
	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Tema 8

Cláusulas avanzadas de selección

Índice

1.	Introducción	1
2.	Agrupación de elementos. GROUP BY y HAVING	1
3.	Subconsultas.....	2
3.1.	Condiciones de búsquedas en subconsultas	3
3.2.	Subconsultas correlacionadas	4
4.	Producto cartesiano.....	5
5.	Combinación de tablas: JOIN.....	5
5.1.	Composición interna natural: NATURAL JOIN	7
6.	Combinación externa: OUTER JOIN	7
7.	Composición de consultas.....	8
7.1.	Operador unión: UNION.....	8
7.2.	Operador Intersección: INTERSECT.....	10
7.3.	Operador diferencia: EXCEPT	11
7.4.	Diferencias entre Union, Intersect y Except.....	11
8.	Resumen:.....	12
9.	Apéndice 1:.....	13
9.1.	Opciones avanzadas de Select.....	13
9.2.	Opciones avanzadas de Insert	14

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

1. Introducción

En este tema vamos a continuar haciendo consultas contra la base de datos. Nos ocuparemos de nuevas cláusulas que acompañan a la sentencia SELECT y que permiten llegar a consultas más complejas.

2. Agrupación de elementos. GROUP BY y HAVING

Hasta ahora hemos utilizado la orden SELECT para recuperar filas de una tabla y la cláusula WHERE para seleccionar (filtrar) el número de filas que se quieren recuperar.

Pero, a veces, nos interesa consultar los datos agrupados según alguna característica. Así, por ejemplo, para saber cuál es la puntuación media de los jugadores que tienen una valoración concreta necesitamos realizar un agrupamiento por valoración. Para ello utilizaremos la cláusula GROUP BY la cual sirve para calcular propiedades de uno o más conjuntos de filas.

```
SELECT valoracion, avg(puntos) FROM jugadores GROUP BY valoracion;
```

El agrupamiento se lleva a cabo mediante la cláusula GROUP BY por las columnas especificadas y en el orden especificado.

CLAUSULA	DESCRIPCIÓN
WHERE	Selecciona las filas que cumplan la condición especificada.
GROUP BY	Agrupar estas filas.
HAVING	Selecciona los grupos de filas que cumplan la condición especificada.
ORDER BY	Clasifica la salida. Ordena los grupos.

La cláusula HAVING es similar a la cláusula WHERE, pero trabaja con grupos de filas; pregunta por una característica de grupo, es decir, pregunta por los resultados de las funciones de grupo, lo cual WHERE no puede hacer.

Los datos seleccionados en la sentencia SELECT que lleva el GROUP BY deben ser: una constante, una función de grupo (mostrada en la tabla siguiente) o una columna expresada en el GROUP BY.

FUNCIÓN	PROPÓSITO
AVG(n)	Calcula el valor medio de 'n'
COUNT ([DISTINCT] * expresión)	Cuenta el número de veces que la expresión evalúa algún dato con valor no nulo. La opción * cuenta todas las filas seleccionadas.
MAX([DISTINCT] expresión)	Calcula el máximo valor de una expresión.
MIN([DISTINCT] expresión)	Calcula el mínimo valor de una expresión
SUM([DISTINCT] expresión)	Obtiene la suma de valores de la 'expresión' distintos de nulos.

Del mismo modo que existe la condición de búsqueda WHERE para filas individuales, también hay una condición de búsqueda para grupos de filas: HAVING. La cláusula HAVING se emplea para controlar cuál de los conjuntos de filas se visualiza. Se evalúa sobre la tabla que devuelve el GROUP BY. No puede existir sin GROUP BY.

EJEMPLO

Visualiza a partir de la tabla EMPLEADOS el número de empleados que hay en cada departamento. Para hacer esta consulta, tenemos que agrupar las filas de la tabla Empleados por departamento (GROUP BY iddepart) y contarlas (COUNT(*)). La consulta es la siguiente:

```
SELECT iddepart, COUNT(*) FROM empleados GROUP BY iddepart
```

COUNT funciona como una función de grupo y da información sobre un grupo de filas, no sobre filas individuales de la tabla.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

La cláusula GROUP BY iddepart obliga a COUNT a contar las filas que se han agrupado por cada departamento.

