Programación Web con PHP y MySQL

Unidad 2: Introducción a MYSQL

Contacto: consultas@elearning-total.com
Web: www.elearning-total.com

Indice

Objetivos

Que el alumno logre:

- Aprender los conceptos de tabla, columna, fila o registro y valor de un dato dentro de la tabla.
- Aprender a definir una llave o clave primaria y cuál es el propósito de la misma.

DDL (DATA DEFINITION LANGUAGE)

Es el encargado de la definición de Bases de Datos, tablas, vistas e índices entre otros.

Admite las siguientes sentencias de definición:

- CREATE
- DROP
- ALTER

Cada una de las cuales se puede aplicar a las tablas, vistas, procedimientos almacenados y triggers de la base de datos.

Entonces DLL:

- Añade una nueva base de datos.
- Suprime una base de datos.
- Añade una nueva tabla a la base de datos.
- Suprime una tabla de la base de datos.
- Modifica la estructura de una tabla existente.
- Añade una nueva vista a la base de datos.
- Suprime una vista de la base de datos.
- Construye un índice para una columna.
- Suprime el índice para una columna.
- Define un alias para un nombre de tabla.
- Suprime un alias para un nombre de tabla.

1) **CREATE:** Se utilice esta sentencia para crear bases de datos, tablas, dominios, aserciones y vistas.

2) **ALTER:** Se utilice esta sentencia para modificar tablas y dominios.

3) **DROP:** Se utilice esta sentencia para borrar bases de datos, tablas, dominios, aserciones y vistas.

Obviamente lo primero que hay que hacer para poder trabajar con bases de datos relacionales es definirlas.

Crear una base de datos

Cada conjunto de relaciones que componen un modelo completo forma una base de datos. Desde el punto de vista de SQL, una base de datos es sólo un conjunto de relaciones (o tablas), y para organizarlas o distinguirlas se accede a ellas mediante su nombre. A nivel de sistema operativo, cada base de datos se guarda en un directorio diferente.

Debido a esto, crear una base de datos es una tarea muy simple. Claro que, en el momento de crearla, la base de datos estará vacía, es decir, no contendrá ninguna tabla.

Vamos a crear y manipular nuestra propia base de datos, al tiempo que nos familiarizamos con la forma de trabajar de MySQL.

CREATE

Este comando crea un objeto dentro de la base de datos. Puede ser la propia base de datos, una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte. Para empezar, crearemos una base de datos, para ello se usa la sentencia CREATE

DATABASE:

CREATE DATABASE nombre_de_la_base;

Se puede completar la orden con la cláusula:

CREATE DATABASE IF NOT EXISTS nombre_de_la_base;

En cuyo caso la nueva base de datos solo se intentará crear si no existe otra con el mismo nombre. Si no usamos IF NOT EXISTS y ya existe una base con ese nombre, MySQL nos avisará del error y no ejecutará acción alguna.

A partir de que la base esté creada, si quisiéramos trabajar con esta base de datos, nos conviene seleccionarla como base de datos por defecto. Esto nos permitirá obviar el nombre de la base de datos en consultas. Para seleccionar una base de datos se usa el comando USE, que no es exactamente una sentencia SQL, sino más bien de una opción de MySQL.

USE nombre_de_la_base;

Crear una tabla

Veamos ahora la sentencia CREATE TABLE que sirve para crear tablas. La sintaxis de esta sentencia es muy compleja, ya que existen muchas opciones y tenemos muchas posibilidades diferentes a la hora de crear una tabla. Las iremos viendo paso a paso, y en poco tiempo sabremos usar muchas de sus posibilidades.

En su forma más simple, la sentencia CREATE TABLE creará una tabla con las columnas que indiquemos. Crearemos, como ejemplo, una tabla que nos permitirá almacenar nombres de personas y sus fechas de nacimiento. Deberemos indicar el nombre de la tabla y los nombres y tipos de las columnas.

CREATE TABLE nombre_de_tabla (nombre_del_campo tipo de campo, …);

Por ejemplo:

CREATE TABLE gente (nombre VARCHAR(40), fecha DATE);

Hemos creado una tabla llamada "gente" con dos columnas: "nombre" que puede contener cadenas de hasta 40 caracteres y "fecha" de tipo fecha.

