

# Curso Aprende SQL con HolaMundo {;)}

## INTRODUCCIÓN: Conceptos Básicos

Concepto	Descripción
BD	Base de Datos
ID	Identificador

### Qué es una base de datos?

Cualquier cosa que agrupe información y que esta tenga un sentido.

Ej: Lista del supermercado, libreta de contacto, Lista de alumnos, etc...

Lo importante es que los datos ser puedan relacionar entre sí (El contacto y su número de teléfono, el alumno y su clave, etc..)

### Maneras de guardar información:

- La mente
- Hoja y lapiz
- Computadora\*\*

La mejor manera de guardar datos o infoprmación es en una computadora, por que puede almacenar grandes cantidades de datos... Para poder acceder a los datos dentro de nuestra base de datos necesitiamos un software de tipo **Relational Data Base Management System (RDBMS)** por ejemplo:

- MySQL
- Postgres
- MaríaDB
- Oracle

Su objetivo es entregarnos un acceso fácil a nuestra base de datos. Parte de **su función es encargarse de la seguridad e integridad** de nuestros datos, así como realizar **respaldos** de nuestras BD, También de **importar y exportar** datos. Así como manejar la **concurrencia**; La **concurrencia** es cuando más de un usuario quiere acceder a la base de datos para hacer consultas o cambio.

También nos da la posibilidad de conectarnos a diferentes lenguajes de programación...

Algunas de las operaciones más comunes que existen dentro de las **RDBMS** son las (**CRUD**):

- **C** → CREATE
- **R** → READ
- **U** → UPDATE
- **D** → DELATE

Para poder acceder a los registros de los BD y realizar estas operaciones, tenemos que escribir lo que se conoce como '**Query**', que se traduce al español como **consulta**.

**Ej:** Estas haciendo una consulta cuando: Buscas alguna duda en Google, creas un hilo en twitter, buscas un perfil en instagram.

### **¿Quiénes utilizan los softwares de gestión de bases de datos?**

Casi todas las empresas a nivel mundial los utilizan.

Empresas como Facebook, lo que hacen es que dese su aplicación de facebook, através de un RDBMS, realizan consultas a la BD, la cual les regresa la información al RDBMS y de ahí a la aplicación de facebook para visualizar la información.

Otro ejemplo es con Amazon cuando ves artículos para comprar. Lo que realmente esta pasando es que al entrar a ver algún artículo estás haciendo una consulta que viaja a un RDBMS que entra a la BD que obtiene la información del artículo, y la regresa de nuevo al RDBMS y este lo regresa a la página para poder ver la información.

### **Tipos de Software para BD:**

Existen dos mundos al momento de elegir un software para almacenar bases de datos: **SQL** y **NoSQL (relacionales y no relacionales)**

- **SQL**

Las datos que se almacenan en un software de tipo SQL se almacenan mediante tablas, con filas y columnas.

ID*	NOMBRE	EDAD
1	Felipe	20
2	-	-

**ID\***= Este es un identificador que se agrega de manera automática al momento de crear un registro.

- **NoSQL**

En este software se almacenan datos de tipo JSON, BSON, BLOB, Key-Value, entre otros datos que no son en formato de tablas.

Nosotros utilizaremos los de tipo SQL para aprender a gestionarlos.

Veremos ahora un ejemplo con una tabla de Productos. Existen varios tipos de perfiles para hacer consultas en las BD, pero en este caso veremos el perfil de tipo **ADMINISTRADOR**, el cuál puede crear productos dentro de un software.

Esta forma que vemos de la tabla de PRODUCTOS tiene una relación que se llama **1→N (uno a n)**, donde un usuario puede crear muchos productos, pero un producto solo puede ser creado por un usuario.

### PRODUCTOS

ID	NOMBRE	CANTIDAD	USUARIO_ID
1	iPad	500	1
-	-	-	-
105	iPhone	100	2

ID = identificador del producto

NOMBRE = Nombre del producto

CANTIDAD = Cantidad de inventario de producto

CREADO POR USUARIO = Usuario que creó el registro del producto

¿Porqué el USUARIO esta escrito como número? Por que podemos tener información del usuario en otra tabla relacionado con este número y esta puede modificarse. ej:

### USUARIO

ID	NOMBRE	EDAD
1	Felipe	20
2	Pedro	24

En este ejemplo estamos viendo como podemos relacionar una tabla a otra.

Ahora ¿Qué pasa si queremos crear una relación donde por ejemplo, tenemos una tabla de ALUMNOS y otra de DEPORTES, y los alumnos pueden estar en más de un deporte?

