

# Lenguaje SQL

## Contenidos

1. Lenguaje de Manipulación de Datos
2. Consultas SELECT básicas
3. Consultas SELECT de AGRUPACIÓN
4. Consultas SELECT multitabla

## Lenguaje de Manipulación de Datos

El **Lenguaje de Manipulación de Datos** (LMD, en inglés Data Manipulation Language, DML) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios de la misma llevar a cabo las tareas de consulta o manipulación de los datos (inserción, borrado, actualización y consultas) basado en el modelo de datos adecuado.



**MySQL Oficial – Sintaxis de las instrucciones DML**

Las **consultas** recuperan información de las tablas de una base de datos mediante la selección de campos, la realización de filtros y la transformación de los datos recuperados.

El uso de consultas permite:

- **Elegir tablas.** Se puede obtener información de una sola tabla o de varias.
- **Elegir campos.** Se pueden especificar los campos a visualizar de cada tabla.
- **Elegir registros.** Se pueden seleccionar los registros a mostrar en la hoja de respuestas dinámica, especificando un criterio.
- **Ordenar registros.** Se puede ordenar la información en forma ascendente o descendente.
- **Realizar cálculos.** Se pueden emplear las consultas para hacer cálculos con los datos de las tablas.

Además, en consultas más complejas se puede:

- **Crear tablas.** Se puede generar otra tabla a partir de los datos combinados de una consulta. La tabla se genera a partir de la hoja de respuestas dinámica.

- **Consultas encadenadas.** Utilizar una consulta como origen de datos para otras consultas (subconsulta). Se pueden crear consultas adicionales basadas en un conjunto de registros seleccionados por una consulta anterior.

## Consultas SELECT básicas

### Consultas con operaciones numéricas y literales

Comenzaremos por utilizar la instrucción SELECT como si fuera una calculadora.

Para ello utilizaremos los siguientes operadores aritméticos:

Operador	Función
+	Suma
-	Resta
*	Producto o Multiplicación
/	División
DIV	División entera
MOD o %	Resto división

### Ejemplos

Los siguientes ejemplos evalúan las expresiones que hay después de la cláusula SELECT y muestran el resultado por pantalla. El último ejemplo muestra tres columnas.

```
SELECT 20 * 150;
```

```
SELECT ((5 * 4.5) / 3) + 7;
```

```
SELECT 100 / 3, 100 DIV 3, 100 MOD 3;
```



#### Referencias

[MySQL – Operadores aritméticos](#)

[MySQL Oficial - Operadores](#)

Para el uso de literales o cadenas de caracteres basta con colocar unas comillas:

```
SELECT 'Felicidades';
```

```
SELECT 'Precio', '25 * 1.21 =', 25 * 1.21;
```

## Consultas con funciones predefinidas de información

A través de funciones ya predefinidas en el MySQL podemos obtener información sobre diferentes aspectos del estado de nuestra conexión y del sistema. Algunos ejemplos son:

La versión de MySQL y nuestro identificador (ID) de conexión.

```
SELECT version(), connection_id();
```

El usuario actual con el que estamos conectados y la base de datos seleccionada.

```
SELECT user(), database();
```

Para saber la fecha y hora (now()) o sólo la fecha current\_date()) del sistema.

```
SELECT now(), current_date();
```



### Referencias

[MySQL – Funciones de información](#)

[MySQL Oficial – Funciones de información](#)

## Consultas básicas a tablas

```
SELECT [DISTINCT] campos
[FROM table_references
[WHERE where_definition]
[ORDER BY {col_name | expr | position} [ASC | DESC] , ...]
[LIMIT [offset,] row_count]
```

Es importante conocer el orden de las cláusulas:

```
SELECT ... FROM ... WHERE ... ORDER ... LIMIT
```

## Utilidad de captura

En muchas ocasiones nos interesaría llevar un registro de las instrucciones SQL ejecutadas y su resultado. Conectados con la utilidad `mysql.exe` a MySQL podemos grabar toda la salida a un archivo a través de un comando:

```
mysql> tee archivo.txt
mysql> ... instrucciones SQL ...
```

Para terminar la salida:

```
mysql> notee
```

## Ejemplos de las primeras consultas

Con XAMPP-MySQL importar la BD de jardinería que facilitará el profesor.

