

**MP0484.**

**Bases de datos**

**UF4. Formulación de consultas avanzadas**

## **4.1. Consultas avanzadas**

# Índice

---

≡ Introducción y objetivos	3
≡ Base de datos de ejemplo	4
≡ Introducción	6
≡ Consultas multitabla	7
≡ Tipos de JOIN	9
≡ [ INNER ] JOIN	11
≡ LEFT [ OUTER ] JOIN	14
≡ RIGHT [ OUTER ] JOIN	16
≡ FULL [ OUTER ] JOIN	18
≡ Union	20
≡ Intersect	24
≡ Minus	26
≡ Funciones de agrupación	28
≡ Consultas agrupadas	32
≡ Ejecución interna	35

# Introducción y objetivos

---

Las consultas avanzadas nos **permiten profundizar en la búsqueda de información**, pudiendo usar en una **sola sentencia las condiciones y resultados intermedios** para poder llegar a la **información requerida**.

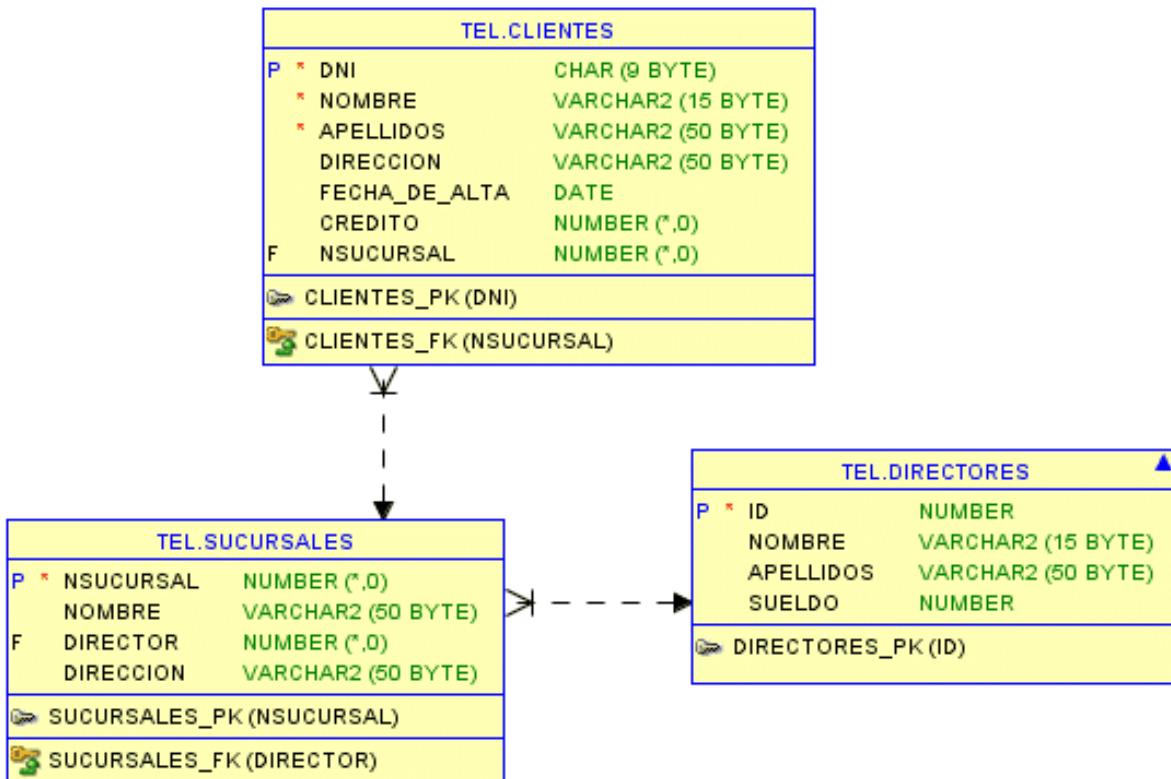
## Objetivos:

- Conocer cómo se pueden obtener datos de diferentes tablas y combinar el resultado
- Unir diferentes consultas en una sola
- Realizar consultas agrupadas

# Base de datos de ejemplo

En esta unidad vamos a seguir trabajando con el modelo de datos de **Entidades bancarias y Clientes.**

## Modelo



## Datos de ejemplo

A continuación se muestran algunos datos de ejemplo de estas tablas.

