CONCEPTUALIZACIÓN SOBRE MySQL

PROGRAMA TÉCNICO EN PROGRAMACIÓN DE SOFTWARE

SERVICIO NACIONAL DE APRENDIZAJE SENA

CENTRO NACIONAL COLOMBO ALEMÁN

BARRANQUILLA (ATL.)

2018

ACTIVIDAD DE APROPIACIÓN No 8 – GUÍA DE APRENDIZAJE BASE DE DATOS.

Para el desarrollo de la actividad, deberás ir avanzando en la lectura del documento e ir interiorizando los conceptos mientras vas transcribiendo y verificando el comportamiento de las secuencias que se definen en cada punto, utilizando MySQL Command Line Client, e iras tomando las capturas de pantalla para casa secuencia transcrita y su respectivo resultado, esto se ira concentrando en un único archivo PDF, y al final el instructor de formación brindara un espacio para la socialización.

MySQL

1. Introducción:

MySQL es un sistema gestor de bases de datos (SGBD, DBMS por sus siglas en inglés) muy conocido y ampliamente usado por su simplicidad y notable rendimiento, disponible para múltiples plataformas. Aunque carece de algunas características avanzadas disponibles en otros SGBD del mercado, es una opción atractiva tanto para aplicaciones comerciales, como de entretenimiento precisamente por su facilidad de uso y tiempo reducido de puesta en marcha. Esto y su libre distribución en Internet bajo licencia GPL le otorgan como beneficios adicionales (no menos importantes) contar con un alto grado de estabilidad y un rápido desarrollo.

MySQL es un SGBD que ha ganado popularidad por una serie de atractivas características:

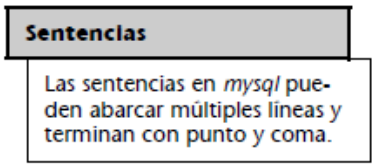
- Está desarrollado en C/C++.
- Se distribuyen ejecutables para cerca de diecinueve plataformas diferentes.
- La API se encuentra disponible en C, C++, Eiffel , Java, Perl, PHP, Python,
- Ruby y TCL.
- Está optimizado para equipos de múltiples procesadores.
- Es muy destacable su velocidad de respuesta.
- Se puede utilizar como cliente-servidor o incrustado en aplicaciones.
- Cuenta con un rico conjunto de tipos de datos.
- Soporta múltiples métodos de almacenamiento de las tablas, con prestaciones
- y rendimiento diferentes para poder optimizar el SGBD a cada caso
- concreto.
- Su administración se basa en usuarios y privilegios.
- Se tiene constancia de casos en los que maneja cincuenta millones de registros,
- sesenta mil tablas y cinco millones de columnas.

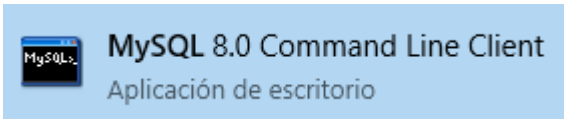
- Sus opciones de conectividad abarcan TCP/IP, *sockets* UNIX y *sockets* NT,
- además de soportar completamente ODBC.
- Los mensajes de error pueden estar en español y hacer ordenaciones correctas
- con palabras acentuadas o con la letra 'ñ'.
- Es altamente confiable en cuanto a estabilidad se refiere.

2. Sentencias:

2.1. Comandos en una sola línea:

El cliente de MySQL en modo interactivo (Command Line Client) nos permite tanto la introducción de sentencias SQL para trabajar con la base de datos (crear tablas, hacer consultas y ver sus resultados, etc.) como la ejecución de comandos propios del SGBD para obtener información sobre las tablas, índices, etc. o ejecutar operaciones de administración, inicialó con el pw.





A continuación presentamos una ejecución de la sentencia *select* con cuatro columnas de datos:

```
mysql> select user(), connection_id(), version(), database();
+----------------+-----------------+-----------+------------+
| user()         | connection_id() | version() | database() |
+----------------+-----------------+-----------+------------+
| root@localhost |              21 | 8.0.12    | NULL       |
+----------------+-----------------+-----------+------------+
1 row in set (0.03 sec)
```

En esta consulta se solicita, a través de funciones incorporadas en el SGBD, el nombre del usuario actual de MySQL, el número de conexión al servidor, la versión del servidor y la base de datos en uso. Las funciones se reconocen por los paréntesis al final. *mysql* entrega sus resultados en tablas, en la que el primer renglón son los encabezados de las columnas. Es importante no dejar espacio entre el nombre de una función y los paréntesis, de otro modo, *mysql* marcará un mensaje de error.