### Ejemplo 1 - Tablas completas

Mostrar todos los campos de todos los clientes.

```
SELECT *
FROM clientes;
```

### Ejemplo 2 - Seleccionar campos a mostrar

Mostrar los campos `CodigoCliente`, `NombreCliente`, `Telefono`, `Ciudad`, `Region`, `Pais` de todos los clientes.

```
SELECT CodigoCliente, NombreCliente, Telefono, Ciudad, Region, Pais
FROM clientes;
```

### Ejemplo 3 – Campos calculados

Mostrar los campos `CodigoCliente`, `NombreCliente`, `LimiteCredito` de todos los clientes añadiendo un campo calculado que se el `LimiteCreditoMensual` como `LimiteCredito/12`. Haced la division entera para evitar decimales. Utilizaremos el **alias de campo** con la palabra reservada **AS**.

```
SELECT CodigoCliente, NombreCliente, LimiteCredito, LimiteCredito DIV 12 AS LimiteCreditoMensual
FROM clientes;
```

#### Ejemplo 4 – No mostrar repetidos

Mostrar las regiones (campo Region) todos los clientes evitando resultandos repetidos.

```
SELECT DISTINCT Region
FROM clientes;
```

#### Ejemplo E52506 – Ordenar registros

Mostrar todos los campos de todos los clientes ordenados por LimiteCredito ordenado descendentemente.

```
SELECT * FROM clientes
ORDER BY LimiteCredito DESC;
```

#### Ejemplo 5 – Limitar el número de registros a mostrar del resultado

Utilizando la consulta del ejercicio anterior E5106 mostrar:

- Sólo los 5 primeros registros

```
SELECT *
FROM clientes
ORDER BY LimiteCredito DESC
LIMIT 5;
```

- Empezando en el tercer registro, mostrar 10 registros

```
SELECT *
FROM clientes
ORDER BY LimiteCredito DESC
LIMIT 2,10;
```

### Consultas con selección de registros

Para poder seleccionar registros es necesario indicar la condición que deben cumplir los campos de un registro para ser mostrado. Para ello se utiliza la sección **WHERE** de la instrucción SQL.

Los operadores relacionales que nos permiten comparar el valor de los campos son:

Operador	Función
=	Igual a
<	Menor que
>	Mayor que
<=	Menor o igual
>=	Mayor ou igual
LIKE	Patrón que debe cumplir un campo cadena
BETWEEN	Intervalo de valores
IN	Conjunto de valores
IS NULL	Es nulo
IS NOT NULL	No es nulo

Veamos unos cuantos ejemplos:

### Ejemplo 1 – Buscar un registro por el valor de su clave


Mostrar todos los campos del cliente con código igual a 12.

 **Nota:** Al ser el campo clave, el resultado sólo mostrará un registro.

```
SELECT *
FROM clientes
WHERE CodigoCliente = 12;
```

### Ejemplo 2 – Buscar registros por el valor de un campo

Mostrar todos los campos de los clientes de la Región Madrid.

 **Nota:** Al no ser un campo clave, el resultado puede mostrar más de un registro.

```
SELECT *
FROM clientes
WHERE Region = 'Madrid';
```

### Ejemplo 3 – Buscar registros por comparación del valor de un campo

Mostrar todos los campos de los clientes con Límite de Crédito mayor de 50000€.

```
SELECT *  
FROM clientes  
WHERE LimiteCredito > 50000;
```

#### Ejemplo 4 – Buscar registros por comparación del valor de un campo

Mostrar todos los campos de los clientes con Límite de Crédito entre 10000€ y 60000€.

```
SELECT *  
FROM clientes  
WHERE LimiteCredito BETWEEN 10000 AND 60000;
```

#### Ejemplo 5 – Buscar registros por comparación del valor de un campo

Mostrar todos los campos de los clientes con la Región nula.

```
SELECT *  
FROM clientes  
WHERE Region IS NULL;
```

#### Ejemplo 5 – Ejemplo combinando con el apartado anterior

Mostrar los Países de los clientes con la Región nula, sin repetir valores.

```
SELECT DISTINCT Pais  
FROM clientes  
WHERE Region IS NULL;
```

#### Ejemplo 7 – Ejemplo con el operador LIKE

Mostrar los Empleados cuyo email contenga "jardin".

```
SELECT *  
FROM Empleados  
WHERE Email LIKE '%jardin%';
```

#### Nota:

% es una cadena de caracteres cualquiera y \_ es un sólo carácter cualquiera.

#### Ejemplo 8 –Ejemplo con IN

Mostrar los productos de las gamas 'Aromáticas', 'Herramientas' y Ornamentales'.

```
SELECT *
FROM productos
WHERE Gama IN ('Aromáticas', 'Herramientas', 'Ornamentales');
```



### Referencias

[MySQL – Consultas](#)

[MySQL – Operadores de comparación](#)

[MySQL Oficial – Operadores de comparación](#)

## Consultas con varias condiciones

Para poder realizar varias condiciones necesitamos combinarlas con operadores lógicos, que en MySQL son:

Operador	Función
AND	Y
OR	O
NOT	No

Veamos unos cuantos ejemplos:

### Ejemplo 1 – Ejemplo con varias condiciones

Mostrar código y nombre de los Productos que sean de la Gama 'Frutales' y CantidadEnStock sea mayor que 50 unidades.