### Sucursales

NSUCRASAL	NOMBRE	DIRECTOR	DIRECCION
1001	Sucursal Centro		12 Avd. del Estilo, 45
1002	Sucursal Oeste		15 Avd. Mediterráneo, 14
1003	Sucursal Este		20 Calle Martinez, 45
1004	Sucursal Norte		(null) Calle Especias, 23

### Clientes

DNI	NOMBRE	APELLIDOS	DIRECCION	FECHA_DE_ALTA	CREDITO	NSUCRASAL
30515454K	Ana	Martin Martin	Calle del Socorro, 1	08/01/18	1500	1001
33358796A	Marta	López Ruiz	Calle Martinez, 76	14/09/17	600	1001
78458784B	Antonio	Castillo Mentas	Calle Soles, 14	(null)	500	1002

### Directores

ID	NOMBRE	APELLIDOS	SUELDO
12	Alberto	Pérez Martín	1800
15	Antonio	López López	1500
20	Silvia	Martín Martín	1300
13	Pedro	García Martín	1900

# Introducción

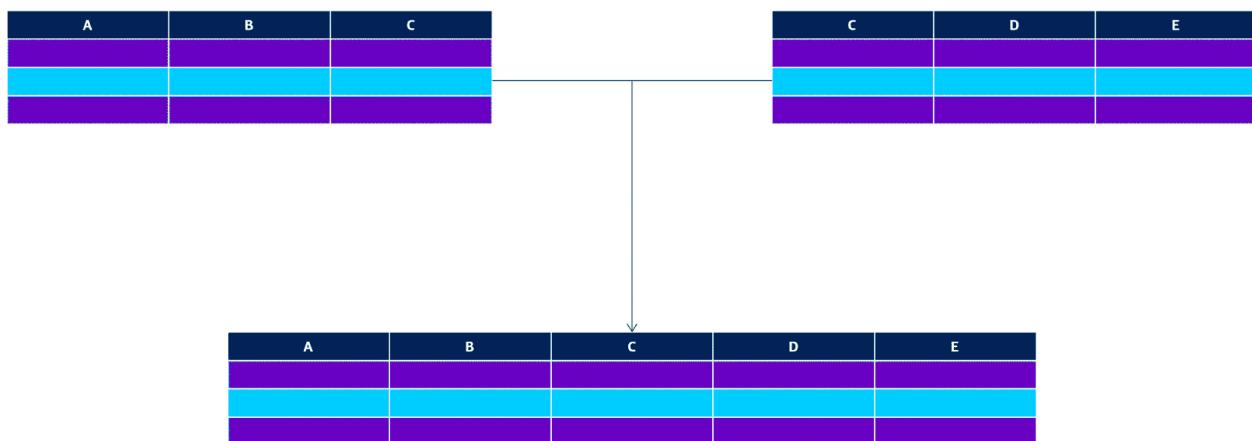
En las consultas que hemos estudiado hasta este momento hemos trabajado con los resultados de **una sola tabla**, es decir los datos que se entregaban **sólo pertenecían a una tabla**.

SQL nos ofrece la posibilidad de **juntar tablas en el resultado final de una consulta** de forma que la información asociada a través de claves foráneas pueda mostrarse en un sólo registro.

La **unión de relaciones** que se almacenan en lugares distintos forma parte de las operaciones del **álgebra relacional**.



La instrucción que se utiliza en SQL es **JOIN** y sus **variantes** nos permiten trabajar con varias tablas. **JOIN** en realidad es el **operador cartesiano de Algebra relacional y combina** los registros de dos tablas tal que su resultado es la **combinación de registros de la primera tabla, con los de la segunda**.



# Consultas multitable

En las primeras consultas con las que hemos trabajado sólo hemos utilizado una tabla, pero hay veces que una consulta necesita columnas de varias tablas. En este caso, las tablas se expresarán a la derecha de la palabra reservada FROM separadas por comas y se relacionarán campos de ambas tablas. Normalmente la relación será entre la clave primaria de la tabla1 y la clave foránea de la tabla2.

La sintaxis general sería

```
SELECT columna1,columna2, ... de las tablas indicadas en el FROM  
FROM tabla1, tabla2...  
WHERE tabla1.columna = tabla2.columna .... ;
```

Ejemplo:

Listar los nombres de sucursales junto con los nombres y apellidos de sus directores.

```
SELECT sucursales.nombre, directores.nombre,  
directores.apellidos FROM sucursales, directores  
WHERE sucursales.director=directores.id;
```

Como se puede observar se están solicitando campos de las dos tablas. En este caso el nombre de la tabla sucursales y el nombre y apellidos de la tabla directores. También comparo el identificador de director en ambas tablas para que sólo muestre los registros relacionados.

condiciones y de esta manera filtrar más el resultado. Por ejemplo si quisieramos excluir al director que se llame 'Antonio' la sentencia sería:

```
SELECT sucursales.nombre, directores.nombre,  
directores.apellidos FROM sucursales,directores  
where sucursales.director=directores.id and directores.nombre<>'Antonio';
```

Cuando combinamos varias tablas, hemos de tener en cuenta una serie de reglas:

- 1 Es posible unir tantas tablas como deseemos.
- 2 En la cláusula SELECT se pueden citar columnas de todas las tablas.
- 3 Si hay columnas con el mismo nombre en las distintas tablas de la cláusula FROM, se deben identificar, especificando NombreTabla.NombreColumna.
- 4 Si el nombre de una columna existe sólo en una tabla, no será necesario especificarla como NombreTabla.NombreColumna. Sin embargo, hacerlo mejoraría la legibilidad de la sentencia SELECT.
- 5 El criterio que se siga para combinar las tablas ha de especificarse en la cláusula WHERE. Si se omite esta cláusula, que especifica la condición de combinación, el resultado será un PRODUCTO CARTESIANO, que emparejará todas las filas de una tabla con cada fila de la otra.



