# Trabajando con Bases de Datos MySQL

Con este capítulo, llegamos al final del curso "PHP para Principiantes". Abarcando esta última unidad, ya estaremos en condiciones de crear aplicaciones funcionales de alto nivel, de complejidad mendia.

Sin dudas, el trabajo con bases de datos, es lo más esperado por cualquier programador que está dando sus primeros pasos, pero entonces ¿por qué dejarlo para el final? Y la respuesta a esta pregunta, es muy simple: porque las bases de datos son el "cristal" de una aplicación. Representan la parte más vulnerable de un sistema informático y de su vulnerabilidad, dependerá la estabilidad o inestabilidad de todo el sistema.

A lo largo del curso, hemos adquirido todas las técnicas, prácticas y herramientas necesarias, para saber como filtrar y securizar datos y recién ahora, estamos listos para poder comenzar a trabajar con bases de datos, en absoluta libertad y confianza.

¡Comencemos!

> ***Una base de datos representa un conjunto de datos pertenecientes a un mismo contexto, que son almacenados de forma sistemática para su posterior uso.*** *Para comprender mejor el concepto de Base de Datos, por favor, dirigirse a* *http://es.wikipedia.org/wiki/Base_de_datos*

# Acerca de MySQL

MySQL es un **servidor de Bases de Datos SQL** (Structured Query Language) que se distribuye en dos versiones:

- Una versión GPL (Software Libre)
- Otra versión privativa, llamada MySQL AB

En este curso, utilizaremos la versión estandar licenciada bajo la GNU General Public License (GPL).

## Instalación y configuración de MySQL

Para instalar MySQL, por línea de comandos, escribe:

```
sudo apt-get install mysql-server mysql-client
```

Durante la instalación, el sistema te pedirá que ingreses una contraseña para la administración de MySQL. Asigna una contraseña que puedas recordar fácilmente y mantenla a salvo ya que deberás utilizarla frecuentemente.

Una vez que finalice la instalación, ejecuta el siguiente comando a fin de securizar el servidor MySQL (esta configuración, es válida también, para servidores de producción):

```
sudo mysql_secure_installation
```

A continuación, el sistema te pedirá que ingreses la contraseña actual para administración de MySQL (la del usuario root de MySQL). Ten en cuenta que la contraseña no será mostrada mientras escribes:

```
Enter current password for root (enter for none):
```

A continuación, te preguntará si deseas modificar esa contraseña. Salvo que desees modificarla, ingresa n:

```
Change the root password? [Y/n] n
```

Ahora la pregunta, será si deseas eliminar usuarios anónimos. Responde que sí:

```
Remove anonymous users? [Y/n] Y
```

Luego, te preguntará si desees desabilitar el acceso remoto al usuario root de MySQL. Por supuesto, responde que sí:

```
Disallow root login remotely? [Y/n] Y
```

La siguiente pregunta será si deseas eliminar la base de datos de prueba y el acceso a ella. También responde que sí:

```
Remove test database and access to it? [Y/n] Y
```

Finalmente, te preguntará si deseas recargar las tablas de privilegios (esto es para asegurar que todos los cambios realizados surjan efecto). Entonces, responde sí, por última vez:

```
Reload privilege tables now? [Y/n] Y
```

# Iniciar, reiniciar y detener el servidor MySQL

En ocasiones necesitarás iniciar, reiniciar o detener el servidor de bases de datos, MySQL.

Las opciones disponibles son:

| | |
|---|---|
| **stop** | detiene el servidor |
| **start** | inicia el servidor |
| **restart** | reinicia el servidor |

Para iniciar, reiniciar o detener el servidor, deberás ejecutar el siguiente comando, seguido de la opción deseada:

```
sudo /etc/init.d/mysql opcion_deseada
```

Lógicamente reemplazando **opcion** por **stop**, **start** o **restart** según si deseas parar, iniciar o reiniciar el servidor.

# Administración de MySQL

Una vez que comencemos a utilizar bases de datos, necesitarás poder acceder a las opciones de aministración de las mismas. Por lo tanto, te recomiendo tener siempre a mano este capítulo, para poder consultarlo con frecuencia.

## Conectarse y desconectarse al servidor

Para conectarte deberás ejecutar el siguiente comando:

```
mysql -u root -p
```

A continuación, deberás ingresar la contraseña del root de MySQL (no es la del root del SO. Es la que hemos configurado durante la instalación de MySQL).

Las `-u` y `-p` significan usuario y password respectivamente.

Te aparecerá un shell interactivo para MySQL:

```
mysql>
```

Allí podremos escribir los comandos necesarios para administrar el servidor de bases de datos.

