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 ≺

	NOMBRE		<u>PUNTOS</u>
1.	Seleccionar columnas	1	0 / 7
	Seleccionar filas	i	0 / 15
	Ordenar resultados	i	0 / 6
4.	Limitar resultados	i	0/2
	Operaciones con 'Strings'	i	0 / 10
6.	Operaciones con fechas	i	0 / 6
7.	Funciones de agregación	i	0/6
8.	Distinct	į	0/6
9.	Introducción a grupos	İ	0 / 9
	Having	ĺ	0/5
11.	Subconsultas	- 1	0 / 4
12.	Combinación de consultas	- 1	0/5
13.	Inserción de registros	- 1	0/9
14.	Borrado y modificación de registros	- 1	0 / 5
15.	Tablas	- 1	0/6
16.	Restricciones	- 1	0 / 11
17 .	Consultas en múltiples tablas	- 1	0/6
18.	Cardinalidad	-	0/6
19.	Tipos de 'Join'	- 1	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	Agrupa 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 <u>27/01/2025</u>.

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' :



1.2 SELECT 'Seleccionamos toda la tabla'



1.3 SELECT 'Seleccionamos una columna'

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



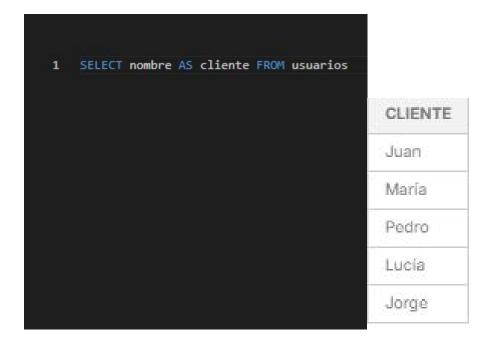
1.4 SELECT 'Seleccionamos varias columnas'

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



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 usurios;":



1.6 SELECT 'Alias (AS) varias columnas'

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



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;":



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.



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:



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;":



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:



2.5 SELECCIONANDO "Filas bajo condición"

A veces querremos 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;":



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;":



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';":



Tenemos que tener en cuenta que la comparación es sensible, es decir, que tenemos que tener cuidado al comparar con MAYUS y MINUS, <u>no es lo mismo 'Camiseta'</u> 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:



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



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';":



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;":



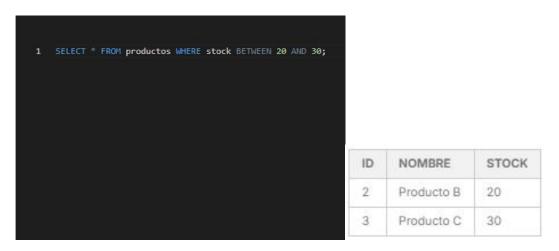
2.11 SELECCIONANDO "Una fecha, otro tipo de dato"

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';":



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;":



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:



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 '%0';":



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;":



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;":



3. ORDENAR RESULTADOS

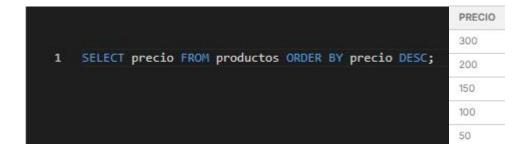
3.1 ORDENANDO "Filas (ASC)"

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;":

		ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
1922		5	Jorge	Martínez	jorgemartinez@gmail.com	555-4321
1	SELECT * FROM usuarios ORDER BY nombre ASC;	1	Juan	Pérez	Juanperez@gmail.com	555-1234
		4	Lucía	Sánchez	luciasanchez@outlook.com	555-5555
		2	Maria	García	mariagarcia@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;":



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:

	ID	NOMBRE	PRECIO
	3	Producto 3	50
1 SELECT * FROM productos ORDER BY precio NULLS LAST;	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;":

