**Conectar con el servidor MySQL**

Lo primero que tendremos que hacer es conectar con el sistema gestor de MySQL. Para ello, desde la línea de comandos invocamos a MySQL. Para ello, simplemente tenemos que escribir el comando "mysql" e indicarle unas opciones de conexión.

% mysql

Con el "%" expresamos el principio de la línea de comandos. Ese principio es el prompt que tengamos en nuestra consola de Linux o MsDOS, que puede ser algo como c:\mysql\bin>. El carácter "%", por tanto, no tenemos que escribirlo.

Con esa sentencia se conecta uno con la base de datos con los parámetros por defecto. Es decir, al servidor local, con usuario y password igual a cadenas vacías.

Lo más normal es que tengamos que indicar algún otro dato para conectar con la base de datos, como el usuario, la clave o la dirección del servidor con el que queremos conectar. La sintaxis sería la siguiente:

% mysql -h nombre\_servidor -u nombre\_usuario –p

% mysql -h nombre\_servidor -u nombre\_usuario –p –port puerto\_mysql

**En el comando mysql el puerto siempre tiene que ir al final.**

Si deseamos conectarnos a la base de datos en local y con nombre de usuario root tendríamos que escribir:

% mysql -h localhost -u root –p

% mysql -h localhost -u root –p –port 3310

% mysql -h localhost -u root –p –P 3310

Lo primero que nos preguntará será el password para el usuario root. Una vez introducida la clave, ya estaremos dentro de la línea de comandos de MySQL. Con ello el prompt cambiará a algo como esto:

mysql>

Podríamos haber indicado la contraseña directamente en la línea de comandos para iniciar la sesión con MySQL, pero esto se desaconseja por razones de seguridad. De todos modos, la sintaxis hubiera sido:

% mysql -h localhost -u root –p mi\_clave

Nos fijamos que entre -h y el nombre del host hay un espacio en blanco, igual que entre -u y el nombre de usuario. Sin embargo, entre -p y la clave no debemos poner ningún espacio.

**Dentro de la consola de MySQL**

Una vez dentro, tendremos a nuestra disposición todas las sentencias de MySQL para el trabajo con la base de datos y el lenguaje SQL.

Lo más normal es que primero te tengas que conectar con una base de datos en concreto, de entre todas las que puedes tener creadas en tu servidor MySQL. Eso se hace con el comando use, seguido del nombre de la base de datos que deseas conectar.

mysql> use mibasedatos;

Esto nos conectaría con la base de datos llamada "mibasedatos".

**Nota:** Hay que fijarse que todas las sentencias dentro de la línea de comandos de MySQL acaban en ";". Si no colocamos el punto y coma, lo más seguro es que no se ejecute el comando y nos vuelva a salir el prompt para que sigamos introduciendo el comando. Si lo que queríamos era ejecutar la sentencia que habíamos escrito antes, con simplemente entrar el ";" será suficiente. Es decir, no debemos escribir de nuevo la sentencia entera, sólo el ";" y volver a apretar "enter".

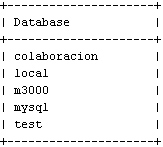
Si queremos ver una lista de las bases de datos alojadas en nuestro servidor podemos escribir el comando show databases. Así:

mysql>show databases;

Con esto nos mostraría una lista de las bases de datos de nuestro servidor. Algo como esto:

mysql> show databases

-> ;



5 rows in set (0.02 sec)

Si queremos crear una base datos, podremos hacerlo con el comando "create database" seguido del nombre de la nueva base de datos.

mysql> create database miprueba;

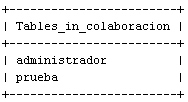
Eso nos creará una base de datos que se llama "miprueba". Como habíamos comentado, si queremos luego usar esa base de datos escribiríamos:

mysql> use miprueba;

Lógicamente, esta base de datos recién creada estará vacía, pero si estuviéramos usando una base de datos ya creada y queremos ver las tablas que tiene escribiríamos el comando "show tables".

mysql> show tables;

