

## Tema 4. Consultas

### Índice

1.- Sentencia SELECT básica.....	2
2.- SELECT con WHERE.....	2
2.1.- Operadores lógicos.....	2
2.2.- LIKE y NOT LIKE.....	3
2.3.- IN y NOT INT.....	3
3.- SELECT DISTINCT.....	4
4.- SELECT con ORDER BY.....	4
4.1.- Limitar el número de resultados.....	4
5.- SELECT con JOINS.....	5
5.1.- Alias.....	6
5.2.- LEFT JOIN.....	6
5.3.- RIGHT JOIN.....	7
6.- SELECT con expresiones.....	7
6.1.- Funciones.....	7
7.- SELECT con agregados.....	9
7.1.- Funciones de agregación comunes.....	9
8.- SELECT con agregados agrupados.....	10
9.- Subconsultas.....	11
9.1.- Subconsultas con comparadores.....	12
9.2.- Subconsultas con IN y NOT IN.....	12
9.3.- Subconsultas en FROM.....	13
9.4.- Subconsultas relacionadas.....	13
9.5.- Subconsultas con EXITS.....	14
10.- Otros usos de subconsultas.....	15
10.1.- INSERT INTO SELECT.....	15
10.2.- DELETE o UPDATE con subconsultas.....	15
11.- Vistas.....	15

En esta unidad vamos a estudiar la sentencia más importante de SQL, que es la sentencia SELECT. La sentencia SQL SELECT se utiliza para recuperar o consultar datos de una o más tablas en su base de datos. Los datos devueltos se almacenan en una tabla de resultados.



## 1.- Sentencia SELECT básica

La consulta más básica que podríamos escribir sería una que seleccione varias columnas (campos) de la tabla con todas las filas.

Sintaxis básica:

```
SELECT columna1, columna2, ... FROM nombreTabla;
```

Si desea seleccionar todos los campos disponibles en la tabla, utilice la siguiente sintaxis:

```
SELECT * FROM nombreTabla;
```

[Prueba y práctica sobre la sentencia SELECT básica en este enlace](#)

## 2.- SELECT con WHERE

La cláusula WHERE se utiliza para filtrar registros. Se utiliza para extraer solo aquellos registros que cumplen una condición especificada.

```
SELECT columna1, columna2, ... FROM nombreTabla WHERE condición;
```

La cláusula WHERE se puede combinar con los operadores AND, OR y NOT.

```
SELECT columna1, columna2, ... FROM nombreTabla  
WHERE condición1 AND condición2 OR NOT (condición3);
```

Además, se puede utilizar con todos los operadores de comparación como puedes ver en la siguiente tabla:

### 2.1.- Operadores lógicos

Operadores	Descripción
> >=	Mayor que, mayor igual que
< <=	Menor que, menor igual que
=	Igual
<> !=	Distinto
IS NULL, IS NOT NULL	Comprueba si el campo es nulo o NO nulo
BETWEEN...AND... NOT BETWEEN... AND...	Comprueba si el valor está en el rango o NO lo está
LIKE , NOT LIKE	Cumple con el patrón
IN ( ), NOT IN ( )	Comprueba si el valor está o no está en a lista de valores
NOT condicion	Da la vuelta al valor de la condición



## 2.2.- LIKE y NOT LIKE

Con este operador puedes comprobar si los datos de una cadena coinciden con un patrón. Puedes utilizar los siguientes dos caracteres comodín en el patrón:

- % -> coincide con cualquier número de caracteres, incluso cero.
- \_ -> coincide solo con UN caracter.

Ejemplos:

*Supongamos que tenemos una tabla y vamos a:*

- *Mostrar los empleados cuyos nombres empiecen por la letra **A**.*

```
SELECT * FROM empleados WHERE nombre LIKE 'A%';
```

- *Mostrar los empleados cuyos nombres terminen por la letra **a**.*

```
SELECT * FROM empleados WHERE nombre LIKE '%a';
```

- *Muestra los empleados que tengan una **a** como segunda letra de su nombre.*

```
SELECT * FROM empleados WHERE nombre LIKE '_a';
```

## 2.3.- IN y NOT IN

Con este operador puedes comprobar si un campo está (o no) dentro de un conjunto de valores.

