo de ventas ni ventas.

Funciones de fecha:

- DATEDIFF(*expr*,*expr2*): devuelve el número de días entre la fecha inicial *expr* y la fecha final *expr2*. Expr debe ser mayor que *expr2*, si no devuelve un número negativo.
- CURDATE(): devuelve la fecha actual.

Ejercicio 9: (Repaso de las consultas simples): Realiza las siguientes consultas:

- a) Obtener una lista de todos los productos indicando para cada uno su idproducto, descripción, precio y precio con I.V.A. incluido (es el precio anterior aumentado en un 21%).
- b) Listar de cada empleado su nombre, nº de días que lleva trabajando en la empresa y su año de nacimiento.
- c) Obtener la lista de los clientes ordenados por código de representante asignado. Se visualizarán todas las columnas de la tabla.
- d) Obtener las oficinas ordenadas por orden alfabético de región y dentro de cada región por ciudad. Si hay más de una oficina en la misma ciudad entonces aparecerá primero la que tenga el número de oficina mayor.

- e) Obtener los pedidos ordenados por fecha de pedido.
- f) Listar toda la información de los pedidos de marzo.
- g) Listar los números de los empleados que tienen una oficina asignada.
- h) Listar los números de las oficinas que no tienen director.
- i) Listar los datos de las oficinas de las regiones del norte y del este. Deben aparecer primero las del norte y después las del este.
- j) Listar los empleados de nombre Juan.
- k) Listar los productos cuyo idproducto acabe en x.

4. Consultas multitable.

- Una *consulta multitable* se caracteriza porque acceden a más de una tabla de la BD.
- Existen dos grupos de consultas multitable:
 - A. **Unión** de tablas
 - B. **Composición** de tablas

4.1 Composición de tablas.

- La *composición de tablas* consiste en concatenar filas de una tabla con filas de otra. En este caso obtenemos una tabla con las columnas de la primera tabla unidas a las columnas de la segunda tabla, y las filas de la tabla resultante son concatenaciones de filas de la primera tabla con filas de la segunda tabla.

Ejemplo 16: Este ejemplo quedaría de la siguiente forma con la composición:

idfab	idproducto	fab	producto
bic	41672	aci	41004
imm	779c	aci	41002

COMPOSICION

idfab	idproducto	fab	producto
bic	41672	aci	41002
bic	41672	aci	41004
imm	779c	aci	41002
imm	779c	aci	41004

- A diferencia de la unión, la composición permite obtener cada fila formada por datos de las dos tablas.

Ejemplo 17: Por ejemplo, queremos listar cada pedido con el nombre del representante que lo ha hecho. Los datos del pedido los tenemos en la tabla de pedidos, pero el nombre del representante está en la tabla de empleados.

numpedido	fechapedido	rep	numemp	nombre
112968	11/01/90	101	101	Antonio Viquer
112992	15/04/90	108	108	Ana Bustamante

COMPOSICION

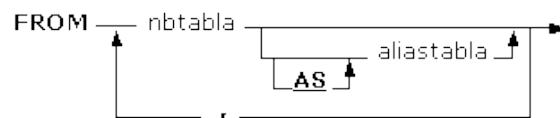
numpedido	fechapedido	rep	numemp	nombre
112968	11/01/90	101	101	Antonio Viquer
112992	15/04/90	108	108	Ana Bustamante

- Los *tipos de composición* de tablas son:

- *Producto cartesiano*
- *Reunión interna*
- *LEFT /RIGHT JOIN*

4.1.1 Producto cartesiano.

- Aplicando el producto cartesiano a dos tablas se obtiene una tabla formada por todas las filas que se puedan generar al concatenar todas las filas de la primera tabla con filas de la segunda tabla. La sintaxis es la siguiente:



- El producto cartesiano se indica poniendo en la cláusula FROM las tablas que queremos componer separadas por comas. Podemos obtener así el producto cartesiano de dos o más tablas.
- En el diagrama anterior *nbtbla* puede ser un nombre de tabla o un nombre de consulta.
- Hay que tener en cuenta que el producto cartesiano obtiene todas las posibles combinaciones de filas, por lo tanto, si tenemos dos tablas de 100 filas cada una el resultado tendrá 100x100 filas, si el producto lo hacemos de estas dos tablas con una tercera de 20 filas el resultado tendrá 200.000 filas (100x100x20). Se observa claramente que el producto cartesiano es una operación costosa, sobre todo si operamos con más de dos tablas o con tablas voluminosas.
- Es posible componer una tabla consigo misma, en cuyo caso es obligatorio utilizar un *nombre de alias* por lo menos para una de las tablas.

Ejemplo 18: Producto cartesiano de la tabla de empleados con ella misma. Obtendremos todas las posibles combinaciones de empleados con empleados.

```
SELECT * FROM empleado, empleado AS emp;
```

- El producto cartesiano no es una operación de las más utilizadas.

