Programación Web con PHP y MySQL

Unidad 2: Introducción a MYSQL

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

Indice

Unidad 2: Introducción a MySQL

DML 4

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.

DML (DATA MANIPULATION LANGUAGE)

Su misión es la manipulación de datos. A través de él podemos seleccionar, insertar, eliminar y actualizar datos.

Utilizaremos por lo general los siguientes comandos:

- SELECT
- INSERT
- UPDATE
- DELETE

Inserción de nuevas filas

La forma más directa de insertar una fila nueva en una tabla es mediante una sentencia INSERT. En la forma más simple de esta sentencia debemos indicar la tabla a la que queremos añadir filas, y los valores de cada columna. Las columnas de tipo cadena o fechas deben estar entre comillas sencillas o dobles, para las columnas numéricas esto no es imprescindible, aunque también pueden estar entrecomilladas.

INSERT INTO personas VALUES (Pedro,'1977-04-19');

INSERT INTO personas VALUES (Juan,'1978-01-15');

O podríamos unificarlo en una sola sentencia:

INSERT INTO personas VALUES (Mariano,'1972-09-07'),

(Verónica,'1976-06-03');

Si no necesitamos asignar un valor concreto para alguna columna, podemos asignarle el valor por defecto indicado para esa columna cuando se creó la tabla, usando la palabra DEFAULT:

INSERT INTO provincia VALUES (Buenos Aires, DEFAULT);

En este caso, como habíamos definido un valor por defecto para población de 5000, se asignará ese valor para la fila correspondiente a Buenos Aires.

Otra opción consiste en indicar una lista de columnas para las que se van a suministrar valores. A las columnas que no se nombren en esa lista se les asigna el valor por defecto. Este sistema, además, permite usar cualquier orden en las columnas, con la ventaja, con respecto a la anterior forma, de que no necesitamos conocer el orden de las columnas en la tabla para poder insertar datos:

INSERT INTO provincia (poblacion, nombre) VALUES (7000000, 'Santa Fe'), (9000000, 'Córdoba'), (3500000, *'Corrientes');*

Cuando creamos la tabla "provincia" definimos tres columnas: 'clave', 'nombre' y 'poblacion' (por ese orden). Ahora hemos insertado tres filas, en las que hemos omitido la clave, y hemos alterado el orden de 'nombre' y 'poblacion'. El valor de la 'clave' se calcula automáticamente, ya que lo hemos definido como auto-incrementado.

Existe otra sintaxis alternativa, que consiste en indicar el valor para cada columna:

INSERT INTO provincia SET nombre='Mendoza', poblacion=8000000;

Una vez más, a las columnas para las que no indiquemos valores se les asignarán sus valores por defecto. También podemos hacer esto usando el valor DEFAULT.

Para que una fila se considere duplicada debe tener el mismo valor que una fila existente para una clave principal o para una clave única. En tablas en las que no exista clave primaria ni índices de clave única no tiene sentido hablar de filas duplicadas. Es más, en esas tablas es perfectamente posible que existan filas con los mismos valores para todas las columnas.

Por ejemplo, en mitabla5 tenemos una clave única sobre la columna 'nombre':

INSERT INTO mitabla5 (id, nombre) VALUES (1, 'Carlos'),

(2, 'Felipe'), (3, 'Antonio'), (4, 'Carlos'), (5,'Juan');

ERROR 1062 (23000): Duplicate entry 'Carlos' for key 1

Si intentamos insertar dos filas con el mismo valor de la clave única se produce un error y la sentencia no se ejecuta.

Reemplazar filas

Existe una sentencia REPLACE, que es una alternativa para INSERT, que sólo se diferencia en que si existe algún registro anterior con el mismo valor para una clave primaria o única, se elimina el viejo y se inserta el nuevo en su lugar.

REPLACE [LOW_PRIORITY | DELAYED]

[INTO] tbl_name [(col_name,...)]

VALUES ({expr | DEFAULT},...),(...),...

REPLACE [LOW_PRIORITY | DELAYED]

[INTO] tbl_name

SET col_name={expr | DEFAULT}, ...