		ID	NOMBRE	SALARIO
-	COLECT 2 FROM 1 1 000FR BY 1 1 155 1 155	1	Juan Pérez	4800
1	SELECT * FROM empleados ORDER BY salario ASC, nombre ASC;	5	Luis Rodriguez	4800
		4	Ana Martínez	5500
		2	María López	5500
		3	Pedro Garcia	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;":

	ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
	1	Juan	Pérez	juanperez@gmail.com	555-1234
1 SELECT * FROM usuarios ORDER BY id LIMIT 3;	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;":

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;":

		EMAIL_UPPER
1000		JUANPEREZ@GMAIL.COM
1	SELECT UPPER(email) AS email_upper FROM usuarios;	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;":

		EMAIL_LOWER
		juan@example.com
1	SELECT LOWER(email) AS email_lower FROM usuarios;	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;":



5.4 COMBINANDO "Funciones"

Se pueden combinar varias funciones a la vez, asi 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;":



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;":



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 carácteres 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:



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;":

		PRIMERAS_LETRAS
		Pér
1	SELECT SUBSTR(apellido, 1, 3) AS primeras_letras FROM usuarios;	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':

```
1 SELECT SUBSTR(apellido, 2, 3) AS tres_caracteres_del_apellido FROM usuarios WHERE nombre = 'Maria';

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');":



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.

		ID	DESCRIPCION	FECHA_LIMITE
1	SELECT * FROM tareas WHERE fecha_limite = DATE('now', '1 day');	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'):

```
1 SELECT monto FROM ganancias WHERE fecha = DATE('now', '-1 day');

75
```

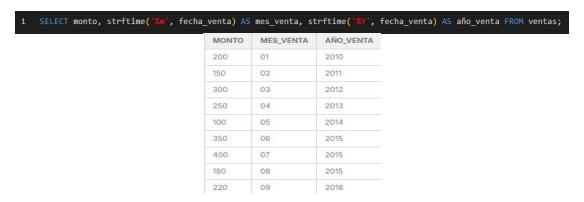
6.4 OBTENIENDO 'Extraer año de transacción'

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:



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



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';":



7. FUNCIONES DE AGREGACIÓN

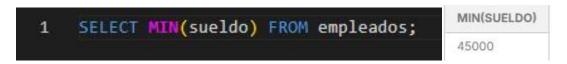
7.1 MAYOR 'Valor de una columna'

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



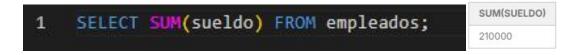
7.2 MENOR 'Valor de una columna'

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



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;":



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;":



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(*)
```

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';

