UTN.BA FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

EXPERTO UNIVERISTARIO EN MySQL Y PHP NIVEL INICIAL

www.sceu.frba.utn.edu.ar/e-learning

Módulo 1

INTRODUCIÉNDONOS EN MySQL Y PHP

Consultas típicas SQL.

Uso de PHPmyAdmin para gestionar y administrar bases de datos.

Presentación de la Unidad:

Llevaremos a la práctica los conceptos aprendidos en la unidad anterior.

Aprenderán a crear una base de datos con sus tablas, a definir los distintos campos que componen la tabla asignándole a cada uno el tipo de dato correspondiente según sus necesidades, a definir la clave primaria de la tabla.

Y mediante las instrucciones básicas que nos proporciona SQL podran dar de alta, de baja, modificar o consultar los datos de la tabla.

Objetivos:

- Aprender a crear tablas, definir tipos de dato para cada una de las columnas.
- Aprender a definir una llave o clave primaria.
- Aprender las distintas sentencias SQL que existen para acceder y administrar los datos de las tablas.

Temario:

CONSIGNAS PARA EL APRENDIZAJE COLABORATIVO

En esta Unidad los participantes se encontrarán con diferentes tipos de consignas que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

1. Los foros asociados a cada una de las unidades.
2. La Web 2.0.
3. Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen las actividades sugeridas y compartan en los foros los resultados obtenidos.

MySQL

Tipos de campos en MySQL.

Retomemos y profundicemos un tema muy importante al momento de crear nuestras bases de datos, ni más ni menos que la selección de los tipos de campos que albergaran los datos en nuestras tablas, MySQL dijimos, soporta un número de tipos de campos (columnas) divididos en varias categorías. Una vez que hemos decidido qué información debemos almacenar, el siguiente paso consiste en elegir el tipo adecuado para cada atributo.

Podemos agrupar los tipos de datos en las siguientes categorías: de caracteres, enteros, de coma flotante, tiempos, bloques, enumerados y conjuntos.

Parámetros

Se define entonces brevemente a una base de datos como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular.

M. Este parámetro se utiliza para indicar el número máximo de caracteres que pueden tener los valores que incluyamos en esta columna. M puede ser cualquier número entero entre 1 y 255. El almacenamiento de un número de caracteres mayor, indefectiblemente, nos acabará causando problemas, sobre todo cuando las relaciones entre las distintas bases de datos sean más complejas.

D. Este parámetro nos permite especificar cuántos números decimales pueden ser almacenados en valores de punto flotante. El máximo valor de este es 30, siempre y cuando el parámetro M nos lo permita.

Atributos

ZEROFILL. Esta opción consigue que la columna siempre ocupe su máxima longitud, dado que en el caso de que no le asignemos ningún valor, el sistema automáticamente lo completará con ceros. De esta manera nos aseguramos de que en una búsqueda siempre encontramos un valor, aunque sea cero. Al activar esta opción, automáticamente se activa la opción UNSIGNED.

UNSIGNED. Esta opción consigue que la columna sólo acepte valores positivos, o cero. Hayque tener precaución y, antes de activarla, tener completamente seguro que no queremos aceptar valores negativos.

BINARY. Por defecto, los caracteres de comparación en MySQL no distinguen entre

mayúsculas y minúsculas. Sin embargo, los caracteres de comparación en columnas BINARY sí admiten mayúsculas y minúsculas.

Los corchetes indican información adicional. En terminología ODBC, a M se le denomina precisión y a
D escala.

Tipos de datos de cadenas de caracteres

CHAR

CHAR

Es un sinónimo de CHAR(1), y puede contener un único carácter.

CHAR()

CHAR(M) [BINARY | ASCII | UNICODE]

Contiene una cadena de longitud constante. Para mantener la longitud de la cadena, se rellena a la derecha con espacios. Estos espacios se eliminan al recuperar el valor. Los valores válidos para M son de 0 a 255, y de 1 a 255 para versiones de MySQL previas a 3.23. Si no se especifica la palabra clave BINARY estos valores se ordenan y comparan sin distinguir mayúsculas y minúsculas. CHAR es un alias para CHARACTER.

