**¿Alguna sabe lo que son las bases de datos?  
¿Qué son?**

¿Diferencias entre MySQL y SQL?

**SQL** es un lenguaje de bases de datos, mientras **MySQL** es un sistema de gestión de bases de datos basado en **SQL**. **SQL** es un lenguaje de programación que necesita estudiarse para comprender su uso, mientras que **MySQL** es un software que se debe descargar, instalar y configurar.

**Para crear una base de datos** vamos a la opción '**Nueva**', ponemos el nombre que queramos, en este caso, 'Kiosko'.

Creamos una tabla con el nombre '**Productos**' y 3 columnas:

-ID tipo ***INT***

-Nombre tipo **VARCHAR** (Varchar sería un String en SQL. Es una cadena de caracteres) con longitud de 40 caracteres.

-Precio tipo ***DOUBLE*** (decimal)

Abajo a la derecha: **\*Guardar\***

**Notas-Vista:**

En la tabla Productos, dentro de **ESTRUCTURA** podemos ver qué columnas tiene nuestra tabla

En la base de datos **Kiosko**, dentro de **ESTRUCTURA** podemos ver cuántas tablas tenemos.

Desde **EXAMINAR** podemos observar todos los cambios de la tabla.

Dentro de la tabla productos debo hacer **SCRIPTS O SENTENCIAS** para cargarle datos a la tabla. ¿Cómo?

Para cargar datos tenemos que ir desde Productos a la pestaña SQL y cargamos filas:

INSERT INTO Productos VALUES (1, "Chupetín", 6.80),

INSERT INTO Productos VALUES (1, "Alfajor Chocolate Blanco", 12),

INSERT INTO Productos VALUES (1, "Alfajor chocolate Negro", 15),

INSERT INTO Productos VALUES (1, "Caramelos", 1);

Es decir, le pedimos que INSERTE/AGREGUE en la tabla productos los VALORES (ID, Nombre, Precio). Deben guiarse por el orden de las columnas de la derecha.

**Le damos a Continuar.**

Desde **EXAMINAR** podemos ver los resultados

**\*TODAS LAS SENTENCIAS DEBEN LLEVAR UN PUNTO Y COMA AL FINAL.**

Generalmente las tablas tienen muchos registros, por lo que verlos desde Examinar puede ser tedioso.

Para ver un registro en particular (o todos) usamos el SELECT:

Desde la pestaña SQL escribimos:

**SELECT** \* **FROM Productos**

La sintaxis correcta es la primera pero el resultado es el mismo.

**SELECT** ID, Nombre, Precio **FROM `Productos**

Esto es una **query (cueri)** o una **consulta.** Le estamos consultando o pidiendo al sistema que nos traiga **TODOS (\*)** los campos de nuestra tabla **Productos.** Como en un futuro vamos a tener muchas tablas en las bases de datos (y puede haber datos que se repitan) **SIEMPRE** hay que aclarar de qué tabla estamos hablando.

**Si yo quisiera editar la query** que acabo de realizar desde la pestaña Examinar selecciono [ [Editar en línea](http://localhost/phpmyadmin/index.php?route=/sql&pos=0&db=clubchicas&table=alumnas) ] y trabajo desde allí.

Ahora bien, puedo traer algunos datos como Nombre y precio o ID y Nombre, etc:

**SELECT** Nombre, Precio **FROM** Productos

**SELECT** ID, Nombre **FROM** Productos

CONDICIONES (Se agregan al final):

**¿Recuerdan el If y else como condicionales?**

Bueno, SQL también tiene su ‘condicional’. Si yo quisiera que la query me trajera todos los campos de mi tabla producto **DONDE** el precio sea menor a 10 debería indicarle:

**SELECT** \* **FROM** Productos

**WHERE** Precio < 10**;** --Traer de la tabla Productos los precios menores a 10.

Traer de la tabla Productos los precios mayores a 6:

**SELECT** \* **FROM** Productos

**WHERE** Precio > 6**;** --Traer de la tabla Productos los precios mayores a 6.

Se puede agregar varias condiciones juntas:

**SELECT** \* **FROM** Productos

**WHERE** Precio < 10 OR precio > 20; --Traer de la tabla Productos los precios menores a 10 y mayores a 20.

### **[Operadores lógicos en SQL](https://sites.google.com/site/tumentororg/postgresql/operadores-logicos-and-or-not" \l ":~:text=Entonces%2C%20para%20establecer%20m%C3%A1s%20de,se%20cumpla%20la%20condici%C3%B3n%20especificada.)**

- and, significa "y",

- or, significa "y/o",

- not, significa "no", invierte el resultado

- (), paréntesis

***En Python se vería así:***

|  |  |  |
| --- | --- | --- |
| and | Devuelve True si ambos operandos son True | a and b |
| or | Devuelve True si alguno de los operandos es True | a or b |
| not | Devuelve True si alguno de los operandos False | not a |

Dejo el link por si quieren ver acercade operadores: https://www.freecodecamp.org/espanol/news/operadores-basicos-en-python-con-ejemplos/  
  
**En Java:**

|  |  |
| --- | --- |
| **OPERADOR** | **DESCRIPCIÓN** |
| **==** | Es igual |
| **!=** | Es distinto |
| **<, <=, >, >=** | Menor, menor o igual, mayor, mayor o igual |
| **&&** | Operador and (y) |
| **||** | Operador or (o) |
| **!** | Operador not (no) |

Operadores lógicos principales en Java

Si yo quisiera ordenar por precio:

**SELECT** \* **FROM** Productos

**WHERE** Precio < 10;

**ORDER BY** precio **DESC** <--Ordenar de mayor a menor (**Descendente**). Si no pongo nada por defecto lo toma como ASC (de menor a mayor)

Si yo quisiera buscar en mis registros un nombre en específico:

**SELECT \* FROM `productos`**

**WHERE Nombre= 'Alfajor Chocolate Blanco';**

Si yo no estoy segura de cómo es el nombre y tiene algo más escrito o quisiera traer todos alfajores:

**SELECT** \* **FROM** Productos

**WHERE** Nombre = “Alfajor%”

Si la palabra que yo busco está en el medio, Es decir que, no es la primera palabra como “Alfajor de chocolate” (puede ser que tenga una palabra adelante y otra atrás) o no es exacto. Sería:

**SELECT \* FROM `productos`**

**WHERE Nombre LIKE '%chocolate%';**

**\*Donde nombre sea parecido a “chocolate”-**

***Entonces a recordar:***

***-Si el nombre es exacto se usa un =***

***-Si conozco alguna parte del nombre voy a usar un ‘like’.  
-Y aviso si tiene una palabra antes uso un % y si tiene algo después uso otro %.***

Si yo quisiera que me traiga todo lo que NO tiene la palabra ‘chocolate’:

**SELECT** \* **FROM** Productos

**WHERE NOT Nombre LIKE '%chocolate%';**

UPDATE-MODIFICAR/ACTUALIZAR:

Si yo quisiera modificar/actualizar valores de mi tabla como por ejemplo: modificar el registro ‘Alfajor Chocolate Blanco’ por ‘Alfajor Blanco’ sería:

**UPDATE** Productos **SET** Nombre = “Alfajor Blanco” WHERE nombre **like** “Alfajor%Bl”;

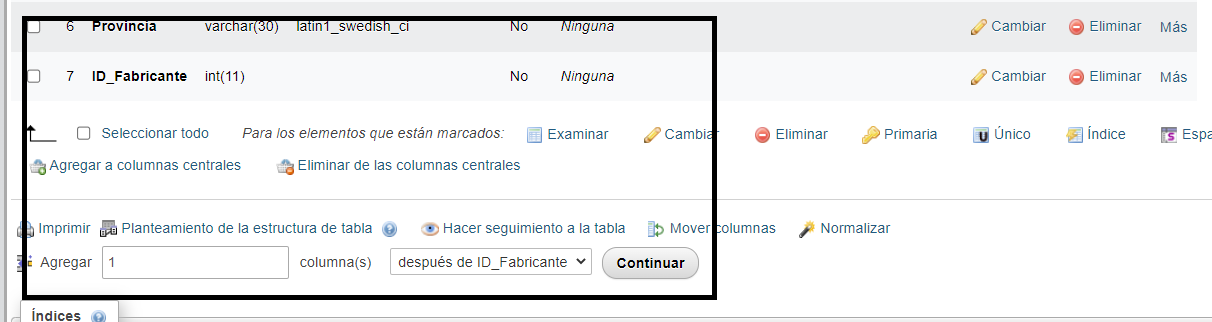
Voy a actualizar la tabla productos, voy a cambiar el nombre DONDE el nombre de ahora es “Alfajor” tiene algo más y dice “Bl” (si no nos acordamos el nombre exacto).

OJO: Si solo pongo esto me va a cambiar TODOS los registros de la tabla, por eso es importante aclarar el WHERE:

**UPDATE** Productos **SET** Nombre = “Alfajor Blanco”;

Para agregar una columna a mi tabla:

Deseo agregar la columna ID\_Fabricante:



Desde estructura ‘Agregar’ y probablemente indique un código como este:  
[ALTER](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/alter-table.html) [TABLE](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/alter-table.html) `codoacodo` ADD `ID\_Fabricante` INT [NOT](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/logical-operators.html%23operator_not) NULL AFTER `Provincia`;

Para eliminar la tabla el código es:

ALTER TABLE `codoacodo` DROP `ID\_Fabricante`;" -OJO. No recomendado-

Si yo quisiera saber cuál es el precio más elevado:

**SELECT** **MAX**(Precio)

**FROM** Productos

Si yo quisiera saber cuál es el producto y precio más caro probablemente haría esto:

**SELECT** **MAX**(Precio), Nombre

**ERROR!**

**FROM** Productos

¿Por qué?  
Porque me indicaría el máximo precio y EL PRIMER NOMBRE QUE ENCUENTRA EN LA TABLA. NO ESTÁ HACIENDO LA RELACIÓN. Tendría que decirle:

**SELECT** ID, Nombre, Precio

**OK!**

**FROM** Productos

**Order by** precio **DESC**

**limit 1 – Me trae el primero que tenga mayor precio, si quisiera podría ponerle 2 para que me traiga los 2 registros con mayor precio y así sucesivamente.**

Si yo quisiera saber cuál es el precio promedio de mis productos?:

**SELECT** **AVG(Precio) –Función Promedio es AVG**

**FROM** Productos

Y si yo quisiera saber cuál es el precio promedio de mis productos POR FABRICANTE?:

**SELECT** **AVG(**Precio), ID\_FABRICANTE **–Función Promedio de Precios y que me muestre a qué fabricante corresponde.**

**FROM** Productos

**GROUP BY** ID\_FABRICANTE –Agrupar promedio por Fabricante

**GROUP BY:** Se usa cuando tengo datos, promedios, máximos, mínimos de diferentes categorías y los quiero agrupar. Quiero saber cuánto de cada categoría.

Si yo quiero saber cuánto es la suma de todos mis productos?

**SELECT** **SUM**(Precio) **–Función Suma**

**FROM `**Productos`

Y cuanto sería, si me compro todos los productos del fabricante Nº2?