Si en la consulta anterior sólo queremos visualizar los departamentos con más de un empleado, tendríamos que utilizar la clausula HAVING:

```
SELECT iddepart, COUNT(*) FROM empleados
GROUP BY iddepart HAVING COUNT(*) > 1;
```

Si queremos ordenar la salida descendientemente por número de empleados, utilizamos la orden ORDER BY:

```
SELECT iddepart, COUNT(*) FROM empleados
GROUP BY iddepart HAVING COUNT(*) > 4
ORDER BY COUNT(*) DESC;
```

La cual es equivalente a:

```
SELECT iddepart, COUNT(*) as "cant" FROM empleados
GROUP BY iddepart HAVING cant > 4
ORDER BY cant DESC
```

COD_EMP	DEPT_NO
1	3
2	5
3	6
4	3
5	3
6	5
7	5
8	3
9	3

DEPT_NO	COUNT(*)
3	5
5	3
6	1

DEPT_NO	COUNT(*)
3	5
5	3

Cuando usamos la cláusula ORDER BY con columnas y funciones de grupo hemos de tener en cuenta que ésta se ejecuta detrás de las cláusulas WHERE, GROUP BY y HAVING. En ORDER BY podemos especificar funciones de grupo, columnas de GROUP BY o su combinación.

Consideraciones a tener en cuenta:

- Cuando se utilizan funciones de grupo en la cláusula SELECT sin que haya GROUP BY el resultado de ejecutar la consulta tiene una sola fila.
- La sentencia SELECT tiene dos cláusulas para realizar restricciones: WHERE y HAVING. Es muy importante saber situar cada restricción en su lugar: las restricciones que se deben realizar a nivel de las filas de la tabla se sitúan en la cláusula WHERE; las restricciones que se deben realizar sobre grupos (normalmente involucran funciones de grupo), se sitúan en la cláusula HAVING.

3. Subconsultas

A veces, para realizar alguna operación de consulta, necesitamos los datos devueltos por otra consulta; así, si queremos obtener los datos de los alumnos que tengan los mismos apellidos que 'Xabi', hemos de averiguar los apellidos de 'Xabi' (primera consulta). Una vez conocido este dato, podemos averiguar qué alumnos tienen los mismos apellidos que 'Xabi' (segunda consulta). Este problema se puede resolver usando una subconsulta, que no es más que una sentencia SELECT dentro de otra SELECT.

El formato de una subconsulta es:

```
SELECT ...
FROM ...
WHERE columna operador (SELECT ...
                        FROM ...
                        WHERE ...
                        );
```

La subconsulta (el comando SELECT entre paréntesis) se ejecutará primero y, posteriormente, el valor extraído es introducido en la consulta principal.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvimiento de Aplicaciones Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

EJEMPLO

Con la tabla jugadores, obtén el nombre de los jugadores con la misma valoración que "Magic Johnson". Para ello, descomponemos el enunciado en dos consultas.

Primero averiguamos cual es la valoración de "Magic Johnson":

```
SELECT valoracion FROM jugadores WHERE nombre="Magic Johnson";
```

Obteniendo como resultado que "Magic Johnson" tiene una valoración de 5

A continuación, visualizamos el nombre de los jugadores con la misma valoración que "Magic Johnson":

```
SELECT nombre FROM jugadores WHERE valoracion=5;
```

Es posible resumir estas dos consultas con una sentencia SELECT que forma parte de la cláusula WHERE:

```
SELECT nombre FROM jugadores WHERE valoracion = (
    SELECT valoracion FROM jugadores WHERE nombre="Magic Johnson"
);
```

3.1. Condiciones de búsquedas en subconsultas

Las subconsultas normalmente aparecen como parte de la condición de búsqueda de una cláusula WHERE o HAVING.

Usaremos las tablas siguientes para los ejemplos:

- Alumnos (Codigo, Nombre, Apellidos, Aula)
- Aulas (Numero, NombreAula, Puestos)

Las condiciones de búsqueda que nos podemos encontrar en una subconsulta son:

- Test de comparación en subconsultas (>, <, <>, <=, >=, =): Compara el valor de una expresión con un **único valor** producido por una subconsulta. Si la subconsulta devuelve más de una fila, se produce un mensaje de error.

EJEMPLO

Obtener los nombre de los alumnos cuya aula sea igual al aula de Frank:


```
SELECT nombre, apellidos, aula FROM alumnos WHERE aula=
    (SELECT aula FROM alumnos WHERE nombre='Frank');
```

- Test de pertenencia a un conjunto devuelto por una subconsulta (IN y not IN): In comprueba si el valor de una expresión coincide (sea igual) con un valor del **conjunto de valores** producidos por una subconsulta mientras que not in comprueba que el valor de una expresión no coincida con algún valor del conjunto de valores devuelto por la subconsulta.