Podemos consultar cuántas tablas y qué nombres tienen en una base de datos, usando la sentencia SHOW TABLES:

SHOW TABLES;

Pero tenemos muchas más opciones a la hora de definir campos. Además del tipo y el nombre, podemos definir valores por defecto, permitir o no que contengan valores nulos, crear una clave primaria, indexar...

La sintaxis para definir campos es:

nombre_col tipo [NOT NULL | NULL] [DEFAULT

valor_por_defecto] [AUTO_INCREMENT] [[PRIMARY] KEY]

[COMMENT 'string'] [definición_referencia]

Veamos cada una de las opciones por separado.

Valores nulos

Al definir cada columna podemos decidir si podrá o no contener valores nulos. El concepto del valor NULL es una fuente común de confusión para los recién llegados a SQL, que frecuentemente piensan que NULL es lo mismo que una cadena de caracteres vacía ''. Esto no es así.

NULL indica que el valor es desconocido. Un valor NULL no es lo mismo que un valor cero o vacío. No hay dos valores NULL que sean iguales. La comparación entre dos valores NULL, o entre un valor NULL y cualquier otro valor, tiene un resultado desconocido porque el valor de cada NULL es desconocido.

Normalmente, los valores NULL indican que los datos son desconocidos, no aplicables o que se agregarán posteriormente. Por ejemplo, la inicial de un cliente puede que no sea conocida en el momento en que éste hace un pedido.

Debemos dejar en claro que, aquellas columnas que son o forman parte de una clave primaria no pueden contener valores nulos. Veremos que, si definimos una columna como clave primaria, automáticamente se impide que pueda contener valores nulos, pero este no es el único caso en que puede ser interesante impedir la asignación de valores nulos para una columna.

La opción por defecto es que se permitan valores nulos, NULL, y para que no se permitan, se usa NOT NULL. Por ejemplo:

CREATE TABLE provincia1 (nombre CHAR(20) NOT NULL, poblacion INT NULL);

Valores por defecto

Para cada columna también se puede definir, opcionalmente, un valor por defecto. El valor por defecto se asignará de forma automática a una columna cuando no se especifique un valor determinado al añadir filas.

Si una columna puede tener un valor nulo, y no se especifica un valor por defecto, se usará NULL como valor por defecto. En el ejemplo anterior, el valor por defecto para poblacion es NULL.

Por ejemplo, si queremos que el valor por defecto para poblacion sea 5000, podemos crear la tabla como:

CREATE TABLE provincia2 (nombre CHAR(20) NOT NULL, poblacion INT NULL DEFAULT 5000);

Claves primarias

También se puede definir una clave primaria sobre una columna, usando la palabra clave KEY o PRIMARY KEY. Sólo puede existir una clave primaria en cada tabla, y la columna sobre la que se define una clave primaria no puede tener valores NULL. Si esto no se especifica de forma explícita, MySQL lo hará de forma automática.

Por ejemplo, si queremos crear un índice en la columna nombre de la tabla de provincias, crearemos la tabla así:

CREATE TABLE provincia3 (nombre CHAR(20) NOT NULL

PRIMARY KEY, poblacion INT NULL DEFAULT 5000);

Usar NOT NULL PRIMARY KEY equivale a PRIMARY KEY, NOT NULL KEY o sencillamente KEY.

Campos autoincrementados

En MySQL tenemos la posibilidad de crear un campo autoincrementado, aunque esta columna sólo puede ser de tipo entero. Si al insertar una fila se omite el valor de la columna autoincrementada o si se inserta un valor nulo para esa columna, su valor se calcula automáticamente, tomando el valor más alto de esa columna y sumándole una unidad. Esto permite crear, de una forma sencilla, una columna con un valor único para cada fila de la tabla.