📌 **Nota:** Mostremos también los campos implicados en las condiciones para comprobar el resultado.

```
SELECT CodigoProducto, Nombre, Gama,
FROM productos
WHERE Gama = 'Frutales' AND CantidadEnStock > 50);
```

### Ejemplo 2 – Ejemplo con varias condiciones

Mostrar código y nombre de los Clientes que sean de la Ciudad 'Madrid' o 'Barcelona'.

📌 **Nota:** Mostremos también los campos implicados en las condiciones para comprobar el resultado.



```
SELECT CodigoCliente, NombreCliente, Ciudad
FROM clientes
WHERE Ciudad = 'Madrid' OR Ciudad = 'Barcelona';
```



### Referencias

[MySQL – Consultas y operadores de comparación](#)

[MySQL Oficial – Consultas y operadores de comparación](#)

## Consultas con funciones

Las funciones nos permiten realizar transformaciones de los datos para obtener información. Existen multitud de funciones que procesan diferentes tipos de datos.

### *Algunas funciones con cadenas de caracteres*

Función	Descripción
<b>CONCAT</b> (cad1, cad2, ...)	Concatena cadenas
<b>UPPER</b> (cad)	Pasa a mayúsculas una cadena
<b>LOWER</b> (cad)	Pasa a minúsculas una cadena
<b>LTRIM</b> (cad)	Elimina de la cadena los espacios iniciales y finales
<b>LEFT</b> (cad, X)	Obtiene los X primeros caracteres de la cadena
<b>RIGHT</b> (cad, X)	Obtiene los X últimos caracteres de la cadena
<b>LENGTH</b> (cad)	Longitud de una cadena
<b>REPLACE</b> (cad, ant, pos)	Obtiene una cadena tomando <b>cad</b> como origen y cambiando <b>ant</b> por <b>pos</b>

### *Algunas funciones numéricas de un campo*

Función	Descripción
<b>RAND</b> ()	Número aleatorio entre 0 y 1
<b>POW</b> (num, exp)	Obtiene la potencia de num^exp

Función	Descripción
<b>FLOOR(num)</b>	Obtiene la parte entera de un número decimal
<b>ROUND(num, X)</b>	Redondea un número a X decimales
<b>SIGN(num)</b>	Devuelve 1 para positivo, 0 para 0 y -1 para negativo
<b>ABS(num)</b>	Obtiene el valor absoluto de un número

### *Algunas funciones de fechas de un campo*

Función	Descripción
<b>YEAR(campo)</b>	Muestra el año del valor de un campo de tipo fecha
<b>MONTH(campo)</b>	Muestra el mes del valor de un campo de tipo fecha
<b>DAY(campo)</b>	Muestra el día del valor de un campo de tipo fecha
<b>DAYNAME(campo)</b>	Muestra el nombre del día de la semana del campo
<b>DAYOFWEEK(campo)</b>	Devuelve un número indicando el día de la semana de un campo fecha: <b>Nota:</b> 1=Domingo, 2=Lunes, 3=Martes, 4=Miércoles, 5=Jueves, 6=Viernes, 7=Sábado.

### *<Algunas funciones numéricas con varios registros*

Función	Descripción
<b>COUNT(*)</b>	Cuenta los registros seleccionados
<b>MIN(campo)</b>	Valor mínimo del campo de los registros seleccionados
<b>MAX(campo)</b>	Valor máximo del campo de los registros seleccionados
<b>SUM(campo)</b>	Suma de los valores del campo
<b>AVG(campo)</b>	Media de los valores del campo



#### Referencias

[MySQL w3schools – Funciones de MySQL](#)

[MySQL Oficial – Funciones](#)

## Ejemplos de funciones para campos calculados y para condiciones

### Ejemplo 1 – Ejemplo con funciones

Mostrar código, nombre y "precio de venta al público" de los productos, pero ese precio debe ser con IVA incluido, es decir, agregarle al PrecioVenta el 21% multiplicándolo por 1.21. Asignar el alias **pvp** al nuevo campo calculado.

```
SELECT CodigoProducto, Nombre, PrecioVenta, ROUND(PrecioVenta * 1.21, 2) AS pvp
FROM Productos;
```

### Ejemplo 2 – Ejemplo con funciones en las condiciones

Seleccionar en la consulta anterior los productos que tengan el nuevo campo pvp mayor de 100€

```
SELECT CodigoProducto, Nombre, PrecioVenta, ROUND(PrecioVenta * 1.21, 2) AS pvp
FROM Productos
WHERE ROUND(PrecioVenta * 1.21, 2) > 100;
```