EJEMPLO

Obtener los apellidos de los alumnos cuyo nombre sea alguno de los nombres que hay en el aula 20:

```
SELECT apellidos FROM alumnos WHERE nombre IN
    (SELECT nombre FROM alumnos WHERE aula=20);
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

- Test de comparación cuantificada (ANY y ALL): Se usan en conjunción con los operadores de comparación (>, <, <>, <=, >=, =).

- ANY: Compara el valor de una expresión con cada valor del **conjunto de valores** producidos por una subconsulta, si alguna de las comparaciones individuales da como resultado verdadero, ANY devuelve verdadero, si la subconsulta no devuelve nada devolverá falso.

EJEMPLO

Obtener los datos de los alumnos cuyo nombre sea igual a algún nombre de los alumnos del aula 20:

```
SELECT * FROM alumnos WHERE nombre =
      ANY (SELECT nombre FROM alumnos WHERE aula=20);
```

- ALL: Compara el valor de una expresión con cada valor del **conjunto de valores** producido por una subconsulta, si todas las comparaciones individuales da como resultado verdadero, ALL devuelve verdadero, en caso contrario devuelve falso.

EJEMPLO

Obtener los datos de los alumnos cuya altura sea menor a cualquier altura de los alumnos del aula 11:

```
SELECT * FROM alumnos WHERE altura <
      ALL (SELECT altura FROM alumnos WHERE aula=11);
```

- Test de existencia (EXISTS y NOT EXISTS): Examina si una subconsulta produce alguna fila de resultados. El test es verdadero si devuelve filas, si no es falso. Para el caso contrario se utiliza el operador NOT EXISTS.

EJEMPLO

Listar las aulas que tengan alumnos:

```
SELECT nombreAula, numero FROM aulas WHERE
      EXISTS (SELECT * FROM alumnos WHERE alumnos.aula = aulas.numero);
```

3.2. Subconsultas correlacionadas

Una subconsulta correlacionada es una consulta anidada que contiene referencias a columnas de las tablas que se encuentran en el FROM de la consulta principal (hace referencia a una o varias columnas de la consulta más externa). En caso de que las tablas se llamen igual y si la subconsulta necesita acceder a esas columnas deberá definirse un alias en la tabla más externa.

Por ejemplo, deseamos obtener los datos de los alumnos cuya altura sea la máxima de su aula:

```
SELECT * FROM alumnos as alum WHERE altura=(SELECT MAX(altura) FROM alumnos WHERE aula= alum.aula);
```

A consulta compara para cada fila da consulta externa, es decir, para cada alumno, la altura de ese alumno con la altura máxima de los alumnos de su misma aula (consulta interna); para referenciar a dicha aula se necesita el alias "alum" usado en la tabla de la consulta externa.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

4. Producto cartesiano

Hasta ahora, en las consultas que hemos realizado sólo se ha utilizado una tabla, indicada a la derecha de la palabra FROM; pero hay veces que una consulta necesita acceder a columnas de varias tablas. En este caso, las tablas se expresarán a la derecha de la palabra FROM.

Sintaxis general:

```
SELECT columnas de las tablas citadas en la cláusula from
FROM tabla1, tabla2,...
```

Se realizara una combinación de todas las filas de la tabla1 con todas las filas de la tabla2. El número de columnas será la suma de las columnas de las dos tablas que participen en el producto cartesiano (si no se hace ninguna selección de columna).

EJEMPLO

En las siguientes tablas: ExAlumnos y Teléfonos. Donde id de teléfono es un FK sobre id de exAlumnos

ExAlumnos			Teléfonos		
id	nombre	asignatura	id	teléfono	marca
1	Juan	BD	1	111111111	Apple
2	Pedro	LM	1	222222222	Lg
3	Javi	BD	1	333333333	Mei zu
			2	444444444	Samsung

El producto cartesiano será:

```
select * from exalumnos, telefonos;
```

id	nombre	asignatura	id	teléfono	marca
1	Juan	BD	1	111111111	Apple
2	Pedro	LM	1	111111111	Apple
3	Javi	BD	1	111111111	Apple
1	Juan	BD	1	222222222	Lg
2	Pedro	LM	1	222222222	Lg
3	Javi	BD	1	222222222	Lg
1	Juan	BD	1	333333333	Mei zu
2	Pedro	LM	1	333333333	Mei zu
3	Javi	BD	1	333333333	Mei zu
1	Juan	BD	2	444444444	Samsung
2	Pedro	LM	2	444444444	Samsung
3	Javi	BD	2	444444444	Samsung

Se puede ver que el producto cartesiano es la combinación de todas las filas de la primera tabla con todas las filas de la segunda. Con las dos primeras tablas podemos saber el teléfono de cada exAlumnos gracias al campo id. Pero al realizar el producto cartesiano surgen una serie de filas con valores incorrectos. Por ejemplo nos dice que Juan tiene el teléfono 444444444 lo cual es incorrecto. Veremos ahora como solucionar este problema.

5. Combinación de tablas: JOIN¹

En este apartado se muestra cómo hacer consultas que involucran a datos de varias tablas. Aunque mediante las subconsultas se ha conseguido realizar consultas de este tipo, aquí se verá que en ocasiones, es posible escribir consultas equivalentes que no hacen uso de subconsultas y que se ejecutan de modo más eficiente. El operador que usaremos es JOIN. A este Join también se le denomina como composición interna: INNER JOIN.

¹ <https://mariadb.com/kb/en/mariadb/join-syntax/>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Para hacer una composición interna se parte de un producto cartesiano y se eliminan aquellas tuplas que no cumplen la condición de la composición.

La sintaxis tradicional² es:

