

APUNTES SQL - Joel Sánchez Fernández

Curso interactivo en el siguiente link:

<https://sqlinteractivo.desafiolatam.com/cursos/1/categoria/330>

La propia página tiene su editor con el que poder trabajar los conceptos que te da en la explicación anterior, me pareció buena forma de aprendizaje y pensé en llevarla a cabo.

> Índice <

<u>NOMBRE</u>	<u>PUNTOS</u>
1. Seleccionar columnas	0 / 7
2. Seleccionar filas	0 / 15
3. Ordenar resultados	0 / 6
4. Limitar resultados	0 / 2
5. Operaciones con 'Strings'	0 / 10
6. Operaciones con fechas	0 / 6
7. Funciones de agregación	0 / 6
8. Distinct	0 / 6
9. Introducción a grupos	0 / 9
10. Having	0 / 5
11. Subconsultas	0 / 4
12. Combinación de consultas	0 / 5
13. Inserción de registros	0 / 9
14. Borrado y modificación de registros	0 / 5
15. Tablas	0 / 6
16. Restricciones	0 / 11
17. Consultas en múltiples tablas	0 / 6
18. Cardinalidad	0 / 6
19. Tipos de 'Join'	0 / 19
20. Normalización	0 / 7

ORDEN	CLAUSULA	DESCRIPCIÓN
1	SELECT	Especifica las columnas que se deben retornar en el resultado.
2	FROM	Especifica las tablas de las cuales se extraerán los datos.
3	WHERE	Filtra registros antes de cualquier agregación o agrupación.
4	GROUP BY	Agrupar registros por una o más columnas.
5	HAVING	Filtra registros después de la agregación.
6	ORDER BY	Ordena los registros retornados por una o más columnas.
7	LIMIT	Limita el número de registros retornados.

Empezado el 27/01/2025.

1. SELECCIONAR COLUMNAS

1.1 SELECT

La consulta que más realizaremos es "SELECT * FROM nombre_tabla;", el punto y coma indica el final de la consulta que queramos hacer.

Ejemplo en web del curso, vamos a buscar la tabla 'asistentes' :

1

SELECT * FROM asistentes

ID	NOMBRE	APELLIDO
1	Juan	Perez
2	Maria	Meza
3	Luis	Gonzalez
4	Rosa	Espinoza
5	Juana	Mendez

1.2 SELECT 'Seleccionamos toda la tabla'

```
1 SELECT * FROM productos;
```

ID	NOMBRE	PRECIO	DESCUENTO
1	Silla Oficina	100	0
2	Mesa Oficina	200	5
3	Lámpara de Escritorio	50	10
4	Monitor de Computadora	150	0
5	Librero de Oficina	300	15

1.3 SELECT 'Seleccionamos una columna'

Vamos a llamar a la columna 'apellido' de la tabla 'usuario' con la siguiente consulta "SELECT apellido FROM usuarios":

```
1 SELECT apellido FROM usuarios
```

APELLIDO
Pérez
García
López
Sánchez
Martínez

1.4 SELECT 'Seleccionamos varias columnas'

Vamos a hacer lo mismo pero ponemos varias columnas de seguido separadas por comas, es decir, "SELECT nombre, precio FROM productos":

<pre>1 SELECT nombre, precio FROM productos;</pre>	<table><tr><th>NOMBRE</th><th>PRECIO</th></tr><tr><td>Producto 1</td><td>9.99</td></tr><tr><td>Producto 2</td><td>19.99</td></tr><tr><td>Producto 3</td><td>14.99</td></tr></table>	NOMBRE	PRECIO	Producto 1	9.99	Producto 2	19.99	Producto 3	14.99
NOMBRE	PRECIO								
Producto 1	9.99								
Producto 2	19.99								
Producto 3	14.99								

1.5 SELECT 'Alias (AS)'

Los alias son nombres alternativos que le podemos dar a la columna que estamos seleccionando. Por ejemplo, "SELECT nombre AS cliente FROM usuarios;":

```
1 SELECT nombre AS cliente FROM usuarios
```

CLIENTE
Juan
María
Pedro
Lucía
Jorge

1.6 SELECT 'Alias (AS) varias columnas'

También podemos hacer lo mismo pero con varias columnas, siendo separadas por comas:

```
1 SELECT nombre AS nombre_usuario, apellido AS apellido_usuario FROM usuarios;
```

NOMBRE_USUARIO	APELLIDO_USUARIO
Juan	Pérez
María	García
Pedro	López
Lucía	Sánchez
Jorge	Martínez

1.7 SELECT 'Alias (AS) nombres largos'

Lo mismo, pero ahora si queremos un nombre separado tenemos que poner comillas doble tal que así **"SELECT nombre, email AS "Correo electrónico" FROM usuarios;"**:

<pre>1 SELECT nombre, email AS "Correo electrónico" FROM usuarios;</pre>	<table><tr><th>NOMBRE</th><th>CORREO ELECTRÓNICO</th></tr><tr><td>Juan Pérez</td><td>juanperez@gmail.com</td></tr><tr><td>María García</td><td>mariagarcia@hotmail.com</td></tr><tr><td>Pedro López</td><td>pedrolopez@yahoo.com</td></tr><tr><td>Lucía Sánchez</td><td>luciasanchez@outlook.com</td></tr><tr><td>Jorge Martínez</td><td>jorgemartinez@gmail.com</td></tr></table>	NOMBRE	CORREO ELECTRÓNICO	Juan Pérez	juanperez@gmail.com	María García	mariagarcia@hotmail.com	Pedro López	pedrolopez@yahoo.com	Lucía Sánchez	luciasanchez@outlook.com	Jorge Martínez	jorgemartinez@gmail.com
NOMBRE	CORREO ELECTRÓNICO												
Juan Pérez	juanperez@gmail.com												
María García	mariagarcia@hotmail.com												
Pedro López	pedrolopez@yahoo.com												
Lucía Sánchez	luciasanchez@outlook.com												
Jorge Martínez	jorgemartinez@gmail.com												

2. SELECCIONAR FILAS

2.1 OPERADOR "Mayor que"

Aquí vamos a empezar a usar la cláusula 'WHERE', pongamos un ejemplo, tenemos una tabla 'pedidos' con la columna precio, pues podríamos recibir las filas con el precio de producto mayor que 100, es decir, **"SELECT * FROM productos WHERE precio > 100;"**.

Tengamos en cuenta, que todas las cláusulas siguen el siguiente orden:

1. SELECT
2. FROM
3. WHERE

Si las cambiamos el orden, nos puede llegar a dar fallo en la sintaxis.

1. `SELECT * FROM productos WHERE descuento > 10;`

ID	NOMBRE	PRECIO	DESCUENTO
5	Librero de Oficina	300	15

2.2 OPERADOR “Mayor o igual que”

Misma función que el anterior, pero añadimos el igual, esto cambia la sintaxis a ‘>=’, la cláusula sigue siendo la misma, solo que ahora es mayor o igual que:

```
1 SELECT * FROM productos WHERE precio >= 50;
```

ID	NOMBRE	PRECIO	DESCUENTO
1	Silla de Oficina	100	0
2	Mesa de Oficina	200	5
3	Lámpara de Escritorio	50	10
4	Monitor de Computadora	150	0
5	Librero de Oficina	300	15

2.3 OPERADOR “Menor que”

Este operador, es menor que, usaremos el mismo símbolo que antes pero a la inversa, es decir, “SELECT * FROM usuarios WHERE id < 3;”:

```
1 SELECT * FROM usuarios WHERE id < 3;
```

ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
1	Juan	Pérez	juanperez@gmail.com	555-1234
2	María	García	mariagarcia@hotmail.com	555-5678

2.4 OPERADOR “Menor o igual que”

Misma función que el anterior, pero añadimos el igual, esto cambia la sintaxis a ‘<=’, la cláusula sigue siendo la misma, solo que ahora es menor o igual que:

```
1 SELECT * FROM productos WHERE precio <= 100;
```

ID	NOMBRE	PRECIO	DESCUENTO
1	Silla de Oficina	100	0
3	Lámpara de Escritorio	50	10

2.5 SELECCIONANDO “Filas bajo condición”

A veces queremos seleccionar columnas con ciertas condiciones, por ejemplo, queremos seleccionar el ‘id’ y el ‘nombre’ de la tabla ‘productos’, los cuales tengan un precio mayor a 30, esto, en sintaxis sería tal que así “SELECT id, nombre FROM productos WHERE precio > 30;”:

```
1 SELECT nombre, precio, cantidad FROM productos WHERE cantidad > 6;
```

NOMBRE	PRECIO	CANTIDAD
Producto 3	60	8

2.6 SELECCIONANDO “Filas bajo condición de igualdad ‘=’ ”

Cuando queramos seleccionar una condición específica, usaremos '=', en un ejemplo sería así `“SELECT nombre FROM usuarios WHERE id = 2;”`:

```
1 SELECT nombre FROM usuarios WHERE id = 2;
```

NOMBRE
María

2.7 SELECCIONANDO “Filas bajo condición de igualdad ‘=’ (String)”

Si queremos comparar textos, tenemos que usar las comillas simples (' '). Ejemplo `“SELECT * FROM productos WHERE nombre = 'Pantalón';”`:

```
1 SELECT * FROM productos WHERE nombre = 'Pantalón';
```

ID	NOMBRE	PRECIO	DESCUENTO
2	Pantalón	30	5

Tenemos que tener en cuenta que la comparación es sensible, es decir, que tenemos que tener cuidado al comparar con MAYUS y MINUS, no es lo mismo 'Camiseta' que 'camiseta'.