Ejercicio 10: (Producto cartesiano) Realiza los siguientes productos cartesianos:

- Todos los campos de las tablas producto y pedido.
- Todos los campos de las tablas empleado y oficina.
- Los campos nombre, edad, oficina y cargo de la tabla empleados, y los campos oficina, ciudad y región de la tabla oficina. No se incluirán en el resultado filas cuyo campo cargo sea “representante”.
- En este apartado efectuaremos el producto cartesiano de la tabla producto consigo misma. Se incluirán todos los campos de la primera relación, pero sólo la descripción del producto de la segunda.

4.1.2 Reunión interna.

- Hay que tener en cuenta las siguientes reglas:
 - Se pueden unir tantas tablas como deseemos.
 - En la cláusula SELECT se pueden citar columnas de todas las tablas.
 - Si hay columnas con el mismo nombre en las distintas tablas de la cláusula FROM, se deben identificar, especificando *NombreTabla.NombreColumna*.
 - **El criterio que se siga para combinar las tablas ha de especificarse en la cláusula WHERE. Si se omite esta cláusula, que especifica la condición de combinación, el resultado será un PRODUCTO CARTESIANO, que emparejará todas las filas de una tabla con cada fila de la otra.**

Ejemplo 19: Composición de las tablas *pedido* y *cliente* usando reunión interna.

```
SELECT * FROM pedido, cliente where pedido.clie = cliente.numclie;
```

- Es posible definir varias condiciones de emparejamiento unidas por los operadores AND y OR poniendo cada condición entre paréntesis.

Ejemplo 20: Composición de las tablas *pedido* y *producto* con dos columnas de emparejamientos. (Este ejemplo no se corresponde con nuestras tablas).

```
SELECT *
FROM pedido, producto
where (pedido.fab = producto.idfab) AND      -- Estos paréntesis son
      (pedido.producto = producto.idproducto); -- innecesarios
```

- Por otra parte, es posible combinar más de dos tablas en una misma consulta.
- Se puede expresar también con el INNER JOIN:

..... FROM tabla1 INNER JOIN tabla2 ON tabla1.col1 = tabla2.col2

En la cláusula ON se ponen las condición/es de unión de las tablas, las restantes condiciones se colocan en el WHERE. Pero para más de dos tablas es muy lioso.

```
SELECT * FROM pedido INNER JOIN producto ON pedidos.fab =  
producto.idfab AND pedido.producto = producto.idproducto;
```

Ejemplo 21: Composición de las tablas *pedido*, *cliente* y *empleado*.

```
SELECT * FROM pedido p, cliente c, empleado  
Where p.cliente = c.numclie AND repclie = numemp);
```

Ejercicio 11: Realiza las siguientes combinaciones:

- a) Selecciona todos los campos de la tabla clientes junto a los datos de los pedidos que han realizado.
- b) Selecciona todos los campos de la tabla clientes junto a los datos de los pedidos que han realizado, incluyéndose únicamente la fecha de cada pedido y el número del pedido. Sólo se incluirán en el resultado a los clientes cuyo límite de crédito sea mayor que 40000.
- c) Seleccionar el nombre de cliente, su límite de crédito, así como el nombre de su representante y la región a la que pertenece la oficina de éste.
- d) Seleccionar el nombre de empleado (con el cargo “Nombre de empleado”), la fecha de su contrato, el código de su oficina, el nombre de la ciudad de la oficina, así como el nombre del jefe del empleado (con el cargo “Nombre del jefe”).
- e) Seleccionar el número de cada pedido, su fecha y su importe, así como la ciudad en la que se encuentra la oficina del representante que tiene asignado. No se incluirán en el resultado los datos de pedidos cuyo importe sea inferior a 40000 ni superior a 60000.

4.2.3 LEFT JOIN y RIGHT JOIN.

- **LEFT JOIN** y **RIGHT JOIN** son otros tipos de composición de tablas que llevan a cabo una *composición externa*. Son una extensión del INNER JOIN.
- Las composiciones vistas hasta ahora (producto cartesiano y reunión interna) son *composiciones internas*. Con una composición interna sólo se obtienen las filas que tienen al menos una fila de la otra tabla que cumpla la condición.

Ejemplo 22: Queremos combinar los empleados con las oficinas para saber la ciudad de la oficina donde trabaja cada empleado.

```
SELECT empleado.*,ciudad FROM empleado, oficina  
WHERE codoficina = oficina;
```

- Supongamos que *codoficina* de empleados puede ser nula, es decir que pueden haber empleados sin oficina asignada. Con la sentencia anterior los empleados que no tienen una oficina asignada, es decir, aquellos que tienen un valor nulo en el campo *codoficina* de la tabla *empleado*, no aparecen en el resultado ya que la condición *codoficina = oficina* será siempre nula para esos empleados.
- En los casos en que queremos que también aparezcan las filas que no tienen una fila coincidente en la otra tabla utilizaremos el LEFT JOIN o RIGHT JOIN.
- La sintaxis de LEFT JOIN es la siguiente:

```
FROM – tabla1 – LEFT JOIN – tabla2 – ON – tabla1.col1 – comp – tabla2.col2
```

- Esta operación consiste en las filas de la tabla de la izquierda que no tienen correspondencia en la tabla de la derecha, y rellenar en esas filas los campos de la tabla de la derecha con valores nulos.