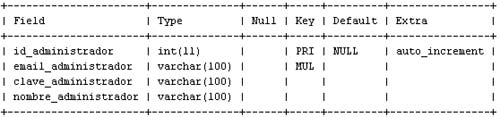
Si no hay tablas, nos dirá algo como "Empty set", pero si tenemos varias tablas dadas de alta en la base de datos que estamos usando, nos saldrá una lista de ellas:



2 rows in set (0.00 sec)

Ahora, si deseamos obtener información sobre una tabla, para saber qué campos tiene y de qué tipo, podremos utilizar el comando describe seguido del nombre de la tabla.

mysql> describe administrador;



4 rows in set (0.11 sec)

**Otras Sentencias SQL**

Desde la consola de MySQL podemos indicar por línea de comandos todo tipo de sentencias en lenguaje SQL, como selecciones, inserciones, actualizaciones, creación de tablas, etc. El mecanismo es el que se puede deducir. Simplemente colocamos la sentencia a ejecutar seguida del punto y coma. Veamos una serie de sentencias seguidas y el resultado de ejecutarlas:

mysql> create table prueba (id\_prueba int);

Query OK, 0 rows affected (0.08 sec)

mysql> insert into prueba (id\_prueba) values (1);

Query OK, 1 row affected (0.00 sec)

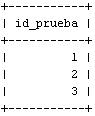
mysql> insert into prueba (id\_prueba) values (2);

Query OK, 1 row affected (0.00 sec)

mysql> insert into prueba (id\_prueba) values (3);

Query OK, 1 row affected (0.00 sec)

mysql> select \* from prueba;



3 rows in set (0.00 sec)

En definitiva, podemos ejecutar todas las sentencias que se han aprendido en el [manual del lenguaje SQL](http://www.desarrolloweb.com/manuales/9/)

**Para salir de la línea de comandos de MySQL**

Una vez hemos terminado de trabajar con MySQL, si queremos cerrar la conexión con el servidor, simplemente escribimos "quit" desde el prompt de MySQL:

mysql> quit

### Conectándose a la base de datos

Esta guía asume que ya tienes creada una base de datos, así como un usuario con los privilegios necesarios para hacer las operaciones que se requieren en la base de datos.

Los cuatro parámetros que necesitamos para establecer una conexión a la base de datos es el host donde reside la base de datos, el nombre de usuario, la contraseña y el nombre de la base de datos que vamos a manipular.

mysql -h [host] -D [base de datos] -u [usuario] -p

Esto te pedirá la contraseña, para que no sea guardada en el historial, por ejemplo:

mysql -h servidor.jveweb.net -D nombre\_base\_de\_datos -u juan -p

Puedes especificar la contraseña en el comando agregando la contraseña junto a -p, no dejes un espacio entre -p y la contraseña para conectarte de esta manera, aunque no usar la contraseña en el comando es recomendable, por ejemplo:

mysql -h servidor.jveweb.net -D nombre\_base\_de\_datos -u juan -p lacontraseña

El parámetro -D para especificar la base de datos a usar desde que nos conectamos también es opcional, si no lo usas puedes ver una lista de las bases de datos disponibles usando show databases; y seleccionar la base de datos a usar con use [nombre base de datos]; en la linea de comandos de mysql, por ejemplo: use **usuarios**;

Si funcionó, obtendremos un resultado similar a este:

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 6324623

Server version: 5.1.39-log MySQL Server

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.

This software comes with ABSOLUTELY NO WARRANTY. This is free software,

and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input

statement.

mysql>

Para terminar la sesión escribe quit. Si te estás conectando a una base de datos ubicada en un host externo, es recomendable el uso de SSL al conectarse a la base de datos, para hacer esto usa el parámetro --ssl

### Enviando peticiones al shell de MySQL

Una vez que estamos en el shell de MySQL, podemos enviar peticiones de MySQL. Para ejecutarlas tenemos que terminarlas con un punto y coma (;), o con \g, por ejemplo:

show tables;

La petición no es ejecutada hasta que el punto y coma es encontrado, esto nos permite escribir peticiones de MySQL en lineas múltiples, por ejemplo:

show  
tables  
;

Si queremos presentar los resultados verticalmente, necesitamos terminar las peticiones con \G en vez de un punto y coma o \g

### Usando el editor

En sistemas basados en Unix como Linux, el comando edit desde dentro del shell de mysql lanza el editor que está definido en la variable de entorno EDITOR. Cuando usamos el comando edit, si habíamos hecho una petición previamente, el editor será abierto con esta petición, esto es muy útil para hacer correcciones a la última petición, de otra manera solo obtendremos un editor vacío para escribir lo que necesitemos. Una vez que terminemos de editar la petición, guardamos, salimos del editor, y entonces usamos un punto y coma o \g para ejecutar la petición(es) que acabamos de escribir.