Esta manera de relacionar tablas utiliza la sintaxis SQL 1992. Veremos en próximos apartados, que es conveniente utilizar la sintaxis SQL 1999, porque separa las condiciones de asociación respecto de las condiciones de selección de registros, lo cual otorga una mayor claridad a las instrucciones SQL.

## Tipos de JOIN

---

Según la comparación entre los campos de las tablas relacionadas existen **dos tipos principales de JOIN**

### NON-EQUI JOIN

Por desigualdad, cuando no existe la correspondencia directa entre campos de tablas. La relación se establecerá usando criterios de rango ( $=<$ ,  $=>$ ,  $<$ ,  $>$ , BETWEEN, ...)

### EQUI JOIN

JOIN sobre dos o más tablas, por igualdad de campos.

---

Dentro de los EQUI JOIN hay que distinguir los siguientes tipos:

**COMBINACIONES INTERNAS**

- INNER JOIN
- NATURAL JOIN

**COMBINACIONES EXTERNAS**

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

Tanto la palabra INNER como la palabra OUTER son opcionales, por lo que es válido utilizar esta nomenclatura:

- JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

# [ INNER ] JOIN

Es el tipo más común de unión. INNER JOIN devuelve todas las filas de varias tablas donde se cumple la condición de unión.

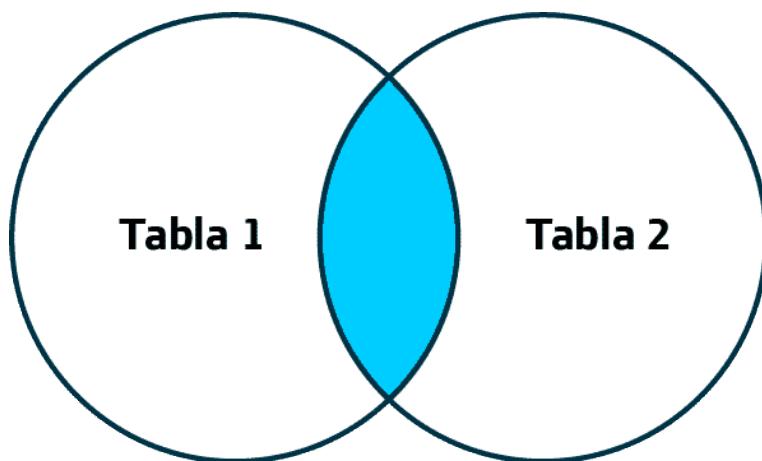
## Sintaxis

La sintaxis para INNER JOIN es:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

## Diagrama visual

En este diagrama visual, INNER JOIN devuelve el área sombreada:



**Ejemplo:**

Se necesitan saber los nombres de los directores y a que sucursal están asignados.

```
SELECT directores.nombre, directores.apellidos,  
sucursales.nombre as NOMBRE_SUCURSAL FROM directores  
INNER JOIN sucursales  
ON directores.id=sucursales.director;
```

**Resultado**

En la siguiente imagen se muestra el resultado obtenido. Como se observa no aparece "Pedro García Martín" porque no está en ambas tablas.

NOMBRE	APELLIDOS	NOMBRE_SUCURSAL
Alberto	Pérez Martín	Sucursal Centro
Antonio	López López	Sucursal Oeste
Silvia	Martín Martín	Sucursal Este

Utilizando la sintaxis SQL 1992 el código sería como se muestra :

```
SELECT directores.nombre, directores.apellidos,  
sucursales.nombre as NOMBRE_SUCURSAL FROM directores,sucursales  
WHERE directores.id=sucursales.director;
```

Sin embargo y tal y como se indicó anteriormente es mejor utilizar JOIN y separar las condiciones de asociación respecto de las condiciones de selección de registros porque se gana en claridad y eficiencia.

En el caso de que necesitemos relacionar más de dos tablas en la misma sentencia SQL, porque necesitemos campos de esas tablas, tendremos que indicar todas las relaciones existentes y los campos por los que están relacionados.

**Ejemplo:**

Indicar el nombre y apellidos del director de la sucursal del cliente con DNI='78458784B'. También se pide mostrar el nombre de la sucursal y el nombre y apellidos del cliente.

```
/* Indicar el nombre y apellidos del director de la sucursal  
del cliente con DNI='78458784B'.
```

```
También se pide mostrar el nombre de la sucursal y  
el nombre y apellidos del cliente */
```

```
SELECT directores.nombre,directores.apellidos,sucursales.nombre,  
clientes.nombre,clientes.apellidos FROM directores  
JOIN sucursales  
ON directores.id=sucursales.director  
JOIN clientes  
ON sucursales.nsucursal=clientes.nsucursal  
WHERE clientes.dni='78458784B';
```

Como se observa en el ejemplo se necesita la relación entre director-sucursal y luego entre sucursal-cliente para conocer qué director pertenece a la sucursal del cliente indicado.



