Insertar registro (INSERT INTO)

Fuente: tutorialesprogramacionya.com

Un registro es una fila de la tabla que contiene los datos propiamente dichos. Cada registro tiene un dato por cada columna.

```
CREATE TABLE usuarios (
        nombre varchar(30),
        clave varchar(10)
);
```

Al ingresar los datos de cada registro debe tenerse en cuenta la cantidad y el orden de los campos.

Ahora vamos a agregar un registro a la tabla:

INSERT INTO usuarios (nombre, clave) VALUES ('MarioPerez','Marito');

Usamos "**INSERT INTO**". Especificamos los nombres de los campos entre paréntesis y separados por comas y luego los valores para cada campo, también entre paréntesis y separados por comas.

La tabla usuarios ahora la podemos graficar de la siguiente forma:

nombre	clave
MarioPerez	Marito

Es importante ingresar los valores en el mismo orden en que se nombran los campos, si ingresamos los datos en otro orden, no aparece un mensaje de error y los datos se guardan de modo incorrecto.

Note que los datos ingresados, como corresponden a campos de cadenas de caracteres, se colocan entre comillas. Las comillas son **OBLIGATORIAS**.

Seleccionar registros

Para ver los registros de una tabla usamos "**SELECT**":

SELECT nombre,clave **FROM** usuarios;

El comando "**SELECT**" recupera los registros de una tabla. Luego del comando SELECT indicamos los nombres de los campos a rescatar.

Seleccionar todos los campos

SELECT * **FROM** libros;

El comando "SELECT" recupera los registros de una tabla. Con el asterisco (*****) indicamos que seleccione todos los campos de la tabla que nombramos.
Podemos especificar el nombre de los campos que queremos ver separándolos por comas:

SELECT titulo,autor,editorial **FROM** libros;

En la sentencia anterior la consulta mostrará sólo los campos "titulo", "autor" y "editorial". En la siguiente sentencia, veremos los campos correspondientes al título y precio de todos los libros:

SELECT titulo,precio **FROM** libros;

Ejemplo:

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
  titulo varchar(100),
  autor varchar(30),
  editorial varchar(15),
  precio float,
  cantidad integer
);

INSERT INTO libros (titulo,autor,editorial,precio,cantidad) VALUES
            ('El aleph','Borges','Emece',45.50,100),
            ('Alicia en el pais de las maravillas','Lewis Carroll','Planeta',25,200),
            ('Matematica estas ahi','Paenza','Planeta',15.8,200);

SELECT titulo,precio FROM libros;

SELECT editorial,cantidad FROM libros;