```
SELECT columnas de las tablas citadas en la cláusula from
FROM tabla1, tabla2,...
WHERE tabla1.columna = tabla2.columna;
```

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

- Es posible unir tantas tablas como deseemos.
- En la cláusula SELECT se pueden citar columnas de todas las tablas que se especifiquen en el FROM.
- Si hay columnas con el mismo nombre en las distintas tablas de la cláusula FROM, se deben identificar anteponiéndole el nombre de la tabla. Por ejemplo: nombreTabla.nombreColumna.
- Si el nombre de una columna existe sólo en una tabla, no será necesario especificarla como nombreTabla.NombreColumna. Pero, hacerlo mejoraría la legibilidad de la sentencia SELECT.
- El criterio que 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.

Veamos un ejemplo para las tablas anteriores:

```
select * from exalumnos, telefonos where exalumnos.id=telefonos.id;
```

id	nombre	asignatura	id	teléfono	marca
1	Juan	BD	1	1111111111	Appl e
1	Juan	BD	1	2222222222	Lg
1	Juan	BD	1	3333333333	Mei zu
2	Pedro	LM	2	4444444444	Samsung

En cualquier caso, la salida sólo contiene las tuplas que emparejan a los exalumnos con sus números de teléfono

Esta sintaxis es equivalente a las marcadas en rojo³, que serán las que usemos en clase:

```
SELECT * FROM exalumnos [INNER] JOIN telefonos WHERE exalumnos.id = telefonos.id;
SELECT * FROM exalumnos [INNER] JOIN telefonos ON exalumnos.id = telefonos.id;
SELECT * FROM exalumnos [INNER] JOIN telefonos USING(id);
```

Donde:

- La coma y JOIN son equivalentes⁴, y la palabra INNER es opcional.
- La condición en la cláusula ON puede ser cualquier expresión válida para una cláusula WHERE, de hecho, en la mayoría de los casos, son equivalentes.
- La cláusula USING nos permite usar una lista de atributos que deben ser iguales en las dos tablas a componer. Al contrario que con on las columnas especificadas en using aparecen una sola vez, y al principio, en el resultado ya que tienen que tener el mismo valor en ambas tablas.

Las mismas sentencias que se realizan mediante Joins se pueden realizar con subconsultas pero los joins suelen ser más eficientes al realizar consultas y los resultados se recuperan con mayor rapidez.

² <http://code.openark.org/blog/mysql/mysql-joins-on-vs-using-vs-theta-style>

³ <http://code.openark.org/blog/mysql/mysql-joins-on-vs-using-vs-theta-style>

⁴ <https://mariadb.com/kb/en/mariadb/comma-vs-join/>

<div> <div>COLEXIO</div> <div>VIVAS S.L.</div> </div>	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

5.1. Composición interna natural: NATURAL JOIN

Natural Join nos permite realizar la unión de dos o más tablas en función de los campos que se llamen igual en las tablas que intervienen en el join.

La unión del ejemplo anterior, con natural join, sería.

EJEMPLO