Para configurar la variable de entorno EDITOR, usamos export, en este ejemplo yo configuro vim ya que es mi editor preferido, pero puedes utilizar uno más fácil como nano. El editor por default es vi:

export EDITOR=vim

Para revisar el valor de la variable de entorno EDITOR, podemos utilizar:

echo $EDITOR

### Procesando un archivo por lotes

Podemos ejecutar un archivo por lotes de peticiones MySQL utilizando:

mysql -u usuario –p contraseña -h host -D nombre\_base\_de\_datos < archivo\_lotes.sql

O, si estamos dentro de el shell de mysql, podemos usar:

source archivo\_lotes.sql

### Peticiones MySQL para manipular tablas

Quien tenga el trabajo de crear scripts y programas que interactúan con MySQL seguramente está familiarizado con estas peticiones, pero ya que utilizo mi propio sitio web como referencia pondré aquí algunas peticiones comunes.

**Listar tablas existentes en la base de datos**

show tables;

**Mostrar información de las tablas en la base de datos**

show tables solo nos mostrará los nombres de las tablas en la base de datos, para ver toda la información sobre las tablas, usa:

show table status;

La información presentada sobre la tabla es:

* Name - El nombre de la tabla
* Engine - Mecanismo de la tabla (MyISAM, InnoDB, Memory, CVS, etc.)
* Version - Número de versión del archivo .frm de la tabla
* Row\_format - El formato de almacenamiento de las filas (Dinámico, Redundante, etc.)
* Rows - Número de filas en la tabla
* Avg\_row\_length - Longitud promedio de las filas
* Data\_length - Longitud del archivo de datos
* Max\_data\_length - La máxima longitud del archivo de datos
* Index\_length - La longitud del archivo índice
* Data\_free - Número de bytes asignados pero no usados
* Auto\_increment - El próximo valor de auto-incremento
* Create\_time - Cuando fue creada la tabla
* Update\_time - Cuando fue actualizado el archivo de datos por última vez
* Check\_time - Cuando fue revisada la tabla por última vez
* Collation - El set de caracteres y la colación de la tabla
* Checksum - El checksum vivo
* Create\_options - Opciones extras utilizadas cuando fue creada la tabla
* Comment - El comentario de la tabla

Podemos especificar de que tabla queremos ver la información utilizando:

show table status like 'nombre\_de\_la\_tabla';

Y podemos buscar en otro campo por cierto valor, por ejemplo, para mostrar todas las tablas que usan el mecanismo de almacenamiento MyISAM, podemos utilizar:

show table status where `Engine` like 'MyISAM';

**Crear una nueva tabla**

Este es un ejemplo del comando para crear una tabla, agregué muchos campos diferentes para la referencia de como declararlos.

create table `nombre\_base\_de\_datos`.`tabla\_de\_prueba` (  
`campo\_id` int( 11 ) unsigned not null auto\_increment comment 'clave primaria',  
`campo\_indice1` int( 11 ) unsigned not null comment 'un índice',  
`campo\_indice2` int( 11 ) unsigned not null comment 'un índice',  
`campo\_indice3` int( 11 ) unsigned not null comment 'un índice',  
`campo\_unico1` int( 11 ) unsigned not null comment 'un campo único',  
`campo\_unico2` int( 11 ) unsigned not null comment 'un campo único',  
`campo\_unico3` int( 11 ) unsigned not null comment 'un campo único',  
`campo\_varchar` varchar( 100 ) not null comment 'un campo varchar',  
`campo\_date` date not null comment 'un campo date',  
`campo\_datetime` datetime not null comment 'un campo datetime',  
`campo\_float` float not null comment 'un campo float',  
`campo\_longtext` longtext not null comment 'un campo longtext',  
`campo\_bool` bool not null comment 'un campo bool',  
`campo\_char` char( 1 ) not null comment 'un campo char',  
`campo\_tinyint` tinyint not null comment 'un campo tinyint',  
primary key ( `campo\_id` ) ,  
index ( `campo\_indice1` , `campo\_indice2`, `campo\_indice3` ) ,  
unique ( `campo\_unico1` , `campo\_unico2`, `campo\_unico3`)  
) engine = myisam character set utf8 collate utf8\_general\_ci comment =  
'comentarios de tabla';