## Ejemplos de funciones con cadenas de caracteres

### Ejemplo 3 – Ejemplo con funciones

Seleccionar de los empleados el nombre completo (NombreCompleto) concatenando el nombre y los dos apellidos.

```
SELECT CONCAT(Nombre, ' ', Apellido1, ' ', Apellido2) AS NombreCompleto
FROM empleados;
```

### Ejemplo 4 – Ejemplo con funciones

Obtener la inicial del nombre de todos los empleados.

```
SELECT LEFT(Nombre, 1) AS inicial
FROM empleados;
```

### Ejemplo 5 – Ejemplo con funciones

Obtener el nombre de los empleados todo en mayúsculas.

```
SELECT UCASE(Nombre)
FROM empleados;
```

## Ejemplo 6 – Ejemplo con funciones

Obtener el código de las oficinas pero sustituyendo el guión por la barra, es decir, el '-' por '/'.

```
SELECT REPLACE(CodigoOficina, '-', '/')
FROM oficinas;
```

## Ejemplo 7 – Ejemplo con funciones

Obtener la primera palabra del nombre del proveedor de todos los productos.

```
SELECT SUBSTRING_INDEX(Proveedor, ' ', 1)
FROM productos;
```

## Ejemplo 8 – Ejemplo con funciones

Obtener el código de los productos pero con el orden de los caracteres invertido.

```
SELECT CodigoProducto, REVERSE(CodigoProducto)
FROM productos;
```

## Ejemplo 9 – Ejemplo con funciones

Obtener un gráfico de barras para el stock de cada producto. Utilizaremos el carácter '|' por cada unidad de stock.

```
SELECT CodigoProducto, REPEAT('|', CantidadEnStock)
FROM productos;
```

## Ejemplo 10 – Ejemplo con funciones

Obtener el email de cada empleado teniendo en cuenta que el usuario es su nombre en minúsculas y el dominio '@iesdoctorbalmis.com'

```
SELECT CONCAT(LCASE(nombre), '@iesdoctorbalmis.com') AS email
FROM empleados;
```

## Ejemplo 11 – Ejemplo con funciones

Obtener la abreviatura del nombre y primer apellidos de los empleados concatenando la inicial del nombre y la inicial del apellido1

```
SELECT CONCAT(LEFT(Nombre, 1), LEFT(Apellido1, 1)) AS inicial
FROM empleados;
```

## Ejemplos de funciones numéricas

### Ejemplo 12 – Ejemplo con funciones

Obtener un número aleatorio entre 0 y 9.

```
SELECT FLOOR(RAND() * 10);
```

## Ejemplos de funciones de fechas

### Ejemplo 13 – Ejemplo con funciones

Mostrar en campos diferentes el año, el mes y el día de la fecha de los pedido.

```
SELECT YEAR(FechaPedido), MONTH(FechaPedido), DAY(FechaPedido)
FROM pedidos;
```

### Ejemplo 14 – Ejemplo con funciones

Mostrar todos los campos de pedidos realizados en enero del 2009.

```
SELECT *
FROM pedidos
WHERE YEAR(FechaPedido) = 2009 AND MONTH(FechaPedido) = 1;
```

### Ejemplo 15 – Ejemplo con funciones

Mostrar todos los campos de pedidos realizados en lunes.

```
SELECT *
FROM pedidos
WHERE DAYOFWEEK(FechaEntrega) = 2;
```

## Ejemplos de funciones numéricas con varios registros

### Ejemplo 16 – Ejemplo con funciones

Mostrar el número total de pagos que se han recibido.

```
SELECT COUNT(*)
FROM pagos;
```

### Ejemplo 17 – Ejemplo con funciones

Mostrar la cantidad en euros total que nos han ingresado.

```
SELECT SUM(Cantidad)
FROM pagos;
```

## Ejemplo 18 – Ejemplo con funciones

Mostrar el importe del pago más pequeño y el importe de pago más grande recibido.

```
SELECT MIN(Cantidad), MAX(Cantidad)
FROM pagos;
```

# Consultas SELECT de AGRUPACIÓN

## Consultas de agrupación

Cuando necesitamos agrupar varios registros para realizar operaciones para sumar, contar, o calcular la media, el mínimo o el máximo, necesitaremos realizar una instrucción SELECT indicando qué registros agrupamos, es decir, qué campos mostramos de los registros comunes y sobre qué campos realizamos la operación.

