BD
Bases de Datos

UD 5
Selección de datos

UD5 – Selección de datos

INDICE

1. Introducción

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.

SQL Server Oficial – Sintaxis de las instrucciones DML

https://docs.microsoft.com/es-es/sql/t-sql/queries/queries?view=sql-server-ver15

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.

2. Consultas SELECT básicas

2.1 Consultas con operaciones numéricas y literales

Comenzaremos por utilizar la instrucción SELECT como si fuera una calculadora. Para ello utilizaremos los **operadores aritméticos**:

Operador	Función
+	Suma
-	Resta
*	Producto o Multiplicación
/	División
%	Resto de la división. Ejemplo. 12%5 = 2, porque el resto de 12 dividido entre 5 es 2.

Ejemplos.

```
SELECT 20*150;
SELECT ((5*4.5)/3)+7;
SELECT 100/3, 100 % 3;
```

Documentación oficial SQL Server – Operadores aritméticos
https://docs.microsoft.com/es-es/sql/t-sql/language-elements/arithmetic-operators-transact-sql?view=sql-server-ver15

Para el uso de literales o cadenas de caracteres basta con colocar comillas simples (están al lado del cero en el teclado). Para concatenar cadenas podemos utilizar el operador +

```
SELECT 'Hola mundo!';
```

	(Sin nombre de columna)
1	Hola mundo!

Si queremos darle un nombre a la columna podemos utilizar **AS NombreCol**

```
SELECT 'Hola mundo!' AS Columna1;
```

	Columna1
1	Hola mundo!

También podemos seleccionar tablas con un nombre diferente al original:

```
SELECT codCliente AS CodCli, NomCliente AS Nombre, Telefono AS Tlf
  FROM CLIENTES;
```

2.6 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
CONCAT(cad1,cad2,...)	Concatena cadenas
UPPER (cad)	Pasa a MAYÚSCULAS una cadena
LOWER (cad)	Pasa a minúsculas una cadena
LTRIM (cad)	Elimina de la cadena los espacios iniciales
RTRIM (cad)	Elimina de la cadena los espacios finales
TRIM (cad)	Elimina de la cadena los espacios iniciales y finales
LEFT (cad, X)	Obtiene los X primeros caracteres de la cadena
RIGHT (cad, X)	Obtiene los X últimos caracteres de la cadena
LEN (cad)	Longitud de una cadena
REPLACE (cad,ant,pos)	Obtiene una cadena tomando **cad** como origen y cambiando la cadena **ant** por **pos**. `SELECT REPLACE('hola mundo', 'hola', 'adios');` > adios mundo

SQL Server Documentación Oficial – Funciones cadenas de caracteres
https://docs.microsoft.com/es-es/sql/t-sql/functions/string-functions-transact-sql?view=sql-server-ver15

Algunas funciones numéricas de un campo

Función	Descripción
RAND()	Número aleatorio entre 0 y 1
POWER(num,exp)	Obtiene la potencia de num^{exp}
FLOOR(num)	Obtiene la parte entera de un número decimal
ROUND(num,X)	Redondea un número a X decimales
SIGN(num)	Devuelve 1 para positivo, 0 para 0 y -1 para negativo
ABS(num)	Obtiene el valor absoluto de un número,es decir, su valor sin signo

Algunas funciones de fechas de un campo

Función	Descripción
YEAR (campo)	Muestra el año del valor de un campo de tipo fecha
MONTH (campo)	Muestra el mes del valor de un campo de tipo fecha
DAY (campo)	Muestra el día del valor de un campo de tipo fecha
DATEADD (datepart, cantidad, fecha)	Datepart puede ser: day, month o year. Ejemplo. `SELECT DATEADD(year, 1, '20191230');`
DATEDIFF (datepart, fechaIni, fechaFin)	Datepart puede ser: day, month o year. Ejemplo. `SELECT DATEDIFF(year, '20060730', '20100830');`
DATENAME (datepart, fecha)	Datepart puede ser: day, month o year. Ejemplo. `SELECT DATENAME(month, '20200630');`

SQL Server Documentación Oficial – Funciones con fechas
https://docs.microsoft.com/es-es/sql/t-sql/functions/datename-transact-sql?view=sql-server-ver15

2.2 Consultas básicas a tablas

La sintaxis para realizar una consulta a una tabla es:

```
SELECT [TOP(n)] [DISTINCT] campo/s
  [FROM NOMBRE_TABLA]
[WHERE condición/es]
[ORDER BY {columna/s } [ASC | DESC] , ...]
```

Es importante conocer el orden de las cláusulas:

SELECT … FROM … WHERE … ORDER BY

2.3 Ejemplos de las primeras consultas

Ejemplo 01 - Tablas completas

Mostrar todos los campos de todos los clientes

```
SELECT *
  FROM CLIENTES;
```

Ejemplo 02 - Seleccionar campos a mostrar

Mostrar los campos codCliente, nombre_cliente, telefono, ciudad, region, pais de todos los clientes

```
SELECT codCliente, nombre_cliente, telefono, ciudad, region, pais
  FROM CLIENTES;
```

Ejemplo 03 – Campos calculados

Mostrar los campos codCliente, nombre_cliente, limite_credito de todos los clientes añadiendo un campo calculado que sea el limiteCreditoMensual como limite_credito/12. Hacer la división entera para evitar decimales.