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(*)
```

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';
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;":



8.2 SELECCIONANDO 'Correos únicos'

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



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;":



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;":

8.5 CONTANDO 'Correos unicos'

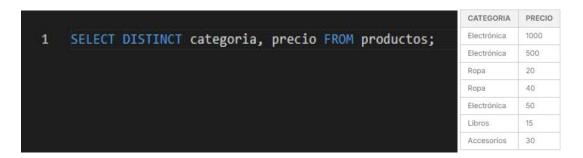
Crea una consulta para contestar cuantos correos únicos existen en la tabla. La columna resultante debe llamarse correos_cant:

1 SELECT COUNT(DISTINCT correo) AS correos_cant FROM usuarios;

CORREOS_CANT

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;":



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;":

		CORREO_UNICO
1	SELECT correo AS correo_unico FROM usuarios GROUP BY correo;	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:



9.3 AGRUPANDO 'Y sumando'

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



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



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:



9.6 AGRUPANDO 'Y sacando el mínimo por grupo'

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



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:



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:

					CORREO	REPETICIONES
1	SELECT correo,	COUNT(correo)	AS repeticiones	FROM usuarios GROUP BY 1 ORDER BY 1;	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:



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:



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:

1	SELECT correo, COUNT(correo) AS cuenta_correos	CORREO	CUENTA_CORREOS
2	FROM correos_corporativos GROUP BY correo	juan.perez@empresa.com	2
4	HAVING cuenta_correos > 1;	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:

8417	AND	CANTIDAD_DE_USUARIOS	DEPARTAMENTO
1	SELECT COUNT(nombre) AS cantidad_de_usuarios, departamento	2	Finanzas
2	FROM empleados	2	Marketing
3	GROUP BY departamento	2	Tecnología
4	<pre>HAVING cantidad_de_usuarios > 1;</pre>	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:

1	SELECT email, AVG(notas) AS promedio_notas	EMAIL	PROMEDIO_NOTAS
2	FROM notas GROUP BY email	Alumno1@ejemplo.com	56.66666666666664
4	HAVING promedio_notas >= 50;	Alumno2@ejemplo.com	53.33333333333333

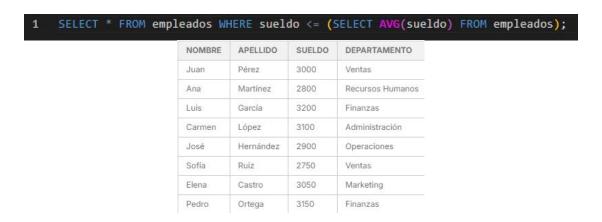
10.5 HAVING 'Y order'

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:



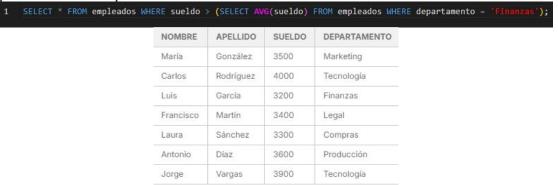
11. SUBCONSULTAS

11.1 SUBCONSULTAS



11.2 SUBCONSULTAS 'Con WHERE'

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

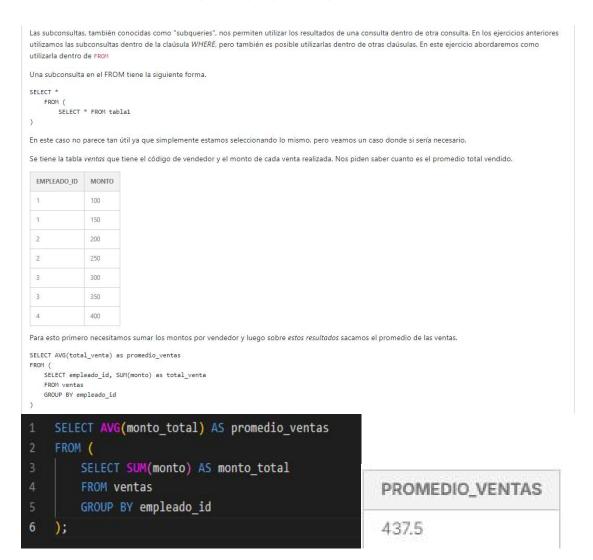


11.3 SUBCONSULTAS 'Con IN'

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:



11.4 SUBCONSULTAS 'Con FROM'



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, columna: 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

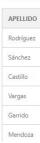
Profesores

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:



		NOMBRES
1	SELECT nombre AS nombres	Ana
2	FROM estudiantes	Carlos
2	UNTON	Juan
7	SELECT nombre AS nombres	Luis
_	FROM profesores;	Maria
,	FROM profesores,	Pedro

12.2 INTRODUCCION 'Cláusula UNION para eliminar duplicados'

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