```
SELECT exalumnos.id, nombre, asignatura, telefono FROM exalumnos NATURAL JOIN telefonos;
```

id	nombre	asignatura	telefono	marca
1	Juan	BD	1111111111	Apple
1	Juan	BD	2222222222	Lg
1	Juan	BD	3333333333	Meizu
2	Pedro	LM	4444444444	Samsung

En este caso no se usa ni *on* ni *using* ya que automáticamente realiza el join en función de las columna que se llamen igual en las dos tablas: en el ejemplo el campo id. En este caso, al igual que con *using*, las columnas utilizadas para realizar el join solo aparecen una vez en el resultado, ya que tienen los mismos valores.

El peligro de usar natural join es que puede tener comportamientos inesperados. Imaginemos las mismas tablas que en el ejemplo anterior, pero cambiando el nombre de la columna marca de telefonos por nombre.

ExAlumnos			Teléfonos		
id	nombre	asignatura	id	teléfono	nombre
1	Juan	BD	1	1111111111	Apple
2	Pedro	LM	1	2222222222	Lg
3	Javi	BD	1	3333333333	Meizu
			2	4444444444	Samsung

El resultado de la consulta anterior sería vacío ya que ahora la unión se haría teniendo en cuenta tanto la columna id como la columna nombre y nombre en ExAlumnos nunca será igual a nombre en Teléfonos.

6. Combinación externa: OUTER JOIN

Existe una variedad de combinación de tablas que se llama OUTER JOIN y que nos permite seleccionar algunas filas de una tabla aunque éstas no tengan correspondencia con las filas de la otra tabla con la que se combina.

Esto puede ser útil por ejemplo si queremos mostrar todos los exalumnos tengan o no tengan un teléfono asociado y, en caso de que sí lo tengan, mostrar el número de dicho teléfono.

Las sintaxis para las composiciones externas es:

```
tabla1 [ NATURAL ] { LEFT | RIGHT } [ OUTER ] JOIN tabla2
```

Existen dos grupos de composiciones externas: izquierda y derecha, dependiendo de cuál de las tablas se lea en primer lugar.

- Composición externa derecha: Se recorre la tabla de la derecha y se buscan tuplas que cumplan la condición en la tabla izquierda. Además de añade al resultado las filas de la tabla de la derecha que no tienen correspondencia en la tabla de la izquierda. Se usa la palabra RIGHT.
- Composición externa izquierda: En estas composiciones se recorre la tabla de la izquierda y se buscan tuplas en la de la derecha. Además de añade al resultado las filas de la tabla de la izquierda que no tienen correspondencia en la tabla de la derecha. Se usa la palabra LEFT.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Es lo mismo usar una combinación derecha de las tablas tabla1 y tabla2 que una combinación izquierda de las tablas tabla2 y tabla1.

```
SELECT * FROM exalumnos LEFT JOIN telefonos USING(id);
```

id	nombre	asignatura	telefono	marca
1	Juan	BD	1111111111	Apple
1	Juan	BD	2222222222	Lg
1	Juan	BD	3333333333	Mei zu
2	Pedro	LM	4444444444	Samsung
3	Javi	BD	NULL	NULL

En el ejemplo se tomaría la primera tupla de exalumnos, con un valor de id igual a 1, y se busca en la tabla teléfonos las tuplas con un valor de id igual a 1. Lo mismo para la segunda tupla, con id igual a 2.

En la tercera el id es 3, y no existe ninguna tupla en teléfonos con un valor de id igual a 3, por lo tanto se combina la tupla de exalumnos con una tupla de teléfonos con todos los atributos igual a NULL (quinta fila).

Por supuesto, también podemos hacer composiciones externas naturales: NATURAL LEFT JOIN y NATURAL RIGHT JOIN.

En su versión actual MariaDB/MySQL no admite las combinaciones externas completas. En estas combinaciones, cada registro de la primera tabla, incluyendo aquellos que no tengan una correspondencia en la segunda, se devuelve junto a cada registro de la segunda tabla, incluyendo aquellos sin correspondencias en la primera.

La sintaxis es la misma que para las otras combinaciones:

```
SELECT campo1, campo2 FROM tabla1 FULL OUTER JOIN tabla2
```

Equivalen a una combinación por la izquierda unida a una combinación por la derecha.

7. Composición de consultas.

7.1. Operador unión: UNION.

El operador UNION se utiliza para combinar los resultados de dos o más consultas en un único resultado

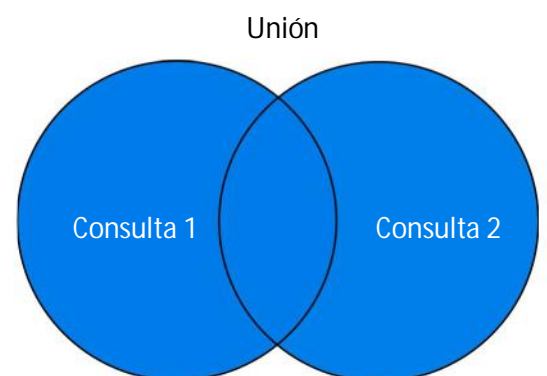
Cada select de la unión tiene que obtener el mismo número de columnas y la columnas de cada select, posición a posición, deben tener tipos de datos compatibles.


Los nombres de las columnas del resultado se obtienen del primer select de la unión. Se puede cambiar este nombre mediante el uso de alias.

Su formato es:

```
SELECT COL1, COL2, ... FROM TABLA1 [WHERE CONDICION]
UNION [ALL | DISTINCT] SELECT COL1, COL2, ... FROM TABLA2 [WHERE CONDICION]
[
  UNION [ALL | DISTINCT] SELECT COL1, COL2, ... FROM TABLA3 [WHERE CONDICION]
  .....
] [ORDER BY COL1[, COL2, ...]] [LIMIT cantidad];
```

Por defecto UNION elimina filas duplicadas del resultado, es el mismo resultado que usando la palabra reservada DISTINCT. Mediante ALL el resultado incluye todas las filas incluyendo las filas duplicadas.



	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Cambio de nombre de una columna de la unión	<pre> Select nombre, nombre as 'unión' from exalumnos unión select marca, marca from telefonos; </pre>	<pre> +-----+-----+ nombre unión +-----+-----+ Juan Juan Pedro Pedro Javi Javi Appl e Appl e LG LG Mei zu Mei zu Samsung Samsung +-----+-----+ </pre>
---	--	---

Unión por defecto elimina filas duplicadas	<pre> Select nombre from exalumnos unión select nombre from exalumnos; </pre>	<pre> +-----+ nombre +-----+ Juan Pedro Javi +-----+ </pre>
--	---	--

Usando ALL se muestran todos los resultados de la Unión	<pre> Select nombre from exalumnos union all select nombre from exalumnos; </pre>	<pre> +-----+ nombre +-----+ Juan Pedro Javi Juan Pedro Javi +-----+ </pre>
---	---	---

El resultado final se puede ordenar mediante *ORDER BY* además de limitar el número de filas del resultado mediante *LIMIT* de la misma manera que se ha visto en temas anteriores.

Ordenar y limitar el número de filas resultantes	<pre> Select nombre, nombre as 'unión' from exalumnos unión select marca, marca from telefonos order by nombre limit 3; </pre>	<pre> +-----+-----+ nombre unión +-----+-----+ Appl e Appl e Javi Javi Juan Juan +-----+-----+ </pre>
--	--	---

Si se desea ordenar o limitar el número de filas de un select de la unión este se ha de encerrar entre paréntesis teniendo en cuenta que para el orden final se usa el orden de la unión (si no se usa limit el order by del select no se tiene en cuenta).

Ordenación de un select de la unión (entre paréntesis). En este caso se utiliza para limitar a solo dos elementos el segundo select.	<pre> Select nombre, nombre as 'unión' from exalumnos unión (select marca, marca from telefonos order by marca desc limit 2) order by nombre; </pre>	<pre> +-----+-----+ nombre unión +-----+-----+ Javi Javi Juan Juan Mei zu Mei zu Pedro Pedro Samsung Samsung +-----+-----+ </pre>
--	--	---

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD	COMPETENCIA				

Identificación del select de cada fila del resultado

```
Select 1 as query, nombre from exalumnos
union
(select 2 as query, marca from telefonos order by marca desc limit 2)
order by query desc, nombre;
```

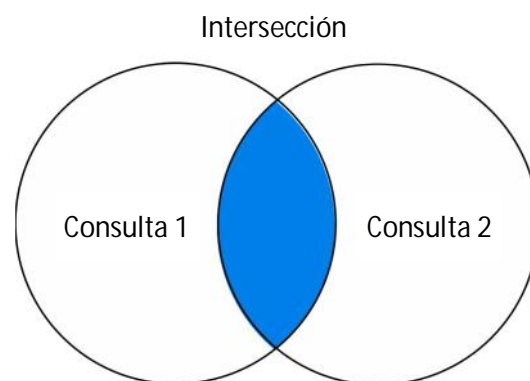
query	nombre
2	Mei zu
2	Samsung
1	Javi
1	Juan
1	Pedro

7.2. Operador Intersección: INTERSECT.

Este operador muestra las filas que son comunes en dos o más consultas. Esta soportado desde la versión 10.3 de MariaDB.

El significado de los distintos elementos es exactamente el mismo que para la unión teniendo en cuenta que Intersect no soporta ALL. Las filas duplicadas se eliminan.

La intersección tiene una prioridad mayor que la unión y la diferencia.



Su formato es:

```
SELECT COL1, COL2, ... FROM TABLA1 [WHERE CONDICION]
INTERSECT [DISTINCT ] SELECT COL1, COL2, ... FROM TABLA2 [WHERE CONDICION]
[
  INTERSECT [DISTINCT ] SELECT COL1, COL2, ... FROM TABLA3 [WHERE CONDICION]
  .....
] [ORDER BY COL1[, COL2, ...]] [LIMIT cantidad];
```

Se seleccionan los ids comunes a ambas tablas