La última línea entregada por *mysql* informa sobre el número de filas encontrado como resultado de la consulta y el tiempo estimado que llevó su realización. Esta medida de tiempo no se debe considerar muy precisa para medir el rendimiento del servidor, se trata simplemente de un valor aproximado que puede verse alterado por múltiples factores.

Observamos que la columna con el nombre de la base de datos actual esta NULL. Esto es natural, ya que no hemos creado aún ninguna base de datos ni le hemos indicado al gestor sobre cuál queremos trabajar.

2.2. Comandos en múltiples líneas:

Los comandos pueden expandirse en varias líneas por comodidad, sobre todo al escribir largas sentencias SQL. El cliente no enviará la sentencia SQL al servidor hasta encontrar el punto y coma, de este modo, el comando anterior puede escribirse así:

```
mysql> select user(),
    -> connection_id(),
    -> version(),
    -> database();
+----------------+-----------------+-----------+------------+
| user()         | connection_id() | version() | database() |
+----------------+-----------------+-----------+------------+
| root@localhost |              21 | 8.0.12    | NULL       |
+----------------+-----------------+-----------+------------+
1 row in set (0.00 sec)
```

También pueden escribirse varios comandos en una sola línea, cada uno debe llevar su respectivo punto y coma:

```
mysql> select now(); select user();
+---------------------+
| now()               |
+---------------------+
| 2018-10-15 19:20:30 |
+---------------------+
1 row in set (0.04 sec)

+----------------+
| user()         |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)
```

Se ejecutarán en el orden que están escritos. Los comandos se pueden cancelar con la combinación \c, con lo que el cliente nos volverá a mostrar el indicador para que escribamos de nuevo la sentencia.

```
mysql> select now(),
    -> uso
    -> ver \c
mysql>
```

2.3. Cadenas de caracteres:

Las cadenas de caracteres pueden delimitarse mediante comillas dobles o simples. Evidentemente, deben cerrarse con el mismo delimitador con el que se han abierto.

```
mysql> select "Hola mundo",'Felicidades';
+------------+-------------+
| Hola mundo | Felicidades |
+------------+-------------+
| Hola mundo | Felicidades |
+------------+-------------+
1 row in set (0.00 sec)
```

Y pueden escribirse en diversas líneas:

```
mysql> select "Este es un texto
    "> en dos renglones";
+----------------------------------+
| Este es un texto
en dos renglones |
+----------------------------------+
| Este es un texto
en dos renglones |
+----------------------------------+
1 row in set (0.00 sec)
```

Al principio, es común olvidar el punto y coma al introducir un comando y, también, olvidar cerrar las comillas. Si éste es el caso, hay que recordar que *mysql* no interpreta lo que está entre comillas, de tal modo que para utilizar el comando de cancelación '\c' es preciso antes cerrar las comillas abiertas:

```
mysql> select "Este es un texto
    "> \c
    "> "\c
mysql>
```

2.4. Utilización de Base de Datos:

2.4.1. Crear una BD: El comando para crear una BD es: ***créate database <nombre_BD>***, en la ejecución que se muestra a continuación se creó una base de datos de nombre ***prueba***.

```
mysql> create database prueba;
Query OK, 1 row affected (0.27 sec)
```

2.4.2. Listar BD en uso: El comando para ver la información de la base de datos que está actualmente en uso es: ***select database***.

```
mysql> select database();
+------------+
| database() |
+------------+
| NULL       |
+------------+
1 row in set (0.00 sec)
```

El campo está vacío porque no estamos haciendo uso de ninguna base de datos.

2.4.3. Listar BD existentes: Para ver las bases de datos existentes en el sistema, se debe efectuar la siguiente consulta: ***show databases***;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| prueba             |
| sakila             |
| sys                |
| world              |
+--------------------+
7 rows in set (0.04 sec)
```

Se puede observar la base de datos de nombre ***prueba*** que creamos anteriormente.

2.4.4. Usar una BD: El comando ***use*** permite abrir una BD para ser utilizada.