## Comandos para administrar MySQL dede el shell interactivo

La siguiente tabla describe los comandos de uso frecuente que necesitarás para administrar el servidor de bases de datos desde el shell interactivo.

Es una buena idea, imprimir esta tabla para tenerla siempre a

mano :)

| COMANDO | DESCRIPCIÓN |
|---|---|
| **show databases;** | Muestra todas las bases de datos creadas en el servidor |
| **use nombre_de_la_base_de_datos;** | Indicar que vas a comenzar a utilizar la base de datos elegida |
| **create database nombre_de_la_db;** | Crear una nueva base de datos |
| **quit** | Salir del shell interactivo |

# Sobre el lenguaje SQL

SQL -siglas de *Structured Query Language*-, es el **lenguaje de consultas a bases de datos**, que nos permitirá crear, modificar, consultar y eliminar tanto bases de datos como sus tablas y registros, desde el shell interactivo de MySQL y también desde PHP.

Como todo **lenguaje informático**, posee su propia **sintaxis**, **tipos de datos** y **elementos**.

En este curso, **abordaremos los conceptos básicos sobre SQL** que nos permitan desarrollar aplicaciones de media complejidad, sin profundizar en el lenguaje en sí, sino solo en aquellos **aspectos mínimamente necesarios relacionados con MySQL**.

## Tipos de datos más comunes (recomendados)

La siguiente tabla, muestra los tipos de datos más comunes, aceptados por versiones la versión 5.0.3 o superior, de MySQL.

| Tipo de dato | Denominación | Especificaciones | Ejemplo |
|---|---|---|---|
| Entero | **INT(N)** | N = cantidad de dígitos | INT(5) |
| Número decimal | **DECIMAL(N, D)** | N = cantidad de dígitos totales<br>D = cantidad de decimales | DECIMAL(10, 2) |
| Booleano | **BOOL** | | BOOL |
| Fecha | **DATE** | | DATE |
| Fecha y hora | **DATETIME** | | DATETIME |
| Fecha y hora automática | **TIMESTAMP** | | TIMESTAMP |
| Hora | **TIME** | | TIME |

| | | | |
|---|---|---|---|
| Año | **YEAR(D)** | D = cantidad de dígitos (2 o 4) | YEAR(4) |
| Cadena de longitud fija | **CHAR(N)** | N = longitud de la cadena – entre 0 y 255 | CHAR(2) |
| Cadena de longitud variable | **VARCHAR(N)** | N = longitud máxima de la cadena – entre 0 y 65532 | VARCHAR(100) |
| Bloque de texto de gran longitud variable | **BLOB** | | BLOB |

# Sintáxis básica de las sentencias SQL

Una sentencia SQL (denominada *"query"* en la jerga informática), es una **instrucción** escrita en lenguaje SQL. Veremos aquí, el tipo de sentencias más habituales.

## Crear tablas en una base de datos

Sintaxis:

```
CREATE TABLE nombre_de_la_tabla(
    nombre_del_campo TIPO_DE_DATO,
    nombre_de_otro_campo TIPO_DE_DATO
);
```

Ejemplo:

```
CREATE TABLE productos(
    producto VARCHAR(125),
    descripcion BLOB,
    precio DECIMAL(6, 2),
    en_stock BOOL
);
```

Explicación:

```
CREATE TABLE productos
```

Crear una nueva tabla llamada "productos"

```
producto VARCHAR(125),
```

Crear un campo llamado producto, de tipo cadena de texto de longitud variable, con una longitud máxima de 125 caracteres

```
descripcion BLOB,
```

Crear un campo llamado descripción, de tipo bloque de texto de gran longitud

```
precio DECIMAL(6, 2),
```

Crear un campo precio de tipo numérico de longitud máxima de 6 digítos de los cuales, solo 2 pueden ser decimales

```
en_stock BOOL
```

Crear un campo llamado “en_stock” del tipo booleano

# Insertar datos en una tabla

Sintaxis:

```
INSERT INTO
    nombre_de_la_tabla(campo1, campo2, campo10..)
    VALUES(dato1, dato2, dato10...);
```

Ejemplo:

```
INSERT INTO
    productos(producto, precio, en_stock)
    VALUES('Bolsa de dormir para alta montaña', 234.65, TRUE);
```

Explicación:

```
INSERT INTO
    productos(producto, precio, en_stock)
```

Insertar un nuevo registro en los campos producto, precio y en_stock de la tabla productos

```
VALUES('Bolsa de dormir para alta montaña', 234.65, TRUE);
```

Con los valores “Bolsa de dormir para alta montaña”, 234.65 y verdadero, respectivamente en cada uno de los campos indicados

