

SQL

Structured Query Language.

# INDICE

3	¿QUÉ ES SQL?	20	INSERT INTO
4	DDL - DML	21	UPDATE
5	TIPOS DE DATOS	22	DELETE
6	TABLAS	23	FUNCIONES MIN-MAX-COUNT
7	CONSTRAINTS	24	FUNCIONES AVG-SUM
8	CONSTRAINTS	25	ALIAS
9	MODIFICAR Y BORRAR TABLAS	26	JOIN - INNER JOIN
10	INDICES	27	EJEMPLO INNER JOIN
11	VISTAS	28	OUTER JOINS : LEFT JOIN-RIGHT J.
12	SELECT - FROM	29	FULL JOIN - SQL UNION
13	WHERE	30	GROUP BY - HAVING.
14	OPERADORES LÓGICOS		
15	OPERADORES DE TEXTO		
16	DISTINCT		
17	ORDER-BY		
18	LIMIT - OFFSET		
19	NULL		

SQL, o **Structured Query Language** es un lenguaje para consultar, manipular y transformar datos de una base de datos relacional.

una base de datos relacional representa una colección de tablas relacionadas.

Cada una de las tablas es similar a una hoja de cálculo de Excel, con un número fijo de columnas y cualquier número de filas de datos.

Y dada una tabla de datos, la consulta más básica que podríamos escribir sería una que seleccione un par de columnas (propiedades) de la tabla con todas las filas...





→ DDL (Data Definition Language): Encargado de la definición de Bases de datos, tablas, índices, etc.

Comandos:

- CREATE TABLE
- CREATE INDEX
- CREATE VIEW



→ DML (Data Manipulation Language): manipulación de datos; seleccionar, insertar, eliminar y actualizar datos.

Comandos:

- SELECT
- UPDATE
- INSERT
- DELETE FROM



# TIPOS de datos

(algunos de ellos)



## NUMÉRICOS:

- INTEGER: Valor Entero
- NUMERIC(n,m): números de hasta 18 dígitos (con decimales) donde  $n$   $\Rightarrow$  representa el total de dígitos admitidos y  $m$   $\Rightarrow$  el número de posiciones decimales.
- DECIMAL(n,m): igual a NUMERIC.
- FLOAT: número de coma flotante.



## ALFANUMÉRICOS:

- CHAR(n): almacena de 1 a 255 caracteres. Fijo.
- VARCHAR(n): datos de cadena de tamaño variable.

## FECHA:



- DATE: Almacena fechas con día, mes y año.
- DATETIME: Almacena fechas con fecha y hora.

## LÓGICOS:

- Bit: Tipo bit. Se aplica lógica booleana.
- Boolean: Cero  $\rightarrow$  false, distinto a cero  $\rightarrow$  true

# tablas

DDL ó el Lenguaje de definición de datos es el encargado de permitir la descripción de los objetos que forman una base de datos: creación de tablas, índices; modificación de las estructuras de tablas, índices y vistas; eliminación de tablas, índices...

## ✦ Creación de tablas:

La instrucción CREATE TABLE se utiliza para crear una nueva tabla en una base de datos.

```
CREATE TABLE nombre-tabla (  
  columna1 tipo-datos  
  columna2 tipo-datos  
  ...  
);
```

ESPECIFICA LOS NOMBRES DE LAS COLUMNAS

ESPECIFICA EL TIPO DE DATOS QUE LA COLUMNA PUEDE CONTENER (INT, DATE, ETC.)



## ✦ Constraints:

Son restricciones que se utilizan para limitar el tipo de dato que puede recibir una columna de una tabla.

Las restricciones se pueden definir cuando se crea una tabla (CREATE TABLE) o posteriormente con la sentencia ALTER TABLE.

## Posibles restricciones:



★ NOT NULL: ESPECIFICA QUE UNA COLUMNA NO ACEPTA EL VALOR NULL, ES DECIR, QUE ESA COLUMNA SIEMPRE DEBE TENER UN VALOR.

```
CREATE TABLE PERSONAS (  
  Nombre varchar(255) NOT NULL,  
  Apellido varchar(255) NOT NULL  
);
```

★ UNIQUE: IDENTIFICA DE MANERA ÚNICA A CADA FILA DE UNA TABLA.