VARCHAR()

VARCHAR(M) [BINARY]

Contiene una cadena de longitud variable. Los valores válidos para M son de 0 a 255, y de 1 a 255 en versiones de MySQL anteriores a 4.0.2. Los espacios al final se eliminan.
Si no se especifica la palabra clave BINARY estos valores se ordenan y comparan sin distinguir mayúsculas y minúsculas. VARCHAR es un alias para CHARACTER

VARYING.

Tipos de datos enteros

TINYINT

TINYINT[(M)] [UNSIGNED] [ZEROFILL]

Contiene un valor entero muy pequeño. El rango con signo es entre -128 y 127. El rango sin singo, de
0 a 255.

BIT, BOOL y BOOLEAN, todos son sinónimos de TINYINT(1).

SMALLINT

SMALLINT[(M)] [UNSIGNED] [ZEROFILL]

Contiene un entero corto. El rango con signo es de -32768 a 32767. El rango sin singo, de 0 a 65535.

MEDIUMINT

MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]

Contiene un entero de tamaño medio, el rango con signo está entre -8388608 y

8388607. El rango sin signo, entre 0 y 16777215.

INT

INT[(M)] [UNSIGNED] [ZEROFILL]

Contiene un entero de tamaño normal. El rango con signo está entre -2147483648 y 2147483647. El rango sin singo, entre 0 y 4294967295.

INTEGER

INTEGER[(M)] [UNSIGNED] [ZEROFILL]

Es sinónimo de INT.

BIGINT

BIGINT[(M)] [UNSIGNED] [ZEROFILL]

Contiene un entero grande. El rango con signo es de -9223372036854775808 a 9223372036854775807. El rango sin signo, de 0 a 18446744073709551615.

Tipos de datos en coma flotante

FLOAT

FLOAT(precision) [UNSIGNED] [ZEROFILL]

Contiene un número en coma flotante, precisión puede ser menor o igual que 24 para números de precisión sencilla y entre 25 y 53 para números en coma flotante de doble precisión. Estos tipos son idénticos que los tipos FLOAT y DOUBLE descritos a continuación. FLOAT(X) tiene el mismo rango que los tipos FLOAT y DOUBLE correspondientes, pero el tamaño mostrado y el número de decimales quedan indefinidos.

FLOAT()

FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]

Contiene un número en coma flotante pequeño (de precisión sencilla). Los valores permitidos son entre -3.402823466E+38 y -1.175494351E-38, 0, y entre 1.175494351E- 38 y 3.402823466E+38. Si se especifica el modificador UNSIGNED, los valores negativos no se permiten. El valor M es la anchura a mostrar y D es el número de decimales. Si se usa sin argumentos o si se usa FLOAT(X), donde X sea menor o igual que 24, se sigue definiendo un valor en coma flotante de precisión sencilla.

DOUBLE

DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]

Contiene un número en coma flotante de tamaño normal (precisión doble). Los valores permitidos están entre -1.7976931348623157E+308 y -2.2250738585072014E-308, 0, y entre 2.2250738585072014E-308 y 1.7976931348623157E+308. Si se especifica el modificador UNSIGNED, no se permiten los valores negativos. El valor M es la anchura a mostrar y D es el número de decimales. Si se usa sin argumentos o si se usa FLOAT(X), donde X esté entre 25 y 53, se sigue definiendo un valor en coma flotante de doble precisión.

DOUBLE
PRECISIO

N REAL
DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL] REAL[(M,D)] [UNSIGNED] [ZEROFILL]
Ambos son sinónimos de DOUBLE.

DECIMAL
DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]

Contiene un número en coma flotante sin empaquetar. Se comporta igual que una columna CHAR: "sin empaquetar" significa que se almacena como una cadena, usando un carácter para cada dígito del valor. El punto decimal y el signo '-' para valores negativos, no se cuentan en M (pero el espacio para estos se reserva). Si D es 0, los valores no tendrán punto decimal ni decimales. El rango de los valores DECIMAL es el mismo que para DOUBLE, pero el rango actual para una columna DECIMAL dada está restringido por la elección de los valores M y D. Si se especifica el modificador UNSIGNED, los valores negativos no están permitidos. Si se omite D, el valor por defecto es 0. Si se omite M, el valor por defecto es 10.

DEC
NUMERIC
FIXED
DEC[(M[,D])] [UNSIGNED] [ZEROFILL]
NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL] FIXED[(M[,D])] [UNSIGNED] [ZEROFILL] Todos ellos son sinónimos de DECIMAL.

