

# SQL. Curso completo de SQL. aprende desde cero. Comandos SQL UDEMY

Ing. Luis Felipe Narvaez Gomez. E-mail: [ing.felipenarvaez017@gmail.com](mailto:ing.felipenarvaez017@gmail.com)

\*APUNTES\*

Desarrollo > Diseño y desarrollo de bases de datos > SQL


## SQL. Curso completo de SQL. Aprende desde cero. Comandos SQL

Aprende SQL desde cero para saber manejar cualquier base de datos

4,4 ★★★★★ (2.795 calificaciones) 9.269 estudiantes

Creado por [Redait Media](#)

Última actualización: 4/2022 🌐 Español 🗣️ Español



Vista previa de este curso

**Has comprado este curso el Sep. 30, 2021**

[Ir al curso](#)

Garantía de reembolso de 30 días

**Este curso incluye:**

- 2.5 horas de video bajo demanda
- 1 artículo
- 2 recursos descargables
- Acceso de por vida
- Acceso en dispositivos móviles y TV
- Certificado de finalización

[Compartir](#) [Regalar este curso](#)

### Lo que aprenderás

- ✓ Usar SQL para hacer cualquier tipo de consulta de una base de datos
- ✓ Aprenderas a crear, modificar y optimizar cualquier base de datos
- ✓ Escribir consultas SQL complejas para una o varias tablas
- ✓ Al finalizar este curso, tendrá el conocimiento para administrar y utilizar mejor cualquier base de datos

### Contenido del curso

6 secciones • 86 clases • 2 h 44 m de duración total

[Ampliar todas las secciones](#)

## DESCRIPCION DE CURSO

Link:

<https://www.udemy.com/course/sql-curso-completo-de-sql-aprende-desde-cero/learn/lecture/13191096#overview>

Descripción General:

Acerca de este curso: Aprende SQL desde cero para saber manejar cualquier base de datos

Por cifras:

Nivel de habilidad: Todos los niveles  
Estudiantes: 9271  
Idiomas: Español  
Subtítulos: Sí  
Clases: 86  
Video: 3 horas en total

Certificados: Consigue el certificado de Udemy al completar todo el curso

Características: Disponible en iOS y Android

Descripción:

Aprenderás a realizar consultas básicas y avanzadas, realizar actualizaciones y modificaciones sobre cualquier base de datos, mediante el lenguaje SQL. Estos conocimientos de SQL sirven para cualquier base de datos del mercado: MySQL, Oracle, DB2, SQL Server, y muchas más. ¡Aprender SQL es una de las formas más rápidas para mejorar tus perspectivas profesionales, ya que es una de las habilidades tecnológicas más demandadas actualmente! ¡En este curso aprenderás rápidamente, mediante ejemplos con explicaciones cortas y sencillas!

Lo que aprenderás

- Usar SQL para hacer cualquier tipo de consulta de una base de datos
- Escribir consultas SQL complejas para una o varias tablas
- Aprenderás a crear, modificar y optimizar cualquier base de datos
- Al finalizar este curso, tendrá el conocimiento para administrar y utilizar mejor cualquier base de datos
- ¿Hay requisitos para realizar el curso?
- No hay requisitos previos. Solo será necesario disponer de un ordenador si se quiere practicar con los ejemplos explicados en el curso
- ¿Para quién es este curso?
- Cualquier persona interesada en aprender sobre SQL para el manejo de bases de datos o para empezar con el análisis de datos

Instructores

Redait Media Ingeniería de software

REDAIT MEDIA es una empresa de IT especializada en software con personal titulado en Ingeniería en Informática con más de 20 años de experiencia en el desarrollo del software.

Somos expertos en la gestión de bases de datos con SQL, y hemos querido compartir con vosotros este curso completo de SQL.

También somos expertos en lenguajes de programación y próximamente crearemos algunos cursos sobre programación.

Valoraciones

4.4

Calificación: 4.386113 de 5

Valoración del curso

Introducción a SQL

SQL es un lenguaje de programación para el acceso a bases de datos

SQL se utiliza para acceder y manipular datos en cualquier base de datos de mercado, como por ejemplo:

MySQL  
ORACLE  
DB2  
SQL Server  
etc.

SQL se compone de sentencias, cada una con una utilidad diferente y las vamos a estudiar en dos secciones, SQL Básico y SQL avanzado; con multitud de ejemplos para entenderlo perfectamente.

## SQL BASICO

Estas son las sentencias básicas para SQL:

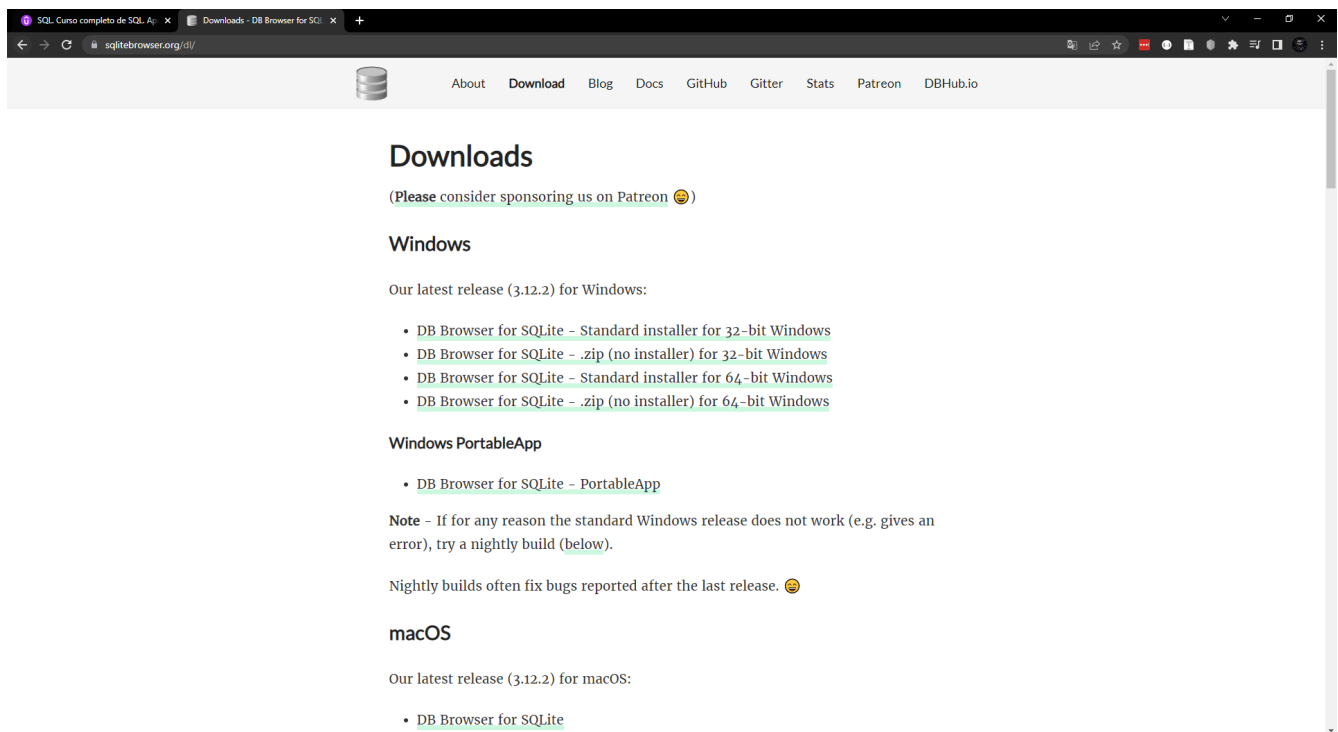
SELECT --> Para consultar datos  
WHERE --> para incluir condiciones  
ORDER BY --> para ordenar resultados  
INSERT --> para insertar datos  
UPDATE --> para actualizar datos  
DELETE --> para borrar datos

## SQL AVANZADO

Estas son las sentencias de SQL de forma de uso avanzado:

LIMIT --> para limitar número de registros  
LIKE --> para buscar por un patrón  
IN --> para seleccionar por múltiples valores  
BETWEEN --> para seleccionar por rango de valores  
ALIAS --> para renombrar  
JOIN --> para combinar tablas  
UNION --> para combinar varios resultados  
CREATE TABLE --> para crear una tabla  
NULL --> para no indicar ningún valor  
UNIQUE --> para identificar de manera única  
PRIMARY KEY --> para identificar cada fila de manera única  
FOREIGN KEY --> para identificar la clave primaria de otra tabla  
DROP --> para borrar completamente una tabla o base de datos  
TRUNCATE --> para borrar solo los datos de la tabla, no la estructura  
CREATE VIEW --> para crear vistas de una tabla  
ALTER TABLE

El Software con el que trabajara este curso es <https://sqlitebrowser.org/> se descarga el necesario para la computadora que se este trabajando.

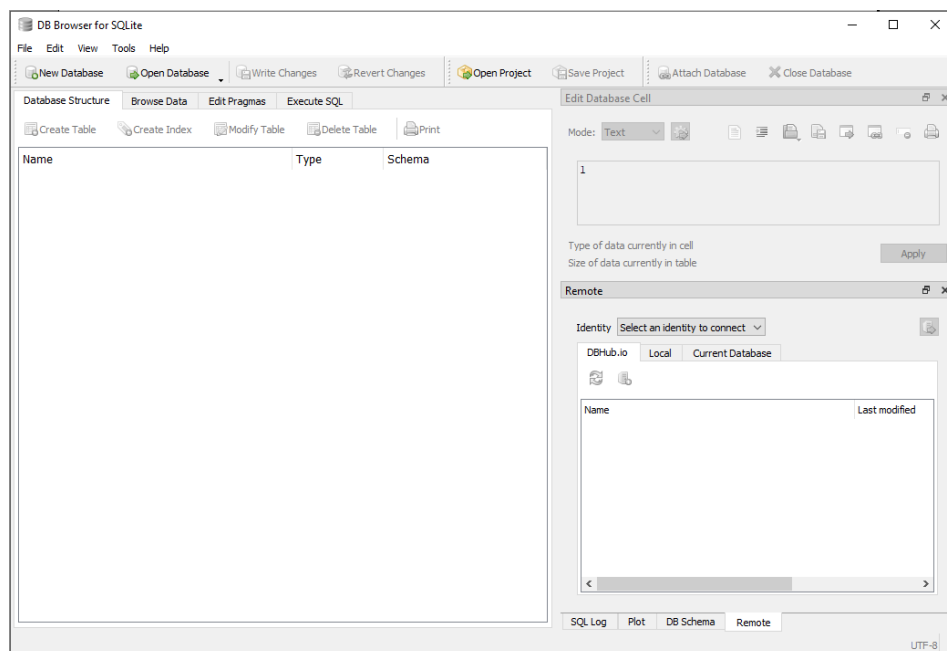


En mi caso seleccione la version para x64-bit Windows. En mi caso lo instale en un destino a parte del sugerido por el programa (Disco Local C es la ruta por defecto), en principio el funcionamiento propio del software no se ve alterado por esto.