• EJEMPLO CON MYSQL



```
CREATE TABLE PERSONAS (  
  ID int NOT NULL,  
  Nombre varchar(255) NOT NULL,  
  Apellido varchar(255) NOT NULL,  
  UNIQUE (ID) → la columna ID tiene un valor  
  diferente para cada fila.  
);
```

- SE PUEDE AÑADIR CON ALTER TABLE
- SE PUEDE CREAR RESTRICCIONES PARA VARIAS COLUMNAS A LA VEZ.

```
{ ALTER TABLE PERSONAS  
  ADD UNIQUE (ID)
```

★ PRIMARY KEY: IDENTIFICA DE MANERA ÚNICA CADA FILA DE UNA TABLA.

La columna definida como clave primaria debe ser UNIQUE y NOT NULL (no puede contener valores nulos).

★ FOREIGN KEY: ES una columna o varias columnas que sirven para señalar cual es la clave primaria de otra tabla.

Las columnas especificadas como FOREIGN KEY, solo podran tener valores que ya existen en la clave primaria de la otra tabla.

```
CREATE TABLE pedidos (  
  OrdenID INT NOT NULL,  
  NumeroOrden INT NOT NULL,  
  IDPersona int,  
  PRIMARY KEY (OrdenID),  
  FOREIGN KEY (IDPersona) REFERENCES personas(IDPersona)  
)
```



★ CHECK: SE utiliza para limitar el rango de valores que puede tener una columna.

- Se pueden definir varias restricciones CHECK en una tabla.

★ DEFAULT: SE utiliza para proporcionar un valor predeterminado para una columna.

Si no se especifica un valor al insertar una fila, entonces se pondrá el valor por defecto (DEFAULT) que tenga cada columna.



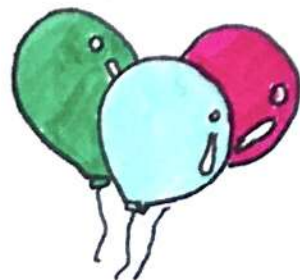


## ✦ Modificación de tablas:

La instrucción ALTER TABLE se usa para agregar, eliminar o modificar columnas de una tabla.

### Añadir columna

```
ALTER TABLE nombreTabla  
ADD nombreColumna tipoDato;
```



### Borrar columna

```
ALTER TABLE nombreTabla  
DROP COLUMN nombreColumna;
```

### Modificar el tipo de dato de una columna

```
ALTER TABLE nombreTabla  
ALTER COLUMN nombreColumna tipoDato
```

## ✦ Borrar tabla:

La instrucción DROP TABLE para eliminar una tabla.



```
DROP TABLE nombreTabla;
```

- DROP INDEX: elimina un índice.
- DROP DATABASE: elimina una base de datos.

# índice

Un índice sirve para buscar datos rápidamente.

Si una columna es un índice de una tabla, al buscar por el valor de esa columna, iremos directamente a la fila correspondiente.

♥ SINTAXIS:

```
CREATE INDEX nombreIndice
```

```
ON nombreTabla (nombreColumna);
```

Admite valores duplicados en su columna.

♥ SINTAXIS PARA ÍNDICE ÚNICO:

```
CREATE UNIQUE INDEX nombreIndice
```

```
ON nombreTabla (nombreColumna);
```

No pueden existir claves duplicadas en el índice.

Ex.

```
CREATE INDEX indice_personas
```

```
ON personas (persona)
```



# Vistas



Una vista es una tabla virtual basada en el conjunto de resultados de una declaración SQL.

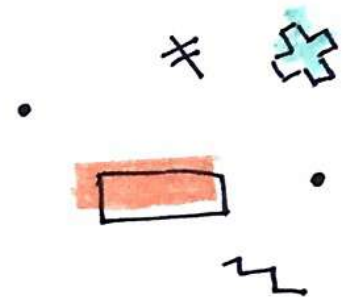
Razones para crear vistas:

- Seguridad.
- Comodidad.

Muestran siempre datos reales de una o varias tablas.

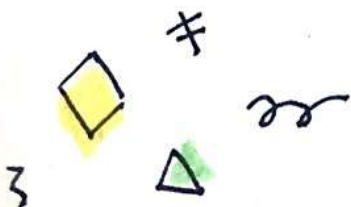
✦ Creación de vistas:

```
CREATE VIEW nombre_vista AS  
SELECT nombre_columna(s)  
FROM nombre_tabla  
WHERE condición;
```



✦ Eliminar vistas:

```
DROP VIEW nombre_vista
```



# Select • From ✨

• **select**: selecciona columnas específicas de nuestra tabla.