Tipos de datos para tiempos

DATE
DATE
Contiene una fecha. El rango soportado está entre '1000-01-01' y '9999-12-31'. MySQL muestra los valores DATE con el formato 'AAAA-MM-DD', pero es posible asignar valores a columnas de este tipo usando tanto números como cadenas.

DATETIME
DATETIME
Contiene una combinación de fecha y hora. El rango soportado está entre '1000-01-01 00:00:00' y '9999-12-31 23:59:59'. MySQL muestra los valores DATETIME con el formato 'AAAA-MM-DD HH:MM:SS', pero es posible asignar valores a columnas de este tipo usando tanto cadenas como números.

TIMESTAMP
TIMESTAMP[(M)]
Contiene un valor del tipo timestamp. El rango está entre '1970-01-01 00:00:00' y algún momento del año 2037. Hasta MySQL 4.0 los valores TIMESTAMP se mostraban como AAAAMMDDHHMMSS, AAMMDDHHMMSS, AAAAMMDD o AAMMDD, dependiendo del si el valor de M es 14 (o se omite),

12, 8 o 6, pero está permitido asignar valores a columnas TIMESTAMP usando tanto cadenas como
números.

Desde MySQL 4.1, TIMESTAMP se devuelve como una cadena con el formato 'AAAA-MM-DD HH:MM:SS'. Para convertir este valor a un número, bastará con sumar el valor 0. Ya no se soportan distintas longitudes para estas columnas.

Se puede asignar fácilmente la fecha y hora actual a uno de estas columnas asignando el valor NULL. El argumento M afecta sólo al modo en que se visualiza la columna TIMESTAMP. Los valores siempre se almacenan usando cuatro bytes. Además, los valores de columnas TIMESTAMP(M), cuando M es 8

ó 14 se devuelven como números, mientras que para el resto de valores se
devuelven como cadenas.

TIME
TIME
Una hora. El rango está entre '-838:59:59' y '838:59:59'. MySQL muestra los valores TIME en el formato 'HH:MM:SS', pero permite asignar valores a columnas TIME usando tanto cadenas como números.

YEAR
YEAR[(2|4)]
Contiene un año en formato de 2 ó 4 dígitos (por defecto es 4). Los valores válidos son entre 1901 y
2155, y 0000 en el formato de 4 dígitos. Y entre 1970-2069 si se usa el formato de 3 dígitos (70-69). MySQL muestra los valores YEAR usando el formato AAAA, pero permite asignar valores a una columna YEAR usando tanto cadenas como números.

Tipos de datos para datos sin tipo o grandes bloques de datos

TINYBLOB
TINYTEXT
TINYBLOB
TINYTEXT
Contiene una columna BLOB o TEXT con una longitud máxima de 255 caracteres.

BLOB
TEXT
BLOB
TEXT
Contiene una columna BLOB o TEXT con una longitud máxima de 65535 caracteres.

MEDIUMBLOB
MEDIUMTEXT
MEDIUMBLOB
MEDIUMTEXT
Contiene una columna BLOB o TEXT con una longitud máxima de 16777215 caracteres.

LONGBLOB
LONGTEXT
LONGBLOB
LONGTEXT
Contiene una columna BLOB o TEXT con una longitud máxima de 4294967298 caracteres.

Tipos enumerados y conjuntos

ENUM
ENUM('valor1','valor2',...)

Contiene un enumerado. Un objeto de tipo cadena que puede tener un único valor, entre una lista de valores 'valor1', 'valor2', ..., NULL o el valor especial de error "". Un ENUM puede tener un máximo de 65535 valores diferentes.

SET
SET('valor1','valor2',...)
Contiene un conjunto. Un objeto de tipo cadena que puede tener cero o más valores, cada uno de los cuales debe estar entre una lista de valores 'valor1', 'valor2', ... Un conjunto puede tener un máximo de 64 miembros.