Ejemplos:

*Imaginemos que tenemos la tabla Coches con un campo **color** (ENUM) y un campo **modelo** (VARCHAR).*

- *Muestra los coches que son **Blanco, Rojo o Negro**.*

```
SELECT * FROM coches WHERE color IN ('Blanco','Rojo','Negro');
```

- *Muestra los coches que NO son **Rojo o Blanco**.*

```
SELECT * FROM coches WHERE color NOT IN ('Blanco','Rojo');
```

- *Muestras los coches que son **Audi y BMW**.*

```
SELECT * FROM coches WHERE marcas IN ('Audi', 'BMW');
```

[Prueba y práctica sobre la sentencia SELECT con WHERE en este enlace](#)



### 3.- SELECT DISTINCT

Aunque los datos de una base de datos pueden ser únicos, los resultados de una consulta en particular pueden no serlo. Tomemos como ejemplo una tabla de Películas, muchas películas diferentes pueden estrenarse el mismo año. En tales casos, SQL proporciona una forma conveniente de descartar filas que tienen un valor de columna duplicado mediante el uso de la palabra clave DISTINCT.

```
SELECT DISTINCT columna, otraColumna, ... FROM nombreTabla WHERE condiciones;
```

### 4.- SELECT con ORDER BY

La mayoría de los datos de las bases de datos reales se agregan sin ningún orden de columnas en particular. Como resultado, puede resultar difícil leer y comprender los resultados de una consulta a medida que el tamaño de una tabla aumenta a miles o incluso millones de filas.

Para ayudar con esto, SQL proporciona una forma de ordenar los resultados por una columna determinada en orden ascendente o descendente utilizando la cláusula ORDER BY.

```
SELECT columna, otraColumna, ... FROM nombreTabla  
WHERE condiciones ORDER BY columna ASC/DESC;
```

Cuando se especifica una cláusula ORDER BY, cada fila se ordena alfanuméricamente según el valor de la columna especificada. En algunas bases de datos, también puede especificar una intercalación para ordenar mejor los datos que contienen texto internacional.

#### 4.1.- Limitar el número de resultados

Otras cláusulas que se utilizan comúnmente con la cláusula ORDER BY son las cláusulas LIMIT y OFFSET, que son una optimización útil para indicar a la base de datos el subconjunto de los resultados en los que está interesado. LIMIT reducirá la cantidad de filas a devolver y la opción OFFSET especificará desde dónde comenzar a contar la cantidad de filas.

```
SELECT columna, otraColumna, ... FROM nombreTabla  
WHERE condiciones ORDER BY columna ASC/DESC  
LIMIT numLimite OFFSET numDesde;
```

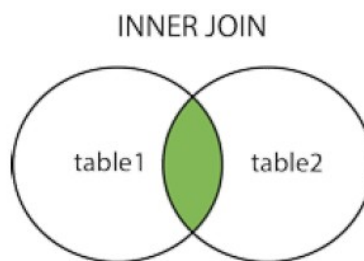
[Prueba y práctica sobre la sentencia SELECT con ORDER BY y LIMIT en este enlace](#)

## 5.- SELECT con JOINS

Al utilizar la cláusula JOIN en una consulta, podemos combinar filas de datos de dos tablas separadas utilizando esta clave única. La primera de las uniones que presentaremos es INNER JOIN. Consulta de selección con INNER JOIN en varias tablas

```
SELECT tabla1.campo1, tabla1.campo2, tabla2.campo1, ...  
FROM tabla1 INNER JOIN tabla2 ON tabla1.pkCampo = tabla2.fkCampo  
WHERE ... ;
```

INNER JOIN es un proceso que combina filas de la primera tabla y la segunda tabla que tienen la misma clave (tal como se define en la restricción ON) para crear una fila de resultados con las columnas combinadas de ambas tablas. Después de unir las tablas, se aplican las otras cláusulas que aprendimos anteriormente.