**SELECT** columna, otra\_columna, ...

• **FROM** mitabla;

SE UTILIZA PARA ESPECIFICAR LA TABLA ♡♡



El resultado de esta consulta será un conjunto de filas y columnas, pero solo con las **columnas que especificamos**.

Si queremos seleccionar todas las columnas de datos de una tabla, podemos utilizar el asterisco (\*)

**SELECT \***

**FROM** mitabla;

Ejemplo: Hagamos de cuenta que tenemos una tabla llamada 'Películas', de la cual sólo queremos obtener el título de las películas y su año de lanzamiento:

**SELECT** título-pelicula, año-pelicula  
**FROM** películas;

titulo-pelicula y año-pelicula serían los nombres de las columnas que queremos de nuestra tabla. ✨

# Where



La cláusula WHERE de SQL se utiliza para especificar una condición al recuperar un conjunto de datos de una tabla o de un conjunto de tablas.

Si se cumple la condición, la consulta devuelve los valores que se relacionan con la condición que se especifique en la cláusula WHERE.

- SELECT columna, otra\_columna, ...  
FROM mitabla  
WHERE condición  
AND/OR otra\_condición  
AND/OR ...;



Se puede incluir una única cláusula de comparación (llamada condición simple) o múltiples cláusulas utilizando los operadores AND u OR (condición compuesta)

Además, la cláusula WHERE puede especificar una condición utilizando la comparación o los operadores lógicos:

=, !=, <, >, <=, >=	OPERADORES NUMÉRICOS ESTÁNDAR	id_columna != 4
BETWEEN ... AND ...	ESTÁ DENTRO DEL RANGO DE DOS VALORES	id_columna BETWEEN 1.5 AND 10.5
NOT BETWEEN ... AND ...	NO ESTÁ DENTRO DEL RANGO DE DOS VALORES	id_columna NOT BETWEEN 1 AND 10

ENTRE OTROS.



¿Sabías? SQL no requiere que escriba todas las palabras clave en mayúsculas, pero por convención ayuda a las personas a distinguir las palabras clave SQL de los nombres de columnas y tablas.

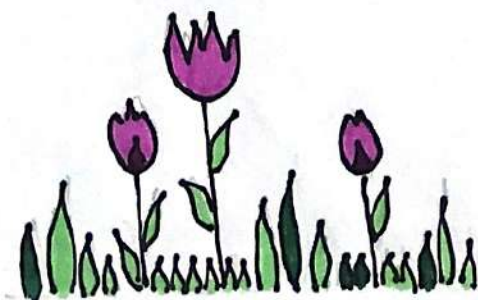


La cláusula WHERE se puede usar con las instrucciones

UPDATE y DELETE además de la instrucción SELECT.

Ej: Si en la tabla 'pelicular' queremos encontrar la película con una fila id de 6:

```
SELECT id, titulo_película FROM películas
WHERE id = 6;
```



Al escribir WHERE con columnas que contienen datos de texto, SQL admite una serie de operadores útiles para hacer cosas tales como la comparación de cadenas que no distingue entre mayúsculas y minúsculas. ✨



OPERADOR	CONDICIÓN	EJEMPLO
=	Comparación de cadena exacta sensible a mayúsculas y minúsculas.	columna_nombre = "abc"
!= o <>	Comparación de desigualdad de cadena exacta sensible a mayúsculas y minúsculas.	columna_nombre != "abcd"
LIKE	Comparación insensible a mayúsculas y minúsculas.	columna_nombre LIKE "ABC"
NOT LIKE	Comparación de desigualdad insensible a mayúsculas y minúsculas.	columna_nombre NOT LIKE "ABCD"
%	SE usa en cualquier lugar de una cadena para que coincida con una secuencia de cero o más caracteres.	columna_nombre LIKE "%a"

ENTRE OTROS...




## Ejemplo:



Seguimos con nuestra tabla 'pelicular' de la cual queremos buscar todas las películas de Toy Story; y su director:

```
SELECT titulo_película, director FROM peliculars
WHERE titulo_película LIKE "Toy Story%";
```



titulo_película	director
Toy Story	John Lasseter
Toy Story 2	John Lasseter
Toy Story 3	Lee Unkrich
Toy Story 4	Josh Cooley



## Distinct

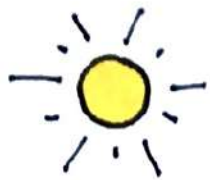
La palabra clave DISTINCT se utiliza para devolver solo valores distintos.