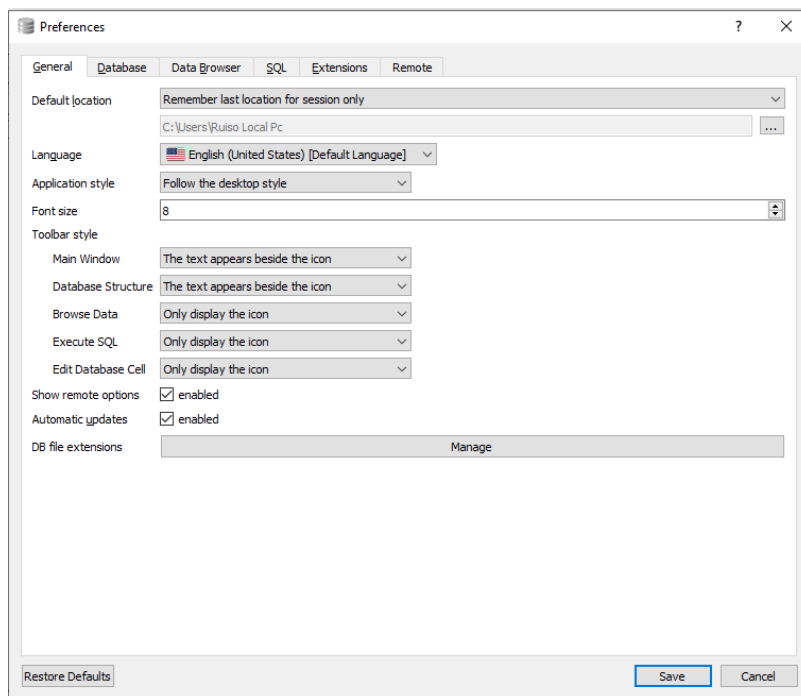
En la computadora se instalan dos accesos para utilizar el Software:

- DB Browser (SQLite)
- DB Browser (SQLCipher)

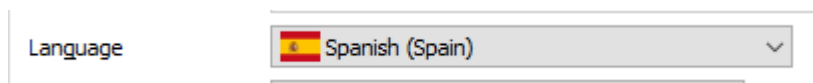
Para razones de este curso, en princpio solo se utilizara la primera opcion. La vista al ejecutar este programa es el siguiente:



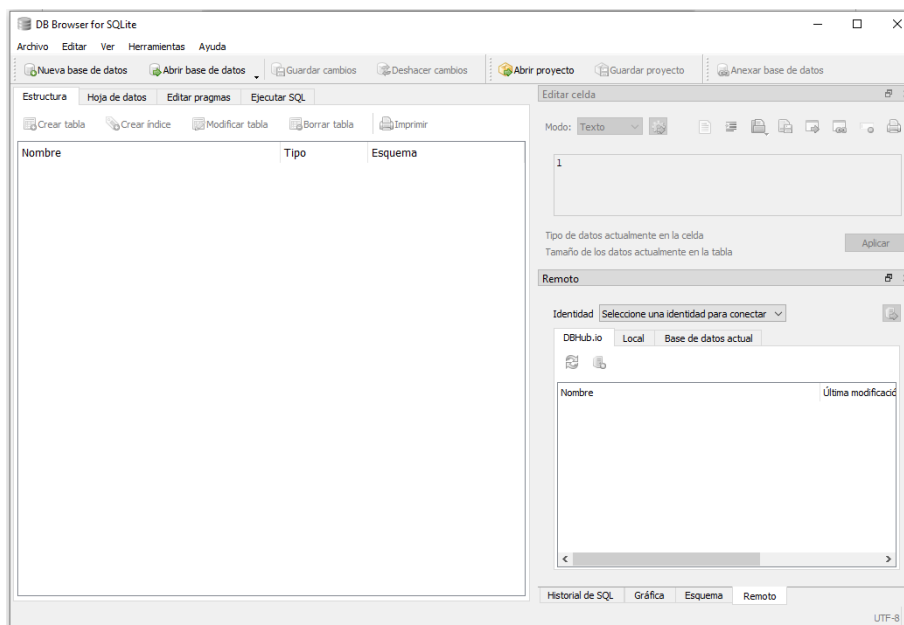
Cambiar el idioma de la aplicación a español llenando a la barra de menus del programa, seleccionar EDIT y en el seleccionar la opción de PREFERENCES.

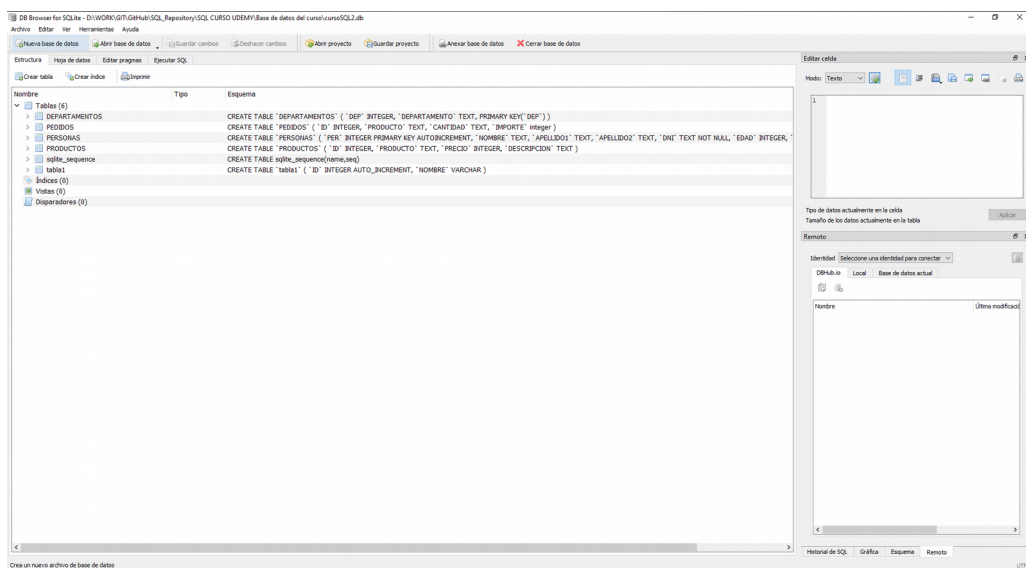


En la sección de GENERAL, la parte destinada a LANGUAGE cambio el idioma a mi preferencia.



Luego doy clic izquierdo en el borde inferior de la ventana SAVE para guardar los cambios. Se debe Reiniciar la aplicación para en su próxima ejecución observar los cambios realizados.





# SELECT

Esta es una de las sentencias mas importantes en el uso del lenguaje SQL para la utilidad de base de datos. La sentencia SELECT permite realizar consultas sobre los datos que contiene una tabla en especifico en una base de datos.

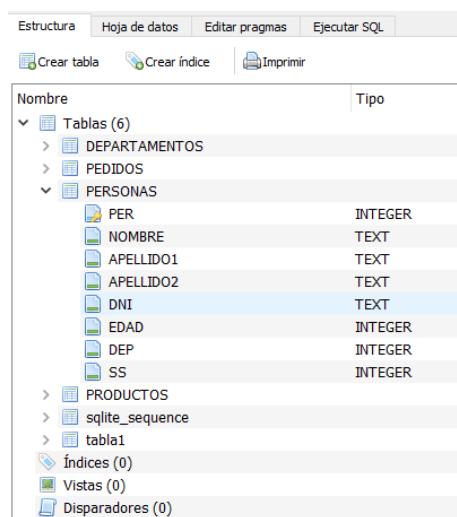
La sintaxis de uso seria la siguiente:

```
SELECT columna_1, columna_2, columna_n  
FROM nombre_tabla;
```

En caso de querer traer todos los registros, se puede utilizar \* para nombrar todas las columnas de la tabla.

Por ejemplo:

( En el Software que estamos utilizando, para hacer una consulta SQL nos dirigimos a la pestaña de EJECUTAR SQL en caso contrario de ver solo la estructura de la base de datos nos dirigimos a ESTRUCTURA y en caso de solo ver los datos con las tablas ir a HOJAS DE DATOS).



La consulta SQL con la sentencia SELECT para mostrar todo el material de una tabla especifica puede ser con \* o con los nombres de las columnas.

```
SELECT *  
FROM PERSONAS;
```

```
SELECT PER, NOMBRE, APELLIDO1, APELLIDO2, DNI, EDAD, DEP, SS  
FROM PERSONAS;
```

Estructura Hoja de datos Editar pragmas Ejecutar SQL							
SQL 1							
1 SELECT * FROM PERSONAS;							
PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2 NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1 NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3 NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2 NULL

Estructura Hoja de datos Editar pragmas Ejecutar SQL							
SQL 1							
1 SELECT PER, NOMBRE, APELLIDO1, APELLIDO2, DNI, EDAD, DEP, SS FROM PERSONAS;							
PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2 NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1 NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3 NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2 NULL

En caso de solo querer mostrar una sola columna o algunas columnas en especifico, solo se deben poner dichas columnas.