Consideremos la siguiente base de datos:

- Propietarios (dni, nombre, dirección, fechaNacimiento)
  - PK (dni)
- Coches (matrícula, marca, color, dniPropietario)
  - PK (matrícula) FK (dniPropietario -> Propietarios)

Ejemplos de SQL con JOIN:

- *Mostrar la matrícula, marca y color de todos los vehículos y el nombre de sus propietarios.*

```
SELECT coches.matricula, coches.marca, coches.color, propietarios.nombre  
FROM coches INNER JOIN propietarios ON propietarios.dni = coches.dniPropietarios;
```

- *Mostrar la matrícula de todos los coches rojos y el nombre de sus propietarios.*

```
SELECT coches.matricula, coches.marca, coches.color, propietarios.nombre  
FROM coches INNER JOIN propietarios ON propietarios.dni = coches.dniPropietarios  
WHERE coches.color = 'Rojo';
```

[Prueba y práctica sobre la sentencia SELECT con JOIN en este enlace](#)



### 5.1.- Alias

Los alias de SQL se utilizan para darle a una tabla o a una columna de una tabla un nombre temporal. Los alias se utilizan a menudo para que los nombres de las columnas sean más legibles. Un alias solo existe mientras dura esa consulta.

Sintaxis para el alias de columnas:

```
SELECT nombreColumna AS aliasColumna FROM nombreTabla;
```

Sintaxis para el alias de tablas:

```
SELECT nombreColumnas FROM nombreTabla AS nombreAlias;
```

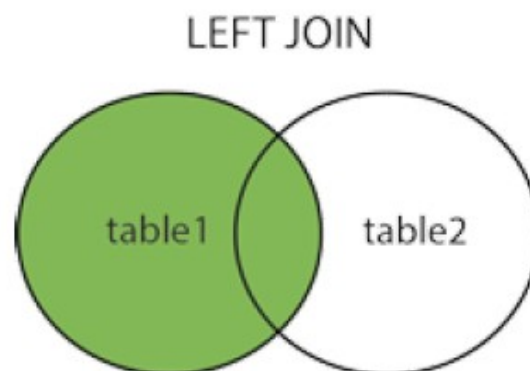
Un ejemplo de SQL con JOIN y ALIAS sería el siguiente:

```
SELECT c.matricula AS 'Matrículas de coche', c.marca AS 'Marcas de coche',  
c.color AS 'Colores', p.nombre AS 'Nombre del propietario'  
FROM coches AS c INNER JOIN propietarios AS p ON p.dni = c.dniPropietarios  
WHERE c.color = 'Rojo';
```

### 5.2.- LEFT JOIN

LEFT JOIN es una combinación que especifica que todos los registros se obtendrán de la tabla del lado izquierdo de la declaración de combinación. Si un registro devuelto de la tabla izquierda no tiene ningún registro coincidente en la tabla del lado derecho de la combinación, se devuelve de todos modos y la columna correspondiente de la tabla derecha devuelve un valor NULL.

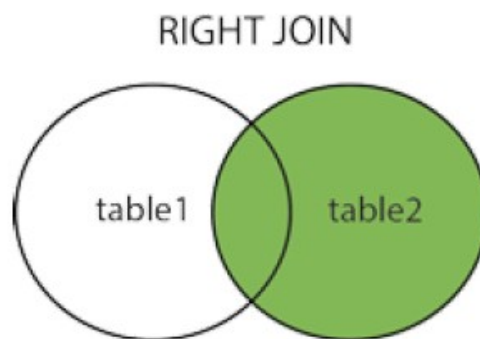
```
SELECT * FROM tabla1 LEFT JOIN tabla2 ON tabla1.campo1 = tabla2.campo2
```



### 5.3.- RIGHT JOIN

La combinación RIGHT JOIN es una combinación que especifica que todos los registros se obtendrán de la tabla del lado derecho de la declaración de combinación, incluso si la tabla de la izquierda no tiene ningún registro coincidente. En este caso, las columnas de la tabla de la izquierda devuelven valores NULL.

```
SELECT * FROM tabla1 RIGHT JOIN tabla2 ON tabla1.campo1 = tabla2.campo2
```



## 6.- SELECT con expresiones

Dentro de la declaración SELECT puedes crear una expresión con un número y un valor de campo usando operadores aritméticos.

```
SELECT expresión AS expNombre FROM nombreTabla;
```

Un ejemplo en la tabla Empleados (**dni**, nombre, salario) puede ser:

```
SELECT dni, nombre, salario, (12*salario+400) AS 'Salario anual' FROM empleados;
```

### 6.1.- Funciones

También puedes usar funciones en una expresión. Una función es una fórmula predefinida que toma uno o más argumentos como entrada, luego procesa los argumentos y devuelve un resultado. Hay diferentes tipos de funciones, como funciones matemáticas, funciones de cadena o funciones de fecha.