SELECT * FROM libros;
```

Seleccionar con WHERE
Existe una cláusula, "WHERE" que es opcional, con ella podemos especificar condiciones para la consulta "SELECT". Es decir, podemos recuperar algunos registros, sólo los que cumplan con ciertas condiciones indicadas con la cláusula "WHERE". Por ejemplo, queremos ver el usuario cuyo nombre es "MarioPerez", para ello utilizamos "WHERE" y luego de ella, la condición:

SELECT nombre, clave FROM usuarios WHERE nombre='MarioPerez';

Para las condiciones se utilizan operadores relacionales. El signo igual(=) es un operador relacional. Para la siguiente selección de registros especificamos una condición que solicita los usuarios cuya clave es igual a 'bocajuniors':

SELECT nombre, clave FROM usuarios WHERE clave='bocajuniors';

Si ningún registro cumple la condición establecida con el "WHERE", no aparecerá ningún registro.

Operadores relacionales

Los operadores relacionales vinculan un campo con un valor para que MySQL compare cada registro (el campo especificado) con el valor dado.
Los operadores relacionales son los siguientes:

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

Podemos seleccionar los registros cuyo autor sea diferente de 'Borges', para ello usamos la condición:

SELECT titulo,autor,editorial FROM libros WHERE autor<>'Borges';

Podemos comparar valores numéricos. Por ejemplo, queremos mostrar los libros cuyos precios sean mayores a 20 pesos:

SELECT titulo,autor,editorial,precio FROM libros WHERE precio>20;

También, los libros cuyo precio sea menor o igual a 30:

SELECT titulo,autor,editorial,precio FROM libros WHERE precio<=30;

Operadores lógicos

Hasta el momento, hemos aprendido a establecer una condición con "WHERE" utilizando operadores relacionales. Podemos establecer más de una condición con la cláusula "WHERE", para ello aprenderemos los operadores lógicos.
Son los siguientes:

- AND, significa "y",
- OR, significa "o",
- NOT, significa "no", invierte el resultado

Los operadores lógicos se usan para combinar condiciones.
Queremos recuperar todos los registros cuyo autor sea igual a "Borges" y cuyo precio no supere los 20 pesos, para ello necesitamos 2 condiciones:

SELECT * FROM libros WHERE (autor='Borges') AND(precio<=20);

Los registros recuperados en una sentencia que une 2 condiciones con el operador "AND", cumplen con las 2 condiciones.
Queremos ver los libros cuyo autor sea "Borges" y/o cuya editorial sea "Planeta":

SELECT * FROM libros WHERE autor='Borges' OR editorial='Planeta';

El operador "NOT" invierte el resultado de la condición a la cual antecede.
Los registros recuperados en una sentencia en la cual aparece el operador "NOT", no cumplen con la condición a la cual afecta el "NO".
Los paréntesis se usan para encerrar condiciones, para que se evalúen como una sola expresión.

Cuando explicitamos varias condiciones con diferentes operadores lógicos (combinamos "AND", "OR") permite establecer el orden de prioridad de la evaluación; además permite diferenciar las expresiones más claramente.
Por ejemplo, las siguientes expresiones devuelven un resultado diferente:

SELECT * FROM libros WHERE (autor='Borges') OR (editorial='Paidos' AND precio<20);

SELECT * FROM libros WHERE (autor='Borges' OR editorial='Paidos') AND precio<20);

Si bien los paréntesis no son obligatorios en todos los casos, se recomienda utilizarlos para evitar confusiones.

Between / In

Existen otros que simplifican algunas consultas:
Para recuperar de nuestra tabla "libros" los registros que tienen precio mayor o igual a 20 y menor o igual a 40, usamos 2 condiciones unidas por el operador lógico "and":

SELECT * FROM libros WHERE precio>=20 AND precio<=40;

Podemos usar "between":

SELECT * FROM libros WHERE precio BETWEEN 20 AND 40;

"between" significa "entre". Averiguamos si el valor de un campo dado (precio) está entre los valores mínimo y máximo especificados (20 y 40 respectivamente).
Si agregamos el operador "not" antes de "between" el resultado se invierte.
Para recuperar los libros cuyo autor sea 'Paenza' o 'Borges' usamos 2 condiciones:

SELECT * FROM libros WHERE autor='Borges' OR autor='Paenza';

Podemos usar "in":

SELECT * FROM libros WHERE autor IN('Borges','Paenza');

Con "in" averiguamos si el valor de un campo dado (autor) está incluido en la lista de valores especificada (en este caso, 2 cadenas).
Para recuperar los libros cuyo autor no sea 'Paenza' ni 'Borges' usamos:

SELECT * FROM libros WHERE autor<>'Borges' AND autor<>'Paenza';

También podemos usar "in" :

SELECT * FROM libros WHERE autor NOT IN('Borges','Paenza');

Con "IN" averiguamos si el valor del campo está incluido en la lista, con "NOT" antecediendo la condición, invertimos el resultado.

Selecciones ordenadas

Podemos ordenar el resultado de un "SELECT" para que los registros se muestren ordenados por algún campo, para ello usamos la cláusula "**ORDER BY**".
Por ejemplo, recuperamos los registros de la tabla "libros" ordenados por el título:

SELECT codigo,titulo,autor,editorial,precio FROM libros ORDER BY titulo;

Aparecen los registros ordenados alfabéticamente por el campo especificado.

Por defecto, si no aclaramos en la sentencia, los ordena de manera ascendente (de menor a mayor). Podemos ordenarlos de mayor a menor, para ello agregamos la palabra clave "**DESC**":

SELECT codigo,titulo,autor,editorial,precio FROM libros ORDER BY editorial desc;

Borrar registro