Generalmente, estas columnas se usan como claves primarias 'artificiales'. MySQL está optimizado para usar valores enteros como claves primarias, de modo que la combinación de clave primaria, que sea entera y autoincrementada es ideal para usarla como clave primaria artificial:

CREATE TABLE provincia4 (clave INT AUTO_INCREMENT

PRIMARY KEY, nombre CHAR(20) NOT NULL, poblacion INT

NULL DEFAULT 5000);

Comentarios

Adicionalmente, al crear la tabla, podemos añadir un comentario a cada columna. Este comentario sirve como información adicional sobre alguna característica especial de la columna, y entra en el apartado de documentación de la base de datos:

CREATE TABLE provincia5 (clave INT AUTO_INCREMENT

PRIMARY KEY COMMENT 'Clave principal', nombre CHAR(50)

NOT NULL, poblacion INT NULL DEFAULT 5000);

Índices

Un índice (o KEY, o INDEX) es un grupo de datos que MySQL asocia con una o varias columnas de la tabla. En este grupo de datos aparece la relación entre el contenido y el número de fila donde está ubicado. Los índices -como los índices de los libros- sirven para agilizar las consultas a las tablas, evitando que mysql tenga que revisar todos los datos disponibles para devolver el resultado.

Los índices se utilizan para buscar las filas con valores de columna específica rápidamente. Sin un índice, MySQL debe comenzar con el registro primero y luego leer a través de toda la tabla para buscar las filas correspondientes. Cuanto más grande sea la tabla, más tarda este proceso. Si la tabla tiene un índice para las columnas que se trate, MySQL puede determinar rápidamente la posición de buscar en el medio del archivo de datos sin tener que mirar todos los datos. Si una tabla tiene 1000 filas, entonces esto es por lo menos 100 veces más rápido que la lectura secuencial.

Podemos crear el índice a la vez que creamos la tabla, usando la palabra INDEX seguida del nombre del índice a crear y columnas a indexar (que pueden ser varias):

INDEX nombre_indice (columna_indexada, columna_indexada2...)

La sintaxis es ligeramente distinta según la clase de índice:

PRIMARY KEY (nombre_columna_1 [,nombre_columna2...])

UNIQUE INDEX nombre_indice (columna_indexada1

[,columna_indexada2 ...])

INDEX nombre_index (columna_indexada1

[,columna_indexada2...])

Tenemos tres tipos de índices. El primero corresponde a las claves primarias, que como vimos, también se pueden crear en la parte de definición de columnas.

Claves primarias

La sintaxis para definir claves primarias es:

definición_columnas… PRIMARY KEY (index_nombre_col,...)

El ejemplo anterior que vimos para crear claves primarias, usando esta sintaxis, quedaría así:

CREATE TABLE provincia6 (nombre CHAR(20) NOT NULL,

poblacion INT NULL DEFAULT 5000, PRIMARY KEY (nombre));

Pero esta forma tiene más opciones, por ejemplo, entre los paréntesis podemos especificar varios nombres de columnas, para construir claves primarias compuestas por varias columnas:

CREATE TABLE mitabla1 (id1 CHAR(2) NOT NULL, id2

CHAR(2) NOT NULL, texto CHAR(30), PRIMARY KEY (id1,

id2));

Índices

El segundo tipo de índice permite definir índices sobre una columna, sobre varias, o sobre partes de columnas. Para definir estos índices se usan indistintamente las opciones KEY o INDEX.

CREATE TABLE mitabla2 (id INT, nombre CHAR(19), INDEX

(nombre));

O su equivalente:

CREATE TABLE mitabla3 (id INT, nombre CHAR(19), KEY

(nombre));

También podemos crear un índice sobre parte de una columna:

CREATE TABLE mitabla4 (id INT, nombre CHAR(19), INDEX

(nombre(4)));

Este ejemplo usará sólo los cuatro primeros caracteres de la columna 'nombre' para crear el índice.

Claves únicas

El tercero permite definir índices con claves únicas, también sobre una columna, sobre varias o sobre partes de columnas. Para definir índices con claves únicas se usa la opción UNIQUE.

La diferencia entre un índice único y uno normal es que en los únicos no se permite la inserción de filas con claves repetidas. La excepción es el valor NULL, que sí se puede repetir.

CREATE TABLE mitabla5 (id INT, nombre CHAR(19), UNIQUE

(nombre));

Una clave primaria equivale a un índice de clave única, en la que el valor de la clave no puede tomar valores NULL. Tanto los índices normales como los de claves únicas sí pueden tomar valores NULL.