Una función matemática ejecuta una operación matemática generalmente basada en valores de entrada que se proporcionan como argumentos y devuelve un valor numérico como resultado de la operación. Las funciones matemáticas operan con datos numéricos como decimales, enteros, float, reales, smallint y tinyint.



Algunas funciones aritméticas son:

- [abs\(number\)](#)
- [ceil\(number\)](#)
- [floor\(number\)](#)
- [mod\(number1, number2\)](#)
- [power\(number1, number2\)](#)
- [sqrt\(number\)](#)

Una función de cadena es una función que toma uno o más caracteres o números como parámetros y devuelve un valor de carácter. Las funciones de cadena básicas ofrecen una serie de capacidades y devuelven un valor de cadena como conjunto de resultados.

Algunas funciones sobre String son:

- `lower(cadena)` -> devuelve la cadena en minúscula.
- `upper(cadena)` -> devuelve la cadena en mayúsculas.
- `trim(cadena)` -> devuelve la cadena eliminando los espacios al principio y al final.
- `length(cadena)` -> devuelve la longitud de la cadena, su número de caracteres.

Puedes consultar más funciones de MySQL en estos enlaces:

- [Funciones numéricas.](#)
- [Funciones con cadenas.](#)
- [Funciones de fecha y hora.](#)

Ejemplos:

- Mostrar el nombre de los empleados en mayúsculas.

```
SELECT upper(nombre) FROM empleados;
```

- Mostrar el nombre y el número de caracteres que tiene el nombre de cada empleado.

```
SELECT upper(nombre), length(nombre) FROM empleados;
```

- Mostrar el nombre y el año de nacimiento de los empleados.

```
SELECT upper(nombre), YEAR(fecNacimiento) FROM empleados;
```



## 7.- SELECT con agregados

Además de las expresiones simples que presentamos en la última lección, SQL también admite el uso de expresiones agregadas (o funciones) que permiten resumir información sobre un grupo de filas de datos.

```
SELECT AGG_FUNC(columna) AS AgrNombre ... FROM nombreTabla WHERE ...;
```

### 7.1.- Funciones de agregación comunes

Función	Descripción
COUNT(*) COUNT(columna)	Una función común que se utiliza para contar la cantidad de filas en el grupo si no se especifica ningún nombre de columna. De lo contrario, cuenta la cantidad de filas en el grupo con valores distintos de NULL en la columna especificada.
MIN(columna)	Encuentra el valor numérico más pequeño en la columna especificada para todas las filas del grupo.
MAX(columna)	Encuentra el valor numérico más grande en la columna especificada para todas las filas del grupo.
AVG(columna)	Encuentra el valor numérico promedio en la columna especificada para todas las filas del grupo.
SUM(columna)	Encuentra la suma de todos los valores numéricos en la columna especificada para las filas del grupo.

Consideremos la siguiente base de datos:

- Propietarios (dni, nombre, dirección, fechaNacimiento)
  - PK (dni)
- Coches (matrícula, marca, color, precio, dniPropietario)
  - PK (matricula) FK (dniPropietario -> Propietarios)

Todos los siguientes ejemplos devolver UN ÚNICO valor:

- Muestra el número de coches Rojos de la base de datos.

```
SELECT COUNT(*) FROM coches WHERE color = 'Rojo';
```

- Muestra el total de propietarios de coches.

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

- Muestra el total de colores distintos de todos los coches.

```
SELECT COUNT(color) FROM coches;
```

- Muestra el precio del coche más caro.

```
SELECT MAX(precio) FROM coches;
```



- Muestra el precio del BMW más barato.

```
SELECT MIN(precio) FROM coches WHERE marca = 'BMW';
```

- Muestra el precio medio de los coches rojos.

```
SELECT AVG(precio) FROM coches WHERE color = 'Rojo';
```

## 8.- SELECT con agregados agrupados

La sentencia GROUP BY en SQL se utiliza para organizar datos idénticos en grupos. La agrupación es una de las tareas más importantes que debe abordar al trabajar con datos. Para agrupar filas en grupos, utilice la cláusula GROUP BY.

La cláusula GROUP BY es una cláusula opcional de la sentencia SELECT que combina filas en grupos según los valores coincidentes en las columnas especificadas. Se devuelve una fila para cada grupo.