2.8 SELECCIONANDO “Filas bajo condición de igualdad (Boolean)”

Este solo puede guardar dos valores, TRUE o FALSE. Imagina que tenemos una tabla ‘productos’ con la columna ‘destacado’, se puede usar **“SELECT * FROM productos WHERE destacado = true;”**, o también se puede utilizar la siguiente manera **“SELECT * FROM productos WHERE destacado = 1;”**, ya que 1 es lo mismo que TRUE:

```
1 SELECT * FROM usuarios WHERE status = 1;
```

NOMBRE	APELLIDO	EMAIL	TELEFONO	STATUS
Juan	Pérez	juanperez@gmail.com	555-1234	1
Pedro	López	pedrolopez@yahoo.com	555-9876	1
Lucía	Sánchez	luciasanchez@outlook.com	555-5555	1

Pasa lo mismo con un valor FALSE o 0 (cero):

```
1 SELECT * FROM productos WHERE destacado = false;
```

ID	NOMBRE	PRECIO	DESTACADO
2	Producto 2	200	0
3	Producto 3	150	0
5	Producto 5	250	0

2.9 OPERADOR “Dos condiciones con operador AND”

WHERE se puede combinar con el operador AND, que se utiliza para añadir otra condición. Si tenemos dos condiciones pero una de ellas no se cumple, no se mostrará nada. Por ejemplo **“SELECT * FROM usuarios WHERE nombre = ‘María’ AND email = ‘mariagarcia@hotmail.com’;”**:

```
1 SELECT * FROM usuarios WHERE nombre = 'María' AND email = 'mariagarcia@hotmail.com';
```

ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
2	María	García	mariagarcia@hotmail.com	555-5678

2.10 OPERADOR “Dos condiciones con operador OR”

WHERE se combina con este operador para varias condiciones, al menos una condición tiene que ser verdadera para que te devuelva resultado. Por ejemplo “SELECT * FROM productos WHERE precio > 1000 OR descuento = 20;”:

```
1 SELECT * FROM productos WHERE precio > 1000 OR descuento = 20;
```

ID	NOMBRE	PRECIO	DESCUENTO
1	Producto1	1500	10
2	Producto2	2000	15
3	Producto3	800	20
4	Producto4	1200	5
5	Producto5	2500	20

2.11 SELECCIONANDO “Una fecha, otro tipo de dato”

Este es el tipo de dato ‘date’ (fecha). Estas tienen un formato YYYY-MM-DD. Vamos a utilizarlo para saber como se usa, si queremos sacar todos los productos de la tabla ‘productos’ creados después de ‘2021-05-01’, usaremos la sentencia “SELECT * FROM productos WHERE fecha_de_creacion >= ‘2021-05-01’;”:

```
1 SELECT * FROM productos WHERE fecha_de_creacion >= '2021-05-01';
```

ID	NOMBRE	PRECIO	FECHA_DE_CREACION
1	Silla de Oficina	100	2022-01-05
3	Lámpara de Escritorio	50	2022-02-10
5	Librero de Oficina	300	2022-03-01

2.12 SELECCIONANDO “Datos con “between””

El BETWEEN se usa seleccionar registros dentro de un rango específico. Esto son datos inclusivos, los cuales incluyen los valores de referencia que le indicamos. Por ejemplo **“SELECT * FROM productos WHERE stock BETWEEN 20 AND 30;”**:

```
1 SELECT * FROM productos WHERE stock BETWEEN 20 AND 30;
```

ID	NOMBRE	STOCK
2	Producto B	20
3	Producto C	30

Hay que tener cuidado, ya que si los valores que marcamos no queremos que sean inclusivos, tendremos que usar la siguiente sintaxis **“SELECT * FROM productos WHERE stock >= 20 AND stock < 30;”**.

2.13 SELECCIONANDO “Filas con “like””

Este se usa cuando queremos buscar algo que empiece por tal letras, es decir, un ejemplo es este **“SELECT * FROM usuarios WHERE apellido LIKE ‘Ma%’**. El % represente cualquier carácter adicional después de las letras:

```
1 SELECT * FROM usuarios WHERE apellido LIKE 'Ma%';
```

ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
5	Jorge	Martínez	jorgemartinez@gmail.com	555-4321

Esto también puede ser al revés, con el % al principio y queremos que termine en ciertas letras. Por ejemplo **"SELECT * FROM usuarios WHERE nombre LIKE '%o'";**

```
1 SELECT * FROM usuarios WHERE nombre LIKE '%o';
```

ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
3	Pedro	López	pedrolopez@yahoo.com	555-9876

El % también representa números, representa cualquier carácter.

2.14 SELECCIONANDO "Registros sin valores nulos"

Algunos registros pueden ser nulos y no tener 'nombre' o 'email':

ID	NOMBRE	EMAIL
1	Juan Pérez	juan.perez@email.com
2	María Gómez	maria.gomez@email.com
3		carlos.diaz@email.com
4		ana.torres@email.com
5	Luis Méndez	luis.mendez@email.com

Para seleccionar los valores que no sean nulos usamos IS NOT NULL. Por ejemplo "SELECT * FROM productos WHERE descuento IS NOT NULL;":

<pre>1 SELECT * FROM productos WHERE descuento IS NOT NULL;</pre>			
ID	NOMBRE	PRECIO	DESCUENTO
1	Silla de Oficina	100	0
3	Lámpara de Escritorio	50	10
4	Monitor de Computadora		0
5	Librero de Oficina	300	15

También podemos seleccionar los valores nulos, si queremos todos los usuarios que no tengan email registrado en la tabla 'usuarios', usaremos "SELECT * FROM usuarios WHERE email IS NULL;":

<pre>1 SELECT * FROM usuarios WHERE email IS NULL;</pre>				
ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
2	María	García		555-5678
4	Lucía	Sánchez		555-5555

3. ORDENAR RESULTADOS

3.1 ORDENANDO "Filas (ASC)"

Vamos a usar el ORDER BY, esta cláusula ordena resultados según una o más columnas, por defecto es de forma ascendente (ASC). Por ejemplo, si queremos ordenar todos los registros por la columna 'nombre' de la tabla 'usuarios' usamos "SELECT * FROM usuarios ORDER BY nombre ASC;":

<pre>1 SELECT * FROM usuarios ORDER BY nombre ASC;</pre>					
ID	NOMBRE	APELLIDO	EMAIL	TELEFONO	
5	Jorge	Martínez	jorgemartinez@gmail.com	555-4321	
1	Juan	Pérez	juanperez@gmail.com	555-1234	
4	Lucía	Sánchez	luciasanchez@outlook.com	555-5555	
2	María	García	maria Garcia@hotmail.com	555-5678	
3	Pedro	López	pedrolopez@yahoo.com	555-9876	

3.2 ORDENANDO “Filas (DESC)”

Vamos a usar el ORDER BY, pero ahora de forma descendente (DESC). Por ejemplo, si queremos ordenar los registros de forma descendente por la columna 'precio' de la tabla 'productos' usamos **“SELECT precio FROM productos ORDER BY precio DESC;”**:

<pre>1 SELECT precio FROM productos ORDER BY precio DESC;</pre>	PRECIO
	300
	200
	150
	100
	50

3.3 ORDENANDO “Filas con valores nulos”

Aquí solamente usaremos la lógica, si ordenamos de forma ascendente al tener un valor nulo será mostrado arriba del todo, ya que está vacío. Pero si ordenamos de forma descendente este valor nulo se mostrará abajo del todo.

3.4 ORDENANDO “Filas con valores nulos al final”

Aquí veremos dos formas de como manejar nuestros valores nulos, la sentencia es **“SELECT * FROM productos ORDER BY precio NULLS LAST / NULLS FIRST;”**. Si ponemos NULLS LAST, nuestros valores nulos se mostrarán abajo del todo de la tabla, en cambio con NULLS FIRST, los nulos estarán arriba del todo:

<pre>1 SELECT * FROM productos ORDER BY precio NULLS LAST;</pre>	ID	NOMBRE	PRECIO
	3	Producto 3	50
	1	Producto 1	100
	5	Producto 5	200
	2	Producto 2	
	4	Producto 4	

3.5 ORDENANDO “Filas con combinaciones de orden”

Esto se utiliza cuando queremos ordenar varias columnas. Por ejemplo, si con la tabla ‘empleados’ queremos ordenar primero por su precio en forma ASC, y luego por su nombre en forma DESC “SELECT * FROM empleados ORDER BY precio ASC, nombre ASC;”:

1	SELECT * FROM empleados ORDER BY salario ASC, nombre ASC;	ID	NOMBRE	SALARIO
		1	Juan Pérez	4800
		5	Luis Rodríguez	4800
		4	Ana Martínez	5500
		2	María López	5500
		3	Pedro García	5500

3.6 ORDENANDO “Filas con combinaciones de orden ASC y DESC”

Supongamos que queremos obtener una lista de todos los productos cuyo precio sea mayor a \$100 y ordenarlos primero por su precio de forma descendente y luego por su nombre de forma ascendente. Podemos hacer esto utilizando la siguiente consulta “SELECT * FROM productos WHERE precio > 100 ORDER BY precio DESC, nombre ASC;”

4. LIMITAR RESULTADOS

4.1 LIMITANDO “Cantidad de resultados”

Aprenderemos una nueva cláusula llamada LIMIT, que limita los resultados. Es útil cuando queremos ver cierta cantidad de registros con la condición. Por ejemplo si queremos ver los 3 primeros usuarios de la tabla ‘usuarios’ según el id “SELECT * FROM usuarios ORDER BY id LIMIT 3;”:

1	SELECT * FROM usuarios ORDER BY id LIMIT 3;	ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
		1	Juan	Pérez	juanperez@gmail.com	555-1234
		2	María	García	mariagarcia@hotmail.com	555-5678
		3	Pedro	López	pedrolopez@yahoo.com	555-9876

4.2 LIMITANDO “Alumnos con mejor nota”

La combinación de las cláusulas ORDER BY y LIMIT nos permiten obtener el valor o valores máximos de una columna. Por ejemplo “SELECT puntaje FROM puntajes ORDER BY puntaje DESC LIMIT 1;”:

1	SELECT * FROM puntajes ORDER BY puntaje DESC LIMIT 1;	ID	PUNTAJE
		1	100

5. OPERACIONES CON STRINGS

5.1 TRANSFORMANDO “String a MAYÚSCULAS”

Veremos ahora la función UPPER(). Sirve nada más y nada menos que para poner todo en MAYUSCULAS. Se usa tal que “SELECT UPPER(email) AS email_upper FROM usuarios;”:

1	SELECT UPPER(email) AS email_upper FROM usuarios;	EMAIL_UPPER
		JUANPEREZ@GMAIL.COM
		MARIAGARCIA@HOTMAIL.COM
		PEDROLOPEZ@YAHOO.COM
		LUCIASANCHEZ@OUTLOOK.COM
		JORGEMARTINEZ@GMAIL.COM

5.2 TRANSFORMANDO “String a MINUSCULAS”

Lo mismo que anterior pero al revés, pasa todo a MINUSCULAS, usaremos la función LOWER(). Como “SELECT LOWER(email) AS email_lower FROM usuarios;”:

1	SELECT LOWER(email) AS email_lower FROM usuarios;	EMAIL_LOWER
		juan@example.com
		maria@example.com
		pedro@example.com
		ana@example.com
		luis@example.com

5.3 QUITANDO “Espacios en blanco en Strings”

Usaremos la función TRIM(). Esta sirve para quitar los espacios al inicio o final del String. Se usa “SELECT TRIM(email) FROM usuarios;”:

<pre>1 SELECT TRIM(nombre), TRIM(email) FROM usuarios;</pre>	TRIM(NOMBRE)	TRIM(EMAIL)
	Juan	juanperez@gmail.com
	María	mariagarcia@hotmail.com
	Pedro	pedrolopez@yahoo.com
	Lucía	luciasanchez@outlook.com
	Jorge	jorgemartinez@gmail.com

5.4 COMBINANDO “Funciones”

Se pueden combinar varias funciones a la vez, así como “SELECT UPPER(TRIM(nombre)) AS nombre_limpio, TRIM(email) FROM usuarios;”.