Ejemplo:

REPLACE INTO provincia (nombre, poblacion) VALUES ('Mendoza', 7200000),

('Buenos Aires', 9200000), ('San Luis', 6000000);

En este ejemplo se sustituyen las filas correspondientes a 'Mendoza' y 'Buenos Aires', que ya existían en la tabla y se inserta la de 'San Luis' que no estaba previamente.

Las mismas sintaxis que existen para INSERT, están disponibles para REPLACE:

REPLACE INTO provincia VALUES ('Buenos Aires', 9500000);

REPLACE INTO provincia SET nombre='Santa Fe', poblacion=10000000;

Actualizar filas

Podemos modificar valores de las filas de una tabla usando la sentencia UPDATE. En su forma más simple, los cambios se aplican a todas las filas, y a las columnas que especifiquemos.

UPDATE [LOW_PRIORITY] [IGNORE] tbl_name

SET col_name1=expr1 [, col_name2=expr2 ...]

[WHERE where_definition]

[ORDER BY ...]

[LIMIT row_count]

Por ejemplo, podemos aumentar en un 10% la población de todas las ciudades de la tabla provincia usando esta sentencia:

*UPDATE provincia SET poblacion=poblacion*1.10;*

Podemos, del mismo modo, actualizar el valor de más de una columna, separándolas en la sección SET mediante comas:

*UPDATE provincia SET clave=clave+10, población = población *0.97;*

En este ejemplo hemos incrementado el valor de la columna 'clave' en 10 y disminuido el de la columna 'poblacion' en un 3%, para todas las filas.

Pero no tenemos por qué actualizar todas las filas de la tabla. Podemos limitar el número de filas afectadas de varias formas.

La primera es mediante la cláusula WHERE. Usando esta cláusula podemos establecer una condición.

Sólo las filas que cumplan esa condición serán actualizadas:

UPDATE ciudad5 SET poblacion=poblacion*1.03 WHERE nombre='Mendoza';

En este caso sólo hemos aumentado la población de las ciudades cuyo nombre sea 'Mendoza'. Las condiciones pueden ser más complejas. Existen muchas funciones y operadores que se pueden aplicar sobre cualquier tipo de columna, y también podemos usar operadores booleanos como AND u OR, desarrollaremos estos operadores más adelante.

Otra forma de limitar el número de filas afectadas es usar la cláusula LIMIT. Esta cláusula permite especificar el número de filas a modificar:

UPDATE provincia SET clave=clave-10 LIMIT 2;

En este ejemplo hemos decrementado en 10 unidades la columna clave de las dos primeras filas. Esta cláusula se puede combinar con WHERE, de modo que sólo las 'n' primeras filas que cumplan una determinada condición se modifiquen. Sin embargo esto no es lo habitual, ya que, si no existen claves primarias o únicas, el orden de las filas es arbitrario, no tiene sentido seleccionarlas usando sólo la cláusula LIMIT.

La cláusula LIMIT se suele asociar a la cláusula ORDER BY. Por ejemplo, si queremos modificar la fila con la fecha más antigua de la tabla 'gente', usaremos esta sentencia:

UPDATE gente SET fecha="1985-04-12" ORDER BY fecha LIMIT 1;

Si queremos modificar la fila con la fecha más reciente, usaremos el orden inverso, es decir, el descendente:

UPDATE gente SET fecha="2001-12-02" ORDER BY fecha DESC LIMIT 1;

Cuando exista una clave primaria o única, se usará ese orden por defecto, si no se especifica una cláusula ORDER BY.

Eliminar filas

Para eliminar filas se usa la sentencia DELETE. La sintaxis es muy parecida a la de UPDATE:

DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM table_name

[WHERE where_definition]

[ORDER BY ...]