### ALUMNOS

ID	NOMBRE	EDAD
1	Peter	24
2	Felipe	20

### DEPORTES

ID	DEPORTE
1	Escalada
2	Futbol

La mejor manera de **relacionar** estas dos tablas **es creando una tercera tabla**, asignandole el nombre de las dos tablas que queremos relacionar, en este caso **ALUMNOS\_DEPORTES**. Dentro de esta agregaremos, además de su ID Automático, agregamos una columna con el ID de ALUMNOS y otra con el ID de DEPORTES. Con esto lograríamos crear una o dos referencias de un alumno con uno de los deportes.

### ALUMNOS\_DEPORTES

ID	ALUMNOS_ID	DEPORTES_ID
1	1	1
2	1	2
3	2	1

Esta relación se llama **N→N (n a n)**.

Nombres

- **ID = Primary Key (llave primaria) → Ej:** columna 'ID' de tabla ALUMNOS\_DEPORTES

Cuando el ID(Identificador) es la primera columna que genera el dato de manera automática

- **ID = Foreign Key (llave foránea)** → Ej: Columna 'ALUMNOS\_ID' de tabla ALUMNOS\_DEPORTES

Cuando el ID(Identificador) es una columna dentro de la tabla sin ser la primera.

\*\*RECUERDA\*\*

**SQL** = Lenguaje de programación para interactuar con Bases de Datos. [Comandos escritos]

**MySQL** = Software de gestión de base de datos. (RDBMS) [Servidor de base de datos]

**MySQL Workbench** = Herramienta Gráfica que nos permite conectarnos con MySQL y trabajar con bases de datos. [Interfaz gráfica]

## CREACIÓN DE BASES DE DATOS

**MySQL Workbench**

\*Base de Datos; Esta se va a encargar de contener muchas tablas, para crear las tablas primero tenemos que crear la base de datos.

**Tipos de datos**

**INT** = Integer (número entero)

**FLOAT** = numeros decimales

**VARCHAR** = cadena de caracteres (palabras/oraciones)

Para crear una Base de Datos entramos a un host donde vamos a crear un comando.

**CREATE DATABASE holamundo;**

(Con este comando podremos revisar las bases de datos existentes y verificar si existe la que acabamos de crear: **SHOW DATABASES;**)

Ahora queremos comenzar a crear tablas para nuestra BD, pero en sí el sistema no sabría en donde poner esa creación, entonces vamos a crear un comando que nos adentra a la BD que acabamos de crear.

**USE holamundo;**

\*\*RECUERDA\*\*

Al final de cada consulta tenemos que terminar con ' ; '

Con nuestra BD creada y dentro de ella, ahora crearemos las tablas que se gestionaran aquí.

Utilizando **CREATE** y nombrando la nueva tabla. Dentro de la consulta (que son las comas() ) anotaremos las columnas que tendrá la tabla y el tipo de dato que tendrán estas columnas.

Comenzaremos con 'id', a la cual le asignaremos el tipo de datos INT, después una columna llamada

'tipo' a la que le asignaremos VARCHAR con una disponibilidad de 255 caracteres. Al igual que 'tipo' creamos una llamada 'estado' con el mismo tipo de dato.

Por último tenemos que especificar que la columna de 'id' es nuestra primary key.

```
CREATE TABLE animales(
```

```
    id INT,  
    tipo VARCHAR(255),  
    estado VARCHAR(255),  
    PRIMARY KEY (id)  
);
```

Una vez creada nuestra tabla, insertaremos la información que llevará dentro. Utilizando `INSERT`. Después de anotar el nombre de la tabla, entre comas () anotaremos las columnas a las que les agregaremos datos, seguido del comando `VALUE` los datos que queremos agregar.

```
INSERT INTO animales (tipo, estado) VALUE ('chanchito', 'feliz');
```

Nos daremos cuenta que esa consulta no funciona de manera correcta y nos a `ERROR`, ya que la columna de 'id' no contiene valor. Esto es por que tenemos que especificar que a la columna 'id' se autoincremente para que se agregen valores automaticos a su columna.

```
ALTER TABLE animales MODIFY COLUMN id INT AUTO_INCREMENT;
```

De esta manera podremos correr el código anterior y ya tener datos dentro de nuestra columna.