Tamaños de almacenamiento

CAMPOS NUMÉRICOS:

Tipo de Campo	Tamaño de Almacenamiento
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 ú 8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D	M+2 bytes sí D > 0, M+1 bytes sí D = 0
NUMERIC(M,D)	M+2 bytes if D > 0, M+1 bytes if D = 0

CAMPOS DE FECHA:

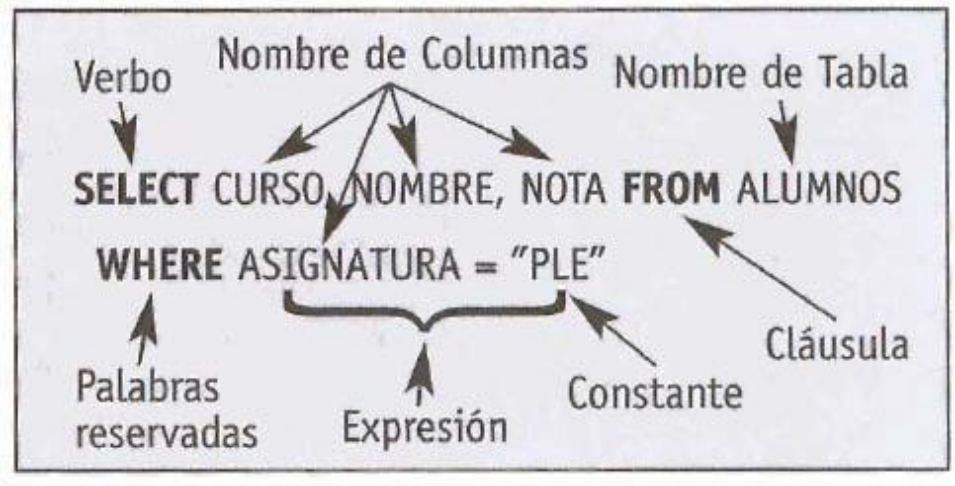
Tipo de Campo	Tamaño de Almacenamiento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

CAMPOS DE TEXTO:

Valor	CHAR(4)	Almacenamiento	VARCHAR(4)	Almacenamiento
''	''	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Componentes sintácticos

La mayoría de sentencias SQL tienen la misma estructura.
Todas comienzan por un verbo (select, insert, update, create), a continuación le sigue una o más clausulas que nos dicen los datos con los que vamos a operar (from, where), algunas de estas son opcionales y otras obligatorias como es el caso del from.



Los mandatos de SQL se dividen en tres grandes grupos diferenciados:

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 clausula:
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.

Las claves foráneas ayudan a mantener la integridad referencial de la base de datos. En general se define un comportamiento para una fila de una tabla hija en relación de una tabla padre. Por ejemplo si borro una fila en la tabla padre borro todas las filas asociadas en la tabla hija. O podría definir que me impida borrar de una fila en la tabla padre si hay filas asociadas en la tabla hija. A esto se lo conoce como restricciones y lo veremos en detalle mas adelante.

Veamos como definir una clave foránea en bases de datos MySQL.

Definimos en una columna de la tabla hija una referencia a una columna de la tabla padre:

CREATE TABLE personas (id INT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(40),
fecha DATE);

CREATE TABLE telefonos (id_tel INT AUTO_INCREMENT PRIMARY KEY,
Id INT,
telefono VARCHAR(40),
INDEX (id_per),
FOREIGN KEY (id_per) REFERENCES personas (id);

Hemos usado una definición de referencia para la columna 'id_per' de la tabla 'telefonos', indicando que es una clave foránea correspondiente a la columna 'id' de la tabla 'personas'.

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;

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_definitio
n] [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.

En la próxima unidad veremos los operadores de los que disponemos.

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;*

DCL (Data Control Laguage)

Un Lenguaje de Control de Datos (DCL por sus siglas en inglés: Data Control Language) es un lenguaje proporcionado por el Sistema de Gestión de Base de Datos que incluye una serie de comandos SQL que permiten al administrador controlar el acceso a los datos contenidos en la Base de Datos.

Las tareas sobre las que se pueden conceder o denegar permisos son las siguientes:

- CONNECT
- SELECT
- INSERT
- UPDATE
- DELETE
- USAGE

Características:

a) Garantiza la seguridad de nuestros datos.
b) Previene el acceso ilegal a nuestros datos.
c) Monitorea el acceso anuestros datos.
d) especifica a cada usuario lo que puede realizar en nuestra base de datos.