**SELECT PER, DNI, EDAD FROM PERSONAS;**

Estructura

Hoja de datos

Edita pragmas

Ejecutar SQL

SQL 1

1

SELECT PER, DNI, EDAD FROM PERSONAS;

	PER	DNI	EDAD
1	1	32887345L	46
2	2	30234863P	23
3	3	29345120	56
4	4	35987125A	35

## DISTINCT

Esta sentencia es especial. Cuando nosotros realizamos una consulta, puede que existan valores repetidos. Es decir que para una columna o varias de estas, los valores almacenados se repitan. Dependiendo de lo que queramos consultar, aveces contar valores repetidos no es una buena accion y ahi donde entra la consulta DISTINCT que evita estos valores repetidos reflejados en la consulta. DISTINCT no funciona con varias columnas a la vez.

Por Ejemplo:

**SELECT NOMBRE  
FROM PERSONAS;**

**SELECT DISTINCT NOMBRE  
FROM PERSONAS;**



1	SELECT NOMBRE FROM PERSONAS;
	NOMBRE
1	ANTONIO
2	LUIS
3	ANTONIO
4	PEDRO

1	SELECT DISTINCT NOMBRE FROM PERSONAS;
	NOMBRE
1	ANTONIO
2	LUIS
3	PEDRO

## WHERE

La sentencia o clausula WHERE se utiliza para hacer un filtro en una consulta, es decir, es empleada cuando queremos que solo se nos refleje una consulta que cumple con un filtro o una premisa o una determinada condicion, esta condicion puede escribirse entre comillas simples o dobles '' "".

Por ejemplo:

```
SELECT NOMBRE, EDAD
FROM PERSONAS
WHERE NOMBRE = 'ANTONIO';
```

1	SELECT NOMBRE, EDAD
2	FROM PERSONAS
3	WHERE NOMBRE = 'ANTONIO';
	NOMBRE EDAD
1	ANTONIO 46
2	ANTONIO 56

Hay veces en que las consultas por letras y los valores en las tablas no coinciden por tener diferencia de mayusculas y minusculas. Una forma de arreglar esto es llevar tanto la consulta como el valor a minuscular **LOWER** o el valor y la consulta a mayusculas **UPPER**.

```
SELECT NOMBRE, EDAD
FROM PERSONAS
WHERE lower(NOMBRE) = lower('ANTONIO');
```

1	SELECT NOMBRE, EDAD
2	FROM PERSONAS
3	WHERE lower(NOMBRE) = lower('ANTONIO');
	NOMBRE EDAD
1	ANTONIO 46
2	ANTONIO 56

```
SELECT NOMBRE, EDAD
FROM PERSONAS
WHERE upper(NOMBRE) = upper('ANTONIO');
```

1	SELECT NOMBRE, EDAD
2	FROM PERSONAS
3	WHERE upper(NOMBRE) = upper('ANTONIO');

	NOMBRE	EDAD
1	ANTONIO	46
2	ANTONIO	56

A la vez las condiciones que se utilizan con WHERE pueden llevar **operadores numericos** tales como >, <, >=, <= y =.

```
SELECT NOMBRE, EDAD
FROM PERSONAS
WHERE EDAD >= 24;
```

1	SELECT NOMBRE, EDAD
2	FROM PERSONAS
3	WHERE EDAD >= 24;

	NOMBRE	EDAD
1	ANTONIO	46
2	ANTONIO	56
3	PEDRO	35

Tambien podemos utilizar otros operadores como los son AND y OR. El condicional AND solo traera una consulta cuando se cumpla la primera condicion y la segunda condicion, es decir ambas premisas se cumplan ( 1 AND 2); que que el operador OR traera la consulta mientras que alguna de las dos condiciones se complan, es decir que se cumpla la primera y no la segunda o que no se cumpla la primera pero si la segunda ( 1 OR 2).

```
SELECT NOMBRE, EDAD
FROM PERSONAS
WHERE EDAD >= 24 AND EDAD <= 35;
```

```

1  SELECT NOMBRE, EDAD
2  FROM PERSONAS
3  WHERE EDAD >= 24 AND EDAD <= 35;

```

	NOMBRE	EDAD
1	PEDRO	35

**SELECT NOMBRE, EDAD**  
**FROM PERSONAS**  
**WHERE EDAD >= 24 OR EDAD <= 35;**

```

1  SELECT NOMBRE, EDAD
2  FROM PERSONAS
3  WHERE EDAD >= 24 OR EDAD <= 35;

```

	NOMBRE	EDAD
1	ANTONIO	46
2	LUIS	23
3	ANTONIO	56
4	PEDRO	35

**SELECT NOMBRE, EDAD**  
**FROM PERSONAS**  
**WHERE lower(NOMBRE) = lower("LUIS") AND EDAD >= 20**

```

1  SELECT NOMBRE, EDAD
2  FROM PERSONAS
3  WHERE lower(NOMBRE) = lower("LUIS") AND EDAD >= 20

```

	NOMBRE	EDAD
1	LUIS	23

**SELECT NOMBRE, EDAD**  
**FROM PERSONAS**  
**WHERE lower(NOMBRE) = lower("LUIS") OR EDAD >= 20**

```

1  SELECT NOMBRE, EDAD
2  FROM PERSONAS
3  WHERE lower(NOMBRE) = lower("LUIS") OR EDAD >= 20

```

	NOMBRE	EDAD
1	ANTONIO	46
2	LUIS	23
3	ANTONIO	56
4	PEDRO	35

## ORDER BY

Es una sentencia que se utiliza para ordenar los datos de una consulta según los datos de una columna específica, el orden puede darse de manera ascendente o descendente del orden alfabético en caso de tratarse de datos que contienen letras o en orden numérico en caso de números.

Hay que tener en cuenta que el orden predeterminado de la sentencia es ASC que es ascendente, en caso de no querer este se debe escribir DESC de descendente. También se puede especificar ASC sin problemas.

Ejemplo:

```
SELECT NOMBRE, APELLIDO1, APELLIDO2, EDAD
FROM PERSONAS
ORDER BY EDAD ASC;
```

1	SELECT NOMBRE, APELLIDO1, APELLIDO2, EDAD			
2	FROM PERSONAS			
3	ORDER BY EDAD ASC			
	NOMBRE	APELLIDO1	APELLIDO2	EDAD
1	LUIS	LOPEZ	PEREZ	23
2	PEDRO	RUIZ	GONZALEZ	35
3	ANTONIO	PEREZ	GOMEZ	46
4	ANTONIO	GARCIA	BENITO	56

```
SELECT NOMBRE, APELLIDO1, APELLIDO2, EDAD
FROM PERSONAS
ORDER BY EDAD DESC;
```

1	SELECT NOMBRE, APELLIDO1, APELLIDO2, EDAD			
2	FROM PERSONAS			
3	ORDER BY EDAD DESC			
	NOMBRE	APELLIDO1	APELLIDO2	EDAD
1	ANTONIO	GARCIA	BENITO	56
2	ANTONIO	PEREZ	GOMEZ	46
3	PEDRO	RUIZ	GONZALEZ	35
4	LUIS	LOPEZ	PEREZ	23

```
SELECT NOMBRE, APELLIDO1, APELLIDO2, EDAD
FROM PERSONAS
ORDER BY NOMBRE;
```

```

1 SELECT NOMBRE, APELLIDO1, APELLIDO2, EDAD
2 FROM PERSONAS
3 ORDER BY NOMBRE;

```

	NOMBRE	APELLIDO1	APELLIDO2	EDAD
1	ANTONIO	PEREZ	GOMEZ	46
2	ANTONIO	GARCIA	BENITO	56
3	LUIS	LOPEZ	PEREZ	23
4	PEDRO	RUIZ	GONZALEZ	35

**SELECT NOMBRE, APELLIDO1, APELLIDO2, EDAD  
FROM PERSONAS  
ORDER BY NOMBRE DESC;**

```

1 SELECT NOMBRE, APELLIDO1, APELLIDO2, EDAD
2 FROM PERSONAS
3 ORDER BY NOMBRE DESC;

```

	NOMBRE	APELLIDO1	APELLIDO2	EDAD
1	ANTONIO	PEREZ	GOMEZ	46
2	ANTONIO	GARCIA	BENITO	56
3	LUIS	LOPEZ	PEREZ	23
4	PEDRO	RUIZ	GONZALEZ	35

## INSERT INTO

La sentencia INSERT INTO se utiliza para registrar nuevos datos en una tabla, los datos corresponden a una fila y se debe tener extremo cuidado de los nombres correctos de las columnas de la tabla especificada, su orden y que en caso de que los datos en la tabla no permitan que no sean nulos, procurar tener un valor en ellos.

Los datos iniciales de la tabla seran:

**SELECT \*  
FROM PERSONAS;**

```

1 SELECT *
2 FROM PERSONAS;

```

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL

Para insertar una nueva fila de datos podemos crear la sentencia de dos maneras, la mejor forma es en la que necesitamos cada uno de los diferentes nombres de las columnas que componen la tabla a la que agregaremos el dato, la otra forma es procurar que el orden de los valores que escribamos coincida con el orden de las columnas reales. Se debe tener precaucion con las columnas de valores que no permiten

nulos y los que ya tienen valores de tipo serial, pues los mismos se generan automáticamente y no hay necesidad de describirlos.

Debemos tener en cuenta que los valores de las columnas de los datos que insertemos tengan el mismo valor de los especificados en la tabla, es decir, en caso de que una columna solo reciba enteros, solo podremos almacenar enteros. Los datos que contienen letras o caracteres especiales diferentes de dígitos se escriben entre comillas sencillas.

Por último en caso de que una columna en su creación se le haya creado la propiedad de UNIQUE no podremos albergar valores que ya existen en la tabla así pertenezcan a otra fila. UNIQUE es una sentencia que permite crear listados únicos de datos para una tabla, ampliamente utilizado en llaves primarias, documentos de identidad, etc.