**Listar los campos en una tabla**

show columns from `tabla\_de\_prueba`;

**Cambiar el nombre de un campo en una tabla**

alter table `tabla\_de\_prueba` change `campo\_indice\_1` `nuevo\_nombre\_campo` int(11) unsigned not null;

**Agregar un campo a una tabla y hacerlo un índice**

alter table `tabla\_de\_prueba` add `nuevo\_campo\_indice` int(11) unsigned not null, add index(`nuevo\_campo\_indice`);

**Quitar un índice de una tabla**

alter table `tabla\_de\_prueba` drop index `nuevo\_campo\_indice`;

**Quitar un campo de una tabla**

alter table `tabla\_de\_prueba` drop `nuevo\_campo\_indice`;

**Agregar campos después de un campo especificado**

alter table `tabla\_de\_prueba` add `a\_borrar` varchar(12) not null after `campo\_date`;

**Agregar campos al comienzo de la tabla**

alter table `tabla\_de\_prueba` add `a\_borrar\_2` varchar(12) not null first;

**Agregar múltiples campos a la tabla**

alter table `tabla\_de\_prueba` add `a\_borrar\_3` varchar(12) not null after `a\_borrar`, add `a\_borrar\_4` varchar(12) not null after `a\_borrar\_3`;

**Borrar campos en una tabla**

alter table `tabla\_de\_prueba` drop `a\_borrar`, drop `a\_borrar\_2`, drop `a\_borrar\_3`, drop `a\_borrar\_4`;

**Renombrar una tabla**

rename table `nombre\_base\_de\_datos`.`nombre\_original` to `nombre\_base\_de\_datos`.`nuevo\_nombre`;

**Cambiar el comentario de una tabla**

alter table `tabla\_de\_prueba` comment='Los comentarios';

**Cambiar el valor de auto incremento de una tabla**

alter table `tabla\_de\_prueba` auto\_increment=3;

**Reparar una tabla**

repair table `tabla\_de\_prueba`;

**Optimizar una tabla**

optimize table `tabla\_de\_prueba`;

**Borrar todos los registros en una tabla**

truncate table `tabla\_de\_prueba`;

**Borrar una tabla**

drop table `tabla\_de\_prueba`;

### Peticiones MySQL para manipular registros en una tabla

Voy a utilizar las siguientes dos tablas ficticias para los ejemplos que voy a estar utilizando referentes a la manipulación de registros.

|  |  |
| --- | --- |
| Tabla de estados (estados) | |
| **estado\_id** | **estado\_nombre** |
| 1 | Jalisco |
| 2 | Guanajuato |
| 3 | Hidalgo |

|  |  |  |  |
| --- | --- | --- | --- |
| Tabla de ciudades (ciudades) | | | |
| **ciudad\_id** | **ciudad\_nombre** | **ciudad\_poblacion** | **estado\_id** |
| 1 | Guadalajara | 1494134 | 1 |
| 2 | Tequila | 33155 | 1 |
| 3 | Zapopan | 1243538 | 1 |
| 4 | Tonalá | 408729 | 1 |
| 5 | Tlaquepaque | 563006 | 1 |
| 6 | Guanajuato | 171709 | 2 |
| 7 | Celaya | 468064 | 2 |
| 8 | León | 1436733 | 2 |
| 9 | Pachuca | 275578 | 3 |
| 10 | Tizayuca | 100562 | 3 |

**Mostrar los registros de una tabla**

select [campos] from `tabla\_nombre` [where condiciones] [order by orden1 asc/desc,orden2 asc/desc] [limit inicio,limite];  
  
select [x.campo,y.campo] from `tabla1` x, `tabla2` y where y.`id`=x.`indice` [condiciones extras] [order by x.campo,y.campo] [limit inicio,limite];

El número de combinaciones que podemos hacer con el comando select es enorme, así que trataré de cubrir algunos usos comunes en los siguientes ejemplos. Los campos que usamos siguiendo el select son los campos que serán mostrados, y el orden en que los especificamos es el orden en el que serán mostrados. Después de esto especificamos la tabla o tablas donde estamos buscando.