Ejemplo 23: A continuación obtenemos una lista de los empleados con los datos de su oficina. El empleado 110, que no tiene oficina, aparece con sus datos, rellenándose los datos de su oficina con valores nulos. Es decir, se obliga a incluir todos los empleados en el resultado.

```
SELECT * FROM empleado LEFT JOIN oficinas ON codoficina = oficina;
```

- Por su parte, la sintaxis del RIGHT JOIN es la siguiente:

```
FROM – tabla1 – RIGHT JOIN – tabla2 – ON – tabla1.col1 – comp – tabla2.col2
```

- Al contrario que LEFT JOIN, esta operación consiste en añadir al resultado de INNER JOIN las filas de la tabla de la derecha que no tienen correspondencia en la tabla de la izquierda, y rellenar en esas filas los campos de la tabla de la izquierda con valores nulos.

Ejemplo 24: A continuación obtenemos una lista de los empleados con los datos de su oficina, y además aparece una fila por cada oficina que no está asignada a ningún empleado con los datos del empleado a nulos. Es decir, se obliga a incluir todas las oficinas en el resultado.

```
SELECT * FROM empleado RIGHT JOIN oficina ON codoficina=oficina;
```

Ejercicio 12: (LEFT JOIN...ON, RIGHT JOIN...ON) Realiza las siguientes combinaciones usando siempre el operador de combinación más apropiado:

- a) Se pretende obtener una relación en la que aparezcan los datos de todas las oficinas junto a los nombres y las edades de los directores de cada una de ellas (si una oficina no tiene director entonces los datos de este aparecerán en blanco).
- b) Obtener una relación que contenga los nombres y cargos de los empleados así como los códigos y la ciudad de las oficinas de los mismos que pertenezcan a la región 'este'. Aparecerán también los empleados que no estén asignados a ninguna oficina.
- c) Una relación en la que aparezcan los empleados con los nombres de sus jefes, aparecerán también los empleados que no tienen jefe.

Ejercicio 13: (Repaso de las consultas multitabla) Realiza las siguientes consultas:

- a) Listar las oficinas del este indicando para cada una de ellas su número, ciudad, números y nombres de sus empleados. Hacer una versión en la que aparecen sólo las que tienen empleados y otra en las que aparezcan también las oficinas del este que no tienen empleados.
- b) Listar los pedidos mostrando su número, nombre del cliente y el límite de crédito del cliente correspondiente.
- c) Listar los datos de cada uno de los empleados, la ciudad y región en donde trabaja.
- d) Listar las oficinas con objetivo superior a 600.000 indicando para cada una de ellas el nombre de su director.
- e) Hallar los empleados que realizaron su primer pedido el mismo día en que fueron contratados.
- f) Listar los empleados con una cuota superior a la de su jefe; para cada empleado sacar sus datos y el número, nombre y cuota de su jefe.

5. Consultas de resumen.

- Una *consulta de resumen* o *consulta sumaria* es una consulta en la que las filas resultantes son un resumen de varias filas de la tabla origen.
- A continuación podemos ver un ejemplo de consulta normal en la que se visualizan las filas de la tabla oficinas ordenadas por región, en este caso cada fila del resultado se corresponde con una sola fila de la tabla oficinas, mientras que la segunda consulta es una consulta resumen, cada fila del resultado se corresponde con una o varias filas de la tabla oficinas.

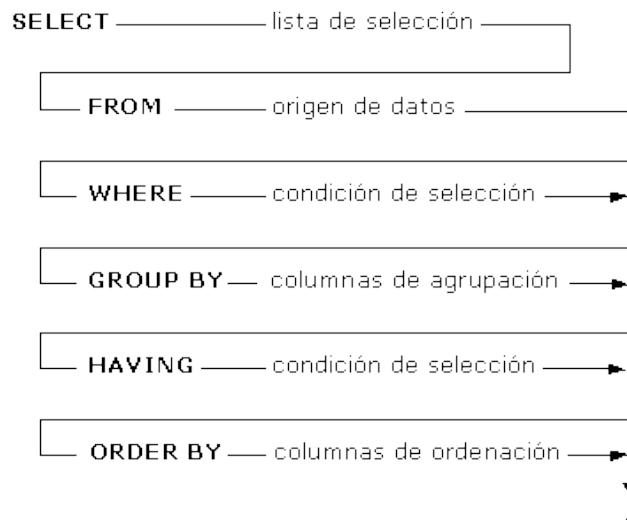

```
SELECT oficina,region,ventas
FROM oficinas
ORDER BY region
```

oficina	region	ventas
24	centro	150.000 Pts
23	centro	
28	este	0 Pts
13	este	368.000 Pts
12	este	735.000 Pts
11	este	693.000 Pts
26	norte	
22	oeste	186.000 Pts
21	oeste	836.000 Pts

```
SELECT region,SUM(ventas)
FROM oficinas
GROUP BY region
```

region	SumaDeventas
centro	150000
este	1796000
norte	
oeste	1022000

- Las consultas de resumen introducen dos nuevas cláusulas a la sentencia SELECT, la cláusula GROUP BY y la cláusula HAVING, que sólo se pueden utilizar en una consulta de resumen. Estas cláusulas se tienen que escribir entre la cláusula WHERE y la cláusula ORDER BY, como podemos ver en el siguiente esquema:



- Detallaremos estas dos nuevas cláusulas posteriormente, antes vamos a introducir otro concepto relacionado con las consultas de resumen, las *funciones de columna*.

5.1 Funciones de columna.

- En la lista de selección de una consulta de resumen aparecen *funciones de columna*, también denominadas funciones de dominio agregadas. Una función de columna se aplica a una columna y obtiene un valor que resume el contenido de la misma. Tenemos las siguientes funciones de columna:

SUM (expresión)	MIN (expresión)
AVG (expresión)	MAX (expresión)
STDEV (expresión)	COUNT (nbcolumna)
STDEVP (expresión)	COUNT (*)

- El argumento de la función indica con qué valores se tiene que operar, por eso *expresión* suele ser el nombre de la columna que contiene los valores a resumir, pero también puede ser cualquier expresión válida que devuelva una lista de valores.
- **Tener en cuenta que si la tabla está vacía o por las condiciones especificadas en la cláusula WHERE, la SELECT no devuelve ninguna fila, todas las funciones de columna excepto COUNT() retornan el valor NULL.**
- La función *SUM()* calcula la suma de los valores indicados en el argumento. Los datos que se suman deben ser de tipo numérico (entero, decimal, coma flotante o monetario...).

Ejemplo 25: A continuación se obtiene una sola fila con el resultado de sumar todos los valores de la columna ventas de la tabla oficinas.



SELECT SUM(ventas) FROM oficina;

- La función *AVG()* calcula el promedio (media aritmética) de los valores indicados en el argumento, también se aplica a datos numéricos.
- Es interesante destacar que el valor nulo no equivale al valor 0, las funciones de columna no consideran los valores nulos mientras que sí consideran el valor 0 como un valor, por lo tanto en la función *AVG()* los resultados no serán los mismos con valores 0 que con valores nulos. Veámoslo con un ejemplo:

Ejemplo 26: Comparativa del uso de funciones de columna sobre columnas que contienen valores nulos o ceros. Supongamos que tenemos las tablas *Tabla1* y *Tabla2* mostradas en las siguientes figuras:

Tabla1 : Tabla	
	col1
	10
	5
	0
	3
	6
	0

Tabla2 : Tabla	
	col1
	10
	5
	3
	6

<i>Consulta</i>	<i>Resultado</i>	<i>Justificación</i>
SELECT AVG(col1) AS media FROM tabla1		En este caso los ceros entran en la media por lo que sale igual a 4 $(10+5+0+3+6+0)/6 = 4$
SELECT AVG(col1) AS media FROM tabla2		En este caso los ceros se han sustituido por valores nulos y no entran en el cálculo por lo que la media sale igual a 6 $(10+5+3+6)/4 = 6$

- Las funciones *MIN()* y *MAX()* determinan los valores menores y mayores, respectivamente. Los valores de la columna pueden ser de tipo numérico, texto o fecha. El resultado de la función tendrá el mismo tipo de dato que la columna. Si la columna es de tipo numérico *MIN()* devuelve el valor menor contenido en la columna, si la columna es de tipo texto *MIN()* devuelve el primer valor en orden alfabético, y si la columna es de tipo fecha, *MIN()* devuelve la fecha más antigua y *MAX()* la fecha más reciente.
- La función *COUNT(nb_columna)* cuenta el número de valores que hay en la columna, los datos de la columna pueden ser de cualquier tipo, y la función siempre devuelve un número entero. Si la columna contiene valores nulos esos valores no se cuentan, si en la columna aparece un valor repetido, lo cuenta varias veces. Para que cuente sólo una vez los valores repetidos se usa *COUNT(DISTINCT nb_columna)*.
- La función *COUNT(DISTINCT lista de columnas separadas por comas)* cuenta las distintas ocurrencias de la lista de columnas considerándolas en como un todo.

COUNT(DISTINCT nombre,ventas)

- COUNT(*)* permite contar filas en vez de valores. Si la columna no contiene ningún valor nulo, *COUNT(nb_columna)* y *COUNT(*)* devuelven el mismo resultado, mientras que si hay valores nulos en la columna, *COUNT(*)* cuenta también esos valores mientras que *COUNT(nb_columna)* no los cuenta.