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 *
2	FROM empleados1
3	UNION ALL
4	SELECT *
5	FROM empleados2;

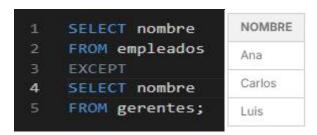
NOMBRE	APELLIDO	EDAD
Juan	Pérez	30
María	González	25
Carlos	Rodriguez	40
Ana	Martínez	22
María	González	25
Carmen	López	25

12.4 INTRODUCCION 'A la intersección'

ntrodu	ucción a intersección	
operador	INTERSECT se utiliza para combinar dos SELECT y d	devolver los resultados que se encuentran en ambas consultas.
r ejemplo	, si tenemos las siguientes dos tablas, clientes1 y clie	entes2:
bla client e	es1:	
NOMBRE		
Juan		
Maria		
Carlos		
Ana		
Luis		
abla client e	ns2:	
NOMBRE		
Ana		
Luis		
Pedro		
Carmen		
Juan		
	contrar los clientes en común utilizando INTERSECT	Taka la sinuitanta farmasi
	re FROM clientes1 INTERSECT SELECT nombre FROM c	
	ado obtendremos:	CITCHUESZ
NOMBRE		
Ana		
Juan		
Luis		
1	SELECT *	CLIENTE
		Ana
2	FROM lista1	34500.00004
		Carmen

1 SELECT * 2 FROM lista1 3 INTERSECT Juan 4 SELECT * 5 FROM lista2; Maria Pedro

12.5 INTRODUCCION 'Operador EXCEPT'



13. <u>ISERCIÓN DE REGISTROS</u>

13.1 AÑADIR 'Registro en una tabla'

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	Lucia	Sánchez	luciasanchez@outlook.com	555-5555

13.2 ESPECIFICANDO 'Valores nulos'

	ID	NOMBRE	PRECIO	STOCK
	1	Camisa	50	20
THE PART THE STATE OF THE STATE	2	Pantalón	80	15
INSERT INTO productos VALUES (7, 'Bolso', 1000, NULL);	3	Zapatos	120	10
	4	Sombrero	30	5
	7	Bolso	1000	

13.3 ESPECIFICANDO 'Columnas'

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

13.3 ESPECIFICANDO 'Columnas (Solo algunas)'

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

INSERT INTO productos (nombre, stock) VALUES
('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
Rolso		10	2025-02-18

13.5 AÑADIR 'Fecha de hoy con formato'

INSERT INTO productos (nombre, precio, stock, fecha) VALUES NOMBRE **PRECIO** STOCK **FECHA** 1000 Gorro 2025-02-18 Camiseta 500 10 2025-02-18 Pantalón 1500 8 2025-02-18 2000 3 2025-02-18 Zapatos 800 Bufanda 12 2025-02-18 Bolso 10 2023-01-01

13.6 AÑADIR 'Multiples valores'

INSERT INTO ventas (producto, cantidad, precio) VALUES	PRODUCTO	CANTIDAD	PRECIO
('Gorro', 5, 1000),	Gorro	5	1000
('Camiseta', 10, 500),	Camiseta	10	500
('Pantalón', 8, 1500);	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	ID	NOMBRE	APELLIDO
('Jane', 'Smith');	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);

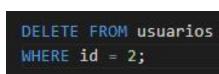
		usuarios (n inchez', 'l	ombre, apellido, email uciasanchez@outlook.co	
ID	NOMBRE	APELLIDO	EMAIL	TELEFONO
1	Juan	Pérez	juanperez@gmail.com	555-1234
2	María	García	mariagarcia@hotmail.com	555-5678
3	Pedro	López	pedrolopez@yahoo.com	555-9876
4	Jorge	Martínez	jorgemartinez@gmail.com	555-4321
5	Lucía	Sánchez	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'



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

14.3 EDITA 'Registros'

	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
UDDATE usuanias SET pagistmada IDUE:	4	Lucía	Sánchez	luciasanchez@outlook.com	1
<pre>UPDATE usuarios SET registrado = TRUE;</pre>	5	Jorge	Martinez	jorgemartinez@gmail.com	1

14.4 EDITA 'Registros usando WHERE'

PDA	WE USUARI	os SET tele	efono = '123-456' WHERE	id = 4;
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	Martinez	jorgemartinez@gmail.com	555-4321

14.5 EDITA 'Múltiples columnas'