Dentro de una tabla, una columna a menudo contiene muchos valores duplicados; y, a veces, solo deseamos enumerar valores diferentes.

```
SELECT DISTINCT columna, otra_columna
FROM mitabla
WHERE condición(es)
```







# Order by



La mayoría de los datos en bases de datos reales se agregan sin ningún orden en particular.

Para ayudar con esto, SQL proporciona una forma de ordenar el conjunto de resultados en orden ascendente o descendente, mediante la palabra clave **ORDER BY**.

Por defecto, se ordena de forma ascendente (con ORDER BY), si queremos ordenar por orden descendente se utiliza la palabra DESC.



```
SELECT columna, otra_columna, ...
```

```
FROM mitabla
```

```
ORDER BY columna ASC/DESC
```

Ejemplo: Si de nuestra tabla 'películas' quisiéramos enumerar todos los directores, alfabéticamente y sin duplicados:

```
SELECT DISTINCT director FROM películas
```

```
ORDER BY director ASC;
```



Con ORDER BY, comúnmente se utilizan las cláusulas **LIMIT** y

## **OFFSET.**


- ✦ **LIMIT**: ESPECIFICA UN NÚMERO LIMITADO DE FILAS QUE DEVOLVERÁ LA CONSULTA.
- ✦ **OFFSET**: OPCIONAL. LA PRIMERA FILA DEVUELTA POR LIMIT SERÁ DETERMINADA POR OFFSET.

Veamos un ejemplo:

DE NUESTRA TABLA 'peliculas' QUEREMOS OBTENER 5 peliculas de Pixar (**LIMIT**) ordenadas alfabéticamente (**ORDER BY**)

CONTANDO DESDE LA TERCER FILA:

```
SELECT titulo-pelicula FROM peliculas  
ORDER BY titulo-pelicula ASC  
LIMIT 5 OFFSET 2;
```



OMITE LA PRIMERA  
Y SEGUNDA FILA,  
DEVOLVERÁ LAS 5 SIGUIENTES.

titulo-pelicula
Cars
Cars 2
Finding Nemo
Monsters University
Monsters, Inc.

# NULL

¿QUÉ ES UN **valor null**? : un campo con un valor NULL es un campo **sin valor**.

un valor nulo es diferente a un valor CERO o un campo que CONTIENE ESPACIOS.

No es posible probar valores NULL con operadores de comparación como =, < o >.

Tendremos que usar los operadores IS NULL y IS NOT NULL.



```
SELECT columna  
FROM mitabla  
WHERE columna IS NULL;
```



```
SELECT columna  
FROM mitabla  
WHERE columna IS NOT NULL;
```



# insert into

INSERT se utiliza para añadir datos a una tabla existente.

La palabra clave VALUES se utiliza para pasar los valores a insertar en las columnas especificadas.

INSERT INTO puede escribirse de dos formas:

- ♥ con valores para todas las columnas:

```
INSERT INTO mitabla  
VALUES (valor1, valor2, valor3, ...);
```

Cada nuevo registro  
→ de datos debe contener  
valores para cada  
columna correspondiente  
en la tabla.

- ♥ para columnas específicas:

```
INSERT INTO mitabla  
(columna1, columna2, ...)  
VALUES (valor1, valor2, ...);
```

En estos casos, el número  
de valores debe coincidir  
con el número de columnas  
especificadas.

Ex:

Agregamos a nuestra tabla una nueva película de Pixar:

```
INSERT INTO peliculas  
VALUES (4, "Brave", "Brenda Chapman", 2012)
```

↓                      ↓                      ↓                      ↓  
id                      título                      directora                      año de estreno  
                            de la  
                            película



# — update



Se utiliza para actualizar los datos existentes de una tabla.

```
UPDATE mitabla
```

```
SET columna1= valor1, columna2=valor2, ...
```

```
WHERE condición;
```

⚠ La cláusula WHERE especifica qué registro (s) se deben actualizar. Si se omite la cláusula WHERE, se actualizarán todos los registros de la tabla.

Ex:

En nuestra tabla 'pelicular', el director de 'Bichos' es incorrecto, en realidad fue dirigido por John Lasseter.