Si queremos agregar esta función desde que estamos creando la tabla, esta consulta se vería de esta manera:

```
CREATE TABLE `animales` (
```

```
    `id` int NOT NULL AUTO_INCREMENT,  
    `tipo` varchar(255) DEFAULT NULL,  
    `estado` varchar(255) DEFAULT NULL,  
    PRIMARY KEY (`id`)  
);
```

Si queremos revisar los datos dentro de la tabla, basta con un `SELECT FROM` para visualizar la tabla.

```
SELECT * FROM animales;
```

En el caso de que queramos revisar registros en específico tenemos la posibilidad de utilizar el comando `WHERE` para especificar los criterios que queremos para consultar el registro. Esto podemos hacerlo más específico aún con `AND`.

```
SELECT * FROM animales WHERE id = 1 AND tipo = 'feliz';
```

Ahora vamos a utilizar el comando `UPDATE` para actualizar la tabla.

```
UPDATE animales SET estado = 'feliz' WHERE id = 3;
```

Continuaremos con un `DELETE`.

```
DELETE FROM animales WHERE estado = 'feliz';
```

**\*\*Atención\*\*** un `DELETE WHERE` sin un 'id' de la llave maestra nos dará como resultado ERROR 1175. Esto es una medida de seguridad para no borrar datos de manera masiva o incluso la tabla completa. La manera correcta de escribir este comando es especificando el 'id' a eliminar.

```
DELETE FROM animales WHERE id = 3;
```



Continuaremos haciendo un `UPDATE` nuevo.

```
UPDATE animales SET estado = 'triste' WHERE tipo = 'dragon';
```

**\*\*Atención\*\*** Tendremos el mismo Error 1175. Ya que nuestro WHERE no cuenta con un 'id' de la llave maestra.

Acabamos de aprender a utilizar las 4 acciones principales que se realizan de manera habitual en las bases de datos (**CRUD**).

C

## R → READ [Profundización de la Acción Lectura]

U

D

Comenzaremos creando una nueva tabla.

```
CREATE TABLE user (
```

```
    id INT NOT NULL auto_increment,  
    name VARCHAR (50) NOT NULL,  
    edad INT NOT NULL,  
    email VARCHAR(100) NOT NULL,  
    PRIMARY KEY (id)  
);
```

Después insertaremos sus valores dentro de la nueva tabla.

```
INSERT INTO user (name, edad, email) VALUES ('Oscar', 25, 'oscar@gmail.com');
```

```
INSERT INTO user (name, edad, email) VALUES ('Lyla', 15, 'layla@gmail.com');
```

```
INSERT INTO user (name, edad, email) VALUES ('Nicolas', 36, 'nico@gmail.com');
```

```
INSERT INTO user (name, edad, email) VALUES ('Chanchito', 7, 'chanchito@gmail.com');
```

Ahora comenzaremos con la acción de READ con el comando `SELECT`.

Con el signo de `' * '` seleccionaremos todas las columnas y el comando `FROM` es para seleccionar la tabla. Primero se selecciona la o las columnas y después la tabla.

```
SELECT * FROM user;
```

Con el comando `LIMIT` limitamos la cantidad de registros que queremos mostrar. Con el ejemplo de abajo (`LIMIT 1`), nos mostrará el primer registro.

```
SELECT * FROM user LIMIT 1;
```

Con el signo `' > '` (mayor que) aplicado en la columna `'edad'`, mostrará todos los registros que de la columna `'edad'` tengan mayor de 15 años.

```
SELECT * FROM user WHERE edad > 15;
```

Con el signo `' > '` (mayor que) y `' = '` (igual que) juntos, aplicado en la columna `'edad'`, mostrará todos los registros que de la columna `'edad'` tengan mayor de 15 años y que sean igual a 15 años.

todos los registros que de la columna edad tengan mayor de 15 años y que sea igual a 15 años.

```
SELECT * FROM user WHERE edad >= 15;
```

Con el comando AND se agregará a la condición de la consulta el siguiente criterio que se aplique, haciendo que solo mostrara los registros que cumplan las dos condicionales , así que en este caso mostraría solos los registros que tengan más de 20 años y que tengan el email escrito.

```
SELECT * FROM user WHERE edad > 20 AND email = 'nico@gmail.com';
```

Con el comando OR los registros que se mostraran serán los que cumplan con uno de los criterios, en este caso los registros que tengan más de 20 o que tengan el mail escrito.

```
SELECT * FROM user WHERE edad > 20 OR email = 'layla@gmail.com';
```

## Negación

Con el signo ' != ' (no es igual a)(Distinto a). Nos mostrará los registros que sean distinto al criterio que escribimos.

```
SELECT * FROM user WHERE email != 'layla@gmail.com';
```

## Rango

Con el comando BETWEEN podremos escribir un rango en donde tenemos que usar el comando AND para definir el minimo y el maximo de los valores.

```
SELECT * FROM user WHERE edad BETWEEN 15 AND 30;
```

## Contiene...

Con el comando LIKE podremos agregar caracteres que queremos que la consulta encuentre. Siempre utilizando (''). Podemos utilizar el signo de ' % ' (porcentaje) para indicar que no importa con qué texto comience o termine, dependiendo en donde pongamos este signo. En este ejemplo buscamos que el registro contenga algun dato con 'gmail', pero como agregamos el signo ' % ' al principio y al final, no importa si el texto contiene algo antes o después de la palabra 'gmail'. Ej: 'layla@gmail.com'

```
SELECT * FROM user WHERE email LIKE '%gmail%';
```