5.5 OBTENIENDO “El largo de un String (LENGTH)”

También existe la función LENGTH, que sirve para obtener cuando mide un String. Se usa tal que “SELECT LENGTH(apellido) FROM usuarios;”:

<pre>1 SELECT LENGTH(apellido) FROM usuarios;</pre>	LENGTH(APELLIDO)
	5
	6
	5
	7
	8

5.6 OBTENIENDO “El nombre más largo de la tabla”

Esto se hace metiendo en la misma consulta, el LENGTH(), y un ORDER BY combinado con LIMIT, para que nos lo ordene por nombre y nos muestre solo uno, el código queda tal que “SELECT LENGTH(email) FROM usuarios ORDER BY email LIMIT 1;”:

```
1 SELECT LENGTH(email) AS email_cadena FROM usuarios ORDER BY LENGTH(email) DESC LIMIT 3;
```

EMAIL_CADENA

24

23

23

5.7 ORDENANDO “Datos y funciones”

También existe la posibilidad de que nos pidan mostrar el correo más largo junto a sus caracteres respectivos en el correo. Para esto usaremos lo aprendido **“SELECT email, LENGTH(email) FROM usuarios ORDER BY LENGTH(email) DESC LIMIT 3;”**:

```
1 SELECT email, LENGTH(email) FROM usuarios ORDER BY LENGTH(email) DESC LIMIT 3;
```

EMAIL	LENGTH(EMAIL)
-------	---------------

luciasanchez@outlook.com	24
--------------------------	----

mariagarcia@hotmail.com	23
-------------------------	----

jorgemartinez@gmail.com	23
-------------------------	----

5.8 CONCATENAR “Strings”

Tiene nombre feo, pero es muy fácil, la función se hace con el operador ‘||’. Esta función sirve para juntar dos columnas con caracteres en medio si queremos. Esta se usa **“SELECT producto || ‘-’ || marca AS producto_marca FROM productos;”**. El ‘-’ indica que entre medias va a ir un guión, también podríamos poner solo un espacio. Aquí juntaremos las columnas marca y producto de la tabla productos, queda así:

```
1 SELECT producto || '-' || marca AS marca_producto FROM productos;
```

MARCA_PRODUCTO

Camiseta-Nike

Zapatos-Adidas

Pantalón-Levis

Gorra-New Era

Bolso-Michael Kors

5.9 SELECCIONANDO “Caracteres de un String (SUBSTR)”

Este se usa con SELECT también, este tiene 3 argumentos. “SUBSTR(string, inicio, largo)”. 1. String: el nombre de la columna o palabra que será utilizada. 2. Inicio: un integer que especifica la posición de inicio desde la cual se extraerán caracteres al String. 3. Largo: la cantidad de caracteres extraídos. Entonces, si queremos las primeras letras de los apellidos de la tabla de usuarios sería así “SELECT SUBSTR (apellido, 1, 3) AS primeras_letras FROM usuarios;”:

<pre>1 SELECT SUBSTR(apellido, 1, 3) AS primeras_letras FROM usuarios;</pre>	PRIMERAS_LETRAS
	Pér
	Gar
	Lóp
	Sán
	Mar

5.10 SELECCIONANDO “Caracteres”

Se tiene una tabla de usuarios con las columnas nombre y apellido. Utilizando la función SUBSTR(), selecciona 3 caracteres del apellido de María, partiendo desde el segundo carácter. Asigna el alias 'tres_caracteres_del_apellido' a la columna creada. Para esto haremos lo siguiente “SELECT SUBSTR(apellido, 2, 3) AS tres_caracteres_del_apellido FROM usuarios WHERE nombre = 'María';”. Usaríamos el WHERE para especificar el apellido de ‘María’:

<pre>1 SELECT SUBSTR(apellido, 2, 3) AS tres_caracteres_del_apellido FROM usuarios WHERE nombre = 'María';</pre>
--

TRES_CARACTERES_DEL_APELLIDO
arc

6. ORDENAR RESULTADOS

6.1 OBTENIENDO 'La fecha de hoy'

Con la función DATE() podemos saber la fecha de hoy, esta se usa con WHERE para saber todos los registros de hoy. Si queremos indicar a la función la fecha de hoy y ponerla como fecha limite de una tarea ponemos "SELECT * FROM tareas WHERE fecha_limite = DATE('now');":

<pre>1 SELECT descripcion FROM tareas WHERE fecha_limite = DATE('now');</pre>	DESCRIPCION
	Tarea 1
	Tarea 2
	Tarea 3
	Tarea 4

6.2 OBTENIENDO 'La fecha de hoy'

En SQL también se pueden sumar fechas metiendo otro argumento en la misma función, es decir, que puede indicar fechas futuras, metiendo en los argumentos DATE('now', '2 week'). Con esto indicaríamos desde hoy hasta pasadas 2 semanas.

<pre>1 SELECT * FROM tareas WHERE fecha_limite = DATE('now', '1 day');</pre>	ID	DESCRIPCION	FECHA_LIMITE
	1	Tarea 1	2025-01-30
	2	Tarea 2	2025-01-30
	3	Tarea 3	2025-01-30

6.3 OBTENIENDO 'La fecha de ayer'

Algo similar a lo anterior, solo que el segundo argumento irá en negativo, es decir, DATE('now', '-1 week'):

<pre>1 SELECT monto FROM ganancias WHERE fecha = DATE('now', '-1 day');</pre>	MONTO
	75
	90

6.4 OBTENIENDO 'Extraer año de transacción'

Para extraer el año de una transacción vamos a usar la siguiente función, será como crear una nueva columna, miralo así. Imagina que nos piden info de una tabla y agregar una columna con su año de venta `"SELECT *, strftime('%Y', fecha_venta) AS año_venta FROM ventas;"`

ID_VENTA	MONTO	FECHA_VENTA	AÑO_VENTA
1	200	2010-01-15	2010
2	150	2011-02-20	2011
3	300	2012-03-10	2012
4	250	2013-04-05	2013
5	100	2014-05-25	2014
6	350	2015-06-18	2015
7	400	2015-07-22	2015
8	180	2015-08-09	2015
9	220	2018-09-30	2018
10	275	2019-10-11	2019

6.5 OBTENIENDO 'Extraer mes de transacción'

Lo mismo que lo anterior, solo que ahora podemos fusionarlo y tenemos un pequeño cambio, la función sera `strftime('%m')`. Podemos combinarlo cogiendo el mes y año de transacción, usando esto:

```
1 SELECT monto, strftime('%m', fecha_venta) AS mes_venta, strftime('%Y', fecha_venta) AS año_venta FROM ventas;
```

MONTO	MES_VENTA	AÑO_VENTA
200	01	2010
150	02	2011
300	03	2012
250	04	2013
100	05	2014
350	06	2015
400	07	2015
180	08	2015
220	09	2018

O tambien lo podemos combinar de la siguiente manera `strftime('%m-%Y')`:

```
1 SELECT monto, strftime('%m-%Y', fecha_venta) AS mes_año FROM ventas;
```

MONTO	MES_AÑO
200	01-2010
150	02-2011
300	03-2012

6.6 OBTENIENDO 'Extracciones con WHERE'

Hasta ahora, hemos utilizado el strftime para SELECT, ahora vamos a usarlo para buscar por alguna fecha específica, que es lo mismo solo que en WHERE. "SELECT * FROM ventas WHERE strftime('%Y', fecha_venta) = '2015';":

1	SELECT * FROM ventas WHERE strftime('%Y', fecha_venta) = '2015';	ID_VENTA	MONTO	FECHA_VENTA
		6	350	2015-06-18
		7	400	2015-07-22
		8	180	2015-08-09

7. FUNCIONES DE AGREGACIÓN

7.1 MAYOR 'Valor de una columna'

Con esta función conseguimos sacar al valor más alto de la columna, se usa "SELECT MAX(N.Columna) FROM empleados;":

1	SELECT MAX(edad) FROM empleados;	MAX(EDAD)
		40

7.2 MENOR 'Valor de una columna'

Misma funcionalidad que antes pero cambiamos el MAX por el MIN para sacar el valor más bajo de la columna:

1	SELECT MIN(sueldo) FROM empleados;	MIN(SUELDO)
		45000

7.3 SUMA 'Elementos de una columna'

En este apartado veremos la función SUM(), que permite sumas todos los valores de una columna, "SELECT SUM(sueldo) FROM empleados;":

1	SELECT SUM(sueldo) FROM empleados;	SUM(SUELDO)
		210000

7.4 PROMEDIO 'De una columna'

Para la media de una columna usaremos la función AVG(), está te muestra la media de toda la columna indicada, se usa "SELECT AVG(sueldo) FROM empleados;":

```
1 SELECT AVG(sueldo) FROM empleados;
```

AVG(SUELDO)
52500

7.5 CONTANDO 'Elementos de una columna'

Para contar los elementos usaremos COUNT(). Contará cada uno de los elementos en la columna "SELECT COUNT(*) FROM empleados;", con esto nos mostrará la cantidad de filas de la tabla:

```
1 SELECT COUNT(*) FROM empleados;
```

COUNT(*)
4

7.6 EJERCICIOS

Utilizando la tabla empleados, calcula la suma de sueldos de todas las personas mayores a 27 años:

```
1 SELECT SUM(sueldo) FROM empleados WHERE edad > 27;
```

SUM(SUELDO)
110000

Utilizando la tabla empleados, calcula el promedio de los sueldos de todas las personas que ganan más de 50,000:

```
1 SELECT AVG(sueldo) FROM empleados WHERE sueldo > 50000;
```

AVG(SUELDO)
57500

Calcula cuantas personas trabajan en el área de marketing:

```
1 SELECT COUNT(*) FROM empleados WHERE departamento = 'Marketing';
```

COUNT(*)
2

Calcula cuantas personas trabajan en total en las areas de finanzas y marketing:

1 SELECT COUNT(*) FROM empleados WHERE departamento = 'Finanzas' OR departamento = 'Marketing';	
COUNT(*)	
4	

Cuenta la cantidad de usuarios cuyo nombre termina con la letra 'a' en la tabla de usuarios:

1 SELECT COUNT(*) FROM usuarios WHERE nombre LIKE '%a';	
COUNT(*)	
2	

8. DISTINCT

8.1 SELECCIONANDO 'Filtros de datos repetidos'

La función DISTINCT sirve para mostrar una única vez los valores repetidos en una columna, por ejemplo si queremos sacar los colores unicos de una columna sin ver los repetidos usamos "SELECT DISTINCT color AS color_unico FROM colores;":

1 SELECT DISTINCT color AS color_unico FROM colores;		COLOR_UNICO
		Rojo
		Azul
		Verde
		Amarillo
		Naranja
		Morado
		Rosa
		Café
		Gris
		Negro
		Blanco

8.2 SELECCIONANDO 'Correos únicos'

Crea una consulta que nos muestre cada correo una única vez. La columna mostrada debe llamarse:

<pre>1 SELECT DISTINCT correo AS correo_unico FROM usuarios;</pre>	CORREO_UNICO
	juan.perez@empresa.com
	maria.gonzalez@empresa.com
	carlos.rodriguez@empresa.com
	ana.martinez@empresa.com
	luis.garcia@empresa.com
	carmen.lopez@empresa.com
	jose.hernandez@empresa.com
	francisco.martin@empresa.com
	laura.sanchez@empresa.com
	antonio.diaz@empresa.com