Ejemplo:

**INSERT INTO PERSONAS (PER, NOMBRE, APELLIDO1, APELLIDO2, DNI, EDAD, DEP, SS) VALUES (5,'Luis','Narvaez','Gomez',456789123,100,2,0);**

```
1  INSERT INTO PERSONAS (PER,NOMBRE,APELLIDO1,APELLIDO2,DNI,EDAD,DEP,SS)
2  VALUES (5,'Luis','Narvaez','Gomez',456789123,100,2,0);
3
4  SELECT *
5  FROM PERSONAS;
```

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL
5	5	Luis	Narvaez	Gomez	456789123	100	2	0

**INSERT INTO PERSONAS VALUES (6,'Felipe','Narvaez','Gomez',456789123,100,2,0);**

```
1  INSERT INTO PERSONAS
2  VALUES (6,'Felipe','Narvaez','Gomez',456789123,100,2,0);
3
4  SELECT *
5  FROM PERSONAS;
```

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL
5	5	Luis	Narvaez	Gomez	456789123	100	2	0
6	6	Felipe	Narvaez	Gomez	456789123	100	2	0

## UPDATE

La sentencia UPDATE se utiliza para actualizar o editar los dato de una tabla. Para que funcione es comun utilizar la sentencia WHERE en conjunto para delimitar la actualizacion a una fila especificada o un rango de datos.

Ejemplo:

Los datos originales son:

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL
5	5	Luis	Narvaez	Gomez	456789123	100	2	0
6	6	Felipe	Narvaez	Gomez	456789123	100	2	0

Suponiendo que la edad de la fila 6 es erronea, actualizaremos ese dato.

**UPDATE PERSONAS**

**SET EDAD = 64**

**WHERE lower(NOMBRE) = lower('Felipe');**

```
1 UPDATE PERSONAS
2 SET EDAD = 64
3 WHERE lower(NOMBRE) = lower('Felipe');
4
5 SELECT *
6 FROM PERSONAS;
```

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL
5	5	Luis	Narvaez	Gomez	456789123	100	2	0
6	6	Felipe	Narvaez	Gomez	456789123	64	2	0

## DELETE

La sentencia DELETE se utiliza para eliminar una fila de datos de una tabla, al igual que la sentencia UPDATE se utiliza en conjunto de la sentencia WHERE para delimitar el campo de edicion a una fila o rango de filas especificas, esto junto con los operadores.

Ejemplo:

Los datos iniciales son:

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL
5	5	Luis	Narvaez	Gomez	456789123	100	2	0
6	6	Felipe	Narvaez	Gomez	456789123	64	2	0

Eliminar una fila seria tal que:

**DELETE**  
**FROM PERSONAS**  
**WHERE lower(NOMBRE) = lower('Luis') AND upper(APELLIDO1)=upper('Narvaez');**

1	<b>DELETE FROM PERSONAS</b>							
2	<b>WHERE lower(NOMBRE) = lower('Luis') AND upper(APELLIDO1)=upper('Narvaez');</b>							
3								
4	<b>SELECT *</b>							
5	<b>FROM PERSONAS;</b>							

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL
5	6	Felipe	Narvaez	Gomez	456789123	64	2	0

**NOTA: Los Ejercicios de SQL BASICO Propuestos en el curso para esta seccion pueden hallarse en la carpeta de repositorio dentro de un archivo sql.**

## LIMIT

La sentencia LIMIT se utiliza para especificar el numero de filas que se quieres mostrar en una consulta.

Ejemplo:

Si quisieramos ver toda la tabla seria:

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL
5	6	Felipe	Narvaez	Gomez	456789123	64	2	0

Si solo quisieramos ver las dos primeras seria:



**SELECT \***  
**FROM PERSONAS LIMIT 2**

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL

## LIKE

LIKE no es como tal una sentencia por si sola, si no mas bien un operador que se utiliza junto con la clausula WHERE cuando se quiee buscar por un patron.

Ejemplo:

Por ejemplo, podemos preguntar por aquellos valores que contengan la letra S en la columna NOMBRE.

**SELECT \***  
**FROM PERSONAS**  
**WHERE NOMBRE LIKE '%S%';**

1	SELECT *
2	FROM PERSONAS
3	WHERE NOMBRE LIKE '%S%';

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL

O que empiece por una o varias letras

**SELECT \***  
**FROM PERSONAS**  
**WHERE NOMBRE LIKE 'A%';**

1	SELECT *
2	FROM PERSONAS
3	WHERE NOMBRE LIKE 'A%';

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL

O Que termine por una letra.

**SELECT \***  
**FROM PERSONAS**  
**WHERE NOMBRE LIKE '%E';**

1	SELECT *
2	FROM PERSONAS
3	WHERE NOMBRE LIKE '%E';

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	6	Felipe	Narvaez	Gomez	456789123	64	2	0

## IN

la sentencia IN no funciona por si sola, asi como vimos en la sentencia LIKE, esta es mas bien un operador que se concatena con la sentencia WHERE y permite seleccionar multiples valores en la clausula, es decir, una lista de valores.

Ejemplo:

Si la tabla original es:

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL
5	6	Felipe	Narvaez	Gomez	456789123	64	2	0

Y solo queremos ver las personas que tienen el apellido 1 de Perez, Garcia y Ruiz ; tenemos:

```
SELECT *
FROM PERSONAS
WHERE lower(APELLIDO1) IN(lower('perez'), lower('garcia'), lower('ruiz'));
```

1	SELECT *
2	FROM PERSONAS
3	WHERE lower(APELLIDO1) IN(lower('perez'), lower('garcia'), lower('ruiz'));

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
3	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL

## BETWEEN

La sentencia BETWEEN no funciona por si sola, es un operador que se utiliza con la clausula WHERE para seleccionar filas que esten dentro de un rango de datos que especifiquemos. Al tratarse de rangos trabajaremos con datos de tipo fecha, horas o numeros. Conjuntamente para enmarcar el rango de A a B utilizamos tambien el operador AND.

Ejemplo:

```
SELECT *  
FROM PERSONAS  
WHERE EDAD BETWEEN '20' AND '40';
```

1	SELECT *							
2	FROM PERSONAS							
3	WHERE EDAD BETWEEN '20' AND '40';							

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
2	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL

## ALIAS “AS”

Los alias AS se utilizan para renombrar durante el tiempo de ejecución de la consulta el nombre de las columnas que nos pueden ser largos o complicados. Dependiendo de la rama de SQL como PostgreSQL a diferencia de MySQL, se puede utilizar AS para tener un objeto al cual referirse a toda una tabla por completo, esto se hace con frecuencia en el momento de hacer consultas que requiera la combinación de datos de diferentes tablas con sentencias avanzadas como INNER.

Ejemplo:

si tuviéramos una consulta originalmente como:

```
SELECT PER, NOMBRE, APELLIDO1  
FROM PERSONAS;
```

1	SELECT PER, NOMBRE, APELLIDO1		
2	FROM PERSONAS;		

	PER	NOMBRE	APELLIDO1
1	1	ANTONIO	PEREZ
2	2	LUIS	LOPEZ
3	3	ANTONIO	GARCIA
4	4	PEDRO	RUIZ
5	6	Felipe	Narvaez

Podríamos renombrar las columnas como:

```
SELECT PER AS 'Id' , NOMBRE As 'Name', APELLIDO1 AS 'Lastname'  
FROM PERSONAS;
```

```

1 SELECT PER AS 'Id' , NOMBRE As 'Name', APELLIDO1 AS 'Lastname'
2 FROM PERSONAS;

```

	Id	Name	Lastname
1	1	ANTONIO	PEREZ
2	2	LUIS	LOPEZ
3	3	ANTONIO	GARCIA
4	4	PEDRO	RUIZ
5	6	Felipe	Narvaez

## JOIN

El uso de la sentencia JOIN, muchas veces en conjunto con la sentencia ON, nos permite hacer consultas entre varias tablas. Esto es gracias a las llaves primarias y foraneas que existen entre tablas. Para poder hacer consultas entre varias tablas, las mismas deben estar relacionadas por una o varias de sus columnas, esto es mediante las columnas de la PRIMARY KEY y las FOREIGN KEY.

Ejemplo:

Tenemos la tabla Personas.

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL
5	6	Felipe	Narvaez	Gomez	456789123	64	2	0

Tenemos la tabla Departamentos.

	DEP	DEPARTAMENTO
1	1	ADMINISTRACION
2	2	INFORMATICA
3	3	COMERCIAL

Como podemos ver, la tabla de personas y la tabla de departamentos estan unidas por la columna DEP de personas que seria la FOREIGN KEY que llama la PRIMARY KEY de la tabla departamentos la columna DEP. Ambas poseen el mismo nombre aunque en principio se podria utilizar nombres distintos y aun asi que las columnas esten enlazadas entre si.

```

SELECT *
FROM PERSONAS
JOIN DEPARTAMENTOS
WHERE PERSONAS.DEP = DEPARTAMENTOS.DEP;

```

1	SELECT *									
2	FROM PERSONAS									
3	JOIN DEPARTAMENTOS									
4	WHERE PERSONAS.DEP = DEPARTAMENTOS.DEP;									

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS	DEP	DEPARTAMENTO
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL	2	INFORMATICA
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL	1	ADMINISTRACION
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL	3	COMERCIAL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL	2	INFORMATICA
5	6	Felipe	Narvaez	Gomez	456789123	64	2	0	2	INFORMATICA

right left ambos join con alias sin alias

Ahora con alias seria:

```
SELECT *
FROM PERSONAS AS P
JOIN DEPARTAMENTOS AS D
WHERE P.DEP = D.DEP;
```

1	SELECT *									
2	FROM PERSONAS AS P									
3	JOIN DEPARTAMENTOS AS D									
4	WHERE P.DEP = D.DEP;									

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS	DEP	DEPARTAMENTO
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL	2	INFORMATICA
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL	1	ADMINISTRACION
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL	3	COMERCIAL
4	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL	2	INFORMATICA
5	6	Felipe	Narvaez	Gomez	456789123	64	2	0	2	INFORMATICA

Existen tres tipos de JOIN el que utilizamos anteriormente es el basico pero existe el INNER, RIGHT y LEFT. Estos se utilizan siempre que tanto la tabla A que este unida a una tabla B aunque compartan columnas en comun, no siempre tengan datos en todas sus filas que confluyan entre ambas tablas, lo mas comun es que tengan un valor Nulo.

Lo anterior se puede imaginar teniendo a la tabla personas con algunas de sus filas en nulos o la tabla departamentos con alguno de sus valores en nulos, estos nulos en la columna que comparten siendo DEP. Si fuese el caso tendríamos tres formas de hacer la consulta.