De la misma manera, esta consulta busca un texto con 'gmail', pero en esta ocasión el texto puede contener cualquier cosa antes de 'gmail' pero nada después, ya que no tiene el signo de ' % ' al final del texto.

```
SELECT * FROM user WHERE email LIKE '%gmail';
```

Para esta situación se mostraran los registros que comienzan con 'oscar' sin importar lo que muestren al final.

```
SELECT * FROM user WHERE email LIKE 'oscar%';
```

## Ordenar

Aquí utilizaremos el comando ORDER BY para indicar el orden que nos mostrará la columna que indiquemos. Y con el comando ASC estaremos indicando que queremos que se orden de orden Ascendente. Para este ejemplo nos mostrará los registros ordenados de manera ascendente en referencia a la columna 'edad'.

```
SELECT * FROM user ORDER BY edad asc;
```

En este ejemplo usamos el comando DESC, que mostrara el orden de los registros de manera descendente.

```
SELECT * FROM user ORDER BY edad desc;
```

## Valores máximos y mínimos

Queremos seleccionar el valor máximo o mínimo de una columna, utilizamos la función max() ó min(), que por ser función llevan paréntesis para indicar la columna que queremos revisar su valor. Despues usaremos AS para indicar el nombre que llevará. Estos comandos solo regresarán el dato único que encuentren.

```
SELECT max(edad) as mayor FROM user;
```

```
SELECT min(edad) as menor FROM user;
```

Si queremos seleccionar columnas en específico, vamos poner el nombre de o las columnas en donde esta normalmente el signo '\*'

```
SELECT id, name FROM user;
```

Podemos ponerle un alias a la columna de esta manera.

```
SELECT id, name as nombre FROM user;
```

---

## JOINS

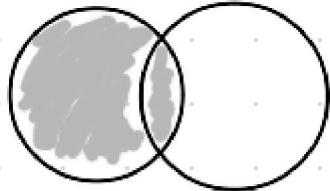
Nuevo script donde veremos consultas JOINS, primero creamos y adaptamos nuestra tabla.

```
CREATE TABLE products(
```

```
    id INT NOT NULL AUTO_INCREMENT,  
    name VARCHAR(50) NOT NULL,  
    created_by INT NOT NULL,  
    marca VARCHAR(50) NOT NULL,
```

```
PRIMARY KEY(id),  
FOREIGN KEY(created_by) references user(id)  
);  
rename TABLE products to product;  
INSERT INTO product (name, created_by, marca)  
values  
    ('ipad', 1, 'apple'),  
    ('iphone', 1, 'apple'),  
    ('watch', 2, 'apple'),  
    ('macbook', 1, 'apple'),  
    ('imac', 3, 'apple'),  
    ('ipad mini', 2, 'apple');  
SELECT * FROM product;
```

## LEFT JOIN



Con un **Left Join** Los resultados son todos los registros de la primera tabla más los productos de la segunda tabla que tienen como llave primaria, las llaves foráneas que existen en la primera tabla.

En este ejemplo la tabla que manda va a ser 'usuarios', en la consulta que vamos a hacer traeremos todos los usuarios que la consulta nos arroje, y si estos registros contienen llaves foráneas de la tabla secundaria que estamos llamando, las traerá también.

Indicaremos que columnas queremos llamar de cada tabla junto con un alias para cada tabla. Después asignaremos con qué columna se van a unir cada Tabla, en este ejemplo unimos la columna 'id' de 'usuarios' y la columna 'created\_by' de 'productos',

```
SELECT u.id, u.email, p.name FROM user u LEFT JOIN product p ON u.id = p.created_by;
```

## LEFT JOIN



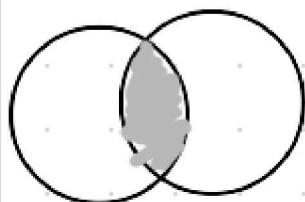


Un Right Join es lo mismo que un Left Join pero al revés.

Para el ejemplo utilizaremos el mismo código pero cambiando el tipo de Join. Entonces el resultado serán las mismas columnas pero con los registros de la tabla de la derecha 'product' y el orden de los registros también lo tomará de la tabla de la derecha.

```
SELECT u.id, u.email, p.name FROM user u RIGHT JOIN product p ON u.id = p.created_by;
```

## INNER JOIN



Un Inner Join nos entregará los registros que siempre y cuando los registros de las dos tablas puedan ser asociados. En este caso nos traerá todos los registros, ya que todos estan relacionados entre sí.

```
SELECT u.id, u.email, p.name FROM user u INNER JOIN product p ON u.id = p.created_by;
```

## CROSS JOIN