```
select id from exalumnos
INTERSECT
select id from telefonos;
```

id
1
2

Ejemplo de precedencia

```
(select a,b from t1)
UNION
(select c,d from t2)
INTERSECT
(select e,f from t3)
UNION
(select 4,4);
```

Primero se ejecuta la intersección de t2 y t3 ya que tiene mayor precedencia y luego se realizan las uniones.

<div> <div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div> </div>	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

7.3. Operador diferencia: EXCEPT.

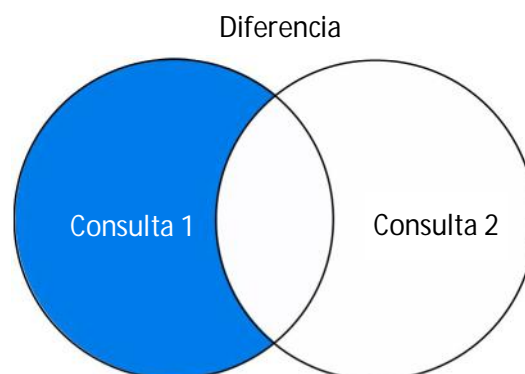
El operador diferencia devuelve todas las filas del primer select menos las filas del segundo select.

El significado de los distintos elementos es exactamente el mismo que para la unión teniendo en cuenta que la diferencia no soporta ALL. Las filas duplicadas se eliminan.

La intersección tiene la misma prioridad que la unión y menor que la intersección.

Su formato es:

```
SELECT COL1, COL2, ... FROM TABLA1 [WHERE CONDICION]
EXCEPT [DISTINCT ] SELECT COL1, COL2, ... FROM TABLA2 [WHERE CONDICION]
[ EXCEPT [DISTINCT ] SELECT COL1, COL2, ... FROM TABLA3 [WHERE CONDICION]
.....
] [ORDER BY COL1[, COL2, ...]] [LIMIT cantidad];
```



Se seleccionan los ids de exalumnos que no estén el teléfonos

```
select id from exalumnos
EXCEPT
select id from telefonos;
```

```
+-----+
| id |
+-----+
| 3 |
+-----+
```

7.4. Diferencias entre Union, Intersect y Except.

Las siguientes consultas muestran, ante los mismos datos, la diferencia de comportamiento de la unión, la intersección y la diferencia.

Tabla de datos a usar en las consultas

```
CREATE TABLE seqs (i INT);
INSERT INTO seqs VALUES (1), (2), (3), (4), (5), (6);
```

Union

```
SELECT i FROM seqs WHERE i<=3
UNION
SELECT i FROM seqs WHERE i>=3;
```

```
+-----+
| i |
+-----+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
+-----+
```

Intersección


```
SELECT i FROM seqs WHERE i<=3
INTERSECT
SELECT i FROM seqs WHERE i>=3;
```

```
+-----+
| i |
+-----+
| 3 |
+-----+
```

Diferencia

```
SELECT i FROM seqs WHERE i<=3
EXCEPT
SELECT i FROM seqs WHERE i>=3;
```


```
+-----+
| i |
+-----+
| 1 |
| 2 |
+-----+
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

8. Resumen:

Para generar los resultados de una instrucción SELECT hay que seguir estos pasos:

1. SELECT: Extrae las columnas seleccionadas de cada fila.
SELECT DISTINCT: Extrae las columnas seleccionadas eliminando filas duplicadas.
2. FROM: Indica la tabla o tablas a usar.
3. WHERE: Condición de filtrado a cada fila.
4. GROUP BY: Agrupa las filas seleccionadas con WHERE.
5. HAVING: Aplica una condición de búsqueda a los grupos de filas. Selecciona los grupos.
6. ORDER BY: Ordena los resultados.
7. LIMIT: Limita el número de resultados devueltos por la consulta.
8. Si la instrucción es una UNION, INTERSECT o EXCEPT se repiten los pasos para cada SELECT.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

9. Apéndice 1:


9.1. Opciones avanzadas de Select

En negrita tenemos las opciones que aún no hemos visto de select.

```
SELECT [ ALL | DISTINCT ]
      [ HIGH_PRIORITY ]
      [ STRAIGHT_JOIN ]
      [ SQL_SMALL_RESULT ] [ SQL_BIG_RESULT ] [ SQL_BUFFER_RESULT ]
      [ SQL_CACHE | SQL_NO_CACHE ] [ SQL_CALC_FOUND_ROWS ]
      columna1 [,columna2, ...]
      [ FROM tabla
        [ WHERE condicion ]
        [ GROUP BY { columna | expr | posición }
        [ HAVING condicion ]
        [ ORDER BY { columna | expr | posición } [ASC | DESC], ... ]
        [ LIMIT {(desplazamiento,) numFilas }
      ]
```