Cuando trabajamos con varias tablas y para evitar ambigüedades entre los nombres de los campos, es conveniente indicar la tabla a la que pertenece el campo.

# LEFT [ OUTER ] JOIN

Otro tipo de unión se llama LEFT OUTER JOIN. Este tipo de unión devuelve todas las filas de la tabla IZQUIERDA especificada en la condición ON y solo aquellas filas de la otra tabla donde los campos combinados son iguales (se cumple la condición de unión).

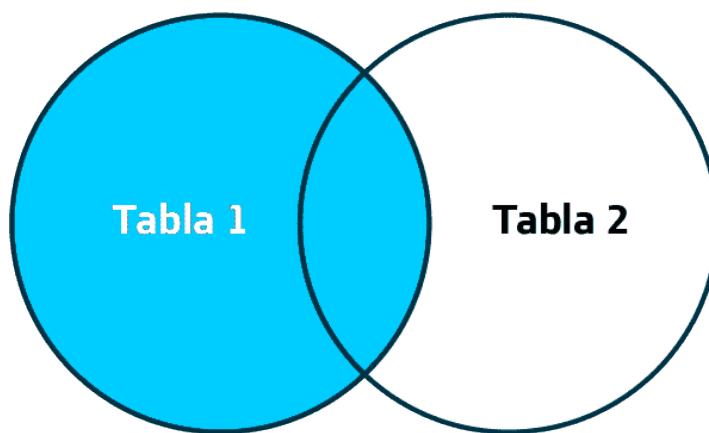
## Sintaxis

La sintaxis para LEFT OUTER JOIN es:

```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

## Diagrama visual

En este diagrama visual, LEFT OUTER JOIN devuelve el área sombreada:



LEFT OUTER JOIN devolverá todos los registros de la tabla 1 y solo aquellos registros de la tabla 2 que se cruzan con la tabla1 .

### Ejemplo:

Se necesitan saber los directores y sus sucursales. Mostrar todos los directores incluso si no tienen asignadas sucursales.

```
SELECT directores.nombre, directores.apellidos,  
sucursales.nombre as NOMBRE_SUCURSAL  
FROM directores  
LEFT JOIN sucursales  
ON directores.id=sucursales.director;
```

### Resultado

Como se puede observar se listan todos los directores, a pesar de que el director "Pedro García Martín" aún no tiene asignada Sucursal.

NOMBRE	APELLIDOS	NOMBRE_SUCURSAL
Alberto	Pérez Martín	Sucursal Centro
Antonio	López López	Sucursal Oeste
Silvia	Martín Martín	Sucursal Este
Pedro	García Martín	

# RIGHT [ OUTER ] JOIN

Otro tipo de unión es RIGHT OUTER JOIN. Este tipo de unión devuelve todas las filas de la tabla DERECHA especificada en la condición ON y solo aquellas filas de la otra tabla donde los campos combinados son iguales (se cumple la condición de unión).

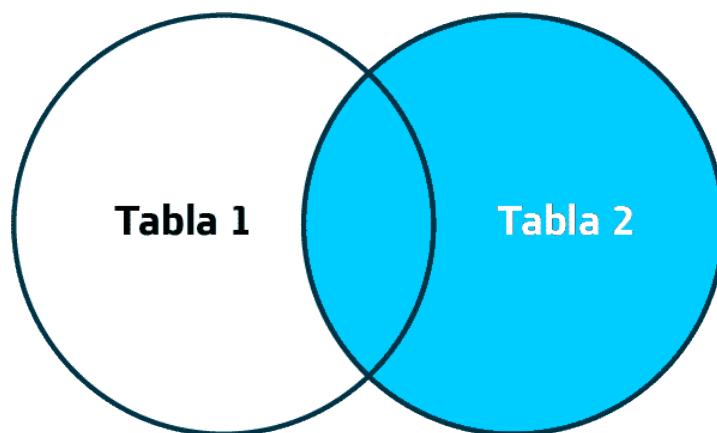
## Sintaxis

La sintaxis para RIGHT OUTER JOIN es:

```
SELECT columns  
FROM table1  
RIGHT [OUTER] JOIN table2  
ON table1.column = table2.column;
```

## Diagrama visual

En este diagrama visual, RIGHT OUTER JOIN devuelve el área sombreada:



RIGHT OUTER JOIN devolverá todos los registros de la tabla 2 y sólo aquellos registros de la tabla 1 que se cruzan con la tabla 2.

### Ejemplo:

Se necesitan saber los directores y sus sucursales asociadas.