A menudo, se utiliza GROUP BY junto con una función de agregación como MIN, MAX, AVG, SUM o COUNT para calcular una medida que proporcione la información de cada grupo.

```
SELECT columnaAgrupada, FuncionAgregada(columna) FROM nombreTabla  
WHERE condicion GROUP BY columna HAVING condicionGrupo  
ORDER BY columna;
```

No es obligatorio incluir una función de agregación en la cláusula SELECT. Sin embargo, si utiliza una función de agregación, calculará el valor de resumen para cada grupo.

Si desea filtrar las filas antes de agruparlas, agregue una cláusula WHERE. Sin embargo, para filtrar grupos, utilice la cláusula HAVING.

Es importante destacar que la cláusula WHERE se aplica antes de agrupar las filas, mientras que la cláusula HAVING se aplica después de agruparlas. En otras palabras, la cláusula WHERE se aplica a las filas, mientras que la cláusula HAVING se aplica a los grupos.

[Prueba y práctica sobre la sentencia SELECT con agregados en este enlace](#)

Para ordenar los grupos, agregue la cláusula ORDER BY después de la cláusula GROUP BY.

Ejemplos:

Considerando la siguiente base de datos

- Departamento (id, nombre)
  - PK (id)
- Empleados (dni, nombre, apellidos, salario, idDepartamento)
  - PK (dni) FK (idDepartamento / Departamento)



Podemos realizar las siguiente consultas:

- Encontrar la cantidad de empleados de cada departamento, debe agrupar a los empleados por la columna departmentID y aplicar la función COUNT a cada grupo:

```
SELECT idDepartamento, COUNT(dni) AS NumEmpleados FROM empleados GROUP BY idDepartamento;
```

- Obtener el nombre del departamento en la consulta anterior, debe unir la tabla Empleados con la tabla Departamentos de la siguiente manera:

```
SELECT e.idDepartamento, d.nombre, COUNT(e.dni) AS NumEmpleados  
FROM empleados AS e INNER JOIN departamentos AS d ON e.idDepartamento = d.id  
GROUP BY e.idDepartamento;
```

- Mostrar los departamentos cuyo número de empleados es mayor a 3 en la consulta anterior, debe utilizar la cláusula HAVING como en la siguiente consulta:

```
SELECT e.idDepartamento, d.nombre, COUNT(e.dni) AS NumEmpleados  
FROM empleados AS e INNER JOIN departamentos AS d ON e.idDepartamento = d.id  
GROUP BY e.idDepartamento HAVING NumEmpleados > 3;
```

- Encontrar el promedio del salario de los empleados de cada departamento, debe agrupar a los empleados por la columna idDepartamento y aplicar la función AVG(salary) a cada grupo como la siguiente consulta:

```
SELECT idDepartamento, AVG(salario) AS SalarioMedioDpto  
FROM empleados GROUP BY idDepartamento;
```

[Prueba y práctica sobre la sentencia SELECT con agregados agrupados en este enlace](#)

## 9.- Subconsultas

Una subconsulta MySQL es una consulta anidada dentro de otra consulta, como SELECT, INSERT, UPDATE o DELETE. Una subconsulta también puede estar anidada dentro de otra subconsulta.

Una subconsulta MySQL se denomina consulta interna, mientras que la consulta que contiene la subconsulta se denomina consulta externa. Una subconsulta se puede utilizar en cualquier lugar donde se utilice esa expresión y debe estar cerrada entre paréntesis.

Por ejemplo, considere la base de datos VentasBD. La siguiente consulta utiliza una subconsulta para devolver los empleados que trabajan en las oficinas ubicadas en Canada.

```
SELECT nombre, apellidos FROM Empleados WHERE codigoOficina IN (  
    SELECT codidoOficina FROM oficina WHERE pais = 'Canada')
```



En este ejemplo:

- La subconsulta devuelve todos los códigos de oficina de las oficinas ubicadas en Canada.
- La consulta externa selecciona el apellido y el nombre de los empleados que trabajan en las oficinas cuyos códigos de oficina se encuentran en el conjunto de resultados devuelto por la subconsulta.

En este caso, esta consulta también se puede realizar mediante INNER JOIN, como se puede ver a continuación:

```
SELECT nombre, apellidos FROM Empleados AS e INNER JOIN Oficinas AS o ON  
e.codOficina = o.codOficina WHERE o.pais = 'Canada'
```

### 9.1.- Subconsultas con comparadores

Puede utilizar operadores de comparación, por ejemplo, =, >, < para comparar un único valor devuelto por la subconsulta con la expresión en la cláusula WHERE.