**INNER**, mostraria una consulta que muestre los nulos de ambas tablas, tanto de A como de B.

**RIGHT** mostraria los nulos de A y por tanto no pondria un valor de B, evitando el conflicto de busqueda.

**LEFT** mostraria los nulos de B y por tanto no pondria valores de A, evitando asi conflictos de busqueda.

Comunmente se utiliza mas el INNER JOIN que los demas. En este tipo de consultas, ademas es posible utilizar la sentencia ON como sustituto de la WHERE esto dado que ademas de hacer JOIN podemos utilizar clausulas o filtros de busqueda, ademas de las alias con AS y limitar numero de columnas visibles.

Ejemplo:

```
SELECT P.PER, P.NOMBRE, D.DEPARTAMENTO
FROM PERSONAS AS P
INNER JOIN DEPARTAMENTOS AS D
ON P.DEP = D.DEP
WHERE EDAD>=40;
```

1	SELECT P.PER, P.NOMBRE, D.DEPARTAMENTO
2	FROM PERSONAS AS P
3	INNER JOIN DEPARTAMENTOS AS D
4	ON P.DEP = D.DEP
5	WHERE EDAD>=40;

	PER	NOMBRE	DEPARTAMENTO
1	1	ANTONIO	INFORMATICA
2	3	ANTONIO	COMERCIAL
3	6	Felipe	INFORMATICA

## UNION

La sentencia UNION se utiliza para acumular resultados de dos sentencias SELECT, estas mismas del numero con el mismo numero de columnas, mismos datos y mismo orden de columnas. Comunmente se utiliz apara poder hacer consultas separadas y juntar sus resultados en una sola respuesta.

Ejemplo:

```
SELECT * FROM PERSONAS WHERE EDAD>=40
UNION
SELECT * FROM PERSONAS WHERE DEP = 1;
```

1	SELECT * FROM PERSONAS WHERE EDAD>=40
2	UNION
3	SELECT * FROM PERSONAS WHERE DEP = 1;

	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	6	Felipe	Narvaez	Gomez	456789123	64	2	0

Quiza en la anterior tabla podemos obtener el mismo resultado solo añadiendo a la primera consulta una clausula como WHERE (EDAD>=40 OR DEP=1) y hubiera funcionado de la misma manera; sin embargo esto se da ya que es una consulta de bajo nivel, en alto nivel las consultas pueden complicarse a un punto dado donde la utilizacion de la sentencia UNION puede ser casi obligatoria para sacarnos de mas de un apuro.

# CREATE TABLE

Una de las sentencias mas utilizadas en el momento de crear una tabla dentro de una base de datos.

Ejemplo:

La creacion de tablas va por niveles, las tablas pueden tener un sin fin de variaciones en su creacion, entre ellas estan:

- Dentro de una tabla se pueden tener datos tablas aisladas de toda la base de datos como tener aquellas que estan enlazadas a muchas otras tablas.
- Se puede tener tablas con una o ninguna llave primaria, del mismo modo se puede tener una o ninguna llave foranea
- Los datos pueden tener muchos tipos de valor dependiendo de la informacion que se quiere almacenar.
- Una tabla puede ser hija o mama de otra con ayuda de las llave primarias y foraneas.
- La tabla puede o no almacenar datos nulos como condicion desde su contruccion.
- La tabla puede tener un comportamiento predefinido al momento de actualizar o borrar un dato o una fila de datos con respecto a si misma y las tablas asociadas a este.
- Las tablas pueden protegerse de edicion al proclamar a un usuario como unico que puede actualizar o borrar los datos o la misma tabla.
- Se puede o no categorizar los datos de una columna en la tabla como unicos, esto haciendo que cada dato sea unico e irrepetible a comparacion de cada uno en la misma columna de cada una de las filas de la tabla.
- Se puede crear un indice o no en las tablas.
- Algunos datos pueden ser de tipo especial como el Tipo SERIAL.
- Etcetera.

No exploraremos todas las opciones en este archivo pues muchas dependen de sentencias unicas que veremos mas adelante pero seran exploradas a medida que se vayan utilizando. La forma mas simple de crear una tabla es:

```
CREATE TABLE prueba(  
    id_prueba INTEGER,  
    columna_prueba TEXT,  
    descripcion_prueba TEXT  
);
```

```
1 CREATE TABLE prueba(  
2     id_prueba INTEGER,  
3     columna_prueba TEXT,  
4     descripcion_prueba TEXT  
5 );  
6  
7 SELECT * FROM prueba;
```

```
Ejecución terminada sin errores.  
Resultado: 0 filas devueltas en 8ms  
En la línea 7:  
SELECT * FROM prueba;
```

En este caso no se devuelven filas de las tablas debido a que no tenemos datos dentro dentro. Si insertamos valores tendríamos:

```
INSERT INTO prueba (id_prueba, columna_prueba, descripcion_prueba)
VALUES(1,"dato1","descripcion 1");
INSERT INTO prueba (id_prueba, columna_prueba, descripcion_prueba)
VALUES(2,"dato2","descripcion 2");
```

```
1  INSERT INTO prueba (id_prueba, columna_prueba, descripcion_prueba)
2  VALUES(1,"dato1","descripcion 1");
3  INSERT INTO prueba (id_prueba, columna_prueba, descripcion_prueba)
4  VALUES(2,"dato2","descripcion 2");
5
6  SELECT * FROM prueba;
```

	id_prueba	columna_prueba	descripcion_prueba
1	1	dato1	descripcion 1
2	2	dato2	descripcion 2

## NOT NULL

La sentencia NOT NULL es mas una restriccion y sirve para el momento de hacer un CREATE TABLE especificar que se quiere que las columnas no puedan contener datos nulos. En caso de que al momento nosotros intentemos guardar un dato como nulo, nos aparecera un error de esto. La columna con NOT NULL no puede estar vacia.

Ejemplo:

```
CREATE TABLE prueba1(
    id_prueba INTEGER NOT NULL,
    columna_prueba TEXT NOT NULL,
    descripcion_prueba TEXT NOT NULL
);
```

```
INSERT INTO prueba1 (id_prueba, columna_prueba, descripcion_prueba)
VALUES(1,"dato1","descripcion 1");
```

```
INSERT INTO prueba1 (id_prueba, columna_prueba, descripcion_prueba)
VALUES(1,"dato2","descripcion 2");
```

```
SELECT * FROM prueba1;
```



1	CREATE TABLE prueba1(
2	id_prueba INTEGER NOT NULL,
3	columna_prueba TEXT NOT NULL,
4	descripcion_prueba TEXT NOT NULL
5	);
6	
7	INSERT INTO prueba1 (id_prueba, columna_prueba, descripcion_prueba)
8	VALUES(1,"dato1","descripcion 1");
9	INSERT INTO prueba1 (id_prueba, columna_prueba, descripcion_prueba)
10	VALUES(1,"dato2","descripcion 2");
11	
12	SELECT * FROM prueba1;

	id_prueba	columna_prueba	descripcion_prueba
1	1	dato1	descripcion 1
2	1	dato2	descripcion 2

## UNIQUE

la sentencia UNIQUE es una restriccion que se utiliza al momento de crear una tabla y nos dice que se identificara de manera unida cada fila de una tabla. Esta restriccion se utiliza cuando queremos que un valor de una columna sea unico entre tablas, por ejmplo al tener los identificadores de ciudadania de una pais, cada persona tiene un ID unico entre toda la poblacion. La sentencia UNIQUE puede utilizarse en una o varias columnas de una tabla tal cual sea pertinente.

Ejemplo:

Hay dos maneras de nombras un dato como unico, la primera forma es ponerlo dentro del CREATE TABLE y el segundo es nombrarlo por fuera como un indice unico.

Primera Forma:

```

1 CREATE TABLE prueba2(
2     id_prueba SERIAL NOT NULL,
3     cedula_ciudadania INTEGER UNIQUE NOT NULL,
4     columna_prueba TEXT NOT NULL,
5     descripcion_prueba TEXT NOT NULL
6 );

```

Segunda Forma:

```

13 CREATE UNIQUE INDEX cedula_unica ON prueba2 (cedula_ciudadania);

```

Para este ejemplo yo utilizare la segunda forma por comodidad y costumbre ( en este ejemplo se utiliza el tipo de dao SERIAL para el id\_prueba, esto hace que los valores de esta columna sean generados automaticamente de forma numerica secuencial, por lo que no tenemos que indicar este valor dentro del INSERT INTO en la mayoria de Programas de alto nivel, sin embargo en el que se esta trabajando en este curso requiere que sea nombrado).

```

CREATE TABLE prueba2(
    id_prueba SERIAL NOT NULL,
    cedula_ciudadania INTEGER NOT NULL,
    columna_prueba TEXT NOT NULL,
    descripcion_prueba TEXT NOT NULL
);

INSERT INTO prueba2 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba)
VALUES(1,205001, "dato1","descripcion 1");

INSERT INTO prueba2 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba)
VALUES(2,205002, "dato2","descripcion 2");

CREATE UNIQUE INDEX cedula_unica ON prueba2 (cedula_ciudadania);

SELECT * FROM prueba2;

```

1	CREATE TABLE prueba2(
2	id_prueba SERIAL NOT NULL,
3	cedula_ciudadania INTEGER NOT NULL,
4	columna_prueba TEXT NOT NULL,
5	descripcion_prueba TEXT NOT NULL
6	);
7	
8	INSERT INTO prueba2 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba)
9	VALUES (1,205001, "dato1","descripcion 1");
10	INSERT INTO prueba2 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba)
11	VALUES (2,205002, "dato2","descripcion 2");
12	
13	CREATE UNIQUE INDEX cedula_unica ON prueba2 (cedula_ciudadania);
14	
15	SELECT * FROM prueba2;

	id_prueba	cedula_ciudadania	columna_prueba	descripcion_prueba
1	1	205001	dato1	descripcion 1
2	2	205002	dato2	descripcion 2

## PRIMARY KEY

La sentencia PRIMARY KEY en español traducido sería LLAVE PRIMARIA , identifica de manera unica a cada fila que compone una tabla de datos, la columna que se nombra como PRIMARY KEY debe tener como valores datos que sean unicos, no pueden ser nulos y deben ser de un unico valor. Cada tabla de nuestra base de datos, solo puede tener una sola llave primaria, en caso de que exista una que contenga mas, seguramente se trate de tablas especiales como las tablas de rompimiento, pero no es el alcance de este curso.NOTA: dentro del repositorio se encuentran trabajos con bases de datos y otros apuntes donde se explica como utilizar este tipo de tablas especiales y restricciones/sentencias de mayor nivel.