```
SELECT directores.nombre, directores.apellidos,  
sucursales.nombre as NOMBRE_SUCURSAL  
FROM directores  
RIGHT JOIN sucursales  
ON directores.id=sucursales.director;
```

### Resultado

Como se puede observar se listan todas las sucursales, a pesar que una de ellas no tiene asociado director. Esto es porque el right join lo hemos hecho sobre la tabla sucursales y se incluyen todos sus registros.

NOMBRE	APELLIDOS	NOMBRE_SUCURSAL
Alberto	Pérez Martín	Sucursal Centro
Antonio	López López	Sucursal Oeste
Silvia	Martín Martín	Sucursal Este Sucursal Norte

# FULL [ OUTER ] JOIN

Otro tipo de unión es FULL OUTER JOIN. Este tipo de unión devuelve todas las filas de la tabla IZQUIERDA y la tabla DERECHA con nulos en el lugar donde no se cumple la condición de unión.

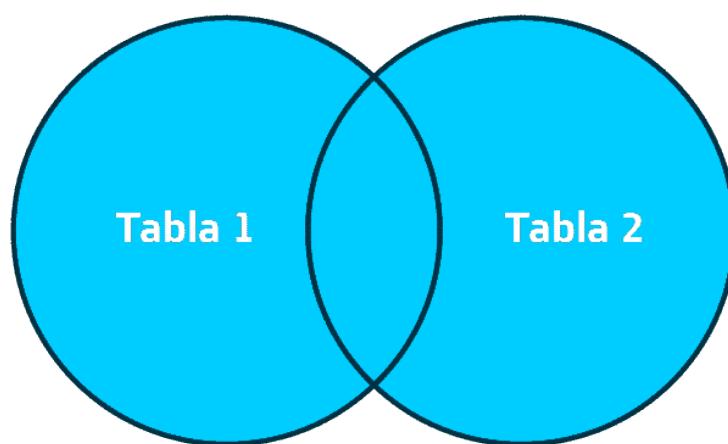
## Sintaxis

La sintaxis para FULL OUTER JOIN es:

```
SELECT columns
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```

## Diagrama visual

En este diagrama visual, FULL OUTER JOIN devuelve el área sombreada:



FULL OUTER JOIN devolverá todos los registros de tabla 1 y tabla 2 .

## Ejemplo

Se necesitan saber los directores y sus sucursales asociadas.

```
SELECT sucursales.nombre as NOMBRE_SUCURSAL,  
directores.nombre, directores.apellidos  
FROM sucursales  
FULL OUTER JOIN directores  
ON sucursales.director=directores.id;
```

## Resultado

En este caso se muestran todos los directores y sucursales, aunque haya directores sin sucursales y sucursales sin directores.

NOMBRE_SUCURSAL	NOMBRE	APELLIDOS
Sucursal Centro	Alberto	Pérez Martín
Sucursal Oeste	Antonio	López López
Sucursal Este	Silvia	Martín Martín
Sucursal Norte	Pedro	García Martín

# Union

Ocasionalmente, es conveniente combinar los resultados de dos o más consultas en una única tabla de resultados totales. SQL permite esta capacidad gracias a la característica **UNION** de la sentencia SELECT.

## Sintaxis

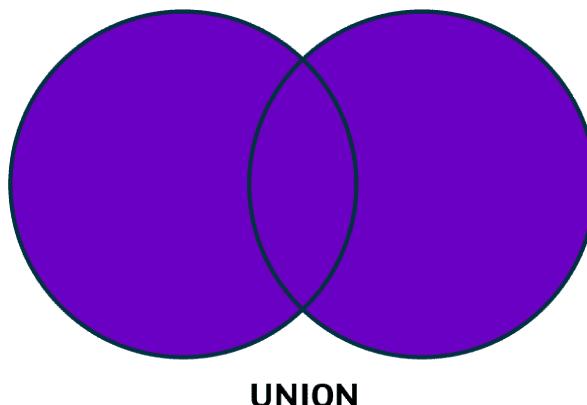
Se utiliza la palabra reservada UNION, y la sintaxis es:

```
SELECT columna1, columna2 FROM tabla1
UNION
SELECT columna1, columna2 FROM tabla2
```

El resultado es una tabla que contendrá la columna1 y columna2 de ambas tablas. Es decir, UNION crea una sola tabla con registros que estén presentes en cualquiera de las consultas. Si están repetidas sólo aparecen una vez. Para mostrar los duplicados se utiliza UNION ALL en lugar de la palabra UNION

## Diagrama visual

Gráficamente sería:



## Ejemplo:

Podríamos unir los nombres y apellidos tanto de clientes como de directores.

```
SELECT nombre, apellidos FROM directores  
UNION  
SELECT nombre, apellidos FROM clientes;
```

## Restricciones

Hay varias restricciones sobre las tablas que pueden combinarse con una operación UNION:

- 1 Ambas tablas deben contener el mismo número de columnas.
- 2 El tipo de datos de cada columna en la primera tabla debe ser el mismo que el tipo de datos de la columna correspondiente en la segunda tabla.
- 3 Ninguna de las dos tablas puede estar ordenadas con la cláusula ORDER BY. Sin embargo, los resultados combinados pueden ser ordenados, según se describe en la sección siguiente.