Ejemplo 27: Dos formas de determinar el número de empleados que hay en la tabla. En este caso las dos sentencias devuelven el mismo resultado, ya que la columna *numemp* no contiene valores nulos.

```
SELECT COUNT(numemp) FROM empleado;
SELECT COUNT(*) FROM empleado;
```

Ejemplo 28: Obtención del número de empleados que tienen una oficina asignada. Esta sentencia nos devuelve el número de valores no nulos que se encuentran en la columna oficina de la tabla empleados.

```
SELECT COUNT(codoficina) FROM empleado;
```

- Se pueden combinar varias funciones de columna en una expresión, pero no se pueden anidar funciones de columna.

Ejemplo 29: Combinación de varias funciones de columna en una misma consulta.

*SELECT (AVG(ventas) * 3) + SUM(cuota) FROM ...*

Ejemplo 30: Anidamiento incorrecto de funciones de columna en MySQL.

SELECT AVG(SUM(ventas)) FROM ...

IMPORTANTE:

Esta característica se tratará a fondo cuando se explique la cláusula GROUP BY

- El origen de los datos que se tomará en una consulta de resumen puede restringirse usando la sentencia WHERE. Además, es posible utilizar consultas de resumen que tengan un origen de datos proveniente de varias tablas.

Ejemplo 31: Diferentes orígenes de datos para una consulta de resumen.

SELECT SUM(ventas) FROM empleado WHERE codoficina = 12;

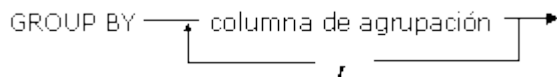
*SELECT SUM(empleado.ventas), MAX(objetivo) FROM empleado
LEFT JOIN oficina ON codoficina=oficina;*

Ejercicio 14: (Funciones de columna) Realiza las siguientes selecciones usando las funciones de columna:

- a) Se pretende calcular la media de las edades de los empleados mayores de 50 años. Este resultado aparecerá con la cabecera de columna “*Media de mayores*”.
- b) Halla la suma de las cuotas de los empleados que trabajan en las oficinas de Valencia o de Castellón.
- c) Halla la suma de los importes del pedido 12345.
- d) Queremos visualizar el número de filas obtenido al realizar el producto cartesiano de la tabla pedidos y la tabla clientes.
- e) Visualizar la mayor edad de entre los empleados contratados antes de 1988.
- f) Visualizar la mayor edad de los empleados cuya oficina esté en la zona este o en la ciudad de A Coruña.
- g) Visualizar la fecha del último pedido que hizo el cliente 1234.
- h) Visualizar la media de los importes asociados al producto *bisagra*.

5.2 La cláusula GROUP BY.

- Hasta ahora las consultas de resumen que hemos visto utilizan todas las filas de la tabla y producen una única fila resultado, sin embargo, es posible obtener subtotales con la cláusula **GROUP BY**.
- Una consulta con una cláusula GROUP BY se denomina *consulta agrupada*, ya que agrupa los datos de la tabla origen y produce una única fila resumen por cada grupo formado. Las columnas indicadas en el GROUP BY se llaman *columnas de agrupación*.
- La sintaxis de esta cláusula es la siguiente:



Ejemplo 32: Uso de GROUP BY para obtener las ventas de cada oficina. Se forma un grupo para cada oficina, con las filas de la oficina, y la suma se calcula sobre las filas de cada grupo.

```
SELECT SUM(ventas) FROM empleado GROUP BY codoficina;
```

La consulta anterior quedaría mejor incluyendo en la lista de selección la oficina para saber a qué oficina corresponde la suma de ventas:

```
SELECT codoficina, SUM(ventas) FROM empleado GROUP BY codoficina;
```

Podemos usar también números de orden en esta cláusula como se hace en la Order By.

```
SELECT codoficina, SUM(ventas) FROM empleado GROUP BY 1;
```

- La columna de agrupación se puede indicar mediante un nombre de columna o cualquier expresión válida basada en una columna y se pueden utilizar los alias de campo.

Ejemplo 33: En la siguiente consulta se agrupan las líneas de pedido por precio unitario y se obtiene, de cada precio unitario, el importe total vendido.

```
SELECT importe/cant , SUM(importe) FROM lineaspedido
GROUP BY importe/cant;
```

Ejemplo 34: En la siguiente consulta se utiliza un alias de columna en el GROUP BY:

```
SELECT importe/cant AS precios, SUM(importe) FROM lineaspedido
GROUP BY precios;
```

```
SELECT importe/cant, SUM(importe) FROM lineaspedido
GROUP BY 1;           – MEJOR
```

- En la lista de selección de una consulta agrupada sólo pueden aparecer:
 - ▲ *valores constantes*
 - ▲ *funciones de columna*
 - ▲ *columnas de agrupación (columnas que aparecen en la cláusula GROUP BY)*
 - ▲ *cualquier expresión basada en las anteriores.*

Ejemplo 35: A continuación podemos observar dos consultas agrupadas. La primera es válida, ya que la lista de selección contiene elementos válidos. Sin embargo, la segunda no está permitida, ya que el campo *ciudad* es una columna simple que no está encerrada en una función de columna, ni está en la lista de columnas de agrupación.

```
SELECT region, SUM(ventas) FROM oficina GROUP BY region;
```

```
SELECT region, ciudad, SUM(ventas) FROM oficina GROUP BY 1;
```

- Se pueden agrupar las filas por varias columnas, en este caso se indican las columnas separadas por una coma y en el orden de mayor a menor agrupación. Se permite incluir en la lista de agrupación hasta 10 columnas.