Utilizaremos el **alias de campo** con la palabra reservada **AS**.

```
SELECT codCliente, nombre_cliente, limite_credito,
       limite_credito / 12 AS limiteCreditoMensual
  FROM CLIENTES;
```

Ejemplo 04 – No mostrar repetidos

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

```
SELECT DISTINCT Region
  FROM CLIENTES;
```

Ejemplo 05 – Ordenar registros

Mostrar todos los campos de todos los clientes ordenados por limite_credito ordenado descendentemente

```
SELECT *
  FROM CLIENTES
 ORDER BY limite_credito DESC;
```

Ejemplo 06 – Limitar el número de registros a mostrar del resultado

Utilizando la consulta del ejercicio anterior muestra únicamente los 5 primeros registros.

```
SELECT TOP(5) *
  FROM CLIENTES
 ORDER BY limite_credito DESC;
```

Algunas funciones numéricas con varios registros

Función	Descripción
COUNT(*) o COUNT(1) **COUNT(campo)**	Cuenta los registros seleccionados. Si simplemente queremos sacar el número de registros sin más campos adicionales, es más óptimo utilizar COUNT(1) que COUNT(*)
MIN (campo)	Valor mínimo del campo de los registros seleccionados
MAX (campo)	Valor máximo del campo de los registros seleccionados
SUM (campo)	Suma de los valores del campo de los registros
AVG (campo)	Media de los valores del campo de los registros

Funciones numéricas de varios registros

Ejemplo 07– Ejemplo con funciones

Mostrar el número total de clientes

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

Ejemplo 08 – Ejemplo con funciones

Mostrar la cantidad en euros de los productos dados de alta

```
SELECT SUM(precio_venta)
  FROM PRODUCTOS;
```

Ejemplo 09 – ***Ejemplo con funciones***

Mostrar el importe del producto más barato y el del producto más caro

```
SELECT MIN(precio_venta), MAX(precio_venta)
  FROM PRODUCTOS;
```

Ejemplo 10 – ***Ejemplo con funciones***

Obtener la media del importe de todos los productos dados de alta

```
SELECT AVG(precio_venta)
  FROM PRODUCTOS;
```

2.4 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 o igual
<>	Distinto
LIKE	Patrón que debe cumplir un campo cadena
BETWEEEN	Intervalo de valores
IN	Conjunto de valores
IS NULL	Es nulo
IS NOT NULL	No es nulo

Veamos algunos ejemplos:

Ejemplo 01 – Buscar un registro por el valor de su clave

Mostrar todos los campos del cliente con código igual a 6.
Nota: Al ser el campo clave, el resultado sólo mostrará un registro.

```
SELECT *
  FROM CLIENTES
 WHERE codCliente = 6;
```

Ejemplo 02 – Ejemplo con IN

Mostrar los clientes que sean de la ciudad de Madrid, Paris o London

```
SELECT *
  FROM CLIENTES
 WHERE ciudad IN ('Madrid', 'Paris', 'London');
```

Ejemplo 03 – Buscar registros por el valor de un campo

Mostrar todos los campos de los clientes de la región de Barcelona
Nota: Al no ser un campo clave, el resultado puede mostrar más de un registro.

```
SELECT *
  FROM CLIENTES
 WHERE region = 'Barcelona';
```

Ejemplo 04 – 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 limite_credito > 50000;
```

Ejemplo 05 – Buscar registros por comparación del valor de un campo

Mostrar todos los campos de los clientes con Límite de Crédito entre 3000€ y 7500€.

```
SELECT *
  FROM CLIENTES
 WHERE limite_credito BETWEEN 3000 AND 7500;
```

Ejemplo 06 – 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 07 – 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;
```

Comodines que podemos utilizar para realizar consultas:

%	es una cadena de caracteres cualquiera
_	es un sólo carácter cualquiera (subrayado)

Ejemplo 08 – Ejemplo con LIKE

Mostrar los Empleados cuyo email contenga "jardin"

```
SELECT *
  FROM EMPLEADOS
 WHERE email LIKE '%jardin%';
```

Ejemplo 09 – Ejemplo con LIKE

Mostrar los Empleados cuyo email esté compuesto por cualquier cadena y un servidor de 9 letras (como jardineria o cualquier otro) y que acabe en .es

```
SELECT *
  FROM EMPLEADOS
 WHERE email LIKE '%@_________.es';
```

SQL Server Documentación Oficial – Operadores de comparación

https://docs.microsoft.com/es-es/sql/t-sql/language-elements/comparison-operators-transact-sql?view=sql-server-ver15

2.5 Consultas con varias condiciones

Para poder realizar consultas con varias condiciones necesitamos combinarlas con operadores lógicos, que en SQL Server (y en otros SGBD) son:

Operador	Función
AND	Y
OR	O
NOT	No

Veamos algunos ejemplos:

Ejemplo 01 – ***Ejemplo con varias condiciones***

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

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

```
SELECT codProducto, nombre, gama, cantidad_en_stock
  FROM PRODUCTOS
 WHERE Gama='Frutales' AND cantidad > 50;
```

Ejemplo 02 – ***Ejemplo con varias condiciones***

Mostrar codigo 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 codCliente, nombre_cliente, ciudad
  FROM CLIENTES
 WHERE Ciudad='Madrid' OR Ciudad='Barcelona';
```

Podemos colocar tantas condiciones como sean necesarias. Es posible que sea necesario utilizar **paréntesis** para que se evalúe una condición antes que otras.

```
(Ciudad = 'Madrid' AND codCliente=2) OR Ciudad='Barcelona'
```

Funciones para campos calculados y para condiciones

Ejemplo 01 – 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 codProducto, nombre, precio_venta,
       ROUND(precio_venta * 1.21, 2) AS PVP
  FROM PRODUCTOS;
```

Ejemplo 02 – Ejemplo con funciones

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

```
SELECT CONCAT(LOWER(nombre),'@gmail.com') AS email
  FROM EMPLEADOS;
```

Funciones con cadenas de caracteres

Ejemplo 03 – Ejemplo con funciones

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

```
SELECT CONCAT(Nombre,' ',Apellido1,' ',Apellido2) AS NomCompleto
FROM EMPLEADOS;
```

Ejemplo 04 – Ejemplo con funciones

Obtener la inicial del nombre de todos los empleados

```
SELECT LEFT(apellido1,1) AS Inicial
  FROM EMPLEADOS;
```

Ejemplo 05 – Ejemplo con funciones

Obtener el nombre de los empleados todo en mayúsculas

```
SELECT UPPER(Nombre)
  FROM EMPLEADOS;
```

Ejemplo 06 – Ejemplo con funciones

Obtener el correo de los empleados, pero sustituyendo “gmail.com” por “gva.es”

```
SELECT REPLACE(email,'jardineria.com','gmail.es')
  FROM EMPLEADOS;
```

Ejemplo 07 – ***Ejemplo con funciones***

Obtener las tres primeras letras del nombre de los productos

```
SELECT SUBSTRING (nombre, 1, 3)
  FROM PRODUCTOS;
```