**SELECT** **SUM**(Precio), ID\_FABRICANTE **–Función Suma**

**FROM `**Productos`

GROUP BY ID\_FABRICANTE –Agrupa suma por Fabricantes

CREAR UNA NUEVA TABLA

Creamos una tabla con el nombre '**Productos**' y 3 columnas:

-ID tipo ***INT y en la categoría de A\_I(AUTO INCREMENT) vamos a darle un check para que el ID se vaya cargando solo a medida que se carguen los fabricantes. Imaginense tener 10000 fabricantes y tener que poner el ID uno por uno. ¿Y si nos equivocamos en un registro y en vez de poner 4 ponemos 5? A veces estas herramientas nos ayudan un poco más. En ese caso se va a volver una PRIMARY KEY. \*Explicar PRIMARY KEY y FOREIGN KEY)***

-Nombre tipo **VARCHAR** (Varchar sería un String en SQL. Es una cadena de caracteres) con longitud de 60 caracteres.

-Direccion tipo ***Varchar con*** longitud de 60 caracteres.

Abajo a la derecha: **\*Guardar\***

Para cargar datos tenemos que ir desde Productos a la pestaña SQL y cargamos filas:

INSERT INTO Productos VALUES (null, "Distribuidora L SA", “Libertad 243”),

INSERT INTO Productos VALUES (null, "Maxikiosko Luz", “Forest 194”),

INSERT INTO Productos VALUES (null, "Potiggi",”Uruguay 22”);`

No vamos a poner el ID. Vamos a ingresar un null, este se va a crear solo, se va fijar automáticamente cual era el último ID y le va a crear uno nuevo.

\*NO SE PUEDEN REPETIR LOS ID\*

Ahora bien, si yo quisiera saber quién produce el alfajor blanco y el alfajor negro?

**SELECT** **\***

**FROM `**Productos`

**WHERE** Nombre LIKE “Alfajor%”

Y si yo quiero tener los datos de ese fabricante en la misma consulta?

**SELECT** **\* FROM `**Productos` **INNER JOIN** Fabricantes **ON** productos.ID\_FABRICANTE = Fabricantes.ID

**WHERE** Productos.Nombre LIKE “Alfajor%”

\*\*Le estoy diciendo que traiga TODO de la tabla Productos y esta tabla Productos la voy a unir (INNER JOIN) a la tabla Fabricantes y debo especificarle el campo que se repite o tienen en común (-mostrar en las tablas- si se fijan bien Productos tiene el ID de Fabricantes y Fabricantes tiene ID. Entonces el ID de Fabricantes es el mismo que el ID\_Fabricante que está en mi tabla producto. OjO, no confundan con el campo ID de la tabla productos, este corresponde al código del producto.). Entonces reitero, uno con fabricantes EN(ON) la tabla productos campo ID\_Fabricantes (cuando quiero hablar de un campo en específico que se puede llamar igual en otra tabla le pongo el nombre de la tabla.campo) que es igual que de la tabla Fabricantes el campo ID\_Fabricantes.

WHERE: Que me traiga el campo Nombre de la tabla Productos, no de Fabricantes.

Si quiero que me traiga campos específicos DE CADA TABLA EN LA CONSULTA:

**SELECT** Productos.ID, Productos.Nombre, precio, Fabricantes.Nombre, Fabricantes.Direccion **FROM `**Productos` **INNER JOIN** Fabricantes **ON** productos.ID\_FABRICANTE = Fabricantes.ID

**WHERE** Productos.Nombre LIKE “Alfajor%”

El ID de la tabla Productos (Productos.ID), nombre de la tabla Productos (Productos.Nombre), precio (no hace falta aclarar de cuál porque no se repite en las dos tablas ¿Se entendió? Solo está en la tabla productos). Luego, el Nombre de la tabla Fabricantes (Fabricantes.Nombre),

La dirección de la tabla Fabricantes (Fabricantes.Direccion).