Ejemplo:

Hay tres formas de nombrar una PRIMARY KEY, la primera es directamente dentro de CREATE TABLE, la segunda tambien dentro del CREATE TABLE pero haciendo uso de la sentencia CONSTRAINT y la tercera fuera del CREATE TABLE es un juego de varias sentencias que parten de ALTER TABLE.

La primera seria:

```
1 CREATE TABLE prueba3(  
2     id_prueba SERIAL PRIMARY KEY NOT NULL,  
3     cedula_ciudadania INTEGER NOT NULL,  
4     columna_prueba TEXT NOT NULL,  
5     descripcion_prueba TEXT NOT NULL  
6 );
```

La segunda seria:

```
1 CREATE TABLE prueba3(  
2     id_prueba SERIAL NOT NULL,  
3     cedula_ciudadania INTEGER NOT NULL,  
4     columna_prueba TEXT NOT NULL,  
5     descripcion_prueba TEXT NOT NULL,  
6     CONSTRAINT PK_PRUEBA PRIMARY KEY (cedula_ciudadania)  
7 );
```

Y la tercera seria:

```
16 ALTER TABLE prueba2 ADD CONSTRAINT PK_PRUEBA PRIMARY KEY (cedula_ciudadania);  
17
```

En mi caso prefiero trabajar con la segunda forma.

```
CREATE TABLE prueba3(  
    id_prueba SERIAL NOT NULL,  
    cedula_ciudadania INTEGER NOT NULL,  
    columna_prueba TEXT NOT NULL,  
    descripcion_prueba TEXT NOT NULL,  
    CONSTRAINT PK_PRUEBA PRIMARY KEY (id_prueba)  
);  
CREATE UNIQUE INDEX cedula ON prueba3 (cedula_ciudadania);  
  
INSERT INTO prueba3 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba)  
VALUES(1,205001, "dato1","descripcion 1");  
INSERT INTO prueba3 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba)  
VALUES(2,205002, "dato2","descripcion 2");  
  
SELECT * FROM prueba3;
```

1	CREATE TABLE prueba3(
2	id_prueba SERIAL NOT NULL,
3	cedula_ciudadania INTEGER NOT NULL,
4	columna_prueba TEXT NOT NULL,
5	descripcion_prueba TEXT NOT NULL,
6	CONSTRAINT PK_PRUEBA PRIMARY KEY (cedula_ciudadania)
7	);
8	CREATE UNIQUE INDEX cedula ON prueba3 (cedula_ciudadania);
9	
10	INSERT INTO prueba3 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba)
11	VALUES(1,205001, "dato1","descripcion 1");
12	INSERT INTO prueba3 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba)
13	VALUES(2,205002, "dato2","descripcion 2");
14	
15	SELECT * FROM prueba3;

	id_prueba	cedula_ciudadania	columna_prueba	descripcion_prueba
1	1	205001	dato1	descripcion 1
2	2	205002	dato2	descripcion 2

## FOREIGN KEY

la sentencia FOREIGN KEY o traducida al español como LLAVE FORANEA es utilizada como restriccion que permite señalar cual es el dato de otra de una columna de nuestra tabla que pertenece o esta enlazada a otra tabla.

La llave externa o llave foranea es normalmente la llave primaria de otra tabla sin embargo no existe restriccion de tomar otra columna diferente de otra tabla, sin embargo por lo general se toma la primary key.

Puede haber tantas llaves foraneas como se quiera y de distintas tablas, lo importante es que estos valores solo pueden tomar valores ya existentes en las otras tablas a las que apuntan.

La tabla que da su PRIMARY KEY a otra para que sea utilizada como FOREIGN KEY y asi queden enlazadas, convierte a la primera en tabla Mama y a la segunda en tabla Hija, pues visto de otro modo, hereda esos datos de A a B.

Ejemplo:

Tal y como sucedia con la llave Primaria hay dos formas de añadir una llave Foranea. La primera se parece a la segunda de PRIMARY KEY sin embargo no utiliza la sentencia CONSTRAINT si no la sentencia REFERENCES la cual permite referenciar la columna a una tabla y a una columna de esa tabla en concreto.

La segunda forma de hacerlo es con ayuda de la sentencia ALTER TABLE la misma permite alterar las propiedades de una tabla una vez a sido creada, en este caso nos permite designar una columna de la tabla como llave foranea, con el añadido que podemos designar comportamientos propios en caso de que el valor de la llave foranea cambie o se elimine.

Para enlazar con una llave primaria de otra tabla debemos ver el croquis de creacion de la misma, el que tenga el icono de una llave en amarillo, sera la columna que tenga el vaor de Primary Key. Vamos a enlazar la tabla con los datos de departamanetos. La llave primaria es DEP.

	DEP	DEPARTAMENTO
1	1	ADMINISTRACION
2	2	INFORMATICA
3	3	COMERCIAL
4	4	PRUEBA
5	5	PRUEBA

Tablas (10)
DEPARTAMENTOS
DEP INTEGER
DEPARTAME... TEXT

La Primera Forma de hacerlo seria:

```
1 CREATE TABLE prueba4(
2     id_prueba SERIAL NOT NULL,
3     cedula_ciudadania INTEGER NOT NULL,
4     columna_prueba TEXT NOT NULL,
5     descripcion_prueba TEXT NOT NULL,
6     departamento INTEGER NOT NULL,
7     CONSTRAINT PK_PRUEBA_4 PRIMARY KEY (id_prueba),
8     FOREIGN KEY ('departamento') REFERENCES 'DEPARTAMENTOS' ('DEP')
9 );
```

La segunda Forma y la que mas me gusta utilizar es con ayuda de la sentencia ALTER TABLE:

```
ALTER TABLE prueba4 ADD CONSTRAINT FK_PRUEBA_4 FOREIGN KEY (departamento)
REFERENCES DEPARTAMENTOS(DEP)
ON DELETE RESTRICT
ON UPDATE CASCADE;
```

Asi tendríamos lo siguiente:

```
CREATE TABLE prueba4(
    id_prueba SERIAL NOT NULL,
    cedula_ciudadania INTEGER NOT NULL,
    columna_prueba TEXT NOT NULL,
    descripcion_prueba TEXT NOT NULL,
    departamento INTEGER NOT NULL,
    CONSTRAINT PK_PRUEBA_4 PRIMARY KEY (id_prueba)
);
CREATE UNIQUE INDEX cedula_4 ON prueba4 (cedula_ciudadania);
ALTER TABLE prueba4 ADD CONSTRAINT FK_PRUEBA_4 FOREIGN KEY (departamento)
REFERENCES DEPARTAMENTOS(DEP)
ON DELETE RESTRICT
ON UPDATE CASCADE;
```

```
INSERT INTO prueba4 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba,
departamento)
VALUES(1,205001, "dato1","descripcion 1",1);
INSERT INTO prueba4 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba,
departamento)
```

VALUES(2,205002, "dato2","descripcion 2",5);

SELECT \* FROM prueba4;

```
1 CREATE TABLE prueba4(  
2     id_prueba SERIAL NOT NULL,  
3     cedula_ciudadania INTEGER NOT NULL,  
4     columna_prueba TEXT NOT NULL,  
5     descripcion_prueba TEXT NOT NULL,  
6     departamento INTEGER NOT NULL,  
7     CONSTRAINT PK_PRUEBA_4 PRIMARY KEY (id_prueba)  
8 );  
9 CREATE UNIQUE INDEX cedula_4 ON prueba4 (cedula_ciudadania);  
10 ALTER TABLE prueba4 ADD CONSTRAINT FK_PRUEBA_4 FOREIGN KEY (departamento)  
11     REFERENCES DEPARTAMENTOS (DEP)  
12     ON DELETE RESTRICT  
13     ON UPDATE CASCADE;  
14  
15 INSERT INTO prueba4 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba, departamento)  
16     VALUES(1,205001, "dato1","descripcion 1",1);  
17 INSERT INTO prueba4 (id_prueba, cedula_ciudadania, columna_prueba, descripcion_prueba, departamento)  
18     VALUES(2,205002, "dato2","descripcion 2",5);  
19  
20 SELECT * FROM prueba4;
```

	id_prueba	cedula_ciudadania	columna_prueba	descripcion_prueba	departamento
1	1	205001	dato1	descripcion 1	1
2	2	205002	dato2	descripcion 2	5

## CHECK

La sentencia CHECK es una de tipo restriccion que permite limitar el rango de valores que puede tener una columna de una tabla en una base de datos.

Ejemplo:

En el siguiente ejemplo se limita desde la creacion de la tabla que el valor de que se guarde en la columna “edad” tendra que ser mayor a cero. Tambien podria ponerse un rango como mayor a cero y menor a 200 por ejemplo “CHECK( edad > 0 AND edad <200).

```
CREATE TABLE prueba5(  
    id_prueba SERIAL NOT NULL,  
    nombre VARCHAR(250) NOT NULL,  
    edad INT NOT NULL,  
    CONSTRAINT PK_PRUEBA_5 PRIMARY KEY (id_prueba),  
    CHECK(edad > 0)  
);
```

## CREATE INDEX

La sentencia CREATE INDEX se utiliza para crear indices en una tabla. Los indices sirven para buscar una fila concretamente de forma mas rapida. Si una columna es indice de una tabla, al buscar por un valor de esa columna, iremos directamente a la fila correspondiente. Las busquedas son mas optimas de

esta manera pues en cambio de no utilizar indices, se tendria que recorrer de forma secuencial las filas de una tabla hasta hayar la que concretamente necesitamos (Esto requiere mas recursos de sistema). Sin embargo la creacion, edicion y actualizacion de indices es demorado, por lo que se recomienda utilizar indices solo en tablas que utilizamos con mayor frecuencia. Del mismo modo se recomienda crear indices de tipo UNIQUE asi como lo vimos la ves pasada con esta sentencia.