```
UPDATE posts
   titulo = "Aprendiendo SQL",
   contenido = "SQL es un lenguaje de programación para gestionar bases de datos relacionales"
                   TITULO
             ID
                                    CONTENIDO
                                                                       AUTOR
                                                                                 FECHA
                   Aprendiendo
                                                                       Autor 1
                                                                                 2022-
                                    SQL es un lenguaje de
                   SQL
                                    programación para gestionar
                                                                                 01-01
                                    bases de datos relacionales
             2
                   Post 2
                                    Contenido del post 2
                                                                       Autor 2
                                                                                 2022-
                                                                                 01-02
             3
                   Post 3
                                    Contenido del post 3
                                                                                 2022-
                                                                       Autor 3
                                                                                 01-03
                                                                                 2022-
                   Post 4
                                    Contenido del post 4
                                                                       Autor 4
                                                                                 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');

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, ('Buenos Aires', 20.0, '2024-01-01'), ('Buenos Aires', 21.0, '2024-01-02'), ('Santiago', 22.0, '2024-01-01'),	fecha) VALUES
('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 desc	ripcion TEXT;	
('Cami	t <mark>alón'</mark> , 2000.00, '		corta'), ezclilla'),	VALUES
(Cam	isa XL', 1000.00,	'Camisa de mar	iga 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;

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
                                                         NOMBRE
                                                                         PRECIO
                                                                                     CODIGO
                                                         Camisa
                                                                         1000
                                                                                      CAM-001
INSERT INTO productos(nombre, precio, codigo) VALUES
   amisa', 1000.00,
                                                                        2000
                                                                                     PAN-001
                                                         Pantalón
   antalón', 2000.00, '
amisa XL', 1000.00,
                                                         Camisa XL
                                                                         1000
                                                                                      CAM-002
```

16.5 RESTRICCIONES 'CHECK'

```
CREATE TABLE productos (
    nombre TEXT NOT NULL,
    precio REAL NOT NULL,
    stock INTEGER CHECK(stock >= θ)
                                                             NOMBRE
                                                                               PRECIO
                                                                                             STOCK
                                                                               1000
                                                                                             10
                                                             Camisa
INSERT INTO productos (nombre, precio, stock) VALUES
   amisa', 1000.00, 10),
antalón', 2000.00, 5)
                                                                               2000
                                                                                             5
                                                             Pantalón
      alón', 2000.00, 5),
sa XL', 1000.00, 3);
                                                              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,	ID	TITLE	CONTENT
title TEXT,	1	Introducción	¡Bienvenido al mundo de la programación!
CONTENT TEXT ; NSERT INTO posts (id, title, content) VALUES	2	Primeros pasos	Sumérgete en los conceptos básicos de la programación.
 Introducción', '¡Bienvenido al mundo de la programación!'), 'Primeros pasos', 'Sumérgete en los conceptos básicos de la programación.'), 'Temas Avanzados', 'Explora conceptos y técnicas avanzadas en 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,	ID	NOMBRE	FECHA_CREACION
fecha_creacion DATE);	1	Ana	2024-01-01
INSERT INTO usuarios (nombre, fecha_creacion) VALUES	2	Gonzalo	2024-01-02
('Ana', '2024-01-01'), ('Gonzalo', '2024-01-02'),	3	Juan	2024-01-03
('Juan', '2024-01-03'), ('Maria', '2024-01-04');	4	María	2024-01-04

16.8 RESTRICCIONES 'Autoincremental'

id INTEGER PRIMARY KEY AUTOINCREMENT, monto REAL NOT NULL, fecha DATE	ID	MONTO	FECHA
);	1	1000	2024 01 01
INSERT INTO transacciones (monto, fecha) VALUES	13	1000	2024-01-01
(1000.00, "2024-01-01"),	2	2000	2024-01-02
(2000.00, "2024-01-02"),	Description of	7.000.000E1	
(3000.00, <mark>'2024-01-03'</mark>);	3	3000	2024-01-03
INSERT INTO transacciones (id, monto, fecha) VALUES (10, 4000.00, '2024-01-04');	10	4000	2024-01-04
INSERT INTO transacciones (monto, fecha) VALUES	11	5000	2024-01-05