La sintaxis para realizar una consulta agrupada a una tabla es:

```
SELECT [DISTINCT] campos
[FROM table_references
[WHERE where_definition]
[GROUP BY {col_name | expr | position} [ASC | DESC], ... ]
[HAVING where_definition]
[ORDER BY {col_name | expr | position} [ASC | DESC] , ...]
[LIMIT [offset,] row_count]
```

### Importante

El orden de las cláusulas obligatoriamente es:

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ORDER ... LIMIT
```

## Ejemplos de las primeras consultas agrupadas

Con XAMPP-MySQL importar la BD de jardinería que facilitará el profesor.

!!! Example Ejemplo 11

### Agrupaciones con condiciones

Mostrar las oficinas que tengan más de 3 empleados asignados y estén ubicadas en España.

```
```sql
SELECT CodigoOficina, COUNT(*) AS NumEmpleados
FROM empleados
WHERE Pais = 'España'
GROUP BY CodigoOficina
HAVING COUNT(*) > 3;
```
```

### Ejemplo 1

#### Agrupaciones con condiciones

Mostrar las gamas de productos cuyo precio máximo sea superior a 100€.

```
SELECT Gama, MAX(PrecioVenta) AS PrecioMaximo
FROM productos
GROUP BY Gama
HAVING MAX(PrecioVenta) > 100;
```

### Ejemplo 2

#### Agrupaciones con condiciones

Mostrar los clientes que hayan realizado más de 2 pagos y cuyo importe total sea superior a 5000€.

```
SELECT CodigoCliente, COUNT(*) AS NumPagos, SUM(Cantidad) AS TotalPagos
FROM pagos
GROUP BY CodigoCliente
HAVING COUNT(*) > 2 AND SUM(Cantidad) > 5000;
```

### Ejemplo 3

#### Agrupaciones con condiciones

Mostrar las ciudades donde el número total de clientes sea mayor a 5 y el límite de crédito promedio sea superior a 20000€.

```
SELECT Ciudad, COUNT(*) AS NumClientes, AVG(LimiteCredito) AS CreditoPromedio
FROM clientes
GROUP BY Ciudad
HAVING COUNT(*) > 5 AND AVG(LimiteCredito) > 20000;
```

## Ejemplo 4

### Agrupaciones con condiciones

Mostrar los empleados que hayan gestionado más de 5 pedidos y cuyo importe total sea superior a 15000€.

```
SELECT CodigoEmpleado, COUNT(*) AS NumPedidos, SUM(Cantidad * PrecioUnidad) AS TotalImp
FROM pedidos
INNER JOIN detallepedidos ON pedidos.CodigoPedido = detallepedidos.CodigoPedido
GROUP BY CodigoEmpleado
HAVING COUNT(*) > 5 AND SUM(Cantidad * PrecioUnidad) > 15000;
```

### Ejemplo 1 - Contar

Mostrar el número de clientes (nombrar el nuevo campo **NumClientes** ) que tenemos en cada ciudad .

```
SELECT Ciudad, COUNT(*) AS NumClientes
FROM clientes
GROUP BY Ciudad;
```

### Ejemplo 2 - Sumar operaciones numéricas

Mostrar el pedido y el importe total ( **ImpTotal** ) de todas las líneas de cada pedido. En la tabla detallepedidos tenemos el **CodigoPedido** , y la **Cantidad** y **PrecioUnidad** de cada artículo del pedido. Deberemos sumar **Cantidad \* PrecioUnidad** de cada línea y luego sumarmas todas.



```
-- Sin Agrupar
SELECT CodigoPedido, Cantidad * PrecioUnidad AS ImpLinea
FROM detallepedidos;

-- Agrupando por Pedido
SELECT CodigoPedido, SUM(Cantidad * PrecioUnidad) AS ImpTotal
FROM detallepedidos
GROUP BY CodigoPedido;
```

Podemos añadir también filtros que deban cumplir los registros mediante condiciones en la cláusula **WHERE** .

### Ejemplo 3 – Agrupaciones con condiciones WHERE

Mostrar el número de artículos ( **NumArticulos** ) de cada Gama cuyas **PrecioVenta** mayor que 20€.

```
SELECT Gama, COUNT(*) AS NumArticulos
FROM productos
WHERE PrecioVenta > 20
GROUP BY Gama;
```

Pero puede ser que la condición a cumplir sea sobre los campos calculados. En estos casos, la condición irá en la cláusula **HAVING**.

### Ejemplo 4 – Agrupaciones con condiciones HAVING

Mostrar las Gamas de artículos que tengan más de 100 diferentes. Mostrar también el número de artículos ( **NumArticulos** ).

```
SELECT Gama, COUNT(*) AS NumArticulos
FROM productos
GROUP BY Gama
HAVING COUNT(*) > 100;
```

Incluso podemos tener consultas que combinen **WHERE** y **HAVING** simultáneamente.

### Ejemplo 5 – Agrupaciones con condiciones WHERE y HAVING

Mostrar los clientes que hayan realizado más de 1 pago de cantidad superior a los 3000€.