8.3 SELECCIONANDO 'Distintos años'

Como aprendimos en ejercicios anteriores, usaremos la función de obtener la fecha de un valor y lo combinaremos con DISTINCT para obtener un año único. "SELECT DISTINCT strftime('%m', fecha_venta) AS mes_unico FROM ventas;":

<pre>1 SELECT DISTINCT strftime('%m', fecha_venta) AS mes_unico FROM ventas;</pre>	MES_UNICO
	01
	02
	04
	06
	07

8.4 SELECCIONANDO 'Valores distintos'

Por ejemplo DISTINCT se puede combinar con la función COUNT(DISTINCT columna). Así podemos sacar los valores únicos y contarlos "SELECT COUNT(DISTINCT telefono) AS telefonos_unicos FROM usuarios;":

<pre>1 SELECT COUNT(DISTINCT telefono) AS telefonos_unicos FROM usuarios;</pre>	TELEFONOS_UNICOS
	3

8.5 CONTANDO 'Correos unicos'

Crea una consulta para contestar cuantos correos únicos existen en la tabla. La columna resultante debe llamarse correos_cant:

1	<code>SELECT COUNT(DISTINCT correo) AS correos_cant FROM usuarios;</code>	CORREOS_CANT
		10

8.6 DISTINCT 'Con multiples columnas'

Usaremos el DISTINCT normal pero solo que ahora añadiremos más columnas para que sean únicas. "SELECT DISTINCT categoria, precio FROM productos;":

1	<code>SELECT DISTINCT categoria, precio FROM productos;</code>	CATEGORIA	PRECIO
		Electrónica	1000
		Electrónica	500
		Ropa	20
		Ropa	40
		Electrónica	50
		Libros	15
		Accesorios	30

9. INTRODUCCION A GRUPOS

9.1 AGRUPANDO 'Valores con GROUP BY'

GROUP BY, es una función muy poderosa de SQL ya que podemos agrupar filas idénticas en una o varias columnas. Pero vamos a ver como obtener los valores que todos sean distintos, ya lo hemos hecho anteriormente pero ahora lo haremos con GROUP BY "SELECT correo AS correo_unico FROM usuarios GROUP BY correo;":

1	<code>SELECT correo AS correo_unico FROM usuarios GROUP BY correo;</code>	CORREO_UNICO
		ana.martinez@empresa.com
		antonio.diaz@empresa.com
		carlos.rodriguez@empresa.com
		carmen.lopez@empresa.com
		francisco.martin@empresa.com

9.2 AGRUPANDO 'Y contando'

GROUP BY es muy utilizado con las funciones SUM(), MAX(), MIN(), AVG() y COUNT(), para obtener información específica de datos. Vamos a contar cuantas veces se repite un valor:

```
1 SELECT correo, COUNT(correo) AS repeticiones FROM usuarios GROUP BY correo;
```

CORREO	REPETICIONES
ana.martinez@empresa.com	1
antonio.diaz@empresa.com	1
carlos.rodriguez@empresa.com	1
carmen.lopez@empresa.com	1
francisco.martin@empresa.com	1
jose.hernandez@empresa.com	1
juan.perez@empresa.com	2
laura.sanchez@empresa.com	1
luis.garcia@empresa.com	1
maria.gonzalez@empresa.com	2

9.3 AGRUPANDO 'Y sumando'

Vamos a hacer lo mismo que antes pero con SUM() para sumar todos los valores de una columna:

```
1 SELECT categoria, SUM(monto) AS monto_total FROM ventas GROUP BY categoria;
```

CATEGORIA	MONTO_TOTAL
Accesorios	390
Electrodomésticos	170
Electrónicos	3250
Libros	35
Mobiliario	435
Ropa	240

9.4 AGRUPANDO 'Y sacando el promedio'

Vamos a sacar la media de nota de los 3 trimestres de los alumnos con la siguiente sentencia

```
1 SELECT nombre_completo, AVG(nota) AS Promedio_Notas FROM estudiantes GROUP BY nombre_completo;
```

NOMBRE_COMPLETO	PROMEDIO_NOTAS
Ana Fernández	8.333333333333334
Carlos García	6
Juan Pérez	7
Luis Morales	6
María Rodríguez	8

9.5 AGRUPANDO 'Y sacando el máximo por grupo'

Sintaxis es la misma de antes, vamos a calcular el monto mas alto por cada categoría:

```
1 SELECT categoria, MAX(monto) AS monto_mas_alto FROM ventas GROUP BY categoria;
```

CATEGORIA	MONTO_MAS_ALTO
Accesorios	250
Electrodomésticos	110
Electrónicos	1200
Libros	20
Mobiliario	150
Ropa	100

9.6 AGRUPANDO 'Y sacando el mínimo por grupo'

Crea una consulta para calcular el monto más bajo por cada categoría:

```
1 SELECT categoria, MIN(monto) AS monto_mas_bajo FROM ventas GROUP BY categoria;
```

CATEGORIA	MONTO_MAS_BAJO
Accesorios	70
Electrodomésticos	60
Electrónicos	50
Libros	15
Mobiliario	45
Ropa	40

9.7 AGRUPANDO 'Y agregando fechas'

Vamos a usar la función para obtener la fecha strftime(%y, columna), combinandola con el group by, así obtendremos el monto total por año/mes/dia:

```
1 SELECT SUM(monto) AS suma_ventas, strftime('%m', fecha_venta) AS mes FROM ventas GROUP BY strftime('%m', fecha_venta);
```

SUMA_VENTAS	MES
200	01
450	02
700	04
580	06
220	09
275	10

9.8 AGRUPANDO 'Sin indicar el nombre de las columnas'

Hay una forma de indicar las cláusulas del SELECT en GROUP BY y ORDER BY, y es de forma numérica. Crea una consulta que nos muestre cada correo una única vez acompañado del número de veces que se repite. Las columnas deben llevar los nombres "correo" y "repeticiones", respectivamente, y deben estar ordenadas alfabéticamente:

```
1 SELECT correo, COUNT(correo) AS repeticiones FROM usuarios GROUP BY 1 ORDER BY 1;
```

CORREO	REPETICIONES
ana.martinez@empresa.com	1
carlos.rodriguez@empresa.com	1
carmen.lopez@empresa.com	2
jose.hernandez@empresa.com	1
juan.perez@empresa.com	3
luis.garcia@empresa.com	1
maria.gonzalez@empresa.com	3

9.9 AGRUPANDO 'Por múltiples columnas'

En SQL se pueden llamar a varias columnas a la vez, entonces pongámoslo en práctica. Calcula el promedio de cada estudiante en cada materia. Las columnas deben llamarse correo, materia y promedio_notas:

```
1 SELECT correo, materia, AVG(nota) AS promedio_notas FROM estudiantes GROUP BY 1, 2;
```

CORREO	MATERIA	PROMEDIO_NOTAS
estudiante1@ejemplo.com	Ciencias	8
estudiante1@ejemplo.com	Historia	8.7
estudiante1@ejemplo.com	Matemáticas	8.5
estudiante2@ejemplo.com	Ciencias	9.5
estudiante2@ejemplo.com	Historia	9.2
estudiante2@ejemplo.com	Matemáticas	9
estudiante3@ejemplo.com	Ciencias	7
estudiante3@ejemplo.com	Historia	7.8
estudiante3@ejemplo.com	Matemáticas	7.5

10. HAVING

10.1 HAVING 'Having'

El HAVING es una cláusula similar al WHERE, cuando queremos tener una cantidad específica de reportes en un mes, se usa HAVING para indicarlo:

```
1 SELECT strftime('%m', fecha_inscripcion) AS mes, COUNT(fecha_inscripcion) AS cantidad_usuarios
2 FROM inscripciones
3 GROUP BY mes
4 HAVING cantidad_usuarios = 1;
```

MES	CANTIDAD_USUARIOS
11	1
12	1

10.2 HAVING 'Buscando duplicados'

El HAVING se usa mucho para la búsqueda de duplicados, es decir, para los que hay mas de uno. Muestra los correos que aparezcan en más de una ocasión. La tabla resultante debe tener dos columnas: una llamada correo, y otra llamada cuenta_correos que muestra la cantidad de repeticiones correspondiente a cada correo:

<pre>1 SELECT correo, COUNT(correo) AS cuenta_correos 2 FROM correos_corporativos 3 GROUP BY correo 4 HAVING cuenta_correos > 1;</pre>	CORREO	CUENTA_CORREOS
	juan.perez@empresa.com	2
	maria.gonzalez@empresa.com	2

10.3 HAVING 'Contando'

Crea una consulta que muestre la cantidad de usuarios y el departamento en donde haya más de un empleado. Las columnas deben llamarse cantidad_de_usuarios y departamento, respectivamente:

<pre>1 SELECT COUNT(nombre) AS cantidad_de_usuarios, departamento 2 FROM empleados 3 GROUP BY departamento 4 HAVING cantidad_de_usuarios > 1;</pre>	CANTIDAD_DE_USUARIOS	DEPARTAMENTO
	2	Finanzas
	2	Marketing
	2	Tecnología
	2	Ventas

10.4 HAVING 'Y promedio'

Crea una consulta para determinar cuales son los estudiantes que aprobaron. El criterio de aprobación es promedio de notas ≥ 50 . Las columnas a mostrar deben ser email y promedio_notas:

<pre>1 SELECT email, AVG(notas) AS promedio_notas 2 FROM notas 3 GROUP BY email 4 HAVING promedio_notas >= 50;</pre>	EMAIL	PROMEDIO_NOTAS
	Alumno1@ejemplo.com	56.666666666666664
	Alumno2@ejemplo.com	53.333333333333336

10.5 HAVING 'Y order'

Dada la siguiente tabla ventas, escribe una consulta SQL para obtener los productos que se han vendido en una cantidad total mayor a 1000, ordenados en orden descendente de cantidad vendida:

```
1 SELECT producto, SUM(cantidad) AS cantidad_total
2 FROM ventas
3 GROUP BY producto
4 HAVING cantidad_total > 1000
5 ORDER BY cantidad_total DESC;
```

PRODUCTO	CANTIDAD_TOTAL
B	2800
D	1900
C	1500
A	1100

11. SUBCONSULTAS

11.1 SUBCONSULTAS

```
1 SELECT * FROM empleados WHERE sueldo <= (SELECT AVG(sueldo) FROM empleados);
```

NOMBRE	APELLIDO	SUELDO	DEPARTAMENTO
Juan	Pérez	3000	Ventas
Ana	Martínez	2800	Recursos Humanos
Luis	García	3200	Finanzas
Carmen	López	3100	Administración
José	Hernández	2900	Operaciones
Sofía	Ruiz	2750	Ventas
Elena	Castro	3050	Marketing
Pedro	Ortega	3150	Finanzas

11.2 SUBCONSULTAS 'Con WHERE'

Selecciona toda la información de los registros que sean mayores al promedio del departamento de finanzas:

```
1 SELECT * FROM empleados WHERE sueldo > (SELECT AVG(sueldo) FROM empleados WHERE departamento = 'Finanzas');
```

NOMBRE	APELLIDO	SUELDO	DEPARTAMENTO
María	González	3500	Marketing
Carlos	Rodríguez	4000	Tecnología
Luis	García	3200	Finanzas
Francisco	Martín	3400	Legal
Laura	Sánchez	3300	Compras
Antonio	Díaz	3600	Producción
Jorge	Vargas	3900	Tecnología

11.3 SUBCONSULTAS 'Con IN'

Se tiene una tabla `estudiantes` y otra tabla con promedios. Muestra los nombres de todos los estudiantes que tengan un promedio de notas sobre 50:

```
1 SELECT nombre
2 FROM estudiantes
3 WHERE estudiante_id IN (SELECT estudiante_id FROM promedios WHERE promedio_notas > 50);
```

NOMBRE
Juan
María

11.4 SUBCONSULTAS 'Con FROM'

Las subconsultas, también conocidas como "subqueries", nos permiten utilizar los resultados de una consulta dentro de otra consulta. En los ejercicios anteriores utilizamos las subconsultas dentro de la cláusula `WHERE`, pero también es posible utilizarlas dentro de otras cláusulas. En este ejercicio abordaremos como utilizarla dentro de `FROM`.