16.9 RESTRICCIONES 'Primary key y TEXT'



16.10 RESTRICCIONES 'Clave FOREIGN (Foránea)'

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

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

```
TIPO DE DATO
                                                                                         RESTRICCIONES
CREATE TABLE transacciones2 (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
                                                                           INTEGER
                                                                                         PRIMARY KEY
    monto REAL,
                                                                monto
                                                                           REAL
    usuario_id INTEGER
                                                                usuario_id
                                                                           INTEGER
                                                                                         FOREIGN KEY (usuario_id) REFERENCES usuarios(id)
INSERT INTO transacciones2 (monto, usuario_id) VALUES
                                                                          TIPO DE DATO
                                                                                         RESTRICCIONES
(200, 3),
(300, 3);
                                                                           INTEGER
                                                                                         PRIMARY KEY
                                                                id
DROP TABLE transacciones;
                                                                           TEXT
ALTER TABLE transacciones2 RENAME TO transacciones;
```

17. CONSULTAS EN MÚLTIPLES TABLAS

17.1 INNER JOIN



17.2 INNER JOIN 'Con el mismo nombre'



17.3 INNER JOIN 'Con algunos atributos'

SELECT	EMAIL	NOMBRE	EDAD	NOTAS
u.email,	juan.perez@example.com	Juan Pérez	30	90
u.nombre,	maria.gonzalez@example.com	Maria González	25	100
u.edad, n.notas	john.doe@example.com	John Doe	40	80
FROM usuarios u	test.user@example.com	Test User	22	0
INNER JOIN notas n	juan.perez@example.com	Juan Pérez	30	100
<pre>ON u.email = n.email;</pre>	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	Agrupa 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.



17.6 INNER JOIN 'Múltiples columnas'

```
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;
Producto C 55
```

18. <u>CARDINALIDAD</u>

18.1 RELACIONES '1 a 1'

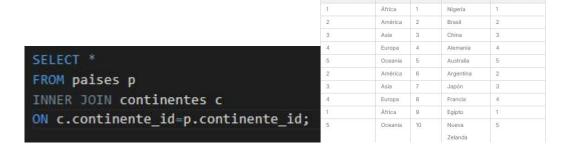
	ID	MODELO	ID	VEHICULO_ID	MATRICULA
SELECT * FROM vehiculos v	1	Toyota Corolla	1	1	ABC-123
LEFT JOIN matriculas m	2	Honda Civic	2	2	XYZ-456
ON v.id = m.vehiculo_id;	3	Ford Focus	3	3	DEF-789

18.2 RELACIONES '1 a n (muchos)'

CONTINENTE_ID

NOMBRE PAIS_ID NOMBRE

CONTINENTE_ID



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



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. <u>TIPOS DE JOIN</u>

19.1 JOIN 'INNER'

EMAIL	NOMBRE	EDAD	EMAIL	NOTAS
) juan.perez@example.com	Juan Pérez	30	juan.perez@example.com	90
0 maria.gonzalez@example.com	n Maria González	25	maria.gonzalez@example.com	100
SELECT * john.doe@example.com	John Doe	40	john.doe@example.com	80
FROM usuarios u Ojuan.perez@example.com INNER JOIN notas n	Juan Pérez	30	juan.perez@example.com	100
ON n.email = u.email O maria.gonzalez@example.com	Maria González	25	maria.gonzalez@example.com	100

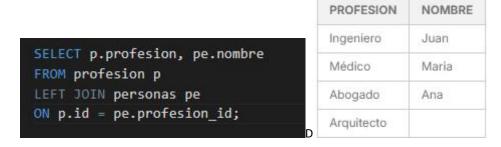
19.2 JOIN 'INNER con diagrama de VENN'

	NOMBRE	TITULO
cricer a senter a stanta	Robert Downey Jr.	Iron Man
SELECT a.nombre, p.titulo	Robert Downey Jr.	Avengers
FROM actores a	Scarlett Johansson	Black Widow
TROM accores a	Chris Hemsworth	Thor
INNER JOIN peliculas p	Chris Hemsworth	Avengers
	Mark Ruffalo	Avengers
ON a.actor_id = p.actor_id;	Chris Evans	Avengers
	Chris Evans	Captain America