Ejemplo:

Si tenemos la tabla:

```
CREATE TABLE prueba5(  
    id_prueba SERIAL NOT NULL,  
    dni INT NOT NULL,  
    nombre VARCHAR(250) NOT NULL,  
    edad INT NOT NULL,  
    CONSTRAINT PK_PRUEBA_5 PRIMARY KEY (id_prueba),  
    CHECK(edad > 0)  
);
```

Y queremos crear un indice sobre la columna de DNI identificacion ciudadana, podemos utilizar el siguiente comando:

--primera forma  
**CREATE INDEX INDICE\_DNI ON prueba5(dni);**

Sabiendo que el DNI es un numero unico entre personas, podemos utilizar INDEX junto con la sentencia UNIQUE para asegurar la no duplicidad del indice en la base de datos.

--segunda forma  
**CREATE UNIQUE INDEX INDICE\_DNI ON prueba5(dni);**

La busqueda entonces podria realizarse de la siguiente manera:

--La busqueda seria tipo  
**SELECT \* FROM prueba5 WHERE DNI='1049070';**

Lo que sucede internamente en la busqueda es que, si no existiera el INDICE, tendria que recorrer la base de datos fila a fila hasta encontrar el dato correspondiente, pero ahora teniendo el indice, la base de datos ira directamente a este valor sin tener que recorrer tooooooda la tabla.

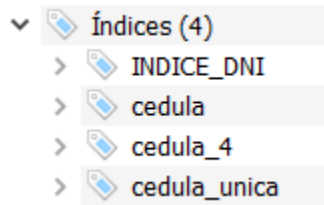
```
1 CREATE TABLE prueba5(  
2     id_prueba SERIAL NOT NULL,  
3     dni INT NOT NULL,  
4     nombre VARCHAR(250) NOT NULL,  
5     edad INT NOT NULL,  
6     CONSTRAINT PK_PRUEBA_5 PRIMARY KEY (id_prueba),  
7     CHECK(edad > 0)  
8 );  
9 --primera forma  
10 CREATE INDEX INDICE_DNI ON prueba5(dni);  
11  
12 --segunda forma  
13 CREATE UNIQUE INDEX INDICE_DNI ON prueba5(dni);  
14  
15 --La busqueda seria tipo  
16 SELECT * FROM prueba5 WHERE DNI='1049070';
```

# DROP

La sentencia DROP se utiliza para BORRAR de manera DEFINITIVA un INDICE, una TABLA o una BASE DE DATOS completa. Despues de su ejecucion, no hay vuelta atrás.

Ejemplo:

Observemos que tenemos varios inidices creados recientemente en la base de datos y queremos eliminarlos.

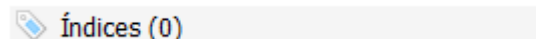


Asi que procedemos a eliminar estos indices con las siguientes sentencias.

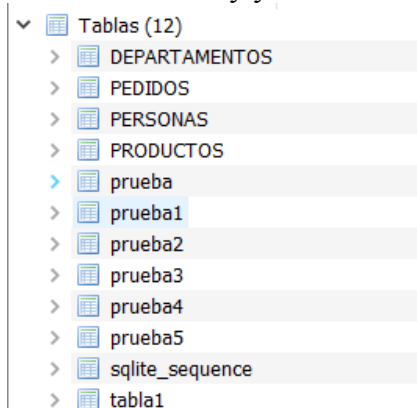
```
1 DROP INDEX INDICE_DNI;
2 DROP INDEX cedula;
3 DROP INDEX cedula_4;
4 DROP INDEX cedula_unica;
```

Ejecución terminada sin errores.  
Resultado: consulta ejecutada con éxito. Tardó 0ms  
En la línea 4:  
DROP INDEX cedula\_unica;

Como resultado tenemos que ya eliminamos estos indices:



Ahora tenemos algunas tablas que podemos borrar. Hay que recordar que una vez eliminemos la tabla, no solo se borrarán todos los datos de todas las filas en todas las columnas de la tabla, si no que también la misma estructura de la tabla de eliminara y ya no existira dicha tabla en la base de datos.



Ahora procedemos a autilizar las sentencias pertinentes para eliminar todas las tablas de prueba.



```

1 DROP TABLE prueba;
2 DROP TABLE prueba1;
3 DROP TABLE prueba2;
4 DROP TABLE prueba3;
5 DROP TABLE prueba4;
6 DROP TABLE prueba5;

```

Ejecución terminada sin errores.  
Resultado: consulta ejecutada con éxito. Tardó 0ms  
En la línea 6:  
DROP TABLE prueba5;

Como Resultado no odremos encontrar la tabla en la base de datos ni la informacion que alvergaba estas tablas.

▼ Tablas (6)

- > DEPARTAMENTOS
- > PEDIDOS
- > PERSONAS
- > PRODUCTOS
- > sqlite\_sequence
- > tabla1

En caso de querer borra una base de datos la sentencia es la misma, en este caso no se ejecutara pero iria asi:

```

1 DROP DATABASE nombre de la base de datos

```

## TRUNCATE

La sentencia TRUNCATE funciona de manera similar a la sentencia DROP, sin embargo se utiliza cuaando queremos eliminar SOLO LOS DATOS de una TABLA, no su estrctura ni desaparecerla de la base de datos.

Ejemplo:

**TRUNCATE TABLE 'Nombre de la tabla';**

## ALTER

La sentencia ALTER la hemos visto con aterioridad en el uso y creacion de llaves primarias y foraneas, sin embargo por si sola se utiliza para añadir, eliminar o modificar columnas de una tabla.

Ejemplo:

Supongamos que ya tenemos una tabla llamada tabla1, esta ya esta en la base de datos pero le faltó crearle una columna.

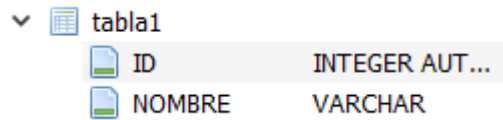


tabla1	ID	INTEGER AUT...
	NOMBRE	VARCHAR

Asi que le vamos a añadir la columna faltante.

**ALTER TABLE tabla1 ADD porque\_aja\_ome VARCHAR(200);**

Dando como resultado:

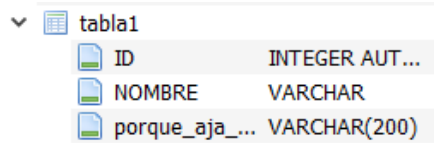


tabla1	ID	INTEGER AUT...
	NOMBRE	VARCHAR
	porque_aja_...	VARCHAR(200)

Pero resulta que la tabla si estaba bien como estaba asi que hay que eliminar esa columna “porque aja ome” y asi aja la eliminamos xD.

**ALTER TABLE tabla1 DROP porque\_aja\_ome;**

Dando como resultado

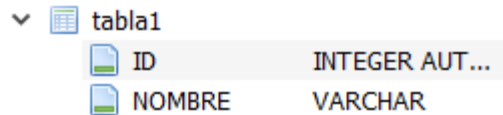


tabla1	ID	INTEGER AUT...
	NOMBRE	VARCHAR

Se debe tener en cuenta que al momento de agregar una columna, esta se rellenara con NULL pues no existen valores para esta columna de manera presedente, ahora bien, si eliminamos la columna con DROP, de la misma manera que funciona la sentencia por si sola, eliminara la columna ( la estructura) y la informacion que estaba en esta tabla.

Las modificaciones sobre las tablas no se limitan a aregar o limitar columnas si no que ALTER es utilizado como ya hemos visto para añadir o liminar PRIMARY KEY , FOREIGN KEY o incluso cambar los tipos de datos de una columna.

## AUTOINCREMENT

La sentencia Auto Increment permite añadir un numero automaticamente de manera incremental a un valor añadido en una tabla, se utiliza normalmente para las llaves primarias ya que los numeros incrementales son unicos y secuenciales y permite diferencia facilmente una fila de datos de otras en una tabla. El incremento es de 1 en 1 y su vaor inicial es 1.

Ejemplo:

```
CREATE TABLE tabla1(
    id_tabla1 INTEGER AUTOINCREMENT NOT NULL,
    descripcion VARCHAR (200) NOT NULL,
    CONSTRAINT PK_Table_1 PRIMARY KEY (id_tabla1)
);
```

Esta sentencia muchas veces es reemplazado con el tipo de dato SERIAL, es decir que la anterior consulta y la siguiente son iguales.

```
CREATE TABLE tabla1(
    id_tabla1 SERIAL NOT NULL,
    descripcion VARCHAR (200) NOT NULL,
    CONSTRAINT PK_Tabla_1 PRIMARY KEY (id_tabla1)
);
```

Sin embargo algunas veces es necesario utilizar el AUTOINCREMENT y no el tipo de dato SERIAL ya que algunos MOTORES de desarrollo de bases de datos no trabajan con SERIAL. En cualquiera de los dos casos al momento de rellenar utilizando la sentencia INSERT INTO el campo de la columna con autoincrement o serial no hay necesidad de de rellenarlo o nombrarlo dentro de la linea de codigo; pues la misma se auto rellenara de manera incremental y cumpliendo a propiedad de que cada nueva fila posea en la columna un dato unico.

## CREATE VIEW

La sentencia CREATE VIEW se utiliza para realizar una vista de una tabla a partir de una consulta que utiliza la sentencia SELECT. Las vistas muestran siempre datos reales de una o de varias tablas. En el momento que se hace una consulta sobre una vista del sistema de base de datos, la misma actualiza los datos para mostrar siempre los reales dentro de una tabla y lo que se esta preguntando en la consulta.

Ejemplo:

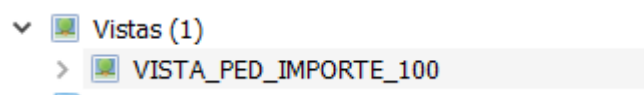
Una consulta general sobre la tabla de PEDIDOS seria normalmente asi:

```
SELECT * FROM PEDIDOS;
```

	ID	PRODUCTO	CANTIDAD	IMPORTE
1	1	IMPRESORA	2	100
2	2	RATON	1	10
3	3	ORDENADOR	1	500
4	4	IMPRESORA	3	150
5	5	ORDENADOR	1	600

Ahora pensemos que queremos crear una VISTA de aquellos pedidos que tengan un importe mayor de 100.

```
CREATE VIEW VISTA_PED_IMPORTE_100
AS SELECT *
FROM PEDIDOS
WHERE IMPORTE>=100;
```



Ahora hemos creado la vista, la vista contiene la consulta que hemos dado por mas compleja que pueda


ser esta. En adelante podemos consultar esta vista y nos arrojará los resultados de la consulta con la que se creo. Los resultados siempre serán aquellos más actualizados con la tabla.