Ejemplo 08 – ***Ejemplo con funciones***

Obtener el nombre de los productos, pero con el orden de los caracteres invertido

```
SELECT nombre, REVERSE(nombre)
  FROM PRODUCTOS;
```

Ejemplo 09 – ***Ejemplo con funciones***

Obtener la abreviatura del nombre y primer apellido 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;
```

Ejemplo 10 – ***Ejemplo con funciones***

Obtener un número aleatorio entre 0 y 9

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

Funciones con fechas

Ejemplo 11 – ***Ejemplo con funciones***

Mostrar todos los campos de los pedidos que se realizaron el mes de enero

```
SELECT *
  FROM PEDIDOS
 WHERE DATENAME (month, fecha_pedido) = 'Enero';
```

Ejemplo 12 – ***Ejemplo con funciones***

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

```
SELECT YEAR(fecha_pedido), MONTH(fecha_pedido), DAY(fecha_pedido)
  FROM PEDIDOS;
```

Ejemplo 13 – ***Ejemplo con funciones***

Mostrar todos los campos de los pedidos que se registraron el mes de diciembre de 2008

```
SELECT *
  FROM PEDIDOS
 WHERE YEAR(fecha_pedido) = 2008
   AND DATENAME(MONTH, fecha_pedido) = 'Diciembre';
```

3. Consultas SELECT de AGRUPACIÓN

3.1 Consultas de agrupación

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

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

```
SELECT [DISTINCT] campos
 [FROM tabla/s
[WHERE condiciones sobre los campos]
[GROUP BY expresión1, expresión2 ,... ]
[HAVING condiciones sobre la agrupación]
 [ORDER BY {col_name | expr | position} [ASC | DESC] , ...]
```

Es importante conocer el orden de las cláusulas:

SELECT … FROM … WHERE … GROUP BY … HAVING … ORDER BY

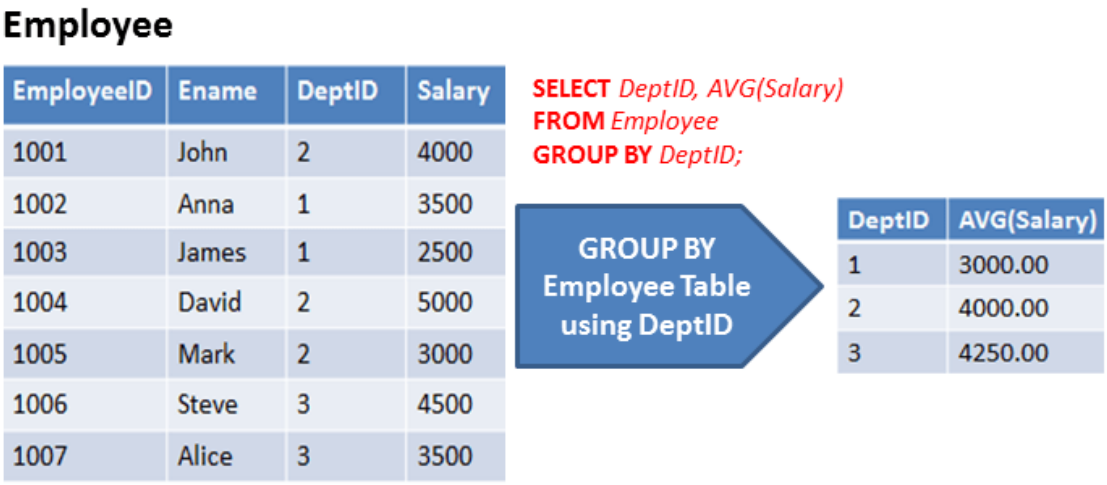
Es importante recalcar que sólo podremos utilizar las funciones de agregado (MIN, MAX, AVG, COUNT, etc.) en la cláusula SELECT y en el HAVING, pero nunca en el WHERE.

Con este tipo de consultas, podremos dar respuesta a preguntas como estas:

- ¿Cuántos clientes son de la ciudad de Madrid?
- ¿Cuál es la cantidad total que ha gastado cada cliente en la tienda virtual?
- ¿Cuál es la media del gasto de los clientes?

Cláusula GROUP BY

Se utiliza con la instrucción SELECT y siempre se coloca después de la cláusula WHERE. Sirve para combinar registros y utilizar funciones de agregado (COUNT(*), MAX, MIN, SUM, AVG) para combinarlos en una columna o en varias. Es conveniente recordar que GROUP BY mostrará un único resultado por grupo de datos.

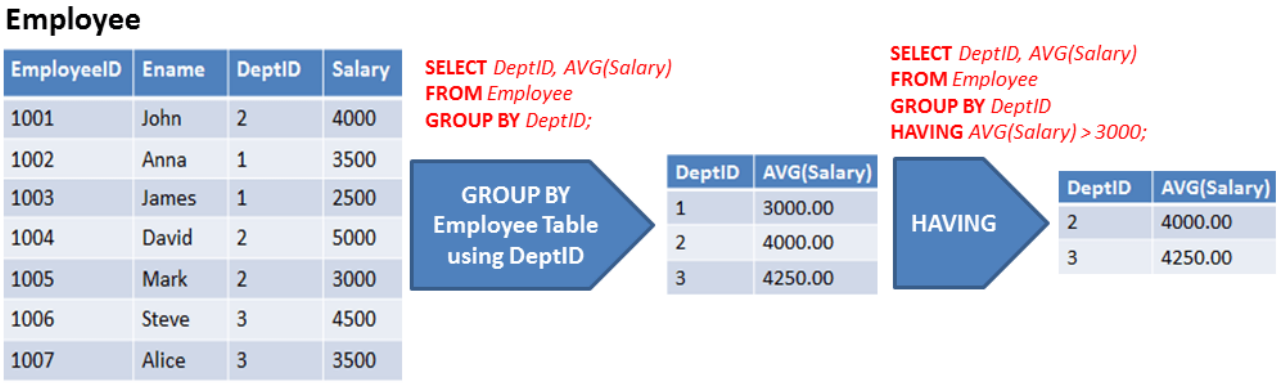


Error típico al realizar estas consultas:

Cuando se ejecuta una SELECT con GROUP BY se comprueba que todas las columnas incluidas en la cláusula SELECT estén dentro de la cláusula GROUP BY, aunque pueden estar en cualquier orden o bien tener funciones de agregado (SUM, MIN, MAX, etc.). Si encuentra alguna columna en el SELECT (que no esté dentro de una función sumaria) que no aparezca en el GROUP BY, devuelve el error: "*Not a GROUP BY expression*"

Cláusula HAVING

Se puede utilizar en las SELECT que tengan cláusula GROUP BY y sirve para poner condiciones sobre la agrupación (es como el WHERE del GROUP BY).



3.2 Ejemplos de las primeras consultas agrupadas

A continuación, veremos algunos ejemplos de consultas agrupadas:

Ejemplo 01 - 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 02 – Sumar operaciones numéricas

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