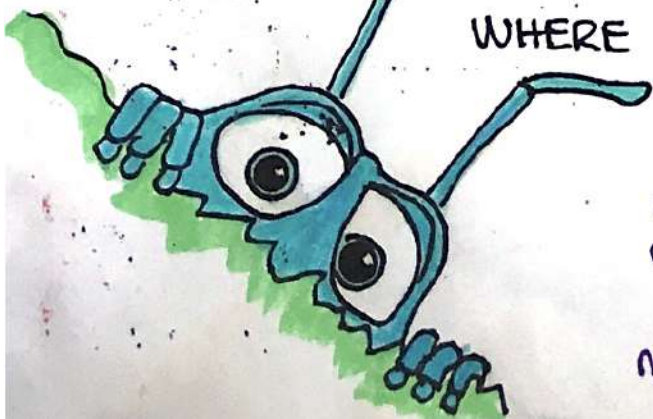
Entonces, actualizamos:

```
UPDATE peliculas  
SET director = "John Lasseter"  
WHERE id=2;
```

columna que se actualizará

ESTE ES EL ID DE LA PELICULA 'Bichos' DE NUESTRA TABLA.

id	titulo-pelicula
1	Toy Story
2	Bichos
3	Cars



# delete



Cuando necesitamos eliminar datos de una tabla utilizamos la declaración DELETE, que especifica la tabla sobre la que actuar, y las filas son especificadas por la cláusula WHERE.

```
DELETE mitabla  
WHERE condición;
```



⚠ Si omitimos WHERE, se eliminan todas las filas.

Ejemplo: ♥

Borraremos de nuestra tabla todas las películas que se lanzaron antes del 2000:

```
DELETE FROM películas  
WHERE año_pelicula < 2000;
```



# funciones MIN() · MAX()

La función **MIN()** devuelve el valor más pequeño de la columna seleccionada.

```
SELECT MIN(columna)
FROM mitabla
WHERE condición;
```



La función **MAX()** devuelve el valor más grande de la columna seleccionada.

```
SELECT MAX(columna)
FROM mitabla
WHERE condición;
```

# count



La función **COUNT()** devuelve el número de filas de la consulta, es decir, el número de registros que cumplen una determinada condición.

```
SELECT COUNT(columna)
FROM mitabla
```

```
SELECT COUNT(*) FROM mitabla
```

**Nota:** los valores NULL se ignoran.

Devolverá el número de filas de una tabla.



# AVG()

15

La función AVG() devuelve el valor promedio de una columna específica. (numérica)

```
SELECT AVG (columna)  
FROM mitabla  
WHERE condición;
```



# ✦ :: SUM()

La función SUM() devuelve la suma total de una columna numérica.

```
SELECT SUM (columna)  
FROM mitabla  
WHERE condición;
```





# ... alias ...

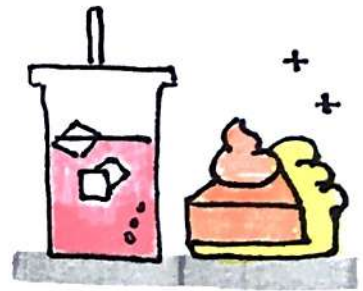


Los alias en SQL se utilizan para dar un nombre temporal a una tabla, o una columna de una tabla.

Son utilizados a menudo para hacer que los nombres de las columnas sean más legibles.

Columna {  
SELECT columna AS nombre-alias  
FROM mitabla;

Tabla {  
SELECT columna(s)  
FROM mitabla AS nombre-alias;



Los alias pueden resultar útiles cuando:

- ♥ Hay más de una tabla involucrada en una consulta.
- ♥ Las funciones se utilizan en la consulta.
- ♥ Los nombres de las columnas son grandes o poco legibles.
- ♥ Dos o más columnas se combinan juntas.

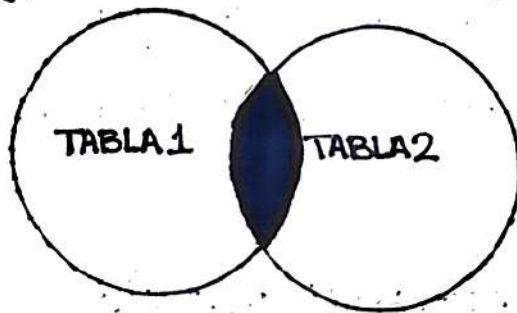


# JOIN

La cláusula **JOIN** se utiliza para combinar filas de dos o más tablas, según una columna relacionada entre ellas.

Existen diferentes tipos de JOIN en SQL:

- INNER JOIN: ES UN PROCESO QUE HACE COINCIDIR FILAS DE LA PRIMERA TABLA CON LAS DE LA SEGUNDA TABLA QUE TIENEN LA MISMA CLAVE (QUE SE DEFINE CON ON) PARA CREAR UN RESULTADO CON LAS COLUMNAS COMBINADAS DE AMBAS TABLAS.