Si quiero cambiar el nombre de la columna en la consulta (para que no se repita el nombre -valga la redundancia- y evitar confusiones)

**SELECT** Productos.ID, Productos.Nombre **AS “Producto”**, precio, Fabricantes.Nombre **AS “FABRICANTE”,** Fabricantes.Direccion **FROM `**Productos` **INNER JOIN** Fabricantes **ON** productos.ID\_FABRICANTE = Fabricantes.ID

**WHERE** Productos.Nombre LIKE “Alfajor%”

Si yo quisiera eliminar una base desde SQL:

[DROP DATABASE](https://mariadb.com/kb/en/drop-database/) clubchicas; --Borrar base de datos y nombre de la misma

Glosario:

SQL (Structured Query Language o Lenguaje de consultas estructuradas): Lenguaje de programación estandarizado que se utiliza para administrar bases de datos relacionales y realizar diversas operaciones con los datos que contienen. las sentencias SQL más utilizadas son select, add, insert, update, delete, create, alter y truncate.

Cotejamiento: El cotejamiento o codificación de una base de datos MySQL, determina los juegos de caracteres que se utilizan en esa base de datos.

UTF-8 es una codificación de caracteres que usa la mayoría de los sitios web. Codifica cada uno de los 1,112,064 puntos de código válidos.

Scripts: Pueden utilizarse para crear la estructura de la base de datos, realizar operaciones en la base de datos (p. ej. rellenarla con datos) y cambiar o eliminar la estructura de la base de datos.

SET especifica las columnas que se deben actualizar y los valores nuevos para las columnas.

Lenguaje de Definición de Datos (DDL) - [**Data Definition Language**](https://es.wikipedia.org/wiki/Lenguaje_de_definici%C3%B3n_de_datos): **permite a los programadores** de un sistema gestor de base de datos, como Postgres, **definir las estructuras que almacenarán los datos** así como los procedimientos o funciones que permitan consultarlos.

Para definir las estructura disponemos de tres sentencias:

* [**CREATE**](https://todopostgresql.com/crear-usuarios-en-postgresql/)**,** se usa para crear una base de datos, tabla, vistas, etc.
* [**ALTER**](https://todopostgresql.com/comandos-postgresql-alter/), se utiliza para modificar la estructura, por ejemplo añadir o borrar columnas de una tabla.
* [**DROP**](https://todopostgresql.com/estructura-de-datos-en-postgresql-create-drop/), con esta sentencia, podemos eliminar los objetos de la estructura, por ejemplo un índice o una secuencia.

## Lenguaje de Manipulación de Datos (DML): También es un lenguaje proporcionado por los sistemas gestores de bases de datos. En inglés, [**Data Manipulation Language**](https://es.wikipedia.org/wiki/Lenguaje_de_manipulaci%C3%B3n_de_datos) **(DML)**.

Utilizando instrucciones de SQL, **permite a los usuarios introducir datos para posteriormente realizar tareas de consultas o modificación** de los datos que contienen las Bases de Datos.

Los elementos que se utilizan para manipular los datos, son los siguientes:

* [**SELECT**](https://todopostgresql.com/manejando-funciones-en-postgresql/)**,** esta sentencia se utiliza para realizar consultas sobre los datos.
* [**INSERT**](https://todopostgresql.com/la-instruccion-insert-en-postgresql/), con esta instrucción podemos insertar los valores en una base de datos.
* [**UPDATE**](https://todopostgresql.com/las-instrucciones-update-y-delete/)**,** sirve para modificar los valores de uno o varios registros.
* [**DELETE**](https://todopostgresql.com/las-instrucciones-update-y-delete/)**,** se utiliza para eliminar las filas de una tabla.

https://mariadb.com/kb/es/basic-sql-statements/

## ¿Diferencias entre Text y Varchar?

El valor de varchar es definido  
El valor de text crece y se adapta a la nececidad.  
  
Por ejemplo un nombre de usuario puede ser varchar(12) esto quiere decir que esa es la longitud maxima que almacena ese campo.  
Por otro lado un comentario, una reseña o un artículo de un blog, como son textos más largos un campo varchar no sería de provecho ya que no podemos conocer que tan largo es el escrito.

Índice:

Optimizador de consultas evalúa cada método disponible para recuperar datos y selecciona el método más eficiente.