```
-- Sin Agrupar
SELECT codPedido, cantidad * precio_unidad AS ImpLinea
  FROM DETALLE_PEDIDOS;
```

```
-- Agrupando por Pedido
SELECT codPedido, SUM(cantidad * precio_unidad) AS ImpTotal
  FROM DETALLE_PEDIDOS
 GROUP BY codPedido;
```

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

Ejemplo 03 – 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 precio_venta > 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 04 – 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 05 – Agrupaciones con condiciones WHERE y HAVING

Mostrar los clientes que hayan realizado más de 1 pago de importe superior a 5€.

```
SELECT codCliente, COUNT(*) AS NumPagos
  FROM pagos
 WHERE total > 5
 GROUP BY codCliente
HAVING COUNT(*) > 1;
```

4. Consultas SELECT de varias tablas

Las bases de datos relacionales almacenan sus datos en varias tablas. Lo normal, en casi cualquier consulta, es requerir datos de varias tablas a la vez. Esto es posible porque los datos de las tablas están ligados por columnas que contienen *claves ajenas* que permiten relacionar los datos de esa tabla con datos de otra tabla.

4.1 Consultas JOIN entre varias tablas

En SQL es posible hacer esto especificando más de una tabla en la cláusula FROM de la instrucción SELECT. La única condición que deberemos tener muy en cuenta será que deberemos utilizar ALIAS para las tablas y en la cláusula WHERE igualar los campos relacionados como claves ajenas.

Ejemplo.

Tabla *DEPARTAMENTOS*

codDpto	nombre
1	Dirección
2	Informática
3	RRHH
4	Administración

Tabla *EMPLEADOS*

codEmpl	nombre	apellido	edad	codDpto
1	Carmen	Sánchez	34	1
2	Felipe	Palomar	18	1
3	Andrés	Fuster	27	2
4	Isabel	Jiménez	54	2
5	Ana	Albert	44	3
6	Javier	Rodríguez	21	4
7	Antonio	Pérez	64	

La columna codDpto en la tabla de empleados es una clave ajena. A través de ella sabemos que Ana Albert, por ejemplo, es del departamento de RRHH.

La consulta que devolverá los datos de los empleados y el nombre del departamento en el que trabajan será:

```
SELECT emple.codEmpl, emple.nombre, emple.apellido,
       emple.edad, dpto.nombre
  FROM DEPARTAMENTOS dpto,
       EMPLEADOS emple
 WHERE dpto.codDpto = emple.codDpto;

NOTAS
1) dpto y emple son alias de las tablas Departamentos y Empleados
2) Si no lo hubiéramos planteado de este modo, no podríamos distinguir el campo
codDpto entre ambas tablas puesto que se llama igual.
3) También podemos utilizar el nombre completo de la tabla como alias.
```

A continuación, veremos varios ejemplos de consultas con varias tablas involucradas:

Ejemplo 01 – SELECT multitabla

Mostrar los valores de la tabla **pagos** añadiendo el campo **nombreCliente**

```
SELECT c.nombre_cliente, p.*
  FROM PAGOS p,
       CLIENTES c
 WHERE p.codCliente =  c.codCliente;
```

¿Cuántos registros tiene la tabla pagos? ___
¿Cuántos registros tiene la tabla clientes? ___
¿Cuántos registros devuelve la consulta anterior? ___
Si se elimina la condición WHERE, ¿cuántos registros obtenemos? Justifícalo.

Ejemplo 02 – SELECT multitabla

Mostrar los valores de la tabla **pedidos** añadiendo el campo **nombreCliente**

```
SELECT c.nombre_cliente, p.*
  FROM PEDIDOS AS p,
       CLIENTES AS c
 WHERE p.codCliente =  c.codCliente;
```

¿Cuántos registros tiene la tabla pedidos? ___
¿Cuántos registros tiene la tabla clientes? ___
¿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 03 – SELECT multitabla

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

```
SELECT trabajadores.*,
       jefes.Nombre nombreJefe, jefes.Apellido1 apellidoJefe
  FROM EMPLEADOS AS trabajadores,
       EMPLEADOS AS jefes
 WHERE trabajadores.codEmplJefe =  jefes.codEmpleado;
```

¿Cuántos registros tienen la tabla empleados? ___
¿Cuántos registros devuelve la consulta anterior? ___
¿Por qué no aparece el codEmpleado 1? ¿Es correcto?
Si se elimina la condición WHERE, ¿cuántos registros obtenemos? Justifícalo.

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

Ejemplo 04 – SELECT multitabla
Mostrar los siguientes valores: • De PEDIDOS: codPedido y fechaPedido • De DETALLE_PEDIDOS: codProducto y cantidad • De PRODUCTOS: nombre y gama

`SELECT ped.codPedido, ped.fecha_pedido,`
`detped.codProducto, detped.cantidad,`
`prod.nombre, prod.gama`
`FROM PEDIDOS ped,`
`DETALLE_PEDIDOS detped,`
`PRODUCTOS prod`
`WHERE ped.codPedido = detped.codPedido`
`AND detped.codProducto = prod.codProducto;`

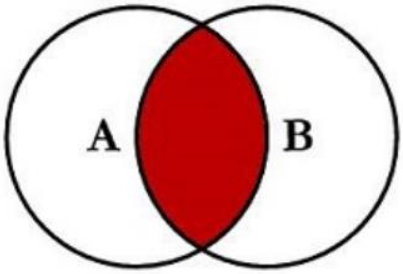
¿Cuántos registros tienen la tabla empleados? ___ ¿Cuántos registros devuelve la consulta anterior? ___ Si se elimina la condición WHERE, ¿cuántos registros obtenemos? Justifícalo.

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.

Además de utilizar los ALIAS para relacionar tablas en las consultas JOIN, también podemos utilizar una sintaxis alternativa para hacerlo que nos permitirá entender mejor las combinaciones derecha, izquierda y externa.

Las sintaxis es la siguiente:

tabla1 INNER JOIN tabla2 ON condicion



SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key

Si fuera necesario añadir condiciones, podremos ponerlas con WHERE después de la condición de la cláusula ON.

tabla1 INNER JOIN tabla2 ON condición
WHERE …

El ejemplo anterior quedaría:

Ejemplo 01 – SELECT multitabla con INNER JOIN

Mostrar los valores de la tabla **pagos** añadiendo el campo **nombreCliente**