Una subconsulta en el `FROM` tiene la siguiente forma.

```
SELECT *
FROM (
    SELECT * FROM tabla1
)
```

En este caso no parece tan útil ya que simplemente estamos seleccionando lo mismo, pero veamos un caso donde sí sería necesario.

Se tiene la tabla `ventas` que tiene el código de vendedor y el monto de cada venta realizada. Nos piden saber cuanto es el promedio total vendido.

EMPLEADO_ID	MONTO
1	100
1	150
2	200
2	250
3	300
3	350
4	400

Para esto primero necesitamos sumar los montos por vendedor y luego sobre *estos resultados* sacamos el promedio de las ventas.

```
SELECT AVG(total_venta) as promedio_ventas
FROM (
    SELECT empleado_id, SUM(monto) as total_venta
    FROM ventas
    GROUP BY empleado_id
)
```

```
1 SELECT AVG(monto_total) AS promedio_ventas
2 FROM (
3     SELECT SUM(monto) AS monto_total
4     FROM ventas
5     GROUP BY empleado_id
6 );
```

PROMEDIO_VENTAS

437.5

12. SUBCONSULTAS

12.1 INTRODUCCION 'Cláusula UNION'

Introducción a la cláusula unión de SQL

El operador **UNION** en SQL se utiliza para combinar el resultado de dos o más SELECT en un solo conjunto de resultados.

La sintaxis básica de UNION es la siguiente:

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

Las columnas que se seleccionan en los SELECT deben tener los mismos nombres de columna, secuencia y tipos de datos.

Veamos un ejemplo:

Supongamos que tenemos dos tablas: 'Estudiantes' y 'Profesores', que contienen una lista de apellidos en cada una. Queremos crear una lista que combine los apellidos de ambas tablas.

Estudiantes

ID	NOMBRE	APELLIDO
1	Juan	Rodríguez
2	María	Sánchez
3	Pedro	Castillo

Profesores

ID	NOMBRE	APELLIDO
1	Alberto	Vargas
2	Carla	Garrido
3	Diego	Mendoza

Al hacer la consulta:

```
SELECT apellido
FROM Estudiantes
UNION
SELECT apellido
FROM Profesores;
```

Nos daría el resultado:

APELLIDO
Rodríguez
Sánchez
Castillo
Vargas
Garrido
Mendoza

```
1 SELECT nombre AS nombres
2 FROM estudiantes
3 UNION
4 SELECT nombre AS nombres
5 FROM profesores;
```

NOMBRES
Ana
Carlos
Juan
Luis
María
Pedro

12.2 INTRODUCCION 'Cláusula UNION para eliminar duplicados'

1	SELECT email AS correos_unicos	CORREOS_UNICOS
2	FROM usuarios	jorgemartinez@gmail.com
3	GROUP BY email	juanperez@gmail.com
4	UNION	luciasanchez@outlook.com
5	SELECT email AS correos_unicos	mariagarcia@hotmail.com
6	FROM usuarios	pedrolopez@yahoo.com
7	GROUP BY email;	

12.3 INTRODUCCION 'Cláusula UNION vs UNION ALL'

tabla1

NOMBRE	EDAD
Juan	30
Maria	25
Carlos	40

tabla2

NOMBRE	EDAD
Juan	30
Luis	30
Carmen	25

Observa que Juan está en ambas tablas.

Podemos combinar ambas tablas utilizando UNION ALL de la siguiente forma:

```
SELECT * FROM tabla1 UNION ALL SELECT * FROM tabla2;
```

Como resultado obtendremos:

NOMBRE	EDAD
Juan	30
Maria	25
Carlos	40
Juan	30
Luis	30
Carmen	25

1	SELECT *	NOMBRE	APELLIDO	EDAD
2	FROM empleados1	Juan	Pérez	30
3	UNION ALL	Maria	González	25
4	SELECT *	Carlos	Rodríguez	40
5	FROM empleados2;	Ana	Martínez	22
		Maria	González	25
		Carmen	López	25

12.4 INTRODUCCION 'A la intersección'

Introducción a intersección

El operador **INTERSECT** se utiliza para combinar dos SELECT y devolver los resultados que se encuentran en ambas consultas.

Por ejemplo, si tenemos las siguientes dos tablas, clientes1 y clientes2:

Tabla **clientes1**:

NOMBRE
Juan
Maria
Carlos
Ana
Luis

Tabla **clientes2**:

NOMBRE
Ana
Luis
Pedro
Carmen
Juan

Podemos encontrar los clientes en común utilizando INTERSECT de la siguiente forma:

```
SELECT nombre FROM clientes1 INTERSECT SELECT nombre FROM clientes2
```

Como resultado obtendremos:

NOMBRE
Ana
Juan
Luis

1	SELECT *	CLIENTE
2	FROM lista1	Ana
3	INTERSECT	Carmen
4	SELECT *	Juan
5	FROM lista2;	Luis
		Maria
		Pedro

12.5 INTRODUCCION 'Operador EXCEPT'

1	SELECT nombre	NOMBRE
2	FROM empleados	Ana
3	EXCEPT	Carlos
4	SELECT nombre	Luis
5	FROM gerentes;	

13. INSERCIÓN DE REGISTROS

13.1 AÑADIR 'Registro en una tabla'

```
1 INSERT INTO usuarios VALUES (7, 'Lucía', 'Sánchez', 'luciasanchez@outlook.com', '555-5555')
```

ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
1	Juan	Pérez	juanperez@gmail.com	555-1234
2	María	García	mariagarcia@hotmail.com	555-5678
3	Pedro	López	pedrolopez@yahoo.com	555-9876
4	Jorge	Martínez	jorgemartinez@gmail.com	555-4321
7	Lucía	Sánchez	luciasanchez@outlook.com	555-5555

13.2 ESPECIFICANDO 'Valores nulos'

ID	NOMBRE	PRECIO	STOCK
1	Camisa	50	20
2	Pantalón	80	15
3	Zapatos	120	10
4	Sombrero	30	5
7	Bolso	1000	

```
INSERT INTO productos VALUES (7, 'Bolso', 1000, NULL);
```

13.3 ESPECIFICANDO 'Columnas'

```
INSERT INTO usuarios (id, nombre, apellido, email, telefono) VALUES (7, 'Lucía', 'Sánchez', 'luciasanchez@outlook.com', '333-3333');
```

ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
1	Juan	Pérez	juanperez@gmail.com	555-1234
2	María	García	mariagarcia@hotmail.com	555-5678
3	Pedro	López	pedrolopez@yahoo.com	555-9876
4	Jorge	Martínez	jorgemartinez@gmail.com	555-4321
7	Lucía	Sánchez	luciasanchez@outlook.com	333-3333

13.3 ESPECIFICANDO 'Columnas (Solo algunas)'

```
INSERT INTO productos (nombre, stock) VALUES  
('Bolso', 10)
```

NOMBRE	PRECIO	STOCK
Gorro	1000	5
Camiseta	500	10
Pantalón	1500	8
Zapatos	2000	3
Bufanda	800	12
Bolso		10

13.4 AÑADIR 'Fecha de hoy'

```
INSERT INTO productos (nombre, stock, fecha) VALUES  
('Bolso', 10, CURRENT_DATE)
```

NOMBRE	PRECIO	STOCK	FECHA
Gorro	1000	5	2025-02-18
Camiseta	500	10	2025-02-18
Pantalón	1500	8	2025-02-18
Zapatos	2000	3	2025-02-18
Bufanda	800	12	2025-02-18
Bolso		10	2025-02-18

13.5 AÑADIR 'Fecha de hoy con formato'

```
INSERT INTO productos (nombre, precio, stock, fecha) VALUES  
('Bolso', NULL, 10, '2023-01-01')
```

NOMBRE	PRECIO	STOCK	FECHA
Gorro	1000	5	2025-02-18
Camiseta	500	10	2025-02-18
Pantalón	1500	8	2025-02-18
Zapatos	2000	3	2025-02-18
Bufanda	800	12	2025-02-18
Bolso		10	2023-01-01

13.6 AÑADIR 'Multiples valores'

```
INSERT INTO ventas (producto, cantidad, precio) VALUES  
('Gorro', 5, 1000),  
('Camiseta', 10, 500),  
('Pantalón', 8, 1500);
```

PRODUCTO	CANTIDAD	PRECIO
Gorro	5	1000
Camiseta	10	500
Pantalón	8	1500

13.7 CAMPO 'Autoincremental'

```
CREATE TABLE empleados (id INTEGER PRIMARY KEY AUTOINCREMENT, nombre TEXT, apellido TEXT);
```

```
INSERT INTO empleados (nombre, apellido) VALUES ('Jane', 'Smith');
```

ID	NOMBRE	APELLIDO
1	Jane	Smith

13.8 AÑADIR 'Registro asumiendo un valor por defecto'

```
CREATE TABLE Productos (ID INTEGER PRIMARY KEY AUTOINCREMENT, Nombre TEXT, Precio INTEGER DEFAULT 10);
```

```
INSERT INTO usuarios (nombre, apellido, email) VALUES ('Lucía', 'Sánchez', 'luciasanchez@outlook.com');
```

ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
1	Juan	Pérez	juanperez@gmail.com	555-1234
2	María	García	mariagarcia@hotmail.com	555-5678
3	Pedro	López	pedrolopez@yahoo.com	555-9876
4	Jorge	Martínez	jorgemartinez@gmail.com	555-4321
5	Lucía	Sánchez	luciasanchez@outlook.com	111-1111