Ejemplo 36: Queremos obtener la suma de las ventas de las oficinas agrupadas por región y ciudad, por tanto se agrupa primero por región y dentro de cada región por ciudad:

```
SELECT region,ciudad,SUM(ventas) FROM oficina  
GROUP BY 1,2;
```

- Todas las filas que tienen valor nulo en el campo de agrupación pasan a formar un único grupo, es decir, se considera el valor nulo como un valor cualquiera a efectos de agrupación.

Ejemplo 37: La siguiente consulta contendrá en el resultado una fila con el campo *director* sin valor alguno y, a continuación, una cantidad en el campo *Objetivos_totales*, esta cantidad corresponde a la suma de los objetivos de las oficinas que no tienen director asignado (campo *director* igual a nulo).

```
SELECT director, sum(objetivo) as 'Objetivos_totales'  
from oficina group by director;
```

Consideraciones sobre el anidamiento de funciones de grupo (SUM, AVG, MAX...)

Las funciones de grupo en MySql **no se pueden anidar, pero sí en Oracle**. Su uso es bastante útil y se explica con este ejemplo.

```
SELECT MIN(SUM(importe)) FROM lineaspedido GROUP BY numpedido;
```

Primero se agruparía las filas de la tabla en tantos grupos como número de pedidos hay, a continuación obtendría el importe menor de todos los pedidos realizados. La función MIN afecta a todos los grupos seleccionando el valor mínimo del sumatorio de los importes de todos ellos.

En MySql ésto mismo se puede conseguir de la siguiente forma:

```
SELECT SUM(importe) FROM lineaspedido  
GROUP BY numpedido ORDER BY 1 LIMIT 1;
```

Pero si se quisiera obtener el importe total menor y además el/los números de pedidos que tienen dicho importe, la SELECT sería como sigue:

```
SELECT numpedido, SUM(importe) FROM lineaspedido  
GROUP BY numpedido HAVING sum(importe) =  
(SELECT min(sum(importe)) FROM lineaspedido GROUP BY numpedido);
```

En MySql podríamos sacar los pedidos que tuvieran el importe mínimo pero no cuál es dicho importe como en la select anterior:

```
SELECT numpedido FROM lineaspedido  
GROUP BY numpedido HAVING sum(importe)=  
(SELECT SUM(importe) FROM lineaspedido  
GROUP BY numpedido ORDER BY 1 LIMIT 1);
```

Realizar una versión del anterior en la que se muestren el/los números de pedido y los códigos de cliente con menor importe.

5.3 La cláusula HAVING.

- La cláusula HAVING nos permite seleccionar filas de la tabla resultante de una consulta de resumen. Su sintaxis es la siguiente:

HAVING ——— condición de selección ———>

- Para la condición de selección se pueden utilizar los mismos tests de comparación descritos en la cláusula WHERE, también se pueden escribir condiciones compuestas (unidas por los operadores OR, AND, NOT).
- En la condición de selección sólo pueden aparecer :
 - ▲ *valores constantes*
 - ▲ *funciones de columna*
 - ▲ *columnas de agrupación (columnas que aparecen en la cláusula GROUP BY)*
 - ▲ *cualquier expresión basada en las anteriores.*

Ejemplo 38: Queremos obtener las oficinas con un promedio de ventas de sus empleados mayor que 500000.

```
SELECT codoficina FROM empleado GROUP BY codoficina HAVING AVG(ventas) > 500000;
```

Ejercicio 15: (GROUP BY Y/O HAVING) Realiza las siguientes selecciones:

- a) La edad media de los trabajadores que tienen el mismo cargo. Se incluirá el cargo en la salida.
- b) La fecha del contrato más antiguo de los empleados de cada oficina. Se incluirá el número de cada oficina.
- c) De entre los empleados que tienen el mismo jefe se obtendrá la cuota máxima. Se incluirá el código del jefe.
- d) El número de empleados que tiene a su cargo cada empleado que es jefe. Se incluirá el código del jefe. Implementa otra versión que además incluya el nombre del jefe.
- e) Para cada región, la máxima venta de sus trabajadores.
- f) Para cada grupo de empleados que tengan la misma cuota se calculará la fecha de contrato más antigua y más moderna. El resultado debe incluir la cuota. No se tendrán en cuenta para calcular el resultado a los empleados cuya edad sea menor de 35.
- g) Para cada producto el importe total vendido.
- h) El importe total de cada producto vendido a partir de 1995. Se incluirá en la salida el código del producto.
- i) La cantidad total de cada producto vendido a partir de 1995. Se incluirá en la salida el código del producto.
- j) El número de pedidos en los que se ha visto involucrado cada producto. Sólo se tendrán en cuenta los productos cuyo precio sea mayor que 200 y menor que 2000.
- k) El importe medio que se ha vendido a cada cliente que tenga un límite de crédito inferior a 50000. En la salida se incluirá el código del cliente.
- l) El importe máximo que se ha vendido a cada cliente. Sólo se tendrán en cuenta los clientes que tengan asignado un representante mayor de 40 años. En la salida se incluirá el código del cliente.
- m) Para cada producto, la mayor edad de los empleados que lo vendieron. Se incluirá en el resultado el código del producto.
- n) Para cada producto, el mayor objetivo de las oficinas de los empleados que lo vendieron. Se incluirá en el resultado el código del producto.