```
SELECT c.nombre_cliente, p.*
  FROM PAGOS p INNER JOIN CLIENTES c ON p.codCliente =  c.codCliente;
```

Veamos otro ejemplo de consulta multitabla:

Ejemplo 02 – SELECT multitabla con INNER JOIN

Mostrar los valores de la tabla **DETALLE_PEDIDOS** pero añadiendo el campo fecha_pedido y estado de la tabla **PEDIDOS**. Comprueba que el resultado contiene el mismo número de registros que **DETALLE_PEDIDOS**.

```
SELECT p.fecha_pedido, p.estado, dp.*
  FROM DETALLE_PEDIDOS dp INNER JOIN PEDIDOS p
          ON dp.codPedido =  p.codPedido;
```

En las consultas multitabla mediante INNER JOIN, 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 03 – SELECT multitabla con INNER JOIN

```
SELECT c.codCliente, c.nombre_cliente, COUNT(p.codPedido) AS num
  FROM CLIENTES c INNER JOIN PEDIDOS p
                       ON c.codCliente = p.codCliente
 GROUP BY c.codCliente, c.nombre_cliente;
```

codCliente	NombreCliente	CUENTA
1	Antonio Gil	1
2	Filomena Suárez	1
3	Álvaro Moreno	1
5	Roberto Vázquez	1
6	Jose Luis Ramírez	2

En la tabla de clientes hay 8 registros, pero el resultado de la consulta sólo muestra 5. Esto ocurre porque hay clientes que no han realizado todavía ningún pedido.

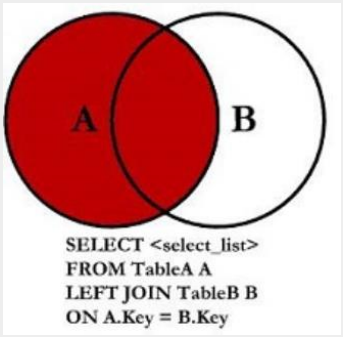
Para evitar esto, se introduce un cambio en la consulta que permite incluir todos los registros de una tabla de la intersección, que en nuestro caso es clientes.

Lo haremos con la cláusula **LEFT** de la relación **clientes-pedidos:**

Ejemplo 04 – SELECT multitabla con LEFT JOIN

```
SELECT c.codCliente, c.nombre_cliente, COUNT(p.codPedido) AS num
  FROM CLIENTES c LEFT JOIN PEDIDOS p
                          ON c.codCliente = p.codCliente
 GROUP BY c.codCliente, c.nombre_cliente;
```

codCliente	NombreCliente	Cuenta
1	Antonio Gil	1
2	Filomena Suárez	1
3	Álvaro Moreno	1
4	David Díaz	0
5	Roberto Vázquez	1
6	Jose Luis Ramírez	2
7	Pepe Domínguez	0
8	Alfonso Prieto	0



Para aquellos clientes que no hayan realizado ningún pedido se le asignará 0 en la cuenta.

Veamos otro ejemplo:

Ejemplo 05 – SELECT multitabla con LEFT JOIN

Mostrar de la tabla productos el codProducto y su nombre, junto a la suma de la cantidad (SumCantidad) pedida en todos los pedidos existentes. (tabla DETALLE_PEDIDOS).
Nota: Ten en cuenta que de los productos que no haya habido ningún pedido debe aparecer 0.

```
SELECT p.codProducto, p.nombre,
       SUM(dp.cantidad) AS SumCantidad
  FROM PRODUCTOS p LEFT JOIN DETALLE_PEDIDOS dp
                          ON p.codProducto = dp.codProducto
GROUP BY p.codProducto, p.Nombre;
```

	codProducto	nombre	SumCantidad
34	FR-19	Camelia Blanco, Chrysler Rojo,...	NULL
35	FR-2	Naranjo -Plantón joven 1 año i...	1
36	FR-20	Landora Amarillo, Rose Gaujar...	NULL
37	FR-21	Kordes Perfect bicolor rojo-am...	NULL
38	FR-22	Pitimini rojo	10
39	FR-23	Rosal copa	8
40	FR-24	Albaricoquero Corbato	NULL
41	FR-25	Albaricoquero Moniqui	NULL
42	FR-26	Albaricoquero Kurrot	NULL
43	FR-27	Cerezo Burlat	NULL
44	FR-28	Cerezo Picota	NULL
45	FR-29	Cerezo Napoleón	120

Para evitar que muestre NULL cuando no haya registros y en su lugar aparezca 0 o cualquier otro valor podemos recurrir a la función integrada **ISNULL**.

ISNULL (valor, valor-siNulo)

- Si valor IS NOT NULL, mostrará valor
- Si valor IS NULL, mostrará valor-siNulo

Ejemplo 06 – SELECT multitabla con LEFT JOIN (mejorado)

Mostrar de la tabla productos el codProducto y su nombre, junto a la suma de la cantidad (SumCantidad) pedida en todos los pedidos existentes. (tabla DETALLE_PEDIDOS).
Nota: Ten en cuenta que de los productos que no haya habido ningún pedido debe aparecer 0.