Definiendo el where podemos especificar múltiples condiciones separadas por espacios. Si estamos utilizando más de una tabla en la búsqueda, necesitamos hacer una comparación de igual entre los campos que ligan ambas tablas. Los últimos ejemplos son sobre hacer estas peticiones en más de una tabla.

Las condiciones pueden ser dadas con operadores de comparación, como =, <, >, <=, >=, <> or !=. La expresión LIKE nos permite hacer comparaciones simple de patrones, podemos utilizar % como carácter comodín. between ... and ... nos permite especificar un rango de valores. Y podemos especificar más de una condición utilizando los operadores lógicos AND o &&, OR o ||, NOT o !, o XOR.

Podemos ordenar el resultado utilizando order by y especificando el campo que queremos utilizar para ordenar, y si queremos que el orden sea ascendente (asc) o descendente (desc), podemos usar más de un campo para hacer el ordenamiento, en cuyo caso el primer campo será usado primariamente para el orden, y el segundo campo será usado cuando el primer campo tenga más de una instancia.

Y finalmente, el valor limit define desde que registro comenzaremos a mostrar, y cuantos registros serán mostrados.

Espero que en los próximos ejemplos esto se volverá mucho más claro, toma las dos tablas como referencia para ver los resultados de cada comando.

select \* from `estados`;

+-----------+---------------+

| estado\_id | estado\_nombre |

+-----------+---------------+

| 1 | Jalisco |

| 2 | Guanajuato |

| 3 | Hidalgo |

+-----------+---------------+

3 rows in set (5.14 sec)

select \* from `ciudades` where `ciudad\_id` = '3';

+-----------+---------------+------------------+-----------+

| ciudad\_id | ciudad\_nombre | ciudad\_poblacion | estado\_id |

+-----------+---------------+------------------+-----------+

| 3 | Zapopan | 1243538 | 1 |

+-----------+---------------+------------------+-----------+

1 row in set (0.90 sec)

select `ciudad\_nombre`,`ciudad\_poblacion` from `ciudades` order by `ciudad\_poblacion` asc;

+---------------+------------------+

| ciudad\_nombre | ciudad\_poblacion |

+---------------+------------------+

| Tequila | 33155 |

| Tizayuca | 100562 |

| Guanajuato | 171709 |

| Pachuca | 275578 |

| Tonalá | 408729 |

| Celaya | 468064 |

| Tlaquepaque | 563006 |

| Zapopan | 1243538 |

| León | 1436733 |

| Guadalajara | 1494134 |

+---------------+------------------+

10 rows in set (0.04 sec)

select `ciudad\_nombre` from `ciudades` where `estado\_id` = '2' order by `ciudad\_nombre` desc;

+---------------+

| ciudad\_nombre |

+---------------+

| León |

| Guanajuato |

| Celaya |

+---------------+

3 rows in set (0.85 sec)

select \* from `ciudades` limit 2,3;

+-----------+---------------+------------------+-----------+

| ciudad\_id | ciudad\_nombre | ciudad\_poblacion | estado\_id |

+-----------+---------------+------------------+-----------+

| 3 | Zapopan | 1243538 | 1 |

| 4 | Tonalá | 408729 | 1 |

| 5 | Tlaquepaque | 563006 | 1 |

+-----------+---------------+------------------+-----------+

3 rows in set (0.06 sec)

select `ciudad\_nombre` from `ciudades` where `ciudad\_nombre` like 'G%';

+---------------+

| ciudad\_nombre |

+---------------+

| Guadalajara |

| Guanajuato |

+---------------+

2 rows in set (0.04 sec)

select \* from `ciudades` where `ciudad\_poblacion` between '500000' and '1000000';  
  
select \* from `ciudades` where `ciudad\_poblacion`>='500000' and `ciudad\_poblacion`<='1000000';