Los nombres de columna de las dos consultas combinadas mediante una UNION no tienen que ser idénticos. Puesto que las columnas de las dos tablas pueden tener nombres diferentes, lo habitual es mostrar el nombre del campo de la primera consulta, pero depende del gestor de base de datos. Por omisión, la operación UNION **elimina las filas duplicadas** como parte de su procesamiento.

La eliminación de filas duplicadas en los resultados de la consulta es un proceso que consume mucho tiempo, especialmente si los resultados contienen un gran número de filas. Si se sabe, en base a las consultas individuales implicadas, que la operación UNION no puede producir filas duplicadas, se debería utilizar específicamente la operación UNION ALL, ya que la consulta se ejecutará mucho más rápidamente.

## Uniones y ordenación

La cláusula ORDER BY no puede aparecer en ninguna de las dos sentencias SELECT combinadas por una operación UNION. No tendría mucho sentido ordenar los dos conjuntos de resultados de ninguna manera, ya que éstos se dirigen directamente a la operación UNION y nunca son visibles al usuario.

Sin embargo, el conjunto combinado de los resultados de la consulta producidos por la operación UNION puede ser ordenado especificando una cláusula ORDER BY después de la segunda sentencia SELECT.

## Uniones múltiples

La operación UNION puede ser utilizada repetidamente para combinar tres o más conjuntos de resultados. De esta manera si tenemos una unión entre una tabla B y una tabla C, la podemos unir y el resultado de esa unión unirla a una tercera tabla A. Tendríamos lo siguiente:

- A UNION (B UNION C)

### Ejemplo:

En este caso le unimos a dos tablas de directores, la tabla de clientes con sus nombres y apellidos.

```
SELECT nombre,apellidos FROM clientes
UNION
SELECT nombre,apellidos FROM directores
UNION
SELECT nombre,apellidos FROM directores1;
```

Los paréntesis que aparecen en la consulta indican que UNION debería ser realizada en primer lugar. De hecho, si todas las uniones de la sentencia eliminan filas duplicadas, o si todas ellas mantienen las filas duplicadas, el orden en que se efectúan no tienen importancia. Estas tres expresiones son equivalentes:

- A UNION (B UNION C)
- (A UNION B) UNION C
- (A UNION C) UNION B

# Intersect

**INTERSECT** devuelve los valores comunes a ambas, con lo que obtenemos una intersección (sólo los registros que están entre los resultados de ambas consultas).

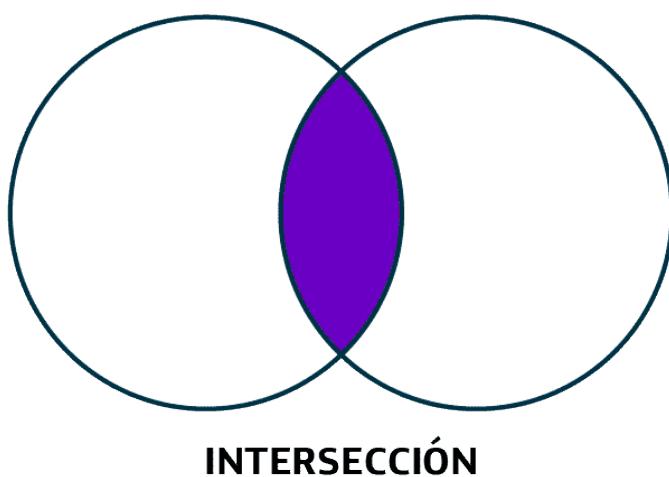
## Sintaxis

La sintaxis sería:

```
SELECT columna1, columna2,... FROM tabla1
INTERSECT
SELECT columna1, columna2,... FROM tabla2
```

## Diagrama visual

La representación gráfica sería la siguiente:



Ejemplo:

Obtener los nombres que son comunes entre la tabla directores y clientes.

```
SELECT nombre FROM directores
INTERSECT
SELECT nombre FROM clientes;
```

---

**Al igual que ocurre con UNION la intersección también puede combinarse con la ordenación ( ORDER BY).**

## Minus

---

**MINUS** devuelve los valores de la primera consulta que no se encuentran en la segunda. Así podemos averiguar qué registros están en una consulta pero no en la otra, calculando la diferencia entre dos conjuntos de registros. Algo realmente útil en ocasiones y difícil de conseguir con instrucciones más simples.