19.3 JOIN 'LEFT'

	EMAIL	NOMBRE	EDAD	EMAIL	DEPARTAMEN	ITO
	ar luding erez@example.com	Juan Pérez	30	juan.perez@example.com	Marketing	
	≀Hidan.perez@example.com	Juan Pérez	30	juan.perez@example.com	RRHH	
SELECT e.email, e.nombre, e.edad, d.email, d.departament	maria.gonzalez@example.com	Maria González	25	maria.gonzalez@example.com		
FROM empleados e	john.doe@example.com	John Doe	40	john.doe@example.com	TI	
LEFT JOIN departamentos d ON e.email = d.email;	francisco@example.com	Test User	22			

19.4 JOIN 'LEFT con diagrama de VENN'



19.5 JOIN 'RIGHT'

FROM empleados e RIGHT JOIN departamentos d ON e.email = d.email;								
EMAIL	NOMBRE	EDAD	EMAIL	DEPARTAMENT				
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.*	PRODUCTO_ID	NOMBRE	PRECIO_ID	PRODUCTO_ID	PRECIO
FROM productos p	1	Producto A	1	1	10.99
LEFT JOIN precios pr	2	Producto B	2	2	15.99
<pre>ON p.producto_id = pr.producto_id;</pre>	3	Producto C			

19.7 JOIN 'Identificando JOIN'

	NOMBRE_AUTOR	TITULO_LIBRO
SELECT a.nombre AS nombre_autor, l.titulo AS titulo_libro FROM autores a	Gabriel García Márquez	Cien Años de Soledad
INNER JOIN libros l	Isabel Allende	La Casa de los Espiritus
ON a.id = 1.id_autor;	J.K. Rowling	Harry Potter y la Piedra Filosofal

19.8 JOIN 'Identificando JOIN pt.2'

	NOMBRE_EMPLEADO	NOMBRE_PROYECTO
SELECT e.nombre AS nombre_empleado, p.nombre_proyecto	Juan Pérez	Desarrollo Web
FROM empleados e	María González	App Móvil
LEFT JOIN proyectos p	Pedro López	19101
ON e.id proyecto = p.id;	redio Lopez	
	Ana Rodríguez	Sistema de Inventario

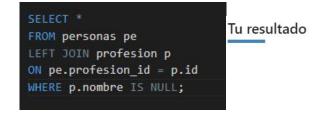
19.9 JOIN 'FULL OUTER JOIN'