# Seleccionar registros

Sintaxis:

```
SELECT    campo1, campo2, campo10
FROM      tabla;
```

Ejemplo:

```
SELECT    producto, precio
```

```
FROM      productos;
```

Explicación:

```
SELECT    producto, precio
```

Seleccionar los campos producto y precio

```
FROM      productos;
```

De la tabla productos

# Modificar registros

Sintaxis:

```
UPDATE  tabla
SET     campo1 = valor,
        campo2 = valor,
        campo10 = valor;
```

Ejemplo:

```
UPDATE        productos
SET         en_stock = FALSE,
            precio = 0;
```

Explicación:

```
UPDATE        productos
```

Actualizar la tabla productos

```
SET         en_stock = FALSE,
```

Modificar el campo en_stock por falso

```
precio = 0;
```

y el campo precio a 0

## Eliminar registros

Sintaxis:

```
DELETE FROM tabla;
```

Ejemplo:

```
DELETE FROM productos;
```

Explicación:

```
DELETE FROM productos;
```

Eliminar todos los registros de la tabla productos

# Consultas avanzadas

Si bien no veremos aquí consultas realmente complejas, ya que el curso se basa en el lenguaje de programación PHP y no, en el lenguaje de consulta SQL, haremos un rápido paseo, por las opciones disponibles en SQL para sentencias más complejas que las anteriores.

## La cláusula WHERE

Las sentencias en SQL, se componen de **cláusulas**. Y **WHERE** es una de ellas. La **cláusula WHERE** nos permite **filtrar registros en una sentencia** SQL.

Esta cláusula, funciona de forma similar a la comparación de expreciones en PHP, utilizando los siguientes **operadores de comparción**:

```
>        mayor que                  <        menor que
=        igual que                  <>       distinto que
>=       mayor o igual que         <=       menor o igual que

BETWEEN n1 AND n2           entre n1 y n2
IS NULL|TRUE|FALSE          es nulo | es verdadero | es falso
IN(valor1, valor2, va...)   contiene
```

Por supuesto, tambien admite **operadores lógicos**:

```
AND (y)              NOT (negación)        OR (o)
```

Veamos algunos ejemplos:

Seleccionar productos donde precio sea menor que 1000:

```
SELECT    producto,
          precio

FROM      productos
WHERE     precio < 1000;
```

Aumentar el 10% del precio de los productos, que actualmente se encuentren entre 150 y 200:

```
UPDATE    productos
SET       precio = (precio * 1.10)
WHERE     precio BETWEEN 150 AND 200;
```

Seleccionar productos donde en_stock no sea falso

```
SELECT    producto,
          precio
FROM      productos
WHERE     en_stock IS NOT FALSE;
```

Eliminar productos cuyos precios sean 100, 200 y/o 300 y además, en_stock sea falso o producto sea nulo:

```
DELETE
FROM      productos
WHERE     precio IN(100, 200, 300)
AND       (en_stock IS FALSE
OR        producto IS NULL);
```

Modificar en_stock a verdadero donde precio sea menor que 50 y producto no sea nulo:

```
UPDATE    productos
SET       en_stock = TRUE
WHERE     precio < 50
AND       en_stock IS NOT NULL;
```

## Ordenando consultas: la cláusula ORDER BY

Es posible además, ordenar los resultados de una consulta, en forma **ascendente** (`ASC`) o **descendente** (`DESC`):

```
SELECT    producto,
          descripcion,
          precio

FROM      productos

WHERE     precio BETWEEN 1 AND 50
AND       en_stock IS NOT FALSE

ORDER BY  precio DESC;
```

También es posible, ordenar los resultados de la consulta, por más de un campo:

```
SELECT    producto,
          descripcion,
          precio

FROM      productos

WHERE     precio BETWEEN 1 AND 50
AND       en_stock IS NOT FALSE

ORDER BY  precio DESC,
          producto ASC;
```

# Alias de tablas y campos

Otra posibilidad que nos da el lenguaje SQL, es utilizar alias para el nombre de los campos y las tablas. Estos alias se asignan mediante la palabra clave reservada, **AS**:

```
SELECT    producto      AS 'Nombre del Producto',
          descripcion   AS Detalles,
          precio        AS Importe

FROM      productos     AS p

WHERE     precio BETWEEN 1 AND 50
AND       en_stock IS NOT FALSE
```

```
ORDER BY  precio DESC,
          producto ASC;
```

Nótese que los alias que contengan caracteres extraños, deben ser encerrados entre comillas simples

# Funciones del lenguaje SQL de MySQL

Es posible también, utilizar diversas funciones propias del lenguaje SQL -ya sea estandar o de MySQL- a fin de poder obtener los datos con cierto formato. Veremos aquellas de uso más frecuente.