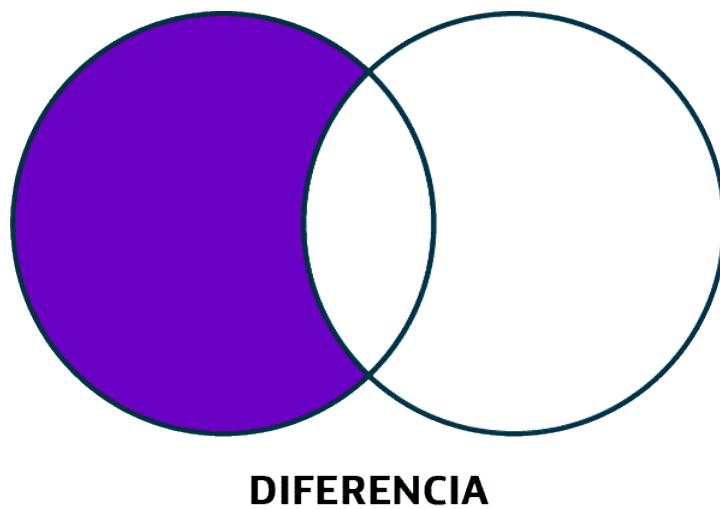
### Sintaxis

La sintaxis sería:

```
SELECT columna1, columna2,... FROM tabla1
MINUS
SELECT columna1, columna2,... FROM tabla2
```

### Diagrama visual

La representación gráfica sería la siguiente:



**Ejemplo:**

Listar los nombres de directores que no se llamen igual que los clientes de las sucursales, sin comprobar los clientes de la sucursal 1001.

```
SELECT nombre FROM directores
MINUS
SELECT nombre FROM clientes where nsucursal<>1001;
```

# Funciones de agrupación

Hasta ahora hemos visto funciones que operan con valores simples; no obstante, hay otras funciones como SUM, AVG y COUNT, que actúan sobre un grupo de filas para obtener un valor.

Estas funciones permiten obtener la edad media de un grupo de alumnos, el alumno más joven, el más viejo, el número total de miembros de un grupo, etc. Los valores nulos son ignorados por las funciones de grupos de valores y los cálculos se realizan sin contar con ellos.

Estas funciones son:

- **SUM:** suma los valores del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
- **AVG:** devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
- **MIN:** devuelve el valor mínimo del campo que especifiquemos.
- **MAX:** devuelve el valor máximo del campo que especifiquemos.
- **COUNT:** devuelve el número total de filas seleccionadas por la consulta.

1

SUM

La función columna SUM() calcula la **suma** de una columna de valores de datos. Los datos de la columna deben tener un tipo numérico. El resultado de la función SUM() tiene el mismo tipo de datos básico que los datos de la columna, pero el resultado puede tener una precisión superior.

**Ejemplo:**

Suma de los sueldos de los directores

```
SELECT SUM(sueldo) FROM directores;
```

2

AVG

La función de columna AVG() calcula el **promedio** de una columna de valores de datos. Al igual que una función SUM(), los datos de la columna deben tener un tipo numérico. Ya que la función AVG() suma los valores de la columna y luego lo divide por el número de valores, su resultado puede tener un tipo de dato diferente al de los valores de columna.

Si se aplica la función AVG() a una columna de enteros, el resultado será un número decimal o un número de coma flotante, dependiendo del gestor concreto que se esté utilizando.

**Ejemplo:**

Media de los sueldos de los directores

```
SELECT AVG(sueldo) FROM directores;
```

3

MIN Y MAX

Las funciones de columna MIN() Y MAX() determinan los valores **menor y mayor** de una columna, respectivamente. Los datos de la columna pueden contener información numérica, de cadena o de fecha/hora. El resultado de la función MIN() y MAX() tiene exactamente el mismo tipo de dato que los datos de la columna.

**Ejemplo:**

Fecha de alta más antigua de los clientes

```
SELECT MIN(fecha_de_alta) FROM clientes;
```

Cuando las funciones de columnas MIN() y MAX() se aplican a datos numéricos, SQL compara los números en orden algebraico (los números negativos grandes son menores que los números negativos pequeños, los cuáles son menores que cero, el cual a su vez es menor que todos los números positivos). Las fechas se comparan secuencialmente (las fechas más antiguas son más pequeñas que las fechas más recientes).

Cuando se utiliza MIN() YMAX() con datos de cadenas, la comparación de las cadenas depende del conjunto de caracteres que se esté siendo utilizado.

4

**COUNT**

La función de columna COUNT() cuenta el número de valores de datos que hay en una columna. Los datos de la columna pueden ser de cualquier tipo. La función COUNT() **siempre devuelve un entero**, independientemente del tipo de datos de la columna.

**Ejemplo:**

Obtener el Número de clientes

```
SELECT count(dni) FROM clientes;
```

La función COUNT, se suele usar con asterisco (\*) como objeto de cálculo, lo que hace que cuente filas independientemente de su contenido. Si usamos una expresión, como COUNT(sueldo), no contaría las filas cuyo sueldo fuera nulo.