Por ejemplo, la siguiente consulta devuelve el cliente que tiene el pago más alto.

```
SELECT dni, idPedido, cantidad FROM Pagos WHERE cantidad = (  
    SELECT MAX(cantidad) FROM Pagos)
```

Además del operador =, puede utilizar otros operadores de comparación, como mayor que (>), mayor o igual que (>=), menor que (<) y menor o igual que (<=).

Por ejemplo, puede buscar clientes cuyos pagos sean mayores que el pago promedio mediante una subconsulta:

```
SELECT dni, idPedido, cantidad FROM Pagos WHERE cantidad > (  
    SELECT AVG(cantidad) FROM Pagos)
```

En este ejemplo:

- Primero, obtenga el pago promedio mediante una subconsulta.
- Luego, seleccione los pagos que sean mayores que el pago promedio devuelto por la subconsulta en la consulta externa.

### 9.2.- Subconsultas con IN y NOT IN

Si una subconsulta devuelve más de un valor, puede utilizar otros operadores como el operador IN o NOT IN en la cláusula WHERE.

```
SELECT nombre FROM Clientes WHERE dni NOT IN (  
    SELECT DISTINCT dni FROM Pedidos)
```



En este ejemplo:

- Primero recuperamos los dnis sin repetir de los pedidos, obteniendo la lista de clientes que han realizado algún pedido.
- Luego seleccionamos los nombre de los clientes que no están en la lista anterior, obteniendo la lista de clientes que no han realizado ningún pedido.

### 9.3.- Subconsultas en FROM

Cuando se utiliza una subconsulta en la cláusula FROM, el conjunto de resultados devuelto por una subconsulta se utiliza como una tabla temporal. Esta tabla se denomina tabla derivada.

La siguiente subconsulta busca la cantidad máxima, mínima y promedio de artículos en los pedidos de venta:

```
SELECT MAX(productos), MIN(productos), AVG(productos) FROM (  
    SELECT numPedido, COUNT(codProductos) AS productos FROM DetallesPedido  
    GROUP BY numPedido) AS TablaTemporal;
```

Tenga en cuenta que es obligatorio crear un alias de la tabla temporal.

### 9.4.- Subconsultas relacionadas

En los ejemplos anteriores, se observa que las subconsultas no dependen la una de la otra. Esto significa que se puede ejecutar la subconsulta como una consulta independiente, por ejemplo:

```
SELECT numPedido, COUNT(numPedido) AS 'Numero productos'  
FROM DetallesPedido GROUP BY numPedido;
```

A diferencia de una subconsulta independiente, una subconsulta correlacionada es una subconsulta que utiliza los datos de la consulta externa. En otras palabras, una subconsulta correlacionada depende de la consulta externa. Una subconsulta correlacionada se evalúa una vez por cada fila de la consulta externa.

El siguiente ejemplo utiliza una subconsulta correlacionada para seleccionar productos cuyos 'precios de compra' sean mayores que el precio de compra promedio de todos los productos en cada línea de productos.

```
SELECT nombreProducto, precio FROM Productos P1 WHERE precio >  
    (SELECT AVG(precio) FROM Productos WHERE Productos.idProducto = P1.idProducto)
```



En este ejemplo, tanto la consulta externa como la subconsulta correlacionada hacen referencia a la misma tabla de productos. Por lo tanto, debemos utilizar un alias de tabla P1 para la tabla de productos en la consulta externa.

A diferencia de una subconsulta normal, no se puede ejecutar una subconsulta correlacionada de forma independiente como esta. Si lo hace, MySQL no conoce la tabla P1 y mostrará un error.

```
SELECT AVG(precio) FROM Productos WHERE idProducto = P1.idProducto;
```

Para cada fila de la tabla de productos (o P1), la subconsulta correlacionada debe ejecutarse una vez para obtener el precio de compra promedio de todos los productos en la línea de productos de esa fila.

Si el precio de compra de la fila actual es mayor que el precio de compra promedio devuelto por la subconsulta correlacionada, la consulta incluye la fila en el conjunto de resultados.