Por lo tanto, las definiciones siguientes son equivalentes:

CREATE TABLE mitabla6 (id INT, nombre CHAR(19) NOT NULL,

UNIQUE (nombre));

Y:

CREATE TABLE mitabla7 (id INT, nombre CHAR(19), PRIMARY

KEY (nombre));

Los índices sirven para optimizar las consultas y las búsquedas de datos. Mediante su uso es mucho más rápido localizar filas con determinados valores de columnas, o seguir un determinado orden. La alternativa es hacer búsquedas secuenciales, que en tablas grandes requieren mucho tiempo.

Claves foráneas

En MySQL sólo existe soporte para claves foráneas en tablas de tipo InnoDB. Sin embargo, esto no impide usarlas en otros tipos de tablas.

La diferencia consiste en que en esas tablas no se verifica si una clave foránea existe realmente en la tabla referenciada, y que no se eliminan filas de una tabla con una definición de clave foránea. Para hacer esto hay que usar tablas InnoDB.

Hay dos modos de definir claves foráneas en bases de datos MySQL.

El primero, sólo sirve para documentar, y, al menos en las pruebas que he hecho, no define realmente claves foráneas. Esta forma consiste en definir una referencia al mismo tiempo que se define una columna:

CREATE TABLE personas (id INT AUTO_INCREMENT PRIMARY

KEY, nombre VARCHAR(40), fecha DATE);

Hemos usado una definición de referencia para la columna 'id' de la tabla 'telefonos', indicando que es una clave foránea correspondiente a la columna 'id' de la tabla 'personas' (1). Sin embargo, aunque la sintaxis se comprueba, esta definición no implica ningún comportamiento por parte de MySQL.

DROP

Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte.

Se puede combinar con la sentencia ALTER (que veremos en breve).

Ejemplo 1:

DROP TABLE TABLA_NOMBRE

Ejemplo 2:

ALTER TABLE TABLA_NOMBRE (DROP COLUMN CAMPO_NOMBRE1)

ELIMINAR UNA TABLA

A veces es necesario eliminar una tabla, ya sea porque es más sencillo crearla de Nuevo que modificarla, o porque ya no es necesaria.

Para eliminar una tabla se usa la sentencia DROP TABLE.

La sintaxis es simple:

DROP TABLE tbl_name [, tbl_name] ...

Por ejemplo:

DROP TABLE provincia6;

Se pueden añadir las palabras IF EXISTS para evitar errores si la tabla a eliminar no existe.

DROP TABLE [IF EXISTS] tbl_name [, tbl_name] ...

Por ejemplo:

DROP TABLE provincia6;

ERROR 1051 (42S02): Unknown table 'provincia6'

Para evitar el mensaje de error:

DROP TABLE IF EXISTS provincia6;

ELIMINAR UNA BASE DE DATOS

De modo parecido, se pueden eliminar bases de datos completas, usando la sentencia

DROP_DATABASE.

La sintaxis también es muy simple:

DROP DATABASE [IF EXISTS] db_name

Hay que tener cuidado, ya que al borrar cualquier base de datos se elimina también cualquier tabla que contenga.

ALTER

Este comando permite modificar la estructura de un objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla modificar un trigger, etc.

Ejemplo 1 (agregar columna a una tabla):

ALTER TABLE TABLA_NOMBRE (ADD NUEVO_CAMPO INT UNSIGNED)

ALTER TABLE Empleados (ADD Salario CURRENCY)(Agrega un campo

Salario de tipo Moneda a la tabla Empleados.)

ALTER TABLE Empleados DROP Salario (Elimina el campo Salario de la tabla Empleados.)

Resumiendo, "ALTER TABLE" se usa para:

- agregar nuevos campos,
- eliminar campos existentes,
- modificar el tipo de dato de un campo,
- agregar o quitar modificadores como "NULL", "UNSIGNED", "AUTO_INCREMENT",
- cambiar el nombre de un campo,
- agregar o eliminar la clave primaria,
- agregar y eliminar índices,
- renombrar una tabla.

"**ALTER TABLE**" hace una copia temporal de la tabla original, realiza los cambios en la copia, luego borra la tabla original y renombra la copia.

Ejemplo: agregar campos a una tabla.

Para ello utilizamos la tabla "libros", definida con la siguiente estructura:

- código, INT UNSIGNED AUTO_INCREMENT, CLAVE PRIMARIA,
- titulo, VARCHAR(40) NOT NULL,
- autor, VARCHAR(30),
- editorial, VARCHAR (20),
- precio, DECIMAL(5,2) UNSIGNED.