```
SELECT p.codProducto, p.nombre,
       ISNULL(SUM(dp.cantidad), 0) AS SumCantidad
  FROM PRODUCTOS p LEFT JOIN DETALLE_PEDIDOS dp
                             ON p.codProducto = dp.codProducto
GROUP BY p.codProducto, p.Nombre;
```

	codProducto	nombre	SumCantidad
34	FR-19	Camelia Blanco, Chrysler Rojo,...	0
35	FR-2	Naranjo -Plantón joven 1 año i...	1
36	FR-20	Landora Amarillo, Rose Gaujar...	0
37	FR-21	Kordes Perfect bicolor rojo-am...	0
38	FR-22	Pitimini rojo	10
39	FR-23	Rosal copa	8
40	FR-24	Albaricoquero Corbato	0
41	FR-25	Albaricoquero Moniqui	0
42	FR-26	Albaricoquero Kurrot	0
43	FR-27	Cerezo Burlat	0
44	FR-28	Cerezo Picota	0
45	FR-29	Cerezo Napoleón	120

Por otra parte, podemos utilizar la cláusula **RIGHT**, la cual permitiría que se mostrarán los de la segunda tabla en la relación, en vez de **LEFT** que muestra la primera.

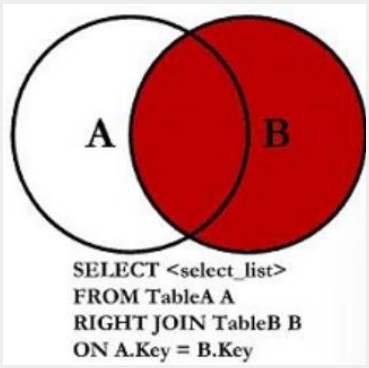
No obstante, podemos utilizar siempre la cláusula **LEFT** sabiendo que la tabla que debemos colocar en la parte izquierda es la que queremos que aparezcan todos los registros, independientemente de si tienen relación con la otra tabla.

Recordatorio: Las consultas realizadas con LEFT JOIN y RIGHT JOIN son totalmente equivalentes controlando la tabla que se coloca a la izquierda o derecha, respectivamente.

La consulta anterior quedaría como sigue utilizando RIGHT JOIN:

```
SELECT p.codProducto, p.nombre,
       ISNULL(SUM(dp.cantidad), 0) AS SumCantidad
  FROM DETALLE_PEDIDOS dp RIGHT JOIN PRODUCTOS p
                          ON p.codProducto = dp.codProducto
GROUP BY p.codProducto, p.Nombre;
```

	codProducto	nombre	SumCantidad
34	FR-19	Camelia Blanco, Chrysler Rojo,...	0
35	FR-2	Naranjo -Plantón joven 1 año i...	1
36	FR-20	Landora Amarillo, Rose Gaujar...	0
37	FR-21	Kordes Perfect bicolor rojo-am...	0
38	FR-22	Pitimini rojo	10
39	FR-23	Rosal copa	8
40	FR-24	Albaricoquero Corbato	0
41	FR-25	Albaricoquero Moniqui	0
42	FR-26	Albaricoquero Kurrot	0
43	FR-27	Cerezo Burlat	0
44	FR-28	Cerezo Picota	0
45	FR-29	Cerezo Napoleón	120

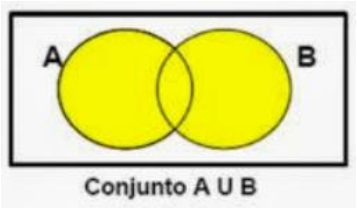


4.2 Consultas de conjuntos

En este punto del tema recuperamos todos los conocimientos aprendidos acerca de los conjuntos del Modelo Relacional. Recordemos que teníamos:

UNION

R ∪S, la unión de R y S define el conjunto de **elementos que están en R, en S o en ambos**. **Un elemento solo aparece una vez, por lo que no habrá tuplas repetidas**.

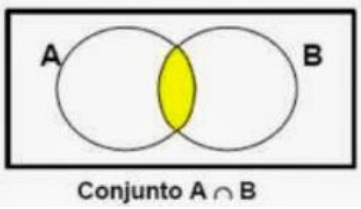


Requisitos para poder utilizarla:

- R y S deben tener esquemas idénticos.
- El orden de las columnas debe ser el mismo

INTERSECCIÓN

R ∩S, define el conjunto de elementos que aparecen en simultáneamente tanto en la relación R como en la relación S.

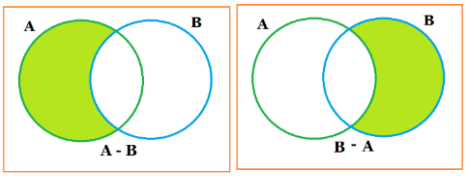


Requisitos para poder utilizarla:

- R y S deben tener esquemas idénticos.
- El orden de las columnas debe ser el mismo

DIFERENCIA

R – S o también denominada diferencia de R y S, define el conjunto de elementos que están en R pero no en S. Es importante resaltar que R - S es diferente a S - R.



Requisitos para poder utilizarla:

- R y S deben tener esquemas idénticos.
- El orden de las columnas debe ser el mismo

A continuación, estudiaremos cada uno de ellos por separado aportando ejemplos para comprender mejor este tipo de consultas.

4.1.1 UNION

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

Por ejemplo, tenemos por un lado los clientes que han realizado pagos en el año 2019:

```
SELECT DISTINCT codCliente
  FROM pagos
 WHERE YEAR(fechaPago) = 2019;
```

codCliente
6

Por otra parte, tenemos los clientes que han realizado pedidos en el año 2021:

```
SELECT DISTINCT codCliente
   FROM pedidos
  WHERE YEAR(fechaPedido) = 2021;
```

codCliente
1
2
3
5
6

La unión de los registros de ambas consultas da como resultado la UNION de ambas consultas.

Sintácticamente se realizaría del siguiente modo:

```
SELECT DISTINCT codCliente
  FROM pagos
 WHERE YEAR(fechaPago) = 2019
UNION
SELECT DISTINCT codCliente
  FROM pedidos
 WHERE YEAR(fechaPedido) = 2021;
```

codCliente
1
2
3
5
6

Recuerda que los registros repetidos NO aparecen en estas consultas.

4.1.2 INTERSECCIÓN

Cuando deseamos obtener únicamente aquellos registros que se encuentran en una tabla/consulta y que también se encuentran en otra tabla/consulta, utilizaremos la palabra reservada **INTERSECT**.

Continuando con el ejemplo anterior, si quisiéramos obtener los registros que están en una consulta y en la otra utilizaremos la siguiente sintaxis:

```
SELECT DISTINCT codCliente
  FROM pagos
 WHERE YEAR(fechaPago) = 2019
INTERSECT
SELECT DISTINCT codCliente
  FROM pedidos
 WHERE YEAR(fechaPedido) = 2021;
```

codCliente
6

4.1.3 DIFERENCIA

Si queremos obtener los registros que están en una tabla MENOS los que están en la otra utilizaremos la palabra reservada **EXCEPT**.

Continuando con el ejemplo anterior, si quisiéramos obtener los registros que están en una consulta y en la otra utilizaremos la siguiente sintaxis:

```
SELECT DISTINCT codCliente
  FROM pagos
 WHERE YEAR(fechaPago) = 2019
EXCEPT
SELECT DISTINCT codCliente
  FROM pedidos
 WHERE YEAR(fechaPedido) = 2021;
```

codCliente

No devuelve ningún registro porque el cliente 6 realizó un pago en 2019 y también realizó un pedido en 2021.

```
SELECT DISTINCT codCliente
  FROM pedidos
 WHERE YEAR(fechaPedido) = 2021
EXCEPT
SELECT DISTINCT codCliente
  FROM pagos
 WHERE YEAR(fechaPago) = 2019;
```

codCliente
1
2
3
5

Devuelve los clientes que realizaron pedidos en 2021 menos los que realizaron pagos en 2019 (por ello, el codCliente 6 NO aparece en el resultado).

4.3 Subconsultas

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

Podemos distinguir **tres tipos de subconsultas**:

1. **Las que aparecen junto con un OPERADOR DE COMPARACIÓN**
   ```
   WHERE expresión operador [ANY | ALL] (subconsulta)
   ```

2. **Aquellas que utilizan LISTAS incorporadas (IN)**
   ```
   WHERE expresión [NOT] IN (subconsulta)
   ```

3. **Las que son PRUEBAS DE EXISTENCIA mediante el operador EXISTS**
   ```
   WHERE [NOT] EXISTS (subconsulta)
   ```

A continuación, estudiaremos cada una de ellas por separado:

1. Subconsultas en WHERE con operador de comparación

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

```
SELECT AVG(importe) AS ImporteMedio
  FROM pagos;
```

ImporteMedio
47.116666

Tabla pagos:

codPago	codPedido	codCliente	importe	fechaPago
3	1	2	23.00	2017-01-05 23:58:00
4	4	6	8.50	2021-08-17 22:00:00
5	6	6	109.85	2019-01-17 23:05:00

Ejemplo 01 – Subconsulta en WHERE con operador de comparación

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

```
SELECT *
  FROM pagos
 WHERE importe > (SELECT AVG(importe) AS ImporteMedio FROM pagos);
```

codPago	codPedido	codCliente	importe	fechaPago
5	6	6	109.85	2019-01-17 23:05:00

Para realizar las consultas podemos especificar **ALL** o **ANY** del siguiente modo:

- > **ALL**: significa "mayor que cualquier valor de la subconsulta" o de dicho de otro modo "mayor que el máximo de todos los valores devueltos por la subconsulta"
- > **ANY**: significa "mayor que algún valor de la subconsulta"

Veamos ejemplos de uso de **ALL** / **ANY**:

Tabla clientes:

codCliente	nombreCliente	telefono	ciudad	region	pais	limiteCredito	fechaAlta
1	Antonio Gil	111111111	Alicante	Comunidad Valenciana	España	1000.00	2019-05-06
2	Filomena Suárez	222222222	Valencia	Comunidad Valenciana	España	1000.00	2020-04-04
3	Álvaro Moreno	333333333	Tarragona	Cataluña	España	2000.00	2007-08-09
4	David Díaz	444444444	Castellón	Comunidad Valenciana	España	500.00	2021-01-15
5	Roberto Vázquez	555555555	Madrid	Madrid	España	300.00	2013-03-02
6	Jose Luis Ramírez	666666666	Alicante	NULL	España	450.00	2019-08-04
7	Pepe Domínguez	777777777	Barcelona	Cataluña	España	750.00	2017-03-05
8	Alfonso Prieto	888888888	NULL	NULL	España	1000.00	2008-01-08

Tabla pagos:

codPago	codPedido	codCliente	importe	fechaPago
3	1	2	23.00	2017-01-05 23:58:00
4	4	6	8.50	2021-08-17 22:00:00
5	6	6	109.85	2019-01-17 23:05:00
6	2	3	510.99	2019-01-18 23:05:00

Ejemplo 02 – Subconsulta en WHERE con operador de comparación ANY

Muestra aquellos clientes que tengan un limiteCredito mayor que algún pago realizado en 2019.

```
SELECT *
  FROM clientes
 WHERE limiteCredito > ANY (SELECT importe FROM pagos
                            WHERE YEAR(pagos.fechaPago) = 2019);
```

codCliente	nombreCliente	telefono	ciudad	region	pais	limiteCredito	fechaAlta
1	Antonio Gil	111111111	Alicante	Comunidad Valenciana	España	1000.00	2019-05-06
2	Filomena Suárez	222222222	Valencia	Comunidad Valenciana	España	1000.00	2020-04-04
3	Álvaro Moreno	333333333	Tarragona	Cataluña	España	2000.00	2007-08-09
4	David Díaz	444444444	Castellón	Comunidad Valenciana	España	500.00	2021-01-15
5	Roberto Vázquez	555555555	Madrid	Madrid	España	300.00	2013-03-02
6	Jose Luis Ramírez	666666666	Alicante	NULL	España	450.00	2019-08-04
7	Pepe Domínguez	777777777	Barcelona	Cataluña	España	750.00	2017-03-05
8	Alfonso Prieto	888888888	NULL	NULL	España	1000.00	2008-01-08

Muestra todos los clientes porque tienen el limiteCredito mayor que cualquier pago realizado (se compara con el menor de ellos que fue de 8,50 €)

Ejemplo 03 – Subconsulta en WHERE con operador de comparación ALL

Muestra aquellos clientes que tengan un limiteCredito mayor que todos los pagos realizados en 2019.