[LIMIT row_count]

La forma más simple es no usar ninguna de las cláusulas opcionales:

DELETE FROM provincia;

De este modo se eliminan todas las filas de la tabla. Pero es más frecuente que solo queramos eliminar ciertas filas que cumplan determinadas condiciones. La forma más normal de hacer esto es usar la cláusula WHERE:

DELETE FROM provincia WHERE clave=2;

También podemos usar las cláusulas LIMIT y ORDER BY del mismo modo que en la sentencia UPDATE, por ejemplo, para eliminar las dos ciudades con más población:

DELETE FROM provincia ORDER BY poblacion DESC LIMIT 2;

Cuando queremos eliminar todas la filas de una tabla, vimos en el punto anterior que podíamos usar una sentencia DELETE sin condiciones. Sin embargo, existe una sentencia alternativa, TRUNCATE, que realiza la misma tarea de una forma mucho más rápida.

La diferencia es que DELETE hace un borrado secuencial de la tabla, fila a fila. Pero TRUNCATE borra la tabla y la vuelve a crear vacía, lo que es mucho más eficiente.

TRUNCATE provincia;

Selección de datos

Ya disponemos de bases de datos, y sabemos cómo añadir y modificar datos.

Ahora aprenderemos a extraer datos de una base de datos. Para ello volveremos a usar la sentencia SELECT. La sintaxis de SELECT es compleja, pero en este capítulo no explicaremos todas sus opciones. Una forma más general consiste en la siguiente sintaxis:

SELECT [ALL | DISTINCT | DISTINCTROW]

expresion_select,...

FROM referencias_de_tablas

WHERE condiciones

[GROUP BY {nombre_col | expresion | posicion}

[ASC | DESC], ... [WITH ROLLUP]]

[HAVING condiciones]

[ORDER BY {nombre_col | expresion | posicion}

[ASC | DESC] ,...]

[LIMIT {[desplazamiento,] contador | contador OFFSET desplazamiento}]

Forma incondicional

La forma más sencilla es la que hemos usado hasta ahora, consiste en pedir todas las columnas y no especificar condiciones.

SELECT * FROM persona;

Limitar las columnas: proyección

Recordemos que una de las operaciones del álgebra relacional era la proyección, que consistía en seleccionar determinados atributos de una relación. Mediante la sentencia SELECT es posible hacer una proyección de una tabla, seleccionando las columnas de las que queremos obtener datos. En la sintaxis que hemos mostrado, la selección de columnas corresponde con la parte "expresion_select". En el ejemplo anterior hemos usado '*', que quiere decir que se muestran todas las columnas.

Pero podemos usar una lista de columnas, y de ese modo sólo se mostrarán esas columnas:

SELECT nombre FROM persona;

Mostrar filas repetidas

Ya que podemos elegir sólo algunas de las columnas de una tabla, es posible que se produzcan filas repetidas, debido a que hayamos excluido las columnas únicas.

Por ejemplo, añadamos las siguientes filas a nuestra tabla:

INSERT INTO persona VALUES ('Alicia', '1972-09-07'), ('Antonio', '1976-06-03');

Vemos que existen dos valores de filas repetidos, para la fecha '1972-09-07' y para '1976-06-03'. La sentencia que hemos usado asume el valor por defecto (ALL) para el grupo de opciones ALL, DISTINCT y DISTINCTROW. En realidad sólo existen dos opciones, ya que las dos últimas: DISTINCT y DISTINCTROW son sinónimos. Si usamos DISTINCT, hará que sólo se muestren las filas diferentes.

Limitar las filas: selección

Otra de las operaciones del álgebra relacional era la selección, que consistía en seleccionar filas de una relación que cumplieran determinadas condiciones.

Lo que es más útil de una base de datos es la posibilidad de hacer consultas en function de ciertas condiciones. Generalmente nos interesará saber qué filas se ajustan a determinados parámetros. Por supuesto, SELECT

permite usar condiciones como parte de su sintaxis, es decir, para hacer selecciones. Concretamente mediante la cláusula WHERE, veamos algunos ejemplos:

*SELECT * FROM persona WHERE nombre="Mariano";*

*SELECT * FROM gente WHERE fecha>="1976-06-03";*

*SELECT * FROM gente WHERE fecha>="1972-09-07" AND fecha "2011-01-01";*

En una cláusula WHERE se puede usar cualquier función disponible en MySQL, excluyendo sólo las de resumen o reunión, que veremos en el siguiente punto. Esas funciones están diseñadas específicamente para usarse en cláusulas GROUP BY.