Si queremos eliminar uno o varios registros debemos indicar cuál o cuáles, para ello utilizamos el comando "DELETE" junto con la cláusula "WHERE" con la cual establecemos la condición que deben cumplir los registros a borrar. Por ejemplo, queremos eliminar aquel registro cuyo nombre de usuario es 'Leonardo':

DELETE FROM usuarios WHERE nombre='Leonardo';

Si solicitamos el borrado de un registro que no existe, es decir, ningún registro cumple con la condición especificada, no se borrarán registros, pues no encontró registros con ese dato.
Hay que tener mucho cuidado en su uso, una vez eliminado un registro no hay forma de recuperarlo. Si por ejemplo ejecutamos el comando:

DELETE FROM usuarios;

Si la tabla tiene 1000000 de filas, **todas ellas serán eliminadas**.

Actualizar registro

Para modificar uno o varios datos de uno o varios registros utilizamos "UPDATE" (actualizar).
Por ejemplo, en nuestra tabla "usuarios", queremos cambiar los valores de todas las claves, por "RealMadrid":

UPDATE usuarios SET clave='RealMadrid';

Utilizamos "UPDATE" junto al nombre de la tabla y "SET" junto con el campo a modificar y su nuevo valor.
El cambio afectará a todos los registros.
Podemos modificar algunos registros, para ello debemos establecer condiciones de selección con "WHERE".

Por ejemplo, queremos cambiar el valor correspondiente a la clave de nuestro usuario llamado 'MarioPerez', queremos como nueva clave 'Boca', necesitamos una condición "WHERE" que afecte solamente a este registro:

UPDATE usuarios set clave='Boca' WHERE nombre='MarioPerez';

Si no encuentra registros que cumplan con la condición del "WHERE", ningún registro es afectado.
Las condiciones no son obligatorias, pero si omitimos la cláusula "WHERE", la actualización afectará a todos los registros.
También se puede actualizar varios campos en una sola instrucción:

UPDATE usuarios set nombre='MarceloDuarte', clave='Marce' WHERE nombre='Marcelo';

Para ello colocamos "UPDATE", el nombre de la tabla, "SET" junto al nombre del campo y el nuevo valor y separado por coma, el otro nombre del campo con su nuevo valor.

Índices

Para facilitar la obtención de información de una tabla se utilizan índices.

El índice de una tabla desempeña la misma función que el índice de un libro: permite encontrar datos **rápidamente**; en el caso de las tablas, localiza registros.
Una tabla se indexa por un campo (o varios).

El índice es un tipo de archivo con 2 entradas: un dato (un valor de algún campo de la tabla) y un puntero.

Un índice posibilita el acceso directo y rápido haciendo más eficiente las búsquedas. Sin índice, se debe recorrer secuencialmente toda la tabla para encontrar un registro.
El objetivo de un índice es acelerar la recuperación de información.
La desventaja es que consume espacio en el disco.

La indexación es una técnica que optimiza el acceso a los datos, mejora el rendimiento acelerando las consultas y otras operaciones. Es útil cuando la tabla contiene miles de registros.

Los índices se usan para varias operaciones:
- para buscar registros rápidamente.
- para recuperar registros de otras tablas empleando "JOIN".

Es importante identificar el o los campos por los que sería útil crear un índice, aquellos campos por los cuales se realizan operaciones de búsqueda con frecuencia.

Hay distintos tipos de índices, a saber:
1) "primary key": es el que definimos como **clave primaria**. Los valores indexados deben ser **únicos** y además **no pueden ser nulos**. MySQL le da el nombre "PRIMARY". Una tabla solamente puede tener una clave primaria.
2) "index": crea un índice común, los valores no necesariamente son únicos y aceptan valores "null". Podemos darle un nombre, si no se lo damos, se coloca uno por defecto. "key" es sinónimo de "index". Puede haber varios por tabla.
3) "unique": crea un índice para los cuales los valores deben ser **únicos y diferentes**, aparece un mensaje de error si intentamos agregar un registro con un valor ya existente.

Una tabla puede tener hasta 64 índices. Los nombres de índices aceptan todos los caracteres y pueden tener una longitud máxima de 64 caracteres. Pueden comenzar con un dígito, pero no pueden tener sólo dígitos.

Una tabla puede ser indexada por campos de tipo numérico o de tipo caracter. También se puede indexar por un campo que contenga valores NULL, excepto los PRIMARY.

"SHOW index" muestra información sobre los índices de una tabla. Por ejemplo:

SHOW index FROM libros;

El índice llamado **PRIMARY** se crea automáticamente cuando establecemos un campo como **clave primaria**, no podemos crearlo directamente. El campo por el cual se indexa puede ser de tipo numérico o de tipo caracter.

Los valores indexados deben ser **únicos** y además **no pueden ser nulos**. Una tabla solamente puede tener una clave primaria por lo tanto, solamente tiene un índice **PRIMARY**.

Una clave primaria es un campo (o varios) que identifica 1 solo registro (fila) en una tabla.

Para un valor del campo clave existe solamente 1 registro. Los valores no se repiten ni pueden ser nulos.

Veamos un ejemplo, si tenemos una tabla con datos de personas, el número de documento puede establecerse como clave primaria, es un valor que no se repite; puede haber personas con igual apellido y nombre, incluso el mismo domicilio (padre e hijo por ejemplo), pero su documento será siempre distinto.

Si tenemos la tabla "usuarios", el nombre de cada usuario puede establecerse como clave primaria, es un valor que no se repite; puede haber usuarios con igual clave, pero su nombre de usuario será siempre distinto.

Veamos un ejemplo definiendo la tabla "libros" con una clave primaria:

```
CREATE TABLE libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar(15),
  primary key(codigo)
 );
```

Podemos ver la estructura de los índices de una tabla con "SHOW index". Por ejemplo:

SHOW index FROM libros;

Aparece el índice PRIMARY creado automáticamente al definir el campo "codigo" como clave primaria.

Ejemplo:

```
DROP TABLE if exists libros;

CREATE TABLE libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar(15),
  primary key(codigo)
 );

show index FROM libros;
```

Establecemos que un campo sea clave primaria al momento de creación de la tabla:

```
CREATE TABLE usuarios (
  nombre varchar(20),
  clave varchar(10),
```

```
  primary key(nombre)
 );
```
Para definir un campo como clave primaria agregamos "primary key" luego de la definición de todos los campos y entre paréntesis colocamos el nombre del campo que queremos como clave.

Si visualizamos la estructura de la tabla con "DESCRIBE" vemos que el campo "nombre" es clave primaria y no acepta valores nulos(más adelante explicaremos esto detalladamente).

Ingresamos algunos registros:
```
INSERT INTO usuarios (nombre, clave)
  VALUES ('Leonardo','hola');
 INSERT INTO usuarios (nombre, clave)
  VALUES ('MarioPerez','Marito');
 INSERT INTO usuarios (nombre, clave)
  VALUES ('Marcelo','River');
 INSERT INTO usuarios (nombre, clave)
  VALUES ('Gustavo','River');
```

Si intentamos ingresar un valor para el campo clave que ya existe, aparece un mensaje de error indicando que el registro no se cargó pues el dato clave existe. Esto sucede porque los campos definidos como clave primaria no pueden repetirse.
Ingresamos un registro con un nombre de usuario repetido, por ejemplo:

INSERT INTO usuarios (nombre, clave) VALUES ('Gustavo','Boca');

Una tabla sólo puede tener una clave primaria. Cualquier campo (de cualquier tipo) puede ser clave primaria, debe cumplir como requisito, que sus valores no se repitan.
Al establecer una clave primaria estamos indexando la tabla, es decir, creando un índice para dicha tabla; a este tema lo veremos más adelante.

```
DROP TABLE IF EXISTS usuarios;

CREATE TABLE usuarios (
  nombre varchar(20),
  clave varchar(10),
  primary key (nombre)
 );

DESCRIBE usuarios;

INSERT INTO usuarios (nombre, clave) VALUES ('Leonardo', 'hola');
INSERT INTO usuarios (nombre, clave) VALUES ('MarioPerez','Marito');
INSERT INTO usuarios (nombre, clave) VALUES ('Marcelo','River');
INSERT INTO usuarios (nombre, clave) VALUES ('Gustavo','River');

INSERT INTO usuarios (nombre, clave) VALUES ('Gustavo','Boca');
```

Index

Dijimos que hay 3 tipos de índices. Hasta ahora solamente conocemos la clave primaria que definimos al momento de crear una tabla.

Vamos a ver el otro tipo de índice, común. Un índice común se crea con "index", los valores no necesariamente son únicos y aceptan valores "null". Puede haber varios por tabla.
Vamos a trabajar con nuestra tabla "libros".

Un campo por el cual realizamos consultas frecuentemente es "editorial", indexar la tabla por ese campo sería útil.
Creamos un índice al momento de crear la tabla:

```
CREATE TABLE libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar(15),
  primary key(codigo),
  index i_editorial (editorial)
 );
```

Luego de la definición de los campos colocamos "index" seguido del nombre que le damos y entre paréntesis el o los campos por los cuales se indexará dicho índice.
"SHOW index" muestra la estructura de los índices:

```
SHOW index FROM libros;
```

Si no le asignamos un nombre a un índice, por defecto tomará el nombre del primer campo que forma parte del índice, con un sufijo opcional (_2,_3,...) para que sea único.

Unique Index

Veamos el otro tipo de índice, único. Un índice único se crea con "**unique**", los valores deben ser únicos y diferentes, aparece un mensaje de error si intentamos agregar un registro con un valor ya existente. Permite valores nulos y pueden definirse varios por tabla.
Podemos darle un nombre, si no se lo damos, se coloca uno por defecto.
Vamos a trabajar con nuestra tabla "libros".