Como podemos notar por medio de este lenguaje ofrecemos seguridad a la administración de nuestras bases de datos, este lenguaje en MySQL está conformado por dos comandos cuya sintaxis mostraremos a continuación:

1.- GRANT: Se utiliza para darle permisos a un usuario en nuestra base de datos la sintaxis es la siguiente.

GRANT privilegios ON basededatos.tabla TO usuario IDENTIFIED BY 'contraseña del usuario';

Donde los privilegios pueden ser:

Privilegio Significado

PRIVILEGIO	SIGNIFICADO
ALL [PRIVILEGES]	Da todos los permisos simples excepto GRANT OPTION
ALTER	Permite el uso de ALTER TABLE
ALTER ROUTINE	Modifica o borra rutinas almacenadas
CREATE	Permite el uso de CREATE TABLE
CREATE ROUTINE	Crea rutinas almacenadas
CREATE TEMPORARY TABLES	Permite el uso de CREATE TEMPORARY TABLE
CREATE USER	Permite el uso de CREATE USER, DROP USER, RENAME USER, y REVOKE ALL PRIVILEGES.

CREATE VIEW	Permite el uso de CREATE VIEW
DELETE	Permite el uso de DELETE
DROP	Permite el uso de DROP TABLE
EXECUTE	Permite al usuario ejecutar rutinas
	almacenadas
FILE	Permite el uso de SELECT ... INTO OUTFILE y LOAD DATA INFILE
INDEX	Permite el uso de CREATE INDEX y DROP
INSERT	Permite el uso de INSERT
LOCK TABLES	Permite el uso de LOCK TABLES en tablas para las que tenga el permiso SELECT
PROCESS	Permite el uso de SHOW FULL PROCESSLIST
REFERENCES	No implementado
RELOAD	Permite el uso de FLUSH
REPLICATION CLIENT	Permite al usuario preguntar dónde están los servidores maestro o esclavo.
REPLICATION SLAVE	Necesario para los esclavos de replicación (para leer eventos del log binario desde el maestro)
SELECT	Permite el uso de SELECT
SHOW DATABASES	Muestra todas las bases de datos
SHOW VIEW	Permite el uso de SHOW CREATE VIEW
SHUTDOWN	Permite el uso de mysqladmin shutdown
SUPER	Permite el uso de comandos CHANGE MASTER, KILL, PURGE MASTER
LOGS, and SET GLOBAL	El comando mysqladmin debug le permite conectar (una vez) incluso si se llega a max_connections
UPDATE	Permite el uso de UPDATE
USAGE	Sinónimo de “no privileges”
GRANT OPTION	Permite dar permisos

Además de los privilegios vemos que debemos indicar la base de datos sobre la cual se le otorgan los privilegios al usuario, si se coloca el * el cual es el selector universal se indica que sobre todas, tendremos varios casos:

a) El asterisco punto asterisco solo: Indicaríamos que el usuario tiene el privilegio indicado sobre todas las tablas y todas las base de datos. Ejemplo:

*GRANT ALL PRIVILEGES ON *.* TO usuario IDENTIFIED BY 'contraseña';*

b) Nombre de la base de datos seguido de punto(.) asterisco (*): Indicamos todas las tablas de esa base de datos.

GRANT ALL PRIVILEGES ON base_datos. TO usuario IDENTIFIED BY 'contraseña';*

c) Nombre de la base de datos seguido de punto(.) y nombre de la tabla: Indicamos a cual tabla en especifica se le darán los privilegios indicados.

GRANT ALL PRIVILEGES ON base_datos.tabla TO usuario IDENTIFIED BY 'contraseña';

Como se puede ver con el GRANT otorgamos privilegios a los usuarios desde nuestro usuario administrador ROOT de manera que solo tengan acceso a lo que el administrador del sistema indique.

2.- REVOKE: Es utilizado para quitarle o revocar los privilegios de un usuario en nuestras base de datos, la sintaxis es la siguiente:

REVOKE privilegios ON basedatos FROM usuario;

Donde los privilegios son igual que los establecidos en la tabla anterior, y la base de datos igualmente se le indica de la manera que se indico con el GRANT.

Como podemos observar el lenguaje DCL es importante en la Administración de nuestras base de datos ya que nos ayuda a mantenerlas seguras y tener control total sobre lo que los usuarios pueden o no hacer dentro de ella.