- o)* Seleccionar el número de pedido e importe de los tres pedidos de menor importe.
- p)* Seleccionar el número de pedido e importe de los 10 pedidos más actuales.
- q)* Lista código de cliente, número de pedido, fecha del pedido e importe total de aquellos clientes cuyo límite de crédito sea mayor de 40000.
- r)* Listar los pedidos mostrando su número, importe, nombre de cliente y el límite de crédito del cliente.
- s)* Listar los pedidos superiores a 25000 de importe incluyendo el nombre del empleado que tomó el pedido y el nombre del cliente que lo solicitó.
- t)* Listar los códigos de los empleados que tienen algún pedido con importe superior a 10000 o que tengan una cuota inferior a 10000.
- u)* Halla la suma de los importes de las líneas de los pedidos en los que se haya solicitado el producto con descripción 'Manivela'.
- v)* La suma de las ventas de los empleados de cada oficina. Se incluirá en el resultado el código de cada oficina. No se incluirán en el resultado las oficinas cuyas ventas totales sean menores de 200000. Hacerlo con HAVING y sin HAVING.
- w)* La suma de las unidades vendidas de cada producto vendido al cliente 1234. No se incluirán en el resultado final aquellas filas cuya suma de unidades al cuadrado sea menor de 1000.
- x)* Por cada empleado, sacar la media del límite de crédito de sus clientes, el código del empleado y su nombre. No se tendrán en cuenta los clientes cuyo límite de crédito sea inferior a 30000. Deben salir aquellos empleados cuyo nombre empiecen por M o A. Por último ordena el resultado por el promedio.

Ejercicio 16: (Repaso de las consultas de resumen) Realiza las siguientes consultas:

- a)* ¿Cuáles son la cuota media y las ventas medias de todos los empleados?
- b)* Halla el importe medio de pedidos, el importe total de pedidos y el precio medio de venta (el precio de venta es el precio unitario en cada pedido).
- c)* Halla el precio medio de los productos.
- d)* ¿Cuál es el importe total de los pedidos realizados por el empleado *Vicente Pantalla*?
- e)* Halla en qué fecha se realizó el primer pedido.
- f)* Halla cuántos pedidos hay de más de 25000. (Tener en cuenta las consideraciones sobre el anidamiento de funciones del apartado GROUP BY).

- g) Lista cuántos empleados están asignados a cada oficina, indicar el número de oficina y cuántos tiene asignados.
- h) Para cada empleado, obtener su número, nombre, e importe vendido por ese empleado a cada cliente indicando el número de cliente.
- i) Para cada empleado cuyos total de importe de los pedidos de sus clientes sumen más de 30000 euros, hallar su importe medio de pedidos. En el resultado aparecerá el número de empleado y su importe medio de pedidos.
- j) Obtén para cada producto, su descripción, precio y cantidad total pedida, incluyendo sólo los productos cuya cantidad total pedida sea superior al 75% del stock; y ordenado por cantidad total pedida.

6. Las subconsultas.

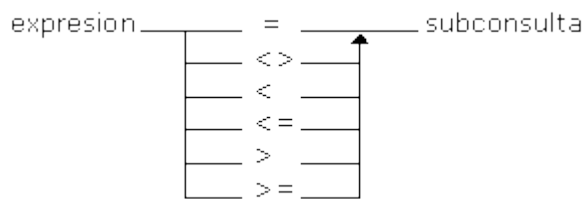
- Una subconsulta es una sentencia SELECT que aparece dentro de otra sentencia SELECT a la que llamaremos consulta principal.
- Una subconsulta se puede incluir *en la lista de selección, en la cláusula WHERE o en la cláusula HAVING* de la consulta principal.
- Una subconsulta tiene la misma sintaxis que una sentencia SELECT normal exceptuando que aparece encerrada entre paréntesis, no puede contener la cláusula ORDER BY, ni puede ser la UNION de varias sentencias SELECT, además tiene algunas restricciones en cuanto a número de columnas según el lugar donde aparece en la consulta principal. Estas restricciones las iremos describiendo en cada caso.
- Cuando se ejecuta una consulta que contiene una subconsulta, la subconsulta se ejecuta por cada fila de la consulta principal.
- Se aconseja no utilizar campos calculados en las subconsultas, ralentizan la consulta.
- Las consultas que utilizan subconsultas suelen ser más fáciles de interpretar por el usuario.
- Se suelen utilizar **subconsultas en las cláusulas WHERE o HAVING** cuando los datos que queremos visualizar están en una tabla pero para seleccionar las filas de esa tabla necesitamos comprobar el valor de un dato que está en otra tabla. En este caso la subconsulta actúa de operando dentro de la condición.