+-----------+---------------+------------------+-----------+

| ciudad\_id | ciudad\_nombre | ciudad\_poblacion | estado\_id |

+-----------+---------------+------------------+-----------+

| 5 | Tlaquepaque | 563006 | 1 |

+-----------+-------------+--------------------+-----------+

1 row in set (0.04 sec)

Nota: Mientras que ambas peticiones regresarían los mismos registros, utilizar between es más rápido que utilizar dos comparaciones, así que si estás lidiando con un rango de valores, siempre utiliza between.

select c.`ciudad\_nombre`,s.`estado\_nombre` from `estados` s, `ciudades` c where c.`estado\_id`=s.`estado\_id` order by c.`ciudad\_nombre`;

+---------------+---------------+

| ciudad\_nombre | estado\_nombre |

+---------------+---------------+

| Celaya | Guanajuato |

| Guadalajara | Jalisco |

| Guanajuato | Guanajuato |

| León | Guanajuato |

| Pachuca | Hidalgo |

| Tequila | Jalisco |

| Tizayuca | Hidalgo |

| Tlaquepaque | Jalisco |

| Tonalá | Jalisco |

| Zapopan | Jalisco |

+---------------+---------------+

10 rows in set (0.06 sec)

select c.`ciudad\_nombre`,s.`estado\_nombre` from `estados` s, `ciudades` c where c.`estado\_id`=s.`estado\_id` and c.`ciudad\_nombre` like 'G%';

+---------------+---------------+

| ciudad\_nombre | estado\_nombre |

+---------------+---------------+

| Guadalajara | Jalisco |

| Guanajuato | Guanajuato |

+---------------+---------------+

2 rows in set (0.05 sec)

select c.`ciudad\_nombre`,s.`estado\_nombre`,c.`ciudad\_poblacion` from `estados` s,`ciudades` c where s.`estado\_id`=c.`estado\_id`;

+---------------+---------------+------------------+

| ciudad\_nombre | estado\_nombre | ciudad\_poblacion |

+---------------+---------------+------------------+

| Guadalajara | Jalisco | 1494134 |

| Tequila | Jalisco | 33155 |

| Zapopan | Jalisco | 1243538 |

| Tonalá | Jalisco | 408729 |

| Tlaquepaque | Jalisco | 563006 |

| Guanajuato | Guanajuato | 171709 |

| Celaya | Guanajuato | 468064 |

| León | Guanajuato | 1436733 |

| Pachuca | Hidalgo | 275578 |

| Tizayuca | Hidalgo | 100562 |

+---------------+---------------+------------------+

10 rows in set (0.05 sec)

select c.`ciudad\_nombre`,s.`estado\_nombre`,c.`ciudad\_poblacion` from `estados` s, `ciudades` c where s.`estado\_id`=c.`estado\_id` order by s.`estado\_nombre` desc,c.`ciudad\_poblacion` asc;

+---------------+---------------+------------------+

| ciudad\_nombre | estado\_nombre | ciudad\_poblacion |

+---------------+---------------+------------------+

| Tequila | Jalisco | 33155 |

| Tonalá | Jalisco | 408729 |

| Tlaquepaque | Jalisco | 563006 |

| Zapopan | Jalisco | 1243538 |

| Guadalajara | Jalisco | 1494134 |

| Tizayuca | Hidalgo | 100562 |

| Pachuca | Hidalgo | 275578 |

| Guanajuato | Guanajuato | 171709 |

| Celaya | Guanajuato | 468064 |

| León | Guanajuato | 1436733 |

+---------------+---------------+------------------+

10 rows in set (0.15 sec)

**Contar los registros de una tabla**

select count(\*) from `tabla\_nombre` [where condiciones];

Las condiciones son opcionales, y pueden tener el mismo formato de las condiciones que utilizamos en peticiones select, esto simplemente nos devuelve el número de registros, por ejemplo:

select count(\*) from `ciudades`;

+----------+

| count(\*) |

+----------+

| 10 |

+----------+

1 row in set (0.06 sec)

**Sumar los registros de una tabla**

select sum(`ciudad\_poblacion`) from `ciudades`;