14. BORRADO Y MODIFICACIÓN DE REGISTROS

14.1 BORRA 'Todos los registros de una tabla'

```
DELETE FROM productos;
```

14.2 BORRA 'Un registro con WHERE'

```
DELETE FROM usuarios  
WHERE id = 2;
```

ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
1	Juan	Pérez	juanperez@gmail.com	555-1234
3	Pedro	López	pedrolopez@yahoo.com	555-9876
4	Lucía	Sánchez	luciasanchez@outlook.com	555-5555
5	Jorge	Martínez	jorgemartinez@gmail.com	555-4321

14.3 EDITA 'Registros'

<pre>UPDATE usuarios SET registrado = TRUE;</pre>	ID	NOMBRE	APELLIDO	EMAIL	REGISTRADO
	1	Juan	Pérez	juanperez@gmail.com	1
	2	María	García	mariagarcia@hotmail.com	1
	3	Pedro	López	pedrolopez@yahoo.com	1
	4	Lucía	Sánchez	luciasanchez@outlook.com	1
	5	Jorge	Martínez	jorgemartinez@gmail.com	1

14.4 EDITA 'Registros usando WHERE'

<pre>UPDATE usuarios SET telefono = '123-456' WHERE id = 4;</pre>				
ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
1	Juan	Pérez	juanperez@gmail.com	555-1234
2	María	García	mariagarcia@hotmail.com	555-5678
3	Pedro	López	pedrolopez@yahoo.com	555-9876
4	Lucía	López	luciasanchez@outlook.com	123-456
5	Jorge	Martínez	jorgemartinez@gmail.com	555-4321

14.5 EDITA 'Múltiples columnas'

<pre>UPDATE posts SET titulo = "Aprendiendo SQL", contenido = "SQL es un lenguaje de programación para gestionar bases de datos relacionales" WHERE id = 1;</pre>				
ID	TITULO	CONTENIDO	AUTOR	FECHA
1	Aprendiendo SQL	SQL es un lenguaje de programación para gestionar bases de datos relacionales	Autor 1	2022-01-01
2	Post 2	Contenido del post 2	Autor 2	2022-01-02
3	Post 3	Contenido del post 3	Autor 3	2022-01-03
4	Post 4	Contenido del post 4	Autor 4	2022-01-04
5	Post 5	Contenido del post 5	Autor 5	2022-01-05

15. TABLAS

15.1 CREAR 'Tabla'

```
CREATE TABLE alumnos (nombre VARCHAR(50));  
  
INSERT INTO alumnos (nombre) VALUES  
('Lucía');
```

NOMBRE
Lucía

15.2 CREAR 'Tabla multiples columnas'

```
CREATE TABLE alumnos (nombre VARCHAR(50), apellido VARCHAR(50), telefono VARCHAR(50));  
  
INSERT INTO alumnos (nombre, apellido, telefono) VALUES  
('Lucía', 'Sánchez', '12345678');
```

NOMBRE	APELLIDO	TELEFONO
Lucía	Sánchez	12345678

15.3 CREAR 'Tabla multiples columnas distintos datos'

```
CREATE TABLE usuarios (nombre VARCHAR(50), apellido VARCHAR(50), edad INTEGER, activo boolean, nacimiento DATE);  
  
INSERT INTO usuarios (nombre, apellido, edad, activo, nacimiento) VALUES  
('Lucía', 'Sánchez', 25, TRUE, '1996-01-01');
```

NOMBRE	APELLIDO	EDAD	ACTIVO	NACIMIENTO
Lucía	Sánchez	25	1	1996-01-01

15.4 CREAR 'Tabla dato REAL'

```
CREATE TABLE temperaturas (temperatura_celsius REAL);  
  
INSERT INTO temperaturas (temperatura_celsius) VALUES  
(23.4),  
(26.5),  
(27.1);
```

TEMPERATURA_CELSIUS
23.4
26.5
27.1

15.5 BORRAR 'Tabla'

```
DROP TABLE temperaturas;

CREATE TABLE temperaturas(ciudad VARCHAR(50), temperatura REAL, fecha DATE);

INSERT INTO temperaturas(ciudad, temperatura, fecha) VALUES
('Buenos Aires', 20.0, '2024-01-01'),
('Buenos Aires', 21.0, '2024-01-02'),
('Santiago', 22.0, '2024-01-01'),
('Santiago', 23.0, '2024-01-02');
```

CIUDAD	TEMPERATURA	FECHA
Buenos Aires	20	2024-01-01
Buenos Aires	21	2024-01-02
Santiago	22	2024-01-01
Santiago	23	2024-01-02

15.6 ACTUALIZAR 'Tabla'

```
ALTER TABLE productos ADD COLUMN descripcion TEXT;

INSERT INTO productos(nombre, precio, descripcion) VALUES
('Camisa', 1000.00, 'Camisa de manga corta'),
('Pantalón', 2000.00, 'Pantalón de mezclilla'),
('Camisa XL', 1000.00, 'Camisa de manga larga');
```

NOMBRE	PRECIO	DESCRIPCION
Valor Antiguo	500	
Camisa	1000	Camisa de manga corta
Pantalón	2000	Pantalón de mezclilla
Camisa XL	1000	Camisa de manga larga

16. RESTRICCIONES

16.1 RESTRICCIONES

```
CREATE TABLE empleados (
    nombre VARCHAR(50) NOT NULL,
    apellido VARCHAR(50)
);

INSERT INTO empleados (nombre, apellido) VALUES
('Pedro', 'Pérez');
```

NOMBRE	APELLIDO
Pedro	Pérez

16.2 RESTRICCIONES 'A tablas existentes'

```
CREATE TABLE patentes2 (
    patente TEXT NOT NULL
);

INSERT INTO patentes2 (patente) VALUES
('ABC123'),
('ABD124');

DROP TABLE patentes;
ALTER TABLE patentes2 RENAME TO patentes;
```

SQL

CREATE TABLE "patentes" (patente TEXT NOT NULL)

16.3 RESTRICCIONES 'Borrar restricción'

```
CREATE TABLE personas2 (  
    nombre TEXT,  
    apellido TEXT,  
    edad INTEGER  
);  
  
DROP TABLE personas;  
ALTER TABLE personas2 RENAME TO personas;
```

SQL

CREATE TABLE "personas" (nombre TEXT, apellido TEXT, edad INTEGER)

16.4 RESTRICCIONES 'UNIQUE'

```
CREATE TABLE productos (  
    nombre TEXT NOT NULL,  
    precio REAL NOT NULL,  
    codigo TEXT UNIQUE  
);  
  
INSERT INTO productos(nombre, precio, codigo) VALUES  
( 'Camisa', 1000.00, 'CAM-001'),  
( 'Pantalón', 2000.00, 'PAN-001'),  
( 'Camisa XL', 1000.00, 'CAM-002');
```

NOMBRE	PRECIO	CODIGO
Camisa	1000	CAM-001
Pantalón	2000	PAN-001
Camisa XL	1000	CAM-002

16.5 RESTRICCIONES 'CHECK'

```
CREATE TABLE productos (  
    nombre TEXT NOT NULL,  
    precio REAL NOT NULL,  
    stock INTEGER CHECK(stock >= 0)  
);  
  
INSERT INTO productos (nombre, precio, stock) VALUES  
( 'Camisa', 1000.00, 10),  
( 'Pantalón', 2000.00, 5),  
( 'Camisa XL', 1000.00, 3);
```

NOMBRE	PRECIO	STOCK
Camisa	1000	10
Pantalón	2000	5
Camisa XL	1000	3

16.6 RESTRICCIONES 'Clave primaria'

Evita valores nulos y que el registro se repita

```
CREATE TABLE posts (  
    id INTEGER PRIMARY KEY,  
    title TEXT,  
    content TEXT  
);  
  
INSERT INTO posts (id, title, content) VALUES  
(1, 'Introducción', '¡Bienvenido al mundo de la programación!'),  
(2, 'Primeros pasos', 'Sumérgete en los conceptos básicos de la programación.'),  
(3, 'Temas Avanzados', 'Explora conceptos y técnicas avanzadas en programación.');
```

ID	TITLE	CONTENT
1	Introducción	¡Bienvenido al mundo de la programación!
2	Primeros pasos	Sumérgete en los conceptos básicos de la programación.
3	Temas Avanzados	Explora conceptos y técnicas avanzadas en programación.

16.7 RESTRICCIONES 'Autoincremental'

```
CREATE TABLE usuarios (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  nombre TEXT NOT NULL,  
  fecha_creacion DATE  
);  
  
INSERT INTO usuarios (nombre, fecha_creacion) VALUES  
(  
  'Ana', '2024-01-01',  
  'Gonzalo', '2024-01-02',  
  'Juan', '2024-01-03',  
  'Maria', '2024-01-04';
```

ID	NOMBRE	FECHA_CREACION
1	Ana	2024-01-01
2	Gonzalo	2024-01-02
3	Juan	2024-01-03
4	María	2024-01-04

16.8 RESTRICCIONES 'Autoincremental'

```
CREATE TABLE transacciones (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  monto REAL NOT NULL,  
  fecha DATE  
);  
  
INSERT INTO transacciones (monto, fecha) VALUES  
(  
  1000.00, '2024-01-01',  
  2000.00, '2024-01-02',  
  3000.00, '2024-01-03');  
INSERT INTO transacciones (id, monto, fecha) VALUES  
(  
  10, 4000.00, '2024-01-04');  
INSERT INTO transacciones (monto, fecha) VALUES  
(  
  5000.00, '2024-01-05';
```

ID	MONTO	FECHA
1	1000	2024-01-01
2	2000	2024-01-02
3	3000	2024-01-03
10	4000	2024-01-04
11	5000	2024-01-05

16.9 RESTRICCIONES 'Primary key y TEXT'

```
CREATE TABLE personas (  
  email TEXT PRIMARY KEY NOT NULL,  
  nombre TEXT,  
  apellido TEXT  
);  
  
INSERT INTO personas (email, nombre, apellido) VALUES  
(  
  'example1@example.com', 'John', 'Doe',  
  'example2@example.com', 'Jane', 'Smith',  
  'example3@example.com', 'Mike', 'Johnson');
```

EMAIL	NOMBRE	APELLIDO
example1@example.com	John	Doe
example2@example.com	Jane	Smith
example3@example.com	Mike	Johnson

16.10 RESTRICCIONES 'Clave FOREIGN (Foránea)'

personas		
COLUMNA	TIPO DE DATO	RESTRICCIONES
id	INTEGER	PRIMARY KEY
nombre	TEXT	
apellido	TEXT	
autos		
COLUMNA	TIPO DE DATO	RESTRICCIONES
id	INTEGER	PRIMARY KEY
patente	TEXT	
persona_id	INTEGER	FOREIGN KEY (persona_id) REFERENCES personas(id)

```
ALTER TABLE articulos ADD COLUMN categoria_id INTEGER REFERENCES categorias(id);
```

SQL

```
CREATE TABLE articulos ( id INTEGER PRIMARY KEY, nombre TEXT, precio REAL ,
categoria_id INTEGER REFERENCES categorias(id))
```