Ejemplo 39: En este ejemplo listamos el número y nombre de los empleados cuya fecha de contrato sea igual a la primera fecha de todos los pedidos de la empresa.

```
SELECT numemp, nombre FROM empleado
WHERE contrato = (SELECT MIN(fechapedido) FROM pedido);
```

1. Test de comparación con subconsulta.

- Es el equivalente al test de comparación simple. Se utiliza para comparar un valor de la fila que se está examinado con un **único valor producido por la subconsulta**. La subconsulta debe devolver una única columna, si no se produce un error. Si la subconsulta no produce ninguna fila o devuelve el valor nulo, el test devuelve el valor nulo, si la subconsulta produce varias filas, SQL devuelve una condición de error.
- La sintaxis es la siguiente:



Ejemplo 40: En este ejemplo se listan las oficinas cuyo objetivo sea superior a la suma de las ventas de los empleados de la oficina 21.

```
SELECT oficina, ciudad FROM oficina
WHERE objetivo > (SELECT SUM(ventas) FROM empleado WHERE
codoficina=21);
```

2. Test de pertenencia a conjunto (IN).

- Este test examina si el valor de la expresión es igual a uno de los valores incluidos en la lista de valores producida por la subconsulta. La subconsulta debe generar una única columna y las filas que sean. Si la subconsulta no produce ninguna fila, el test da falso. Tiene la siguiente sintaxis:

expresion IN subconsulta

Ejemplo 41: En este caso la subconsulta obtiene la lista de los números de oficina del *este*, mientras que la consulta principal obtiene los empleados cuyo número de oficina sea uno de los números de oficina del *este*. Por lo tanto se listan los empleados de las oficinas del *este*.

```
SELECT numemp, nombre, codoficina FROM empleado
WHERE codoficina IN (SELECT oficina FROM oficina WHERE region =
'este');
```

Ejercicio 17: (Subconsultas) Utiliza subconsultas para obtener los siguientes resultados:

- Las ciudades de las oficinas, los nombres de los empleados y sus fechas de contrato, de todos los empleados cuyas ventas sean menores que la media de ventas de todas las oficinas.

- b) Selecciona el nombre de aquellos empleados cuyas oficinas estén en el norte o en el oeste. Hacerlo de dos formas distintas.

Ejercicio 18: (*Test de comparación con subconsulta*): Utiliza subconsultas para obtener los siguientes resultados:

- a) Los nombres de los empleados, que sean representantes, cuyas ventas sean menores que el total de ventas de todas las oficinas de las regiones este y centro juntas.
- b) Selecciona el código y el nombre del cliente cuyo empleado representante trabaje en una oficina del oeste.
- c) Selecciona todos los datos de aquellos clientes cuyos representantes tienen una cuota mayor que sus ventas. Hacerlo de dos formas distintas.

Ejercicio 19: (operador IN) Utiliza subconsultas para obtener los siguientes resultados:

- a) Los nombres de los clientes y su límite de crédito, para todos los clientes que tengan un representante que pertenezca a una oficina que tenga objetivo y venta mayores que 10000.
- b) Saca empleados que estén en las oficinas del Oeste.
- c) Saca empleados que no están en las oficinas del Oeste.
- d) Saca la descripción de productos para los que se hayan hecho líneas de pedidos donde la cantidad sea mayor que 9.
- e) Saca empleadoscuyo jefe sea director en oficinas del Oeste.

Ejercicio 20: (Repaso de las subconsultas) Realiza las siguientes consultas utilizando subconsultas:

- a) Listar los nombres de los clientes que tienen asignado el representante Jaumes López, *Alvaro* .
- b) Listar los vendedores (*numemp*, *nombre*, y *nº de oficina*) que trabajan en oficinas que tienen ventas superiores a su objetivo.
- c) Listar los vendedores que no trabajan en oficinas dirigidas por el empleado 108.
- d) Listar los productos (*idproducto* y *descripción*) para los cuales no se haya hecho ningún pedido de 250 euros o más.
Hacer una segunda versión donde aparezcan también los productos que nunca hayan sido pedidos.

- e)* Listar los números de clientes asignados a Viguer Sánchez, Antonio que al menos hayan hecho un pedido superior a 30000.
- f)* Listar todos los datos de los clientes del representante del ejercicio anterior que nunca hayan hecho un pedido.
- g)* Listar las oficinas que tengan un objetivo menor que la suma de las cuotas de los empleados de las oficinas del Oeste.