### 9.5.- Subconsultas con EXISTS

Cuando se utiliza una subconsulta con el operador EXISTS o NOT EXISTS, la subconsulta devuelve un valor booleano VERDADERO o FALSO. La siguiente consulta ilustra una subconsulta utilizada con el operador EXISTS:

```
SELECT * FROM NombreTabla WHERE EXISTS( subconsulta );
```

En la consulta anterior, si la subconsulta devuelve alguna fila, la subconsulta EXISTS devuelve VERDADERO; de lo contrario, devuelve FALSO.

EXISTS y NOT EXISTS se utilizan a menudo en las subconsultas correlacionadas.

Echemos un vistazo a las tablas Orders y Orderdetails de SalesDB:

La siguiente consulta busca pedidos de venta cuyos valores totales sean mayores a 60000.

```
SELECT Pedidos.numPedidos, SUM(precioUnidadd * cantidadPedido) AS total
FROM DetallePedidos INNER JOIN Pedidos
ON Pedidos.numPedidos=DetallePedidos.numPedidos
GROUP BY numPedidos HAVING SUM(precioUnidadd * cantidadPedido) > 60000;
```

Puede utilizar la consulta anterior como una subconsulta correlacionada para encontrar clientes que realizaron al menos un pedido de venta con un valor total mayor a 60000 utilizando el operador EXISTS:

```
SELECT idCliente, nombreCliente FROM Clientes WHERE EXISTS
(SELECT Pedidos.numPedidos, SUM(precioUnidadd * cantidadPedido)
FROM DetallePedidos INNER JOIN Pedidos ON Pedidos.numPedidos=DetallePedidos.numPedidos
WHERE Pedidos.customerNumber = Clientes.idCliente
GROUP BY numPedidos HAVING SUM(precioUnidadd * cantidadPedido) > 60000);
```



## 10.- Otros usos de subconsultas

### 10.1.- INSERT INTO SELECT

Anteriormente, aprendió a insertar una o más filas en una tabla mediante la instrucción INSERT con una lista de valores de columna especificados en la cláusula VALUES.

Además de utilizar valores de fila en la cláusula VALUES, puede utilizar el resultado de una instrucción SELECT como fuente de datos para la instrucción INSERT.

A continuación, se muestra la sintaxis de la instrucción INSERT INTO SELECT:

```
INSERT INTO NombreTabla(listaColumnas)
SELECT listaCampos FROM otraTabla WHERE condicion...;
```

En esta sintaxis, en lugar de utilizar la cláusula VALUES, puede utilizar una sentencia SELECT. La sentencia SELECT puede recuperar datos de una o más tablas.

La sentencia INSERT INTO SELECT es muy útil cuando desea copiar datos de otras tablas a una tabla o resumir datos de varias tablas en una tabla.

### 10.2.- DELETE o UPDATE con subconsultas

En el comando DELETE o UPDATE, puede utilizar una subconsulta para filtrar los datos que desea eliminar o actualizar.

Ejemplos:

```
DELETE FROM NombreTabla WHERE campo IN/NOT IN (subconsulta)
```

```
UPDATE NombreTabla SET campo=nuevoValor WHERE otroCampo IN/NOT IN (subconsulta)
```

## 11.- Vistas

Por definición, una vista es una consulta con nombre almacenada en el catálogo de la base de datos.

Para crear una nueva vista, utilice la instrucción CREATE VIEW. Esta instrucción crea una vista pagosCliente basada en la consulta anterior:

```
CREATE VIEW PagosClientes AS SELECT nombreCliente, fechaPago, cantidad
FROM Clientes INNER JOIN Pagos ON Clientes.idCliente=Pagos.idCliente;
```

Una vez que ejecuta la instrucción CREATE VIEW, MySQL crea la vista y la almacena en la base de datos.



Ahora, puede hacer referencia a la vista como una tabla en instrucciones SQL. Por ejemplo, puede consultar datos de la vista `customerPayments` utilizando la instrucción `SELECT`:

```
SELECT * FROM PagosClientes;
```

Como puede ver, la sintaxis es mucho más sencilla.

Tenga en cuenta que una vista no almacena físicamente los datos. Cuando emite la instrucción `SELECT` en la vista, MySQL ejecuta la consulta subyacente especificada en la definición de la vista y devuelve el conjunto de resultados. Por este motivo, a veces, se hace referencia a una vista como una **tabla virtual**.