	ID_EMPLEADO	NOMBRE	ID_DEPARTAMENTO	ID_DEPARTAMENTO	NOMBRE
	1	Juan Pérez	1	1	Recursos Humanos
	2	Ana Gómez	2	2	ТІ
	3	Luis López			
FROM empleados e	4	María Díaz	3	3	Ventas
<pre>FULL OUTER JOIN departamentos d ON e.id_departamento = d.id_departamento;</pre>	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
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
                                                5
                                   103
                                                              2023-03-20
1
               Cliente A
                                   101
                                                1
                                                              2023-01-10
 2
               Cliente B
 3
               Cliente C
                                   102
                                                3
                                                              2023-02-15
               Cliente D
 4
```

19.11 JOIN 'LEFT EXCLUDING'



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

19.12 JOIN 'RIGHT EXCLUDING'

SELECT *					
FROM cursos c	ID	NOMBRE	DOCENTE_ID	ID	NOMBRE
RIGHT JOIN docentes d	10	HOMBRE	DOOLINI E_ID	ID	HOWDRE
ON d.id = c.docente_id				2	Felipe
WHERE c.id IS NULL;				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'

The second second	p.nombre, oductos p	v.cantidad,	v.fecha	
NATURAL	JOIN ven	tas v;		

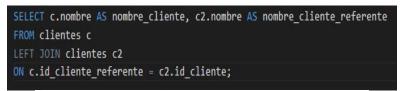
NOMBRE	CANTIDAD	FECHA
Laptop	2	2024-01-01
Smartphone	1	2024-01-01
Camiseta	3	2024-01-02
Zapatos	1	2024-01-02

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'



NOMBRE_CLIENTE	NOMBRE_CLIENTE_REFERENTE
Juan	
Maria	Juan
Pedro	Juan
Ana	Maria
Luis	Maria

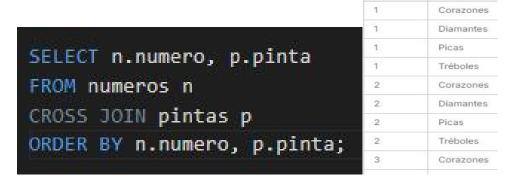
19.17 JOIN 'Natural SELF JOIN pt.2'

SELECT a.nombre, a2.nombre AS nombre_amigo_conectado FROM amigos a INNER JOIN amigos a2	NOMBRE	NOMBRE_AMIGO_CONECTADO
	Laura	Carlos
	Miguel	Carlos
ON a.id amigo conectado = a2.id amigo;	Ana	Laura
on a.iu_amigo_coneccado = az.iu_amigo,	Luis	Miguel

19.18 JOIN 'CROSS JOIN'

NUMERO

PINTA



19.19 JOIN 'CROSS JOIN'

<pre>SELECT u.email, COUNT(n.notas) AS cantidad_notas</pre>	EMAIL	CANTIDAD_NOTAS
FROM usuarios u	francisco@example.com	0
LEFT JOIN notas n	john.doe@example.com	1
ON u.email = n.email	juan.perez@example.com	2
GROUP BY u.email	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;	ID	NOMBRE	APELLIDO	ESTADO_CIVIL
DELETE FROM personas WHERE id = 3;	7	Juan	Pérez	Divorciado
DELETE FROM personas WHERE id = 4; DELETE FROM personas WHERE id = 5;	8	María	González	Soltera
DELETE FROM personas WHERE id = 6;	9	Pedro	Rodriguez	Casado

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

	ID	PRODUCTO	CATEGORÍA	PRECIO
1G INTEGER PRIMARY KEY AUTOINCREMENT, producto TEXT, categoría TEXT,	1	Manzana	Fruta	0.5
precio REAL	2	Pan	Panadería	1
);	23	\$200 A \$2,000 C	Carrier	10021
INSERT INTO productos2 (producto, categoría, precio) VALUES ('Manzana', 'Fruta', 0.50),	3	Leche	Lácteos	1.2
('Pan', 'Panaderia', 1.00), ('Leche', 'Lácteos', 1.20),	4	Manzana	Fruta	0.55
('Manzana', 'Fruta', 0.55), ('Pan', 'Panadería', 1.10),	5	Pan	Panadería	1.1
('Queso', 'Lácteos', 2.50); DROP TABLE productos;	6	Queso	Lácteos	2.5

20.3 GRUPOS 'Grupos repetitivos'

```
INSERT INTO proyectos (id_empleado, id_proyecto) VALUES

INSERT INTO proyectos (id_empleado, id_proyecto) VALUES

(1, 1),
apellido TEXT,
);
INSERT INTO empleados (nombre, apellido) VALUES

(1, 2),
INSERT INTO empleados (nombre, apellido) VALUES

(1, 3);
CREATE TABLE proyectos (
id_empleado INTEGER,
id_proyecto INTEGER,
id_proyecto INTEGER,
FOREIGN KEY (id_empleado) REFERENCES empleados(id)
);
INSERT INTO proyectos (id_empleado, id_proyecto) VALUES

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	NOMBRE_DEL_ESTUDIANTE	NOMBRE_DEL_CURSO	NOTA
FROM notas n LEFT JOIN estudiantes e	Juan	Matemáticas	90
ON n.id_de_estudiante = e.id_de_estudiante	Juan	Historia	85
LEFT JOIN cursos c	Ana	Matemáticas	95
ON n.id_de_curso = c.id_de_curso;	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,

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);
```