Dónde:

- **HIGH_PRIORITY**: Da prioridad al SELECT sobre instrucciones INSERT que se estén ejecutando al mismo tiempo. Solo afecta a tablas que solo disponen de bloqueo de tabla (MyISAM, MEMORY, MERGE).
- **STRAIGHT_JOIN**: Fuerza al optimizador a ejecutar el JOIN en el orden exacto en que se especifica en la sentencia SELECT. Se usa cuando sabemos que el optimizador produce una orden que de hecho no es óptimo.
- **SQL_SMALL_RESULT**: Puede usarse con GROUP BY o DISTINCT para decir al optimizador que el conjunto de resultados es pequeño. En este caso, MySQL usa tablas temporales rápidas para almacenar la tabla resultante en lugar de usar ordenación. En MySQL 5.x, esto normalmente no hará falta.
- **SQL_BIG_RESULT**: Se puede usar con GROUP BY y DISTINCT para decirle al optimizador que el resultado va a ser muy grande. Eso le dice al optimizador que debe usar tablas temporales en disco.
- **SQL_BUFFER_RESULT**: Fuerza a que los resultados sean almacenados en una tabla temporal. Esto ayuda a MySQL a liberar los bloqueos de tabla rápidamente y ayuda en casos en que tarda mucho tiempo en enviar el resultado al cliente.
- **SQL_CACHE**: le dice a MySQL que almacene el resultado de la consulta en la caché de consultas si está usando un valor de query_cache_type de 2 o DEMAND. Para una consulta que use UNION o subconsultas, esta opción afecta a cualquier SELECT en la consulta.
- **SQL_NO_CACHE**: le dice a MySQL que no almacene los resultados de consulta en la caché de consultas. Para una consulta que use UNION o subconsultas esta opción afecta a cualquier SELECT en la consulta.
- **SQL_CALC_FOUND_ROWS**: le dice a MySQL que calcule cuántos registros habrán en el conjunto de resultados, sin tener en cuenta ninguna cláusula LIMIT. El número de registros pueden encontrarse con SELECT FOUND_ROWS().

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Bases de datos				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	3	DATA:	2021/2022
	UNIDAD COMPETENCIA					

9.2. Opciones avanzadas de Insert

En negrita tenemos las opciones que aún no hemos visto de insert.

```
INSERT [ LOW_PRIORITY | HIGH_PRIORITY ] [ IGNORE ]
[ INTO ] tbl_name [ (col_name,...) ]
{VALUES | VALUE} ({expr | DEFAULT},...),(...),...
[ ON DUPLICATE KEY UPDATE
    col_name=expr[, col_name=expr] ...
]
```

Donde:

- **LOW_PRIORITY**: la ejecución de INSERT se retrasa hasta que no hay otros clientes leyendo de la tabla. Esto incluye a otros clientes que comiencen a leer mientras que los clientes existentes están leyendo, y mientras el comando INSERT LOW_PRIORITY está en espera. Es posible, por lo tanto, para un cliente que realice un comando INSERT LOW_PRIORITY esperar durante mucho tiempo (o incluso para siempre) en un entorno de muchas lecturas. Tener en cuenta que LOW_PRIORITY no debe usarse normalmente con tablas MyISAM y que hacerlo deshabilita inserciones concurrentes.
- **HIGH_PRIORITY**: deshabilita el efecto de la opción --low-priority-updates si el servidor se arrancó con esa opción. Hace que las inserciones concurrentes no se usen.
- **IGNORE**: en un comando INSERT, los errores que ocurren mientras se ejecuta el comando se tratan como advertencias. Por ejemplo, sin IGNORE, un registro que duplique un índice UNIQUE existente o valor PRIMARY KEY en la tabla hace que un error de clave duplicada en el comando se aborte. Con IGNORE, el registro todavía no se inserta, pero no se muestra error. Las conversiones de datos dispararían errores y abortarían el comando si no se especificara IGNORE. Con IGNORE, los valores inválidos se ajustan al valor más cercano y se insertan; las advertencias se producen pero el comando no se aborta. Se puede determinar con la función mysql_info() de la API de C cuántos registros se insertan realmente en la tabla.
- **ON DUPLICATE KEY UPDATE**: si se inserta un registro que duplicaría un valor en un índice UNIQUE o PRIMARY KEY, se realiza un UPDATE del antiguo registro.