```
SELECT CodigoCliente, COUNT(*) AS NumPagos
FROM pagos
WHERE Cantidad > 3000
GROUP BY CodigoCliente
HAVING COUNT(*) > 1;
```

!!! Example Ejemplo 6 \*\*Agrupaciones con condiciones\*\* Mostrar las oficinas que tengan más de 5 empleados asignados.

```
```sql
SELECT CodigoOficina, COUNT(*) AS NumEmpleados
FROM empleados
GROUP BY CodigoOficina
HAVING COUNT(*) > 5;
```
```

### Ejemplo 7

#### Agrupaciones con condiciones

Mostrar las gamas de productos cuyo precio medio sea superior a 50€.

```
SELECT Gama, AVG(PrecioVenta) AS PrecioMedio
FROM productos
GROUP BY Gama
HAVING AVG(PrecioVenta) > 50;
```

### Ejemplo 8

#### Agrupaciones con condiciones

Mostrar los clientes que hayan realizado pagos por un importe total superior a 10,000€.

```
SELECT CodigoCliente, SUM(Cantidad) AS TotalPagos
FROM pagos
GROUP BY CodigoCliente
HAVING SUM(Cantidad) > 10000;
```

### Ejemplo 9

#### Agrupaciones con condiciones

Mostrar los empleados que hayan gestionado más de 3 pedidos.

```
SELECT CodigoEmpleado, COUNT(*) AS NumPedidos
FROM pedidos
GROUP BY CodigoEmpleado
HAVING COUNT(*) > 3;
```

### Ejemplo 10

#### Agrupaciones con condiciones

Mostrar las ciudades donde el número total de clientes sea mayor a 10.

```
SELECT Ciudad, COUNT(*) AS NumClientes
FROM clientes
GROUP BY Ciudad
HAVING COUNT(*) > 10;
```

### Ejemplo 5

#### Agrupaciones con condiciones

Agrupaciones con condiciones **WHERE** y **HAVING**

Mostrar los clientes que hayan realizado más de 1 pago de cantidad superior a los 3000€.

```
SELECT CodigoCliente, COUNT(*) AS NumPagos
FROM pagos
WHERE Cantidad > 3000
GROUP BY CodigoCliente
HAVING COUNT(*) > 1;
```

## Consultas SELECT multitable

### Consultas multitable de cruce (relación 1-N)

Las consultas multitable nos van a permitir procesar información de varias tablas conjuntamente. La sintaxis es la misma que hemos visto anteriormente para la sintaxis SELECT, pero **en la cláusula FROM tendremos varias tablas**.

#### Ejemplo 1 – SELECT multitable

Mostrar los valores de la tabla **pagos** añadiendo el campo **NombreCliente**.

```
SELECT clientes.NombreCliente, pagos.*
FROM pagos, clientes
WHERE pagos.CodigoCliente = clientes.CodigoCliente;
```

¿Cuántos registros tienen la tabla pagos?

¿Cuántos registros tienen la tabla clientes?

¿Cuántos registros devuelve la consulta anterior?

Si se elimina la condición WHERE, ¿cuántos registros obtenemos?. Justifícalo.

#### Ejemplo 2 – SELECT multitable

Mostrar los valores de la tabla **pedidos** añadiendo el campo **NombreCliente**.

```
SELECT clientes.NombreCliente, pedidos.*
FROM pedidos, clientes
WHERE pedidos.CodigoCliente = clientes.CodigoCliente;
```

¿Cuántos registros tienen la tabla pedidos?

¿Cuántos registros tienen la tabla clientes?

¿Cuántos registros devuelve la consulta anterior?

Si se elimina la condición WHERE, ¿cuántos registros obtenemos?. Justifícalo.

### Ejemplo 3 – SELECT mutitable

Mostrar los valores de la tabla **empleados** añadiendo los campos **Ciudad** y **Pais de la Oficina**.

```
SELECT oficinas.Ciudad, oficinas.Pais, empleados.*
FROM oficinas, empleados
WHERE oficinas.CodigoOficina = empleados.CodigoOficina;
```

¿Cuántos registros tienen la tabla oficinas?

¿Cuántos registros tienen la tabla empleados?

¿Cuántos registros devuelve la consulta anterior?

Si se elimina la condición WHERE, ¿cuántos registros obtenemos?. Justifícalo.

En **relaciones reflexivas**, debemos cruzar una tabla consigo misma. Para poder hacer esto, tenemos que asignar un **alias** a cada tabla para diferenciarlas.

Veamos un ejemplo.

### Ejemplo 4 – SELECT mutitable

Mostrar los valores de la tabla **empleados** añadiendo los campos **Nombre y Apellido1 de su jefe**.