## Contar la cantidad de registros: COUNT()

```
SELECT    COUNT(producto)   AS Cantidad
FROM      productos;
```

## Sumar totales: SUM()

```
SELECT    SUM(precio)   AS Total
FROM      productos;
```

## Concatenar cadenas: CONCAT()

```
SELECT    producto,
          CONCAT('USD ', precio, '.-')    AS Precio
FROM      productos;
```

Nótese que las cadenas de caracteres deben encerrarse entre comillas simples y que el operador de concatenación para esta función, es la coma.

## Convertir a minúsculas y mayúsculas: LCASE() y UCASE()

```
SELECT    UCASE(producto),
          LCASE(descripcion)
FROM      productos;
```

## Reemplazar datos: REPLACE()

```
SELECT    REPLACE(descripcion, '\n', '<br/>')    AS Descripcion
FROM      productos;
```

Reemplaza '\n' por '<br/>'

## Obtener los primeros o últimos caracteres: LEFT() y RIGHT()

```
SELECT    LEFT(producto, 50)
FROM      productos;
```

## Redonder números: ROUND()

```
SELECT    ROUND(precio, 2)
FROM      productos;
```

Retornará los precios con 2 decimales

## Obtener solo la fecha de un campo DATETIME o TIMESTAMP: DATE()

```
SELECT    DATE(campo_datetime)
FROM      tabla;
```

## Obtener una fecha formateada: DATE_FORMAT()

```
SELECT    DATE_FORMAT(campo_fecha, '%d/%m/%Y')
FROM      tabla;
```

Aplican los mismos patrones de formato de fecha que en PHP

## Obtener el registro con el valor máximo y mínimo: MAX() y MIN()

```
SELECT    MAX(precio)
FROM      productos;
```

Retorna el producto con el precio más caro

```
SELECT    MIN(precio)
FROM      productos;
```

Retorna el producto con el precio más barato

# Optimización de bases de Datos

A continuación, encontrarás una lista de consejos que SIEMPRE debes seguir, al momento de crear nuevas tablas y escribir sentencias SQL.

## Todos los registros deben tener un ID único

Cuando crees tablas, asígnales un campo **id** de tipo autonumérico incremental y establécelo como índice primario. Cuando agregues registros, este campo se completará automáticamente, con un número incremental, que te servirá para optimizar tus consultas y contar con un campo que te permita reconocer el registro como único.

```
CREATE TABLE productos(
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    producto VARCHAR(125)
);
```

El campo **id**, será como cualquier otro y lo podrás seleccionar en un SELECT o utilizarlo e cualquier cláusula WHERE.

# Crear índices en las tablas

Todas las tablas deben tener un índice. El índice se asigna a uno o más campos, y es utilizado por SQL para filtrar registros de forma más rápida. Debes crear índices con precaución, ya que de la misma forma que se aceleran las consultas, se retrasa la inserción y actualización de registros, puesto que la base de datos, deberá actualizar los índices cada vez que se agreguen o modifiquen datos.

Cuando una consulta es ejecutada, MySQL tratará de encontrar primero la respuesta en los campos índice, y lo hará en el orden que los índices hayan sido creados.

¿Cuándo agregar índices? Cuando vayas a utilizar una combinación de campos en la cláusula WHERE. Por ejemplo, si filtrarás a menudo, los datos de la tabla producto por su campo precio y en_stock, que precio y en_stock sean un índice de múltiples campos:

```
CREATE TABLE productos(
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    producto VARCHAR(125),
    precio DECIMAL(10, 2),
    en_stock BOOL,
    descripcion BLOB,
    INDEX(precio, en_stock)
);
```

# Indica cuáles campos no pueden ser nulos

SQL te da la posibilidad de indicar qué campos no pueden estar nulos. Indicar que un campo no debe estar nulo, te ayudará a no almacenar registros defectuosos en tu base de datos.

```
CREATE TABLE productos(
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    producto VARCHAR(125) NOT NULL,
    precio DECIMAL(10, 2) NOT NULL,
    en_stock BOOL,
    descripcion BLOB NOT NULL,
    INDEX(precio, en_stock)
);
```

## Utiliza el motor InnoDB

El motor de bases de datos InnoDB, te permitirá crear tablas relaciones optimizando su rendimiento. Al momento de crear tus tablas, indica que utilizarás el motor InnoDB:

```
CREATE TABLE productos(
    id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    producto VARCHAR(125) NOT NULL,
    precio DECIMAL(10, 2) NOT NULL,
    en_stock BOOL,
    descripcion BLOB NOT NULL,
    INDEX(precio, en_stock)
) ENGINE=InnoDB;
```

# Obtener mayor información

- Descarga el Manual de MySQL
- Aprende sobre el lenguaje SQL, gratis en 1KeyData