### Ejemplo:

Si contamos según el campo fecha\_de\_alta de clientes obtendremos el valor 2 y si lo contamos con \* obtendremos el valor 3, porque hay un cliente que no tiene fecha de alta.

```
/* Obtenemos 2 porque hay un cliente que no tiene fecha_de_alta */  
SELECT count(fecha_de_alta) FROM clientes;  
  
/* Obtenemos 3 */  
SELECT count(*) FROM clientes;
```

# Consultas agrupadas

La cláusula **GROUP BY** unida a un **SELECT** permite agrupar filas según las columnas que se indiquen como parámetros, y se suele utilizar en conjunto con las **funciones de agrupación**, para obtener datos resumidos y agrupados por las columnas que se necesiten.

Las columnas indicadas en la cláusula **GROUP BY** se denominan columnas de agrupación de la consulta ya que son las que determinan cómo se dividen las filas en grupos.

Así por ejemplo si queremos saber la suma de créditos que tienen aprobadas las sucursales a sus clientes, lo podríamos hacer así:

```
SELECT nsucursal, SUM(credito) FROM clientes GROUP BY nsucursal;
```

## Restricciones para GROUP BY

Las columnas de agrupación deben ser columnas efectivas de las tablas designadas en la cláusula **FROM** de la consulta. No se pueden agrupar las filas basándose en el valor de una expresión calculada.

También hay restricciones sobre los elementos que pueden aparecer en la lista de selección de una consulta agrupada. Todos los elementos de la lista de selección deben tener un único valor por cada grupo de filas. Básicamente, esto significa que un elemento de selección en una consulta agrupada puede ser:

- Una constante.
- Una función de columna.
- Una columna de agrupación, que por definición tiene el mismo valor en todas las filas del grupo.
- Una expresión que afecte a combinaciones de los anteriores.

Lo más frecuente es que en una consulta agrupada se incluya siempre una columna de agrupación y una función de agrupación en su lista de selección.

## Condiciones de búsqueda de grupos (HAVING)

Al igual que la cláusula WHERE puede ser utilizada para seleccionar y rechazar filas individuales que participan en una consulta, la cláusula HAVING puede ser utilizada para seleccionar y rechazar grupos de filas.

El formato de la cláusula HAVING es análogo al de la cláusula WHERE y consiste en la palabra clave HAVING seguida de una condición de búsqueda.

Por ejemplo si queremos sacar sólo las sucursales cuyo total de crédito concedido sea mayor de 1000 Euros, lo podríamos hacer así:

```
SELECT nsucursal, SUM(credito) FROM clientes  
GROUP BY nsucursal HAVING sum(credito) > 1000;
```

## Restricciones para HAVING

La cláusula HAVING se utiliza para incluir o excluir grupos de filas de los resultados de la consulta, por lo que la condición de búsqueda que especifica debe ser aplicable al grupo en su totalidad en lugar de a filas individuales. Esto significa que un elemento que aparezca dentro de la condición de búsqueda en una cláusula HAVING puede ser:

- Una constante
- Una función de agrupación, que produzca un único valor resumen de las filas del grupo.
- Una columna de agrupación, que por definición tiene el mismo valor en todas las filas del grupo.
- Una expresión que afecte a combinaciones de los anteriores

Lo más frecuente es que la condición de búsqueda de la cláusula HAVING incluya siempre al menos una función de agrupación. Si no lo hiciera, la condición de búsqueda podría expresarse con la cláusula WHERE y aplicarse a filas individuales. El modo más fácil de averiguar si una condición de búsqueda pertenece a la cláusula WHERE o a la cláusula HAVING es recordar cómo se aplican ambas cláusulas:

- La cláusula WHERE se aplica a filas individuales, por lo que las expresiones que contiene deben ser calculables para filas individuales.
- La cláusula HAVING se aplica a grupos de filas, por lo que las expresiones que contengan deben ser calculables para un grupo de filas.

# Ejecución interna

---

## ¿Cómo se ejecuta internamente una consulta agrupada?

El gestor de base de datos a la hora de ejecutar una consulta SQL sigue unos pasos para optimizar el rendimiento.

Esto depende del gestor de Base de datos, pero los pasos más habituales son los siguientes:

- 1 Primero se forma la tabla origen de datos según la cláusula FROM.
- 2 Se seleccionan del origen de datos las filas según la cláusula WHERE.
- 3 Se forman los grupos de filas según la cláusula GROUP BY.
- 4 Por cada grupo se obtiene una fila en la tabla resultante con los valores que aparecen en las cláusulas GROUP BY, HAVING y en la lista de selección.
- 5 Se seleccionan de la tabla resultante las filas según la cláusula HAVING.
- 6 Se eliminan de la tabla resultante las columnas que no aparecen en la lista de selección.
- 7 Se ordenan las filas de la tabla resultante según la cláusula ORDER BY.

---

**Este orden es importante conocerlo de cara a depurar consultas e intentar optimizarlas.**



**PROEDUCA**