Necesitamos agregar el campo "cantidad", de tipo SMALLINT UNSIGNED NOT NULL, tipeamos:

```
ALTER TABLE libros ADD cantidad SMALLINT UNSIGNED NOT NULL;
```

Usamos "ALTER TABLE" seguido del nombre de la tabla y "ADD" seguido del nombre del nuevo campo con su tipo y los modificadores.

Agreguemos otro campo a la tabla:

```
ALTER TABLE libros ADD edicion DATE;
```

Si intentamos agregar un campo con un nombre existente, aparece un mensaje de error indicando que el campo ya existe y la sentencia no se ejecuta. Cuando se agrega un campo, si no especificamos, lo coloca al final, después de todos los campos existentes; podemos indicar su posición (luego de qué campo debe aparecer) con "AFTER":

```
ALTER TABLE libros ADD CANTIDAD TINYINT UNSIGNED AFTER autor;
```

"ALTER TABLE" nos permite alterar la estructura de la tabla, podemos usarla para eliminar un campo. Continuamos con nuestra tabla "libros".

Para eliminar el campo "edicion" tipeamos:

ALTER TABLE libros DROP edicion;

Entonces, para borrar un campo de una tabla usamos "ALTER TABLE" junto con

"DROP" y el nombre del campo a eliminar.

Si intentamos borrar un campo inexistente aparece un mensaje de error y la acción no se realiza.

Podemos eliminar 2 campos en una misma sentencia:

ALTER TABLE libros DROP editorial, DROP cantidad;

Si se borra un campo de una tabla que es parte de un índice, también se borra el índice.

Si una tabla tiene sólo un campo, éste no puede ser borrado. Hay que tener cuidado al eliminar un campo, éste puede ser clave primaria. Es posible eliminar un campo que es clave primaria, no aparece ningún mensaje:

ALTER TABLE libros DROP codigo;

Si eliminamos un campo clave, la clave también se elimina.

Con "ALTER TABLE" podemos modificar el tipo de algún campo incluidos sus atributos. Continuamos con nuestra tabla "libros", definida con la siguiente estructura:

- código, int unsigned,
- titulo, varchar(30) not null,
- autor, varchar(30),
- editorial, varchar (20),
- precio, decimal(5,2) unsigned,
- cantidad int unsigned.

Queremos modificar el tipo del campo "cantidad", como guardaremos valores que no superarán los 50000 usaremos SMALLINT UNSIGNED, tipeamos:

ALTER TABLE libros MODIFY cantidad SMALLINT UNSIGNED;

Usamos "ALTER TABLE" seguido del nombre de la tabla y "MODIFY" seguido del nombre del nuevo campo con su tipo y los modificadores. Queremos modificar el tipo del campo "titulo" para poder almacenar una longitud de 40 caracteres y que no permita valores nulos, tipeamos:

ALTER TABLE libros MODIFY titulo VARCHAR(40) NOT NULL;

Hay que tener cuidado al alterar los tipos de los campos de una tabla que ya tiene registros cargados. Si tenemos un campo de texto de longitud 50 y lo cambiamos a 30 de longitud, los registros cargados en ese campo que superen los 30 caracteres, se cortarán.

Igualmente, si un campo fue definido permitiendo valores nulos, se cargaron registros con valores nulos y luego se lo define "NOT NULL", todos los registros con valor nulo para ese campo cambiarán al valor por defecto según el tipo (cadena vacía para tipo texto y 0 para numéricos), ya que "NULL" se convierte en un valor inválido.

Si definimos un campo de tipo DECIMAL (5,2) y tenemos un registro con el valor "900.00" y luego modificamos el campo a "decimal(4,2)", el valor "900.00" se convierte en un valor inválido para el tipo, entonces guarda en su lugar, el valor límite más cercano, "99.99".