**SELECT \* FROM VISTA\_PED\_IMPORTE\_100;**

	ID	PRODUCTO	CANTIDAD	IMPORTE
1	1	IMPRESORA	2	100
2	3	ORDENADOR	1	500
3	4	IMPRESORA	3	150
4	5	ORDENADOR	1	600

En caso de que queramos eliminar la vista podemos conjugarla con la sentencia DROP de la siguiente manera:

**DROP VIEW VISTA\_PED\_IMPORTE\_100;**

 Vistas (0)

## NULL

No es propiamente una sentencia, sin embargo se encarga de definir o representar un valor como desconocido o como vacío. Los valores Nulos pueden estar en cualquier tipo de dato y en cualquier columna de cualquier tabla de una base de datos, otra cosa es si es mejor o no dependiendo la circunstancia de tener esos valores.

Los valores nulos no pueden operarse en consultas de  $>$  o  $<$  o  $=$  sin embargo tiene sus propios operadores especiales.

Ejemplo:

En el momento de crear una tabla podemos definir si las columnas pueden albergar o no datos nulos. En caso de que no se especifique, el valor predeterminado está en permitir que se puedan insertar valores Nulos. Si se quieren valores nulos se puede tener:

```
CREATE TABLE nombre_tabla (  
    columna_1 TIPODATO NULL,  
    columna_2 TIPODATO NULL,  
    columna_3 TIPODATO NULL  
);
```

o también se puede por defecto:

```
CREATE TABLE nombre_tabla (  
    columna_1 TIPODATO,  
    columna_2 TIPODATO,  
    columna_3 TIPODATO  
);
```

En caso de que no se quiera nulos en las columnas se puede tener:

```
CREATE TABLE nombre_tabla (  
    columna_1 TIPODATO NOT NULL,  
    columna_2 TIPODATO NOT NULL,  
    columna_3 TIPODATO NOT NULL  
);
```

Ahora pensemos en hacer una consulta sobre una tabla, por ejemplo:

```
SELECT *  
FROM PRODUCTOS;
```

1	SELECT *
2	FROM PRODUCTOS;

	ID	PRODUCTO	PRECIO	DESCRIPCION
1	1	IMPRESORA	50	IMPRESORA DE TINTA
2	2	RATON	10	RATON OPTICO
3	3	ORDENADOR	500	NULL
4	4	TECLADO	15	NULL

Si queremos ver los datos nulos utilizaremos IS NULL.

```
SELECT *  
FROM PRODUCTOS  
WHERE DESCRIPCION IS NULL;
```

1	SELECT *
2	FROM PRODUCTOS
3	WHERE DESCRIPCION IS NULL;

	ID	PRODUCTO	PRECIO	DESCRIPCION
1	3	ORDENADOR	500	NULL
2	4	TECLADO	15	NULL

Para NO querer ver filas con datos nulos utilizaremos IS NOT NULL

```
SELECT *  
FROM PRODUCTOS  
WHERE DESCRIPCION IS NOT NULL;
```

1	SELECT *
2	FROM PRODUCTOS
3	WHERE DESCRIPCION IS NULL;

	ID	PRODUCTO	PRECIO	DESCRIPCION
1	3	ORDENADOR	500	NULL
2	4	TECLADO	15	NULL

En caso de querer insertar un valor en una tabla que sea nulo, simplemente ponemos en el campo el tipo NULL.

**INSERT INTO nombre\_tabla (columna\_1, columna\_2) VALUES (NULL, NULL);**

## SUM

La sentencia SUM es un operador que se utiliza en consulta para sumar valores de dos o mas filas en una columna. Es usado con valores numericos y permite hacer operaciones aritmeticas en rangos o totales de columnas de valores

Ejemplo:

**SELECT SUM(CANTIDAD) AS 'Total de Pedidos'**  
**FROM PEDIDOS;**

1	SELECT SUM(CANTIDAD) AS 'Total de Pedidos'
2	FROM PEDIDOS;

	Total de Pedidos
1	8

## AVG

La sentencia AVG permite hayar el valor promedio de una columna especifica de una tabla.

Ejemplo:

**SELECT AVG(IMPORTE) AS 'Prmedio Importes'**  
**FROM PEDIDOS;**

1	SELECT AVG(IMPORTE) AS 'Prmedio Importes'
2	FROM PEDIDOS;

	Prmedio Importes
1	272.0

## COUNT

Esta sentencia permite contar la cantidad de filas o elementos que existen en una columna o dentro de un rango de la columna según se de por clausula.

Ejemplo:

```
SELECT COUNT(NOMBRE) AS 'Cantidad de Empleados'  
FROM PERSONAS;
```

1	SELECT COUNT(NOMBRE) AS 'Cantidad de Empleados'
2	FROM PERSONAS;
Cantidad de Empleados	
1	5

```
SELECT COUNT(NOMBRE) AS 'Cantidad de Empleados Mayores de 40'  
FROM PERSONAS  
WHERE EDAD>40;
```

1	SELECT COUNT(NOMBRE) AS 'Cantidad de Empleados Mayores de 40'
2	FROM PERSONAS
3	WHERE EDAD>40;
Cantidad de Empleados Mayores de 40	
1	3

## MIN y MAX

Estas sentencias como lo dictan sus nombres, permiten conocer el minimo vaor y el maximo valor de una columna con valores numericos.

Ejemplo:

```
SELECT MIN(EDAD) AS 'Minima edad de los Empleados'  
FROM PERSONAS;
```

1	SELECT MIN(EDAD) AS 'Minima edad de los Empleados'
2	FROM PERSONAS;
Minima edad de los Empleados	
1	23

```
SELECT MAX(EDAD) AS 'Maxima edad de los Empleados'  
FROM PERSONAS;
```

1	SELECT MAX(EDAD) AS 'Maxima edad de los Empleados'
2	FROM PERSONAS;

	Maxima edad de los Empleados
1	64

## CURRENT TIME , CURRENT\_DATE y CURRENT\_TIMESTAMP

La funcion por sentencia CURRENT TIME nos permite tener la hora actual y nos permite anexarla en una consulta. La funcion tambien puede utilizare en conjuncion con otras sentencias, como al momento de insertar un dato en una tabla con INSERT INTO utilizando CURRENT\_TIME como el valor en la columna.

La funcion por sentencia CURRENT DATE nos permite saber la fecha actual y a su vez anexarla dentro de una consulta. Al igual que la anterior consulta , podemos insertarla como valor al utilizar la consulta la sentencia INSERT INTO.

La funcion por sentencia CURRENT TIMES TAMP nos permite saber tanto la fecha como la hora actual. La favorita para en siertos casos lenar datos de una tabla que sean de tipo DATE.

Ejemplo:

**SELECT CURRENT\_TIME AS 'Hora Actual';**

1	SELECT CURRENT_TIME AS 'Hora Actual';
---	---------------------------------------

	Hora Actual
1	21:31:42

**SELECT CURRENT\_TIME AS 'Hora Consulta', \*  
FROM PERSONAS;**

1	SELECT CURRENT TIME AS 'Hora Consulta', *
2	FROM PERSONAS;

	Hora Conulta	PER	NOMBRE	APELLIDO1	APELLIDO2	DNI	EDAD	DEP	SS
1	21:33:06	1	ANTONIO	PEREZ	GOMEZ	32887345L	46	2	NULL
2	21:33:06	2	LUIS	LOPEZ	PEREZ	30234863P	23	1	NULL
3	21:33:06	3	ANTONIO	GARCIA	BENITO	29345120	56	3	NULL
4	21:33:06	4	PEDRO	RUIZ	GONZALEZ	35987125A	35	2	NULL
5	21:33:06	6	Felipe	Narvaez	Gomez	456789123	64	2	0



**SELECT CURRENT\_DATE AS 'Fecha Consulta';**

1	SELECT CURRENT_DATE AS 'Fecha Consulta';
	Fecha Consulta
1	2022-04-25

**SELECT CURRENT\_DATE AS 'Fecha Consulta', CURRENT\_TIME AS 'Hora Consulta'  
FROM PERSONAS;**

1 SELECT CURRENT\_DATE AS 'Fecha Consulta', CURRENT\_TIME AS 'Hora Consulta'

2 FROM PERSONAS;

	Fecha Consulta	Hora Consulta
1	2022-04-25	21:39:00
2	2022-04-25	21:39:00
3	2022-04-25	21:39:00
4	2022-04-25	21:39:00
5	2022-04-25	21:39:00

**SELECT CURRENT\_TIMESTAMP AS 'Momento de la consulta';**

1	SELECT CURRENT_TIMESTAMP AS 'Momento de la consulta';
	Fecha Consulta
1	2022-04-25 21:41:47

**CREATE TABLE tiempos(  
id\_tiempo INT NOT NULL,  
hora\_actual time NOT NULL,  
fecha\_actual date NOT NULL,  
tiempo\_actual datetime NOT NULL  
);  
INSERT INTO tiempos( id\_tiempo, hora\_actual, fecha\_actual, tiempo\_actual)  
VALUES( 1, CURRENT\_TIME, CURRENT\_DATE, CURRENT\_TIMESTAMP);**

**SELECT \* FROM tiempos;**

1	CREATE TABLE tiempos(
2	id_tiempo INT NOT NULL,
3	hora_actual time NOT NULL,
4	fecha_actual date NOT NULL,
5	tiempo_actual datetime NOT NULL
6	);
7	INSERT INTO tiempos( id_tiempo, hora_actual, fecha_actual, tiempo_actual)
8	VALUES( 1, CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP);
9	
10	SELECT * FROM tiempos;

	id_tiempo	hora_actual	fecha_actual	tiempo_actual
1	1	21:51:00	2022-04-25	2022-04-25 21:51:00

**NOTA:** La resolucion de los ejercicios avanzados se encuentran dentro del repositorio.