También se puede aplicar lógica booleana para crear expresiones complejas.

Disponemos de los operadores **AND, OR, XOR Y NOT**.

Agrupar filas

Es posible agrupar filas en la salida de una sentencia SELECT según los distintos valores de una columna, usando la cláusula GROUP BY. Esto, en principio, puede parecer redundante, ya que podíamos hacer lo mismo usando la opción DISTINCT. Sin embargo, la cláusula GROUP BY es más potente:

SELECT fecha FROM persona GROUP BY fecha;

La primera diferencia que observamos es que si se usa GROUP BY la salida se ordena según los valores de la columna indicada. En este caso, las columnas aparecen ordenadas por fechas. Otra diferencia es que se eliminan los valores duplicados aún si la proyección no contiene filas duplicadas, por ejemplo:

SELECT nombre, fecha FROM persona GROUP BY fecha;

Pero la diferencia principal es que el uso de la cláusula GROUP BY permite usar funciones de resumen o reunión. Por ejemplo, la función COUNT(), que sirve para contar las filas de cada grupo:

SELECT fecha, COUNT(*) AS cuenta FROM persona GROUP BY fecha;

Esta sentencia muestra todas las fechas diferentes y el número de filas para cada fecha.

Existen otras funciones de resumen o reunión, como MAX(), MIN(), SUM(), AVG(), STD(), VARIANCE()...

Estas funciones también se pueden usar sin la cláusula GROUP BY siempre que no se proyecten otras columnas:

SELECT MAX(nombre) FROM persona;

Esta sentencia muestra el valor más grande de 'nombre' de la tabla 'gente', es decir, el último por orden alfabético.

Cláusula HAVING

La cláusula HAVING permite hacer selecciones en situaciones en las que no es possible usar WHERE. Veamos un ejemplo completo:

CREATE TABLE muestras (ciudad VARCHAR(40), fecha DATE, temperatura TINYINT);

INSERT INTO muestras (ciudad,fecha,temperatura) VALUES ('Madrid', '2005-03-17', 23), ('París', '2005-03-17', 16), ('Berlín', '2005-03-17', 15), ('Madrid', '2005-03-18', 25), ('Madrid', '2005-03-19', 24), ('Berlín', '2005-03-19', 18);

SELECT ciudad, MAX(temperatura) FROM muestras GROUP BY ciudad HAVING MAX(temperatura)>16;

Nos daría como resultado:

| Berlín | 18 |

| Madrid | 25 |

La cláusula WHERE no se puede aplicar a columnas calculadas mediante funciones de reunión, como en este ejemplo.

Ordenar resultados

Además, podemos añadir una cláusula de orden ORDER BY para obtener resultados ordenados por la columna que queramos:

*SELECT * FROM persona ORDER BY fecha;*

Existe una opción para esta cláusula para elegir el orden, ascendente o descendente. Se puede añadir a continuación ASC o DESC, respectivamente. Por defecto se usa el orden ascendente, de modo que el modificador ASC es opcional.

*SELECT * FROM persona ORDER BY fecha DESC;*

Limitar el número de filas de salida

Por último, la cláusula LIMIT permite limitar el número de filas devueltas:

*SELECT * FROM persona LIMIT 3;*

Esta cláusula se suele usar para obtener filas por grupos, y no sobrecargar demasiado al servidor, o a la aplicación que recibe los resultados. Para poder hacer esto la cláusula LIMIT admite dos parámetros. Cuando se usan los dos, el primero indica el número de la primera fila a recuperar, y el segundo el número de filas a recuperar. Podemos, por ejemplo, recuperar las filas de dos en dos:

*SELECT * FROM persona LIMIT 0,2;*

*SELECT * FROM persona LIMIT 2,2;*

*SELECT * FROM persona LIMIT 4,2;*

*SELECT * FROM persona LIMIT 6,2;*

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.