¿Sabías?

Es posible que veas consultas en las que INNER JOIN se escriben simplemente como JOIN.



Veamos un ejemplo:

## Peliculas

id	titulo_pelicula	año_pelicula	director
1	Toy Story	1995	John Lasseter
2	Bichos	1998	John Lasseter
3	Monsters Inc	2001	Pete Docter
4	Cars	2006	John Lasseter
5	UP	2008	Pete Docter

## Calificación

pelicula-id	rating	duracion_pelicula
3	8.2	92
2	7.4	95
1	8.5	81
5	8.3	101
4	7.2	117

Y quisiéramos enumerar las películas por su calificación:

```
SELECT titulo_pelicula, rating
FROM peliculas
JOIN calificacion
ON peliculas.id = calificacion.peliculas-id
ORDER BY rating DESC;
```



titulo_pelicula	rating
Toy Story	8.5
UP	8.3
Monsters Inc	8.2
Bichos	7.4
Cars	7.2

+  
.  
+

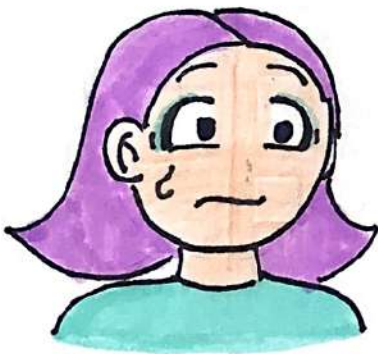
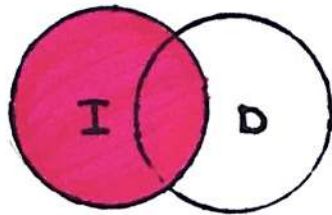
+  
.  
+



# OUTER JOINS

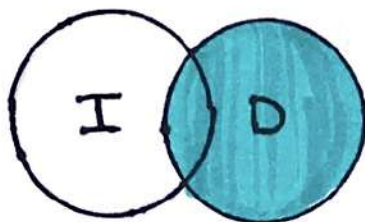


- LEFT JOIN: Devuelve **TODOS** los registros de la tabla izquierda (aunque no cumplan condiciones) y los registros coincidentes de la tabla derecha.

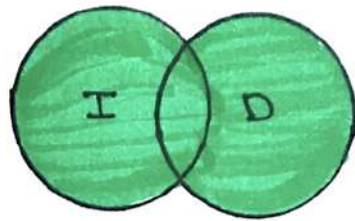


```
SELECT columna, otra.columna
FROM tabla1
LEFT JOIN tabla2 (LEFT/RIGHT/FULL JOIN)
ON tabla1.id = otra.tabla.blabla_id
WHERE condición
ORDER BY columna, ... ASC/DESC
:
```

- RIGHT JOIN: Devuelve **TODOS** los registros de la tabla derecha (aunque no cumplan las condiciones) y los registros coincidentes de la tabla izquierda.



- FULL JOIN: DEVUELVE **TODOS** los registros, tanto de la tabla izquierda como la derecha, aunque no cumplan las condiciones.



```
SELECT columna, otra.columna  
FROM tabla1  
FULL JOIN tabla2  
ON tabla1.id = otra_tabla.blabla_id  
WHERE condición(s)  
ORDER BY columna ASC/DESC  
LIMIT...
```

# SQL UNION



La sentencia SQL UNION se utiliza para combinar el conjunto de resultados de dos o más sentencias SELECT.

Las dos sentencias SELECT tienen que tener el mismo número de columnas, con el mismo tipo de dato y en el mismo orden.



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

# group By

La instrucción **GROUP BY** se utiliza para juntar filas de resultados que coincidan en el valor de alguna columna seleccionada.



```
SELECT columna(s)  
FROM tabla  
WHERE condición  
GROUP BY columna(s);
```

→ SE PUEDE AGRUPAR EN VARIAS COLUMNAS.

# having

**Having** se utiliza para incluir condiciones con alguna función SQL del tipo SUM, MAX...

Como la cláusula WHERE no se puede utilizar con esas funciones, utilizamos HAVING.

```
SELECT columna1, SUM(columna2)  
FROM tabla  
GROUP BY columna  
HAVING SUM(columna2) < número;
```

→ **función**

**condición**