16.11 RESTRICCIONES 'Pk y Fks'

PK = Primary Key

FK = Foreign Key

transacciones

```
CREATE TABLE transacciones2 (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  monto REAL,
  usuario_id INTEGER
);
INSERT INTO transacciones2 (monto, usuario_id) VALUES
(100, 3),
(200, 3),
(300, 3);
DROP TABLE transacciones;
ALTER TABLE transacciones2 RENAME TO transacciones;
```

COLUMNA	TIPO DE DATO	RESTRICCIONES
id	INTEGER	PRIMARY KEY
monto	REAL	
usuario_id	INTEGER	FOREIGN KEY (usuario_id) REFERENCES usuarios(id)

usuarios

COLUMNA	TIPO DE DATO	RESTRICCIONES
id	INTEGER	PRIMARY KEY
nombre	TEXT	
apellido	TEXT	

17. CONSULTAS EN MÚLTIPLES TABLAS

17.1 INNER JOIN

```
SELECT *
FROM usuarios
INNER JOIN notas
ON email1 = email2;
```

EMAIL1	NOMBRE	EDAD	EMAIL2	TÉLFONO
juan.perez@example.com	Juan Pérez	30	juan.perez@example.com	555-123-4567
juan.perez@example.com	Juan Pérez	30	juan.perez@example.com	999-888-7777
maria.gonzalez@example.com	Maria González	25	maria.gonzalez@example.com	444-987-6543
maria.gonzalez@example.com	Maria González	25	maria.gonzalez@example.com	333-111-0000
john.doe@example.com	John Doe	40	john.doe@example.com	777-555-8888
test.user@example.com	Test User	22	test.user@example.com	111-222-3333

17.2 INNER JOIN 'Con el mismo nombre'

```
SELECT *
FROM usuarios u
INNER JOIN notas n
ON u.email = n.email;
```

EMAIL	NOMBRE	EDAD	EMAIL	NOTAS
juan.perez@example.com	Juan Pérez	30	juan.perez@example.com	90
maria.gonzalez@example.com	Maria González	25	maria.gonzalez@example.com	100
john.doe@example.com	John Doe	40	john.doe@example.com	80
test.user@example.com	Test User	22	test.user@example.com	0
juan.perez@example.com	Juan Pérez	30	juan.perez@example.com	100
maria.gonzalez@example.com	Maria González	25	maria.gonzalez@example.com	100

17.3 INNER JOIN 'Con algunos atributos'

```
SELECT
    u.email,
    u.nombre,
    u.edad,
    n.notas
FROM usuarios u
INNER JOIN notas n
ON u.email = n.email;
```

EMAIL	NOMBRE	EDAD	NOTAS
juan.perez@example.com	Juan Pérez	30	90
maria.gonzalez@example.com	Maria González	25	100
john.doe@example.com	John Doe	40	80
test.user@example.com	Test User	22	0
juan.perez@example.com	Juan Pérez	30	100
maria.gonzalez@example.com	Maria González	25	100

17.4 INNER JOIN 'Sin resultados'

¿Qué sucedería si los emails presentes en una tabla no se encuentran en la otra tabla al momento de unir los datos?

Tabla **usuarios**

EMAIL	NOMBRE	EDAD
juan.perez@example.com	Juan Pérez	30
maria.gonzalez@example.com	Maria González	25
john.doe@example.com	John Doe	40
test.user@example.com	Test User	22

Tabla **datos_contacto**

EMAIL	TELÉFONO
alvaro.sanchez@example.com	555-123-4567
maria.fernandez@example.com	444-987-6543
francisca.medina@example.com	777-555-8888

La respuesta es bien sencilla: si no hay ningún dato común entre ambas tablas, no obtendremos resultados.

Utilizando lo aprendido previamente, selecciona todos los registros de la unión de las tablas **usuarios** y **datos_contacto**. Observa el resultado.

```
SELECT
    usuarios.*,
    datos_contacto.telefono
FROM
    usuarios
JOIN
    datos_contacto
ON
    usuarios.email = datos_contacto.email;
```

Al ejecutar esta consulta, no obtendremos ningún resultado ya que no hay emails comunes entre las tablas **usuarios** y **datos_contacto**.

17.5 INNER JOIN 'Orden de cláusulas'

COMANDO	SE LEE COMO:
SELECT	Selecciona estos datos.
FROM	De esta tabla.
JOIN	Únelos con esta tabla.
WHERE	Filtra los valores que cumplan tal condición.
GROUP BY	Agrupar los resultados por este criterio.
HAVING	Filtra por estos criterios agrupados.
ORDER BY	Ordena los resultados por este otro criterio.
LIMIT	Limita los resultados a esta cantidad.


```
SELECT *
FROM usuarios u
INNER JOIN notas n
ON u.email = n.email
WHERE u.email = 'juan.perez@example.com';
```

EMAIL	NOMBRE	EDAD	EMAIL	NOTAS
juan.perez@example.com	Juan	30	juan.perez@example.com	90
	Pérez			
juan.perez@example.com	Juan	30	juan.perez@example.com	100
	Pérez			

17.6 INNER JOIN 'Múltiples columnas'

```
SELECT
  p.Nombre,
  SUM(v.Cantidad) AS total_vendido
FROM
  Productos p
JOIN
  Ventas v ON p.ProductoID = v.ProductoID
GROUP BY
  p.ProductoID, p.Nombre
ORDER BY
  total_vendido DESC
LIMIT 1;
```

NOMBRE	TOTAL_VENDIDO
Producto C	55

18. CARDINALIDAD

18.1 RELACIONES '1 a 1'

```
SELECT *
FROM vehiculos v
LEFT JOIN matriculas m
ON v.id = m.vehiculo_id;
```

ID	MODELO	ID	VEHICULO_ID	MATRICULA
1	Toyota Corolla	1	1	ABC-123
2	Honda Civic	2	2	XYZ-456
3	Ford Focus	3	3	DEF-789

18.2 RELACIONES '1 a n (muchos)'

```
SELECT *
FROM paises p
INNER JOIN continentes c
ON c.continente_id=p.continente_id;
```

CONTINENTE_ID	NOMBRE	PAIS_ID	NOMBRE	CONTINENTE_ID
1	África	1	Nigeria	1
2	América	2	Brasil	2
3	Asia	3	China	3
4	Europa	4	Alemania	4
5	Oceanía	5	Australia	5
2	América	6	Argentina	2
3	Asia	7	Japón	3
4	Europa	8	Francia	4
1	África	9	Egipto	1
5	Oceanía	10	Nueva Zelanda	5

18.3 RELACIONES 'n (muchos) a n (muchos)'

```
SELECT *
FROM objetos o
INNER JOIN objetos_colores oc
ON oc.objeto_id=o.objeto_id
INNER JOIN colores c
ON oc.color_id=c.color_id;
```

OBJETO_ID	OBJETO	OBJETOS_COLORES_ID	OBJETO_ID	COLOR_ID	COLOR_ID	COLOR
1	Roj	Mesa	1	1	1	Rojo
1	Az	Mesa	2	1	2	Azul
2	Roj	Silla	3	2	1	Rojo
3	Ver	Cama	4	3	3	Verde

18.4 RELACIONES 'Tabla intermedia'

```
INSERT INTO profesores_alumnos (profesor_id, alumno_id) VALUES (1, 2);
INSERT INTO profesores_alumnos (profesor_id, alumno_id) VALUES (2, 3);

SELECT p.profesor_id, p.nombre, pa.profesor_id, pa.alumno_id, a.alumno_id, a.nombre
FROM profesores p
JOIN profesores_alumnos pa ON p.profesor_id = pa.profesor_id
JOIN alumnos a ON pa.alumno_id = a.alumno_id;
```

PROFESOR_ID	NOMBRE	PROFESOR_ID	ALUMNO_ID	ALUMNO_ID	NOMBRE
1	Julia	1	1	1	Marta
1	Julia	1	2	2	Elena
2	Pedro	2	3	3	Juan

18.5 RELACIONES 'Sin restricción de unicidad'

```
SELECT u.usuario_id, p.libro_id, COUNT(p.libro_id) AS veces
FROM usuarios u
INNER JOIN pedidos p
ON p.usuario_id=u.usuario_id
GROUP BY u.usuario_id, p.libro_id
HAVING COUNT(p.libro_id) > 1;
```

USUARIO_ID	LIBRO_ID	VECES
1	1	2
2	2	2

18.6 RELACIONES 'Con restricción de unicidad'

```
SELECT e.nombre, e.puesto, COUNT(ep.proyecto_id) AS cantidad_proyectos
FROM empleados e
INNER JOIN empleados_proyectos ep
ON ep.empleado_id=e.id
GROUP BY e.nombre, e.puesto;
```

NOMBRE	PUESTO	CANTIDAD_PROYECTOS
Carlos López	Gerente	3
Juan Pérez	Desarrollador	2
María García	Analista	2

19. TIPOS DE JOIN

19.1 JOIN 'INNER'

```
SELECT *
FROM usuarios u
INNER JOIN notas n
ON n.email = u.email
```

EMAIL	NOMBRE	EDAD	EMAIL	NOTAS
juan.perez@example.com	Juan Pérez	30	juan.perez@example.com	90
maria.gonzalez@example.com	Maria González	25	maria.gonzalez@example.com	100
john.doe@example.com	John Doe	40	john.doe@example.com	80
juan.perez@example.com	Juan Pérez	30	juan.perez@example.com	100
maria.gonzalez@example.com	Maria González	25	maria.gonzalez@example.com	100

19.2 JOIN 'INNER con diagrama de VENN'

```
SELECT a.nombre, p.titulo
FROM actores a
INNER JOIN peliculas p
ON a.actor_id = p.actor_id;
```

NOMBRE	TITULO
Robert Downey Jr.	Iron Man
Robert Downey Jr.	Avengers
Scarlett Johansson	Black Widow
Chris Hemsworth	Thor
Chris Hemsworth	Avengers
Mark Ruffalo	Avengers
Chris Evans	Avengers
Chris Evans	Captain America

19.3 JOIN 'LEFT'

```
SELECT e.email, e.nombre, e.edad, d.email, d.departamento
FROM empleados e
LEFT JOIN departamentos d
ON e.email = d.email;
```

EMAIL	NOMBRE	EDAD	EMAIL	DEPARTAMENTO
juan.perez@example.com	Juan Pérez	30	juan.perez@example.com	Marketing
juan.perez@example.com	Juan Pérez	30	juan.perez@example.com	RRHH
maria.gonzalez@example.com	Maria González	25	maria.gonzalez@example.com	
john.doe@example.com	John Doe	40	john.doe@example.com	TI
francisco@example.com	Test User	22		

19.4 JOIN 'LEFT con diagrama de VENN'

```
SELECT p.profesion, pe.nombre
FROM profesion p
LEFT JOIN personas pe
ON p.id = pe.profesion_id;
```