```
SELECT jefes.Nombre, jefes.Apellido1, trabajadores.*
FROM empleados as trabajadores, empleados as jefes
WHERE trabajadores.CodigoJefe = jefes.CodigoEmpleado;
```

¿Cuántos registros tienen la tabla empleados?

¿Cuántos registros devuelve la consulta anterior?

Si se elimina la condición WHERE, ¿cuántos registros obtenemos?. Justifícalo.

También podemos relacionar más de una tabla. Veamos un ejemplo.

### Ejemplo 5 – SELECT mutitable

Mostrar los siguientes valores:

- De pedidos: CodigoPedido y FechaPedido
- De detallepedidos: CodigoProducto y Cantidad
- De productos: Nombre y Gama

```
SELECT pedidos.CodigoPedido, pedidos.FechaPedido, detallepedidos.CodigoProducto, detallepedidos.Cantidad
FROM pedidos, detallepedidos, productos
WHERE pedidos.CodigoPedido = detallepedidos.CodigoPedido AND detallepedidos.CodigoProducto = productos.CodigoProducto;
```

¿Cuántos registros tienen la tabla empleados?

¿Cuántos registros devuelve la consulta anterior?

Si se elimina la condición WHERE, ¿cuántos registros obtenemos?. Justifícalo.

Se puede observar que cuando se muestran datos de la tabla padre de relaciones con cardinalidad 1-N, estos se repiten ya que pueden tener muchos hijos. En el ejemplo anterior se puede ver claramente que los datos de la tabla pedidos se repiten tantas veces como líneas de detalle tengan.

## Consultas multitable mediante INTERSECCIÓN

Las consultas de intersección se pueden ejecutar con la cláusula

```
tabla1 INNER JOIN tabla2 ON condicion
```

El ejemplo anterior quedaría:

### Ejemplo 1 – SELECT multitable con INNER JOIN

Mostrar los valores de la tabla **pagos** añadiendo el campo **NombreCliente**.

```
SELECT clientes.NombreCliente, pagos.*  
FROM (pagos INNER JOIN clientes ON pagos.CodigoCliente = clientes.CodigoCliente);
```

Veamos otro ejemplo de consulta multitable:

### Ejemplo 2 – SELECT multitable con INNER JOIN

Mostrar los valores de la tabla **detallepedidos** pero añadiendo el campo FechaPedido y Estado de la tabla pedidos. Comprueba que el resultado contiene el mismo número de registros que detallepedidos.

```
SELECT pedidos.FechaPedido, pedidos.Estado, detallepedidos.*  
FROM (detallepedidos INNER JOIN pedidos ON detallepedidos.CodigoPedido = pedidos.CodigoPedido);
```

En las **consultas multitable mediante intersección, es posible que haya registros que no se muestren por no haber cruce entre ellos**. Por ejemplo, si queremos saber el número de pedidos realizados por cada cliente, podríamos pensar en realizar la siguiente consulta:

### Ejemplo 3 – SELECT multitable con INNER JOIN

```
SELECT clientes.CodigoCliente, clientes.NombreCliente, COUNT(pedidos.CodigoPedido)
FROM clientes INNER JOIN pedidos ON clientes.CodigoCliente = pedidos.CodigoCliente
GROUP BY clientes.CodigoCliente, clientes.NombreCliente;
```

En la tabla de clientes hay 36 registros, pero el resultado de la consulta sólo muestra 19. Esto ocurre porque hay clientes que no han realizado todavía ningún pedido. Para evitar esto, se introduce un cambio en la consulta que permite incluir todos los registros de una tabla de la intersección, que en nuestro caso es clientes. Lo haremos con la cláusula **LEFT** de la relación **clientes-pedidos**:

#### Ejemplo 4 – SELECT mutitabla con LEFT JOIN

```
SELECT clientes.CodigoCliente, clientes.NombreCliente, COUNT(pedidos.CodigoPedido)
FROM clientes LEFT JOIN pedidos ON clientes.CodigoCliente = pedidos.CodigoCliente
GROUP BY clientes.CodigoCliente, clientes.NombreCliente;
```

Veamos otro ejemplo:

#### Ejemplo 5 – SELECT mutitabla con LEFT JOIN

Mostrar el CodigoProducto y nombre de la tabla productos junto el la suma de la cantidad (SumCantidad) pedida en todos los pedidos existentes. (tabla detallepedidos).

🚩 **Nota:** Tened en cuenta que de los productos que no haya habido ningún pedido debe aparecer 0.