```
CREATE TABLE libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar(15),
  unique i_codigo(codigo)
```

);

Luego de la definición de los campos colocamos "unique" seguido del nombre que le damos y entre paréntesis el o los campos por los cuales se indexará dicho índice.

Eliminar índice

Para eliminar un índice usamos "DROP index". Ejemplo:

DROP index i_editorial ON libros;

Se elimina el índice con "DROP index" seguido de su nombre y "ON" seguido del nombre de la tabla a la cual pertenece.

Podemos eliminar los índices creados con "index" y con "unique" pero no el que se crea al definir una clave primaria. Un índice PRIMARY se elimina automáticamente al eliminar la clave primaria.

Agregar índice a tabla existente

Podemos agregar un índice a una tabla existente.
Para agregar un índice común a la tabla "libros" tipeamos:

CREATE index i_editorial ON libros (editorial);

Entonces, para agregar un índice común a una tabla existente usamos "CREATE index", indicamos el nombre, sobre qué tabla y el o los campos por los cuales se indexará, entre paréntesis.
Para agregar un índice único a la tabla "libros" tipeamos:

CREATE unique index i_tituloeditorial ON libros (editorial);

Para agregar un índice único a una tabla existente usamos "CREATE unique index", indicamos el nombre, sobre qué tabla y entre paréntesis, el o los campos por los cuales se indexará.
Un índice PRIMARY no puede agregarse, se crea automáticamente al definir una clave primaria.

Campos autoincrementables

Un campo de tipo entero puede tener otro atributo extra 'auto_increment'. Los valores de un campo 'auto_increment', se inician en 1 y se incrementan en 1 automáticamente.

Se utiliza generalmente en campos correspondientes a códigos de identificación para generar valores únicos para cada nuevo registro que se inserta.

Sólo puede haber un campo "auto_increment" y debe ser clave primaria (o estar indexado).
Para establecer que un campo autoincremente sus valores automáticamente, éste debe ser entero (integer) y debe ser clave primaria:

```
CREATE TABLE libros(
  codigo int auto_increment,
  titulo varchar(50),
  autor varchar(50),
  editorial varchar(25),
  primary key (codigo)
 );
```

Para definir un campo autoincrementable colocamos "auto_increment" luego de la definición del campo al crear la tabla.
Hasta ahora, al ingresar registros, colocamos el nombre de todos los campos antes de los valores; es posible ingresar valores para algunos de los campos de la tabla, pero recuerde que al ingresar los valores debemos tener en cuenta los campos que detallamos y el orden en que lo hacemos.
Cuando un campo tiene el atributo "auto_increment" no es necesario ingresar valor para él, porque se inserta automáticamente tomando el último valor como referencia, o 1 si es el primero.

Para ingresar registros omitimos el campo definido como "auto_increment", por ejemplo:

```
INSERT INTO libros (titulo,autor,editorial) VALUES('El aleph','Borges','Planeta');
```

Este primer registro ingresado guardará el valor 1 en el campo correspondiente al código.
Si continuamos ingresando registros, el código (dato que no ingresamos) se cargará automáticamente siguiendo la secuencia de autoincremento.
Un campo "auto_increment" funciona correctamente sólo cuando contiene únicamente valores positivos.

Está permitido ingresar el valor correspondiente al campo "auto_increment", por ejemplo:

```
INSERT INTO libros (codigo,titulo,autor,editorial)
  VALUES(6,'Martin Fierro','Jose Hernandez','Paidós');
```

Pero debemos tener cuidado con la inserción de un dato en campos "auto_increment".
Debemos tener en cuenta que:

- si el valor está repetido aparecerá un mensaje de error y el registro no se ingresará.
- si el valor dado saltea la secuencia, lo toma igualmente y en las siguientes inserciones, continuará la secuencia tomando el valor más alto.
- si el valor ingresado es 0, no lo toma y guarda el registro continuando la secuencia.

Like

Hemos realizado consultas utilizando operadores relacionales para comparar cadenas. Por ejemplo, sabemos recuperar los libros cuyo autor sea igual a la cadena "Borges":

SELECT * FROM libros WHERE autor='Borges';

Los operadores relacionales nos permiten comparar valores numéricos y cadenas de caracteres. Pero al realizar la comparación de cadenas, busca coincidencias de cadenas completas.
Imaginemos que tenemos registrados estos 2 libros:
-El Aleph de Borges;
-Antología poética de J.L. Borges;

Si queremos recuperar todos los libros cuyo autor sea "Borges", y especificamos la siguiente condición:

SELECT * FROM libros WHERE autor='Borges';

sólo aparecerá el primer registro, ya que la cadena "Borges" no es igual a la cadena "J.L. Borges".
Esto sucede porque el operador "=" (igual), también el operador "<>" (distinto) comparan cadenas de caracteres completas. Para comparar porciones de cadenas utilizamos los operadores "LIKE" y "NOT LIKE".

Entonces, podemos comparar trozos de cadenas de caracteres para realizar consultas.
Para recuperar todos los registros cuyo autor contenga la cadena "Borges" debemos tipear:

SELECT * FROM libros WHERE autor LIKE "%Borges%";

El símbolo "%" (porcentaje) reemplaza cualquier cantidad de caracteres (incluyendo ningún caracter). Es un caracter comodín. "LIKE" y "NOT LIKE" son operadores de comparación que señalan igualdad o diferencia.

Para seleccionar todos los libros que comiencen con "A":

SELECT * FROM libros WHERE titulo LIKE 'A%';

Note que el símbolo "%" ya no está al comienzo, con esto indicamos que el título debe tener como primera letra la "A" y luego, cualquier cantidad de caracteres.
Para seleccionar todos los libros que no comiencen con "A":

SELECT * FROM libros WHERE titulo NOT LIKE 'A%';

Así como "%" reemplaza cualquier cantidad de caracteres, el guión bajo "_" reemplaza un caracter, es el otro caracter comodín. Por ejemplo, queremos ver los libros de "Lewis Carroll" pero no recordamos si se escribe "Carroll" o "Carrolt", entonces tipeamos esta condición:

SELECT * FROM libros WHERE autor LIKE "%Carrol_";

Si necesitamos buscar un patrón en el que aparezcan los caracteres comodines, por ejemplo, queremos ver todos los registros que comiencen con un guión bajo, si utilizamos

'_%', mostrará todos los registros porque lo interpreta como "patrón que comienza con un caracter cualquiera y sigue con cualquier cantidad de caracteres". Debemos utilizar "_%", esto se interpreta como 'patrón que comienza con guión bajo y continúa con cualquier cantidad de caracteres".

Es decir, si queremos incluir en una búsqueda de patrones los caracteres comodines, debemos anteponer al caracter comodín, la barra invertida "\", así lo tomará como caracter de búsqueda literal y no como comodín para la búsqueda. Para buscar el caracter literal "%" se debe colocar "\%".

Funciones matemáticas

Existen en MySQL funciones que nos permiten contar registros, calcular sumas, promedios, obtener valores máximos y mínimos. Ya hemos aprendido "count()", veamos otras.
La función "sum()" retorna la suma de los valores que contiene el campo especificado. Por ejemplo, queremos saber la cantidad de libros que tenemos disponibles para la venta:

SELECT sum(cantidad) FROM libros;

También podemos combinarla con "WHERE". Por ejemplo, queremos saber cuántos libros tenemos de la editorial "Planeta":

SELECT sum(cantidad) FROM libros WHERE editorial ='Planeta';

Para averiguar el valor máximo o mínimo de un campo usamos las funciones "max()" y "min()" respectivamente. Ejemplo, queremos saber cuál es el mayor precio de todos los libros:

SELECT max(precio) FROM libros;

Queremos saber cuál es el valor mínimo de los libros de "Rowling":

SELECT min(precio) FROM libros WHERE autor LIKE '%Rowling%';

La función avg() retorna el valor promedio de los valores del campo especificado. Por ejemplo, queremos saber el promedio del precio de los libros referentes a "PHP":

SELECT avg(precio) FROM libros WHERE titulo LIKE '%PHP%';

Estas funciones se denominan "funciones de agrupamiento" porque operan sobre conjuntos de registros, no con datos individuales.
Tenga en cuenta que no debe haber espacio entre el nombre de la función y el paréntesis, porque puede confundirse con una referencia a una tabla o campo. Las siguientes sentencias son distintas:

SELECT count(*) FROM libros;
SELECT count (*) FROM libros;

La primera es correcta, la segunda incorrecta.