```
SELECT *
  FROM clientes
 WHERE limiteCredito > ALL (SELECT importe FROM pagos
                            WHERE YEAR(pagos.fechaPago) = 2019);
```

codCliente	nombreCliente	telefono	ciudad	region	pais	limiteCredito	fechaAlta
1	Antonio Gil	111111111	Alicante	Comunidad Valenciana	España	1000.00	2019-05-06
2	Filomena Suárez	222222222	Valencia	Comunidad Valenciana	España	1000.00	2020-04-04
3	Álvaro Moreno	333333333	Tarragona	Cataluña	España	2000.00	2007-08-09
7	Pepe Domínguez	777777777	Barcelona	Cataluña	España	750.00	2017-03-05
8	Alfonso Prieto	888888888	NULL	NULL	España	1000.00	2008-01-08

Aparecen los clientes cuyo limiteCredito es más mayor que el pago de más importe (se compara con el mayor de todos ellos que fue de 510,99 €)

2. 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 04 – Subconsulta en WHERE con operador de IN

Mostrar la gama de los productos que de los que se haya pedido más de 2 unidades

```
SELECT DISTINCT gama
  FROM productos
 WHERE codProducto IN (SELECT codProducto
                         FROM lineapedidos
                        WHERE cantidad > 2 );
```

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

3. Subconsultas en WHERE con operador EXISTS

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

Ejemplo 05 – Subconsulta en WHERE con operador de EXISTS

Utiliza una subconsulta correlacionada para obtener los datos de los clientes que hayan realizado algún pedido.

```
SELECT DISTINCT codcliente
  FROM clientes
 WHERE EXISTS (SELECT *
                 FROM pedidos
                WHERE codCliente = clientes.codCliente);
```

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

Subconsultas en la cláusula FROM

No es un tipo de subconsulta, sino un recurso al que podemos recurrir en caso de necesitarlo. Podemos utilizar subconsultas como tablas y colocarlas en la cláusula FROM.

Debemos recordar que estas subconsultas en el FROM deben contener ALIAS para que funcionen correctamente.

Ejemplo 06 – Subconsulta en FROM

Aunque la siguiente consulta se puede obtener mediante una consulta JOIN 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.codOficina = OficinasMadrid.codOficina;
```

La consulta anterior sin usar subconsulta sería:

```
SELECT empl.*
  FROM empleados empl,
       Oficinas ofic
 WHERE empl.codOficina = ofic.codOficina
   AND ofic.Ciudad = 'Madrid';
```

Ambas consultas devuelven lo mismo, "Empleados que trabajen en la oficina de Madrid":

codEmpleado	nombre	apellido1	apellido2	telefono	email	codEmplJefe	codOficina
1	Pepe	Martínez	Rodríguez	111111111	pepe_informatica@gmail.com	6	1
4	Isabel	Pérez	Blanco	444444444	isabel_informatica@gmail.com	6	1
6	Víctor	Muñoz	Romero	666666666	jefe_empresa@gmail.com	NULL	1

5. Vistas

Una vista es una tabla virtual cuyo contenido está definido por una consulta. Al igual que una tabla, una vista consta de un conjunto de columnas y filas de datos con un nombre asociado a ellos.

Sin embargo, una vista no existe como conjunto de valores de datos almacenados en una base de datos, es decir, NO es en ningún caso una tabla. Las filas y las columnas de datos proceden de tablas a las que *se hace referencia* en la consulta que define la vista y se producen de forma dinámica cuando se hace referencia a la vista.

Una vista actúa como filtro de las tablas subyacentes a las que se hace referencia en ella. La consulta que define la vista puede provenir de una o de varias tablas, o bien de otras vistas de la base de datos actual u otras bases de datos.

Las vistas suelen usarse para centrar, simplificar y personalizar la percepción de la base de datos para cada usuario.

A continuación, estudiaremos cómo crear vistas, modificarlas y borrarlas.

Creación de vistas

Para crear una vista, utilizaremos la siguiente sintaxis;

```
CREATE VIEW VNombreVista AS

SELECT …
```

Es una buena práctica poner una V como la primera letra del nombre de la vista.

A continuación, se muestra un ejemplo para crear una vista de una de las consultas de los ejemplos anteriores:

```
CREATE VIEW VJefeTrabajadores AS
SELECT jefes.Nombre nombreJefe, jefes.Apellido1 apellidoJefe,
       trabajadores.*
  FROM EMPLEADOS AS trabajadores,
       EMPLEADOS AS jefes
 WHERE trabajadores.codEmplJefe =  jefes.codEmpleado;
```

Para poder ver el contenido de la vista, haremos la SELECT igual que si de una tabla se tratara:

```
SELECT *
  FROM VJefeTrabajadores;
```

nombreJefe	apellidoJefe	codEmpleado	nombre	apellido1	apellido2	telefono	email	codEmplJefe	codOficina
Víctor	Muñoz	1	Pepe	Martínez	Rodríguez	111111111	pepe_informatica@gmail.com	6	1
Víctor	Muñoz	2	Ana	García	Navarro	222222222	ana_marketing@gmail.com	6	3
Víctor	Muñoz	3	Carlos	González	Molina	333333333	carlos_rrhh@gmail.com	6	2
Víctor	Muñoz	4	Isabel	Pérez	Blanco	444444444	isabel_informatica@gmail.com	6	1
Víctor	Muñoz	5	Marta	Fernández	Ramos	555555555	marta_informatica@hotmail.com	6	2

Modificación de vistas

La modificación de vistas se realiza de acuerdo con la siguiente sintaxis:

```
ALTER VIEW VNombreVista AS

SELECT …
```

De este modo se actualizará la consulta que utiliza la vista.

Borrado de vistas

Por último, para eliminar vistas utilizaremos la siguiente sintaxis:

```
DROP VIEW VNombreVista;
```

De este modo se borrará la vista de la base de datos.