PROFESION	NOMBRE
Ingeniero	Juan
Médico	Maria
Abogado	Ana
Arquitecto	

19.5 JOIN 'RIGHT'

```
SELECT e.email, e.nombre, e.edad, d.email, d.departamento
FROM empleados e
RIGHT JOIN departamentos d
ON e.email = d.email;
```

EMAIL	NOMBRE	EDAD	EMAIL	DEPARTAMENTO
juan.perez@example.com	Juan Pérez	30	juan.perez@example.com	Marketing
juan.perez@example.com	Juan Pérez	30	juan.perez@example.com	RRHH
				Finanzas
			john.doe@example.com	TI

19.6 JOIN 'LEFT y RIGHT'

```
SELECT p.*, pr.*
FROM productos p
LEFT JOIN precios pr
ON p.producto_id = pr.producto_id;
```

PRODUCTO_ID	NOMBRE	PRECIO_ID	PRODUCTO_ID	PRECIO
1	Producto A	1	1	10.99
2	Producto B	2	2	15.99
3	Producto C			

19.7 JOIN 'Identificando JOIN'

```
SELECT a.nombre AS nombre_autor, l.titulo AS titulo_libro
FROM autores a
INNER JOIN libros l
ON a.id = l.id_autor;
```

NOMBRE_AUTOR	TITULO_LIBRO
Gabriel García Márquez	Cien Años de Soledad
Isabel Allende	La Casa de los Espíritus
J.K. Rowling	Harry Potter y la Piedra Filosofal

19.8 JOIN 'Identificando JOIN pt.2'

```
SELECT e.nombre AS nombre_empleado, p.nombre_proyecto
FROM empleados e
LEFT JOIN proyectos p
ON e.id_proyecto = p.id;
```

NOMBRE_EMPLEADO	NOMBRE_PROYECTO
Juan Pérez	Desarrollo Web
María González	App Móvil
Pedro López	
Ana Rodríguez	Sistema de Inventario

19.9 JOIN 'FULL OUTER JOIN'

```
SELECT *  
FROM empleados e  
FULL OUTER JOIN departamentos d  
ON e.id_departamento = d.id_departamento;
```

ID_EMPLEADO	NOMBRE	ID_DEPARTAMENTO	ID_DEPARTAMENTO	NOMBRE
1	Juan Pérez	1	1	Recursos Humanos
2	Ana Gómez	2	2	TI
3	Luis López			
4	Maria Díaz	3	3	Ventas
5	Carlos Fernández	2	2	TI

19.10 JOIN 'FULL OUTER JOIN 2a manera'

```
SELECT c.id_cliente, c.nombre_cliente, p.id_pedido, p.id_cliente, p.fecha_pedido  
FROM clientes c  
LEFT JOIN pedidos p ON c.id_cliente = p.id_cliente  
  
UNION  
  
SELECT c.id_cliente, c.nombre_cliente, p.id_pedido, p.id_cliente, p.fecha_pedido  
FROM clientes c  
RIGHT JOIN pedidos p ON c.id_cliente = p.id_cliente;
```

ID_CLIENTE	NOMBRE_CLIENTE	ID_PEDIDO	ID_CLIENTE	FECHA_PEDIDO
		103	5	2023-03-20
1	Cliente A	101	1	2023-01-10
2	Cliente B			
3	Cliente C	102	3	2023-02-15
4	Cliente D			

19.11 JOIN 'LEFT EXCLUDING'

```
SELECT *  
FROM personas pe  
LEFT JOIN profesion p  
ON pe.profesion_id = p.id  
WHERE p.nombre IS NULL;
```

Tu resultado

No hay nada, ya que en este ejercicio los valores que mostramos son todos NULL.

19.12 JOIN 'RIGHT EXCLUDING'

```
SELECT *  
FROM cursos c  
RIGHT JOIN docentes d  
ON d.id = c.docente_id  
WHERE c.id IS NULL;
```

ID	NOMBRE	DOCENTE_ID	ID	NOMBRE
			2	Felipe
			3	Susana

19.13 JOIN 'FULL OUTER EXCLUDING'

```
SELECT e.id_empleado, e.nombre, e.id_departamento, d.departamento  
FROM empleados e  
LEFT JOIN departamentos d  
ON e.id_departamento = d.id_departamento  
WHERE d.id_departamento IS NULL  
  
UNION  
  
SELECT e.id_empleado, e.nombre, d.id_departamento, d.departamento  
FROM empleados e  
RIGHT JOIN departamentos d  
ON e.id_departamento = d.id_departamento  
WHERE e.id_empleado IS NULL;
```

ID_EMPLEADO	NOMBRE	ID_DEPARTAMENTO	DEPARTAMENTO
		40	Marketing
3	María	30	
4	Carlos		

19.14 JOIN 'Natural JOIN'

```
SELECT p.nombre, v.cantidad, v.fecha  
FROM productos p  
NATURAL JOIN ventas v;
```

NOMBRE	CANTIDAD	FECHA
Laptop	2	2024-01-01
Smartphone	1	2024-01-01
Camiseta	3	2024-01-02
Zapatos	1	2024-01-02

19.15 JOIN 'Natural LEFT JOIN'

```
SELECT e.nombre, i.curso, i.fecha  
FROM estudiantes e  
NATURAL LEFT JOIN inscripciones i;
```

NOMBRE	CURSO	FECHA
Carlos	Física	2024-03-03
Carlos	Matemáticas	2024-03-01
Laura	Historia	2024-03-02
Miguel	Química	2024-03-04
Ana		

19.16 JOIN 'Natural SELF JOIN'

```
SELECT c.nombre AS nombre_cliente, c2.nombre AS nombre_cliente_referente
FROM clientes c
LEFT JOIN clientes c2
ON c.id_cliente_referente = c2.id_cliente;
```

NOMBRE_CLIENTE	NOMBRE_CLIENTE_REFERENTE
Juan	
Maria	Juan
Pedro	Juan
Ana	Maria
Luis	Maria

19.17 JOIN 'Natural SELF JOIN pt.2'

```
SELECT a.nombre, a2.nombre AS nombre_amigo_conectado
FROM amigos a
INNER JOIN amigos a2
ON a.id_amigo_conectado = a2.id_amigo;
```

NOMBRE	NOMBRE_AMIGO_CONECTADO
Laura	Carlos
Miguel	Carlos
Ana	Laura
Luis	Miguel

19.18 JOIN 'CROSS JOIN'

```
SELECT n.numero, p.pinta
FROM numeros n
CROSS JOIN pintas p
ORDER BY n.numero, p.pinta;
```

NUMERO	PINTA
1	Corazones
1	Diamantes
1	Picas
1	Tréboles
2	Corazones
2	Diamantes
2	Picas
2	Tréboles
3	Corazones

19.19 JOIN 'CROSS JOIN'

```
SELECT u.email, COUNT(n.notas) AS cantidad_notas
FROM usuarios u
LEFT JOIN notas n
ON u.email = n.email
GROUP BY u.email
```

EMAIL	CANTIDAD_NOTAS
francisco@example.com	0
john.doe@example.com	1
juan.perez@example.com	2
maria.gonzalez@example.com	2

20. NORMALIZACIÓN

20.1 GRUPOS 'Introducción a la primera forma normal'

```
DELETE FROM personas WHERE id = 1;
DELETE FROM personas WHERE id = 2;
DELETE FROM personas WHERE id = 3;
DELETE FROM personas WHERE id = 4;
DELETE FROM personas WHERE id = 5;
DELETE FROM personas WHERE id = 6;
```

ID	NOMBRE	APELLIDO	ESTADO_CIVIL
7	Juan	Pérez	Divorciado
8	María	González	Soltera
9	Pedro	Rodríguez	Casado

20.2 GRUPOS 'Convirtiendo la primera forma normal (1fn)'

```
id INTEGER PRIMARY KEY AUTOINCREMENT,
producto TEXT,
categoria TEXT,
precio REAL
);
INSERT INTO productos2 (producto, categoria, precio) VALUES
('Manzana', 'Fruta', 0.50),
('Pan', 'Panadería', 1.00),
('Leche', 'Lácteos', 1.20),
('Manzana', 'Fruta', 0.55),
('Pan', 'Panadería', 1.10),
('Queso', 'Lácteos', 2.50);
DROP TABLE productos;
```

ID	PRODUCTO	CATEGORÍA	PRECIO
1	Manzana	Fruta	0.5
2	Pan	Panadería	1
3	Leche	Lácteos	1.2
4	Manzana	Fruta	0.55
5	Pan	Panadería	1.1
6	Queso	Lácteos	2.5

20.3 GRUPOS 'Grupos repetitivos'

```
CREATE TABLE empleados (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  nombre TEXT,
  apellido TEXT
);
INSERT INTO empleados (nombre, apellido) VALUES
('Juan', 'Pérez');
CREATE TABLE proyectos (
  id_empleado INTEGER,
  id_proyecto INTEGER,
  FOREIGN KEY (id_empleado) REFERENCES empleados(id)
);
INSERT INTO proyectos (id_empleado, id_proyecto) VALUES
(1, 1),
(1, 2),
(1, 3);
SELECT e.nombre, e.apellido, p.id_proyecto
FROM empleados e
INNER JOIN proyectos p
ON e.id = p.id_empleado;
```

NOMBRE	APELLIDO	ID_PROYECTO
Juan	Pérez	1
Juan	Pérez	2
Juan	Pérez	3

20.4 GRUPOS 'Grupos repetitivos pt.2'

```
SELECT d.nombre_departamento, COUNT(p.departamento_id) AS cantidad_personas
FROM departamentos d
INNER JOIN personas p
ON d.id = p.departamento_id
GROUP BY d.nombre_departamento;
```

NOMBRE_DEPARTAMENTO	CANTIDAD_PERSONAS
Marketing	2
Recursos Humanos	1
TI	2

20.5 GRUPOS 'Dependencias parciales (2fn)'

```
SELECT e.nombre_del_estudiante, c.nombre_del_curso, n.nota
FROM notas n
LEFT JOIN estudiantes e
ON n.id_de_estudiante = e.id_de_estudiante
LEFT JOIN cursos c
ON n.id_de_curso = c.id_de_curso;
```

NOMBRE_DEL_ESTUDIANTE	NOMBRE_DEL_CURSO	NOTA
Juan	Matemáticas	90
Juan	Historia	85
Ana	Matemáticas	95
Ana	Ciencia	88

20.6 GRUPOS 'Dependencias transitivas (2fn)'

```
CREATE TABLE paises (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  pais TEXT
);
INSERT INTO paises (pais) VALUES
('Estados Unidos'),
('Brasil'),
('Francia'),
('India'),
('Corea del Sur');
CREATE TABLE musicos (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  musico TEXT,
  edad_musico INTEGER,
  pais_id REFERENCES paises(id)
);
INSERT INTO musicos (musico, edad_musico, pais_id) VALUES
('Beyoncé', 42, 1),
('Gilberto Gil', 81, 2),
('David Guetta', 56, 3),
('A. R. Rahman', 57, 4),
('RM', 29, 5);
```