```
SELECT productos.CodigoProducto, productos.Nombre, SUM(detallepedidos.Cantidad) AS SumCantidad
FROM productos LEFT JOIN detallepedidos ON productos.CodigoProducto = detallepedidos.CodigoPr
GROUP BY productos.CodigoProducto, productos.Nombre;
```

La cláusula **RIGHT** permitiría que se mostrarán los de la segunda tabla en la relación, en vez de **LEFT** que muestra la primera.

## Consultas multitabla mediante UNION

En ocasiones necesitamos unir el resultado de dos consultas. Para ello el resultado debe mostrar los mismo campos y con los mismos tipos de datos.

Por ejemplo, tenemos por un lado los clientes que han realizado pagos posteriores al 31/01/2009:

```
SELECT DISTINCT CodigoCliente
FROM pagos
WHERE FechaPago > '2009-01-31';
```

Por otra parte, tenemos los clientes que han realizado pedidos posteriores al 31/03/2009:

```
SELECT DISTINCT CodigoCliente
FROM pedidos
WHERE FechaPedido > '2009-03-31';
```

Los clientes que están en las dos consultas son 16, 23 y 27, pero hay clientes que han realizado pagos y no pedidos, y viceversa.

### Ejemplo 1 – SELECT multitabla con UNION

Deseamos saber el CodigoCliente de los clientes que han realizado pagos posteriores al 31/03/2009 o pedidos posteriores al 31/03/2009.

```
SELECT DISTINCT CodigoCliente
FROM pagos
WHERE FechaPago > '2009-01-31'
UNION
SELECT DISTINCT CodigoCliente
FROM pedidos
WHERE FechaPedido > '2009-03-31';
```

## Consultas multitabla mediante SUBCONSULTAS

Una **subconsulta** es una instrucción SELECT que se usa dentro de otra instrucción SELECT.

Una **subconsulta** será **correlacionada** si aparecen datos de la consulta, es decir, si no se puede ejecutar de forma independiente.

Existen varias situaciones donde usaremos subconsultas. A continuación veremos algunos ejemplos.

### Subconsultas en WHERE con operador de comparación

Para el siguiente ejemplo, primero buscamos la cantidad media que pagan los clientes.

```
SELECT AVG(Cantidad) AS CantidadMedia
FROM pagos;
```



## Ejemplo 1 – Subconsulta en WHERE con operador de comparación

Mostrar los registros de pagos que tengan cantidades superiores a la media.

```
SELECT *  
FROM pagos  
WHERE Cantidad > (SELECT AVG(Cantidad) AS CantidadMedia FROM pagos);
```

## Subconsultas en WHERE con operador IN

El operador IN devuelve verdadero si el valor del campo de un registro está en el conjunto de valores devuelto por la subconsulta.

## Ejemplo 2 – Subconsulta en WHERE con operador de IN

Mostrar la Gama de los productos que de los que se haya pedido más de 30 unidades.

```
SELECT DISTINCT Gama  
FROM productos  
WHERE CodigoProducto IN (SELECT CodigoProducto FROM detallepedidos WHERE Cantidad > 30);
```

También puede usarse de forma negativa con **NOT IN**

## Subconsultas en WHERE con operador EXISTS

El operador EXISTS es verdadero si la subconsulta devuelve al menos un registro.

## Ejemplo 3 – Subconsulta en WHERE con operador de EXISTS

Utiliza una subconsulta correlacionada para obtener los datos de los empleados que tenga algún cliente asignado.

```
SELECT *  
FROM empleados  
WHERE EXISTS (SELECT * FROM Clientes WHERE CodigoEmpleadoRepVentas = empleados.CodigoEmpleadoRepVentas);
```

También puede usarse de forma negativa con **NOT EXISTS**

## Subconsultas en FROM

Podemos utilizar subconsultas como tablas y colocarlas en la cláusula FROM.

## Ejemplo 4 – Subconsulta en FROM

Aunque la siguiente consulta se puede obtener mediante una consulta de intersección típica, usaremos una subconsulta para probar su funcionamiento en FROM de forma sencilla. Las subconsultas de FROM deben tener un alias que asignaremos con AS.

Muestra los datos de los empleados que trabajen en oficinas de Madrid.

```
SELECT empleados.*  
FROM empleados, (SELECT * FROM oficinas WHERE Ciudad = 'Madrid') AS OficinasMadrid  
WHERE empleados.CodigoOficina = OficinasMadrid.CodigoOficina;
```

La consulta anterior sin usar subconsulta sería:

```
SELECT empleados.*  
FROM empleados, oficinas  
WHERE empleados.CodigoOficina = oficinas.CodigoOficina AND oficinas.Ciudad = 'Madrid';
```