+-------------------------+

| sum(`ciudad\_poblacion`) |

+-------------------------+

| 6453216 |

+-------------------------+

1 row in set (0.05 sec)

**Insertar un registro**

insert into `estados` (`estado\_nombre`) values ( "Oaxaca");  
  
insert into `ciudades` (`ciudad\_nombre`,`ciudad\_poblacion`,`estado\_id`) values ('Oaxaca','258008',LAST\_INSERT\_ID());

En este caso agrego un nuevo estado, y una nueva ciudad, nota que en el campo estado\_id estoy utilizando como valor la función LAST\_INSERT\_ID(), que me da el valor de la última ID que fue insertada. Si quisiera insertar más de un registro con esta ID, podemos usar la misma petición de insertar para la inserción de más de un campo. Voy a tomar la petición pasada e insertar en vez de eso tres registros:

insert into `estados` (`estado\_nombre`) values( "Oaxaca");  
  
insert into `ciudades` (`ciudad\_nombre`,`ciudad\_poblacion`,`estado\_id`) values  
('Oaxaca','258008',LAST\_INSERT\_ID()),  
(`Salina Cruz`,`76219`,LAST\_INSERT\_ID()),  
(`Zaragoza`,`85869`,LAST\_INSERT\_ID());

Otra función útil que utilizo para llenar un valor es la función NOW() en campos de tipo datetime, utilizo esto mucho para manejar el tiempo de creación o modificación de registros. Por ejemplo, asumiendo que tuviéramos un campo llamado tiempo\_de\_creacion, podríamos usar:

insert into `estados` (`estado\_nombre`,`tiempo\_de\_creacion`) values ('Sonora', NOW());

**Actualizar un registro**

update `ciudades` set `ciudad\_nombre`='Algún nombre',`ciudad\_poblacion`='1000000' where `ciudad\_id`='5';

Cuando estamos actualizando un registro, necesitamos especificar que registro es el que queremos actualizar, usualmente la llave primaria es utilizada para este propósito debido a que es un valor único. Por supuesto, podemos modificar muchos registros en la misma petición si más de un registro cumple con la condición. Por ejemplo digamos que todos los registros creados el 12 de Febrero de 2010 se volverán "activos" cambiando el valor de un campo llamado activo de '0' a '1', aquí está lo que haríamos:

update `cuentas` set `activo`='1' where `fecha\_de\_creacion` between '2010-02-12 00:00:00' and '2010-02-12 23:59:59';

**Borrar un registro**

delete from `estados` where `estado\_id`='8';

La petición delete es más sencilla que una petición update pero algo similar, todos y cada uno de los registros que cumplan la condición(es) serán borrados. Se muy cuidadoso con esta petición, si estás apuntando a registros específicos, siempre usa la llave primaria para no golpear otro registro por accidente.

### Peticiones MySQL preparadas

La razón por la que aprendí sobre peticiones preparadas fue debido a la seguridad que ofrecen cuando estoy utilizando PHP para realizar las peticiones a la base de datos en un sitio web. Sin embargo, el uso de peticiones preparadas desde la linea de comandos nos ofrece la habilidad de definir una petición una vez y entonces llamarla cuantas veces queramos, cambiando solamente el parámetro que usamos. Por ejemplo, para mostrar un registro de la tabla de ciudades, mostrando el nombre del estado en vez de estado\_id, siempre tendría el mismo formato:

select c.`ciudad\_nombre`,s.`estado\_nombre` from `estados` s, `ciudades` c where c.`estado\_id`=s.`estado\_id` and c.`ciudad\_id` = ?;

En el ejemplo, estamos utilizando el ? como un apartado para la ID de el registro que queremos mostrar de esta manera. Esto es útil también cuando estamos utilizando una petición como esta desde PHP y necesitamos usar un valor que recibimos de alguien más. Antes de la existencia de peticiones preparadas, necesitábamos filtrar muy cuidadosamente la entrada de un usuario para poder prevenir un ataque de inyección de sql que tiene el potencial de borrar nuestra base de datos o darle acceso a una persona no autorizada. Al separar la lógica de MySQL de los datos evitamos este problema, ya que MySQL no interpreta el parámetro, simplemente lo maneja como datos. Otra ventaja es que es más rápido utilizar peticiones preparadas de MySQL.