Si intentamos definir "auto_increment" un campo que no es clave primaria, aparece un mensaje de error indicando que el campo debe ser clave primaria. Por ejemplo:

ALTER TABLE libros MODIFY codigo INT UNSIGNED

AUTO_INCREMENT;

"ALTER TABLE" combinado con "MODIFY" permite agregar y quitar campos y atributos de campos. Para modificar el valor por defecto ("DEFAULT") de un campo podemos usar también "MODIFY" pero debemos colocar el tipo y sus modificadores, entonces resulta muy extenso, podemos setear sólo el valor por defecto con la siguiente sintaxis:

ALTER TABLE libros ALTER autor SET DEFAULT 'Varios';

Para eliminar el valor por defecto podemos emplear:

ALTER TABLE libros ALTER autor DROP DEFAULT;

Con "ALTER TABLE" podemos también cambiar el nombre de los campos de una tabla. Continuamos con nuestra tabla "libros", definida con la siguiente estructura:

- código, int unsigned auto_increment,
- nombre, varchar(40),
- autor, varchar(30),
- editorial, varchar (20),
- costo, decimal(5,2) unsigned,
- cantidad int unsigned,
- clave primaria: código.

Queremos cambiar el nombre del campo "costo" por "precio", tipeamos:

ALTER TABLE libros CHANGE costo precio DECIMAL (5,2);

Usamos "ALTER TABLE" seguido del nombre de la tabla y "CHANGE" seguido del nombre actual y el nombre nuevo con su tipo y los modificadores. Con "CHANGE" cambiamos el nombre de un campo y también podemos cambiar el tipo y sus modificadores. Por ejemplo, queremos cambiar el nombre del campo "nombre" por "titulo" y redefinirlo como "NOT NULL", tipeamos:

ALTER TABLE libros CHANGE nombre titulo VARCHAR(40) NOT NULL;

Con "ALTER TABLE" podemos agregar una clave primaria a una tabla existente.

Continuamos con nuestra tabla "libros", con la misma estructura. Para agregar una clave primaria a una tabla existente usamos:

ALTER TABLE libros ADD PRIMARY KEY (codigo);

Usamos "ALTER TABLE" con "ADD PRIMARY KEY" y entre paréntesis el nombre del campo que será clave. Si intentamos agregar otra clave primaria, aparecerá un mensaje de error porque (recuerde) una tabla solamente puede tener una clave primaria.

Para que un campo agregado como clave primaria sea autoincrementable, es necesario agregarlo como clave y luego redefinirlo con "MODIFY" como "AUTO_INCREMENT". No se puede agregar una clave y al mismo tiempo definir el campo autoincrementable. Tampoco es posible definir un campo como autoincrementable y luego agregarlo como clave porque para definir un campo

"AUTO_INCREMENT" éste debe ser clave primaria.

También usamos "alter table" para eliminar una clave primaria.

ALTER TABLE libros DROP PRIMARY KEY;

Con "ALTER TABLE" y "DROP PRIMARY KEY" eliminamos una clave primaria definida al crear la tabla o agregada luego. Si queremos eliminar la clave primaria establecida en un campo "AUTO_INCREMENT" aparece un mensaje de error y la sentencia no se ejecuta porque si existe un campo con este atributo DEBE ser clave primaria. Primero se debe modificar el campo quitándole el atributo "AUTO_INCREMENT" y luego se podrá eliminar la clave. Si intentamos establecer como clave primaria un campo que tiene valores repetidos, aparece un mensaje de error y la operación no se realiza.

Podemos además cambiar el nombre de una tabla con "ALTER TABLE". Para cambiar el nombre de una tabla llamada "libros" por "ejemplares" usamos esta sintaxis:

ALTER TABLE libros RENAME ejemplares;

Entonces usamos "alter table" seguido del nombre actual, "rename" y el nuevo nombre.

También podemos cambiar el nombre a una tabla usando la siguiente sintaxis:

RENAME TABLE datos TO contactos;

La renombración se hace de izquierda a derecha, con lo cual, si queremos intercambiar los nombres de dos tablas, debemos tipear lo siguiente:

RENAME TABLE amigos TO auxiliar, contactos TO amigos, auxiliar TO contactos;

Resumen

En esta Unidad…

En la presente unidad trabajamos con los parámetros de SQL necesarios para comenzar a interactuar con nuestras bases de datos, para prepararnos para incorporar contenido dinámico a nuestros sitios.

En la próxima Unidad…

En la próxima unidad seguiremos trabajando con MySQL.