Hasta donde se, las peticiones preparadas solo funcionan con SELECT, INSERT, UPDATE, REPLACE, DELETE y CREATE TABLE. Veamos un ejemplo utilizando la petición previa. Primero que nada, creamos la petición preparada y la nombramos mostrar\_ciudad:

prepare mostrar\_ciudad from "select c.`ciudad\_nombre`,s.`estado\_nombre` from `estados` s, `ciudades` c where c.`estado\_id`=s.`estado\_id` and c.`ciudad\_id` = ?";

Después preparamos el parámetro, llamado una\_ciudad, en este caso:

set @una\_ciudad = "2";

Y ejecutamos la petición preparada mostrar\_ciudad usando el parámetro una\_ciudad:

execute mostrar\_ciudad using @una\_ciudad;

+---------------+---------------+

| ciudad\_nombre | estado\_nombre |

+---------------+---------------+

| Tequila | Jalisco |

+---------------+---------------+

1 row in set (0.04 sec)

## ¿Cómo crear un nuevo usuario?

Vamos empezando por crear un usuario nuevo desde la consola de MySQL:

CREATE USER 'nombre\_usuario'@'localhost' IDENTIFIED BY 'contrasena';

CREATE USER 'nombre\_usuario'@'%' IDENTIFIED BY 'contrasena';

Lamentablemente, a este punto el nuevo usuario no tiene permisos para hacer algo con las bases de datos. Por consecuencia si el usuario intenta identificarse (con la contraseña establecida), no será capaz de acceder a la consola de MySQL.

Por ello, lo primero que debemos hacer es proporcionarle el acceso requerido al usuario con la información que requiere.

GRANT ALL PRIVILEGES ON \*.\* TO 'nombre\_usuario'@'localhost';

GRANT ALL PRIVILEGES ON DAI.\* TO 'alumno'@'%';

Los asteriscos en este comando hacen referencia a la base de datos y la tabla (respectivamente) a la cual el nuevo usuario tendrá acceso; específicamente este comando permite al usuario leer, editar, ejecutar y realizar todas las tareas en todas las bases de datos y tablas.

Una vez que has finalizado con los permisos que deseas configurar para tus nuevos usuarios, hay que asegurarse siempre de refrescar todos los privilegios.

FLUSH PRIVILEGES;

Tus cambios ahora surtirán efecto.

## ¿Cómo otorgar permisos de usuario diferentes?

Una pequeña lista del resto de los posibles permisos que los usuarios pueden tener:

* ALL PRIVILEGES: como mencionamos previamente esto permite a un usuario de MySQL acceder a todas las bases de datos asignadas en el sistema.
* CREATE: permite crear nuevas tablas o bases de datos.
* DROP: permite eliminar tablas o bases de datos.
* DELETE: permite eliminar registros de tablas.
* INSERT: permite insertar registros en tablas.
* SELECT: permite leer registros en las tablas.
* UPDATE: permite actualizar registros seleccionados en tablas.

GRANT OPTION: permite asignar privilegios de usuarios.

REVOKE OPTION: permite remover privilegios de usuarios.

Para proporcionar un permiso a usuario específico, puedes utilizar ésta estructura:

GRANT [permiso] ON [nombre de bases de datos].[nombre de tabla] TO ‘[nombre de usuario]’@'localhost’;

Si deseas darle acceso a cualquier base de datos o tabla, asegúrate de insertar un asterisco en lugar del nombre de la base de datos o tabla.

Cada vez que tu actualizas o cambias permisos, asegúrate de refrescar los privilegios mediante FLUSH PRIVILEGES;.

Si necesitas remover un permiso, la estructura es casi idéntica a la que los asigna:

REVOKE [permiso] ON [nombre de base de datos].[nombre de tabla] FROM ‘[nombre de usuario]’@‘localhost’;

Así como puedes borrar bases de datos con DROP, también puedes usar el comando DROP para borrar usuarios:

DROP USER ‘usuario\_prueba’@‘localhost’;

Para probar el nuevo usuario, debes cerrar sesión escribiendo quit y volviendo a iniciar sesión con éste comando en la consola:

mysql -u [nombre de usuario]-p