```
mysql> use prueba
Database changed
```

Si listamos la base de datos utilizada nos debe mostrar:

```
mysql> select database();
+------------+
| database() |
+------------+
| prueba     |
+------------+
1 row in set (0.00 sec)
```

Es posible realizar consultas en una base de datos sin utilizar el comando ***use***, en ese caso, todos los nombres de las tablas deben llevar el nombre de la base de datos a que pertenecen de la forma: ej: *prueba.productos*, donde ***prueba*** es la BD y ***productos*** una tabla de la misma.

2.4.5. Eliminar una BD: Para eliminar una base de datos, usaremos la sentencia drop database <nombre_BD>:

```
mysql> drop database prueba;
Query OK, 0 rows affected (0.23 sec)
```

Verificamos que ha sido removida:

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| world              |
+--------------------+
6 rows in set (0.00 sec)
```

Luego creamos la base de datos ***prueba*** nuevamente y la usamos, para seguir colocando en práctica los comandos.

MySQL es sensible al uso de mayúsculas y minúsculas, tanto en la definición de bases de datos, como de tablas o columnas.

2.5. Creación y utilización de tablas:

2.5.1. Crear tablas de la BD: para la creación de tablas en una BD utilizamos el comando: **créate table <nombre_tabla>.**

```
mysql> create table personas(
    -> nombre char(30),
    -> direccion char(40),
    -> telefono char(15)
    -> );
Query OK, 0 rows affected (3.41 sec)
```

En este caso, la sentencia create table construye una nueva tabla en la base de datos en uso. La tabla contiene tres columnas, *nombre*, *dirección* y *teléfono*, todas de tipo carácter y de longitudes 30, 40 y 15 respectivamente. Si se intenta guardar en ellas valores que sobrepasen esos límites, serán truncados para poderlos almacenar. Por ese motivo, es importante reservar espacio suficiente para cada columna. Si se prevé que muchos registros ocuparán sólo una fracción del espacio reservado, se puede utilizar el tipo varchar, similar a char, con la diferencia de que el valor ocupará un espacio menor al especificado si la cadena es más corta que el máximo indicado, ahorrando así espacio de almacenamiento.

2.5.2. Mostrar tablas en la BD en uso: La consulta de las tablas que contiene la BD se realiza con la sentencia show de la siguiente manera:

```
mysql> show tables;
+-----------------+
| Tables_in_prueba |
+-----------------+
| personas        |
+-----------------+
1 row in set (0.00 sec)
```

El comando *show* es útil para mostrar información sobre las bases de datos, tablas, variables y otra información sobre el SGBD. Podemos utilizar ***help show*** en el intérprete de comandos para obtener todas las variantes de esta sentencia.

```
mysql> help show
Name: 'SHOW'
Description:
SHOW has many forms that provide information about databases, tables,
columns, or status information about the server. This section describes
those following:

SHOW {BINARY | MASTER} LOGS
SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW CHARACTER SET [like_or_where]
SHOW COLLATION [like_or_where]
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
SHOW CREATE DATABASE db_name
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION func_name
SHOW CREATE PROCEDURE proc_name
SHOW CREATE TABLE tbl_name
SHOW CREATE TRIGGER trigger_name
SHOW CREATE VIEW view_name
SHOW DATABASES [like_or_where]
SHOW ENGINE engine_name {STATUS | MUTEX}
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW EVENTS
SHOW FUNCTION CODE func_name
SHOW FUNCTION STATUS [like_or_where]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW MASTER STATUS
SHOW OPEN TABLES [FROM db_name] [like_or_where]
SHOW PLUGINS
SHOW PROCEDURE CODE proc_name
SHOW PROCEDURE STATUS [like_or_where]
```

2.5.3. Mostrar contenido de la tabla: se utiliza el comando ***describe <nombre_tabla>***.

```
mysql> describe personas;
+-----------+----------+------+-----+---------+-------+
| Field     | Type     | Null | Key | Default | Extra |
+-----------+----------+------+-----+---------+-------+
| nombre    | char(30) | YES  |     | NULL    |       |
| dirección | char(40) | YES  |     | NULL    |       |
| teléfono  | char(15) | YES  |     | NULL    |       |
+-----------+----------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

2.5.4. Eliminar tablas en la BD: las tablas pueden eliminarse con ***drop table <nombre_tabla>*.**

```
mysql> drop table personas;
Query OK, 0 rows affected (1.53 sec)
```

Alternativamente puede utilizarse el comando: ***drop table if exists <nombre_tabla>***.

```
mysql> drop table if exists personas;
Query OK, 0 rows affected, 1 warning (0.09 sec)
```

Este comando también aplica para eliminar las BD: ***drop database if exists <nombre_BD>***.

2.5.5. Atributos de columnas: son los siguientes:

Atributo	Significado
null	Se permiten valores nulos, atributo por omisión si no se especifica lo contrario.
not null	No se permiten valores nulos.
default valor	Valor por omisión que se asigna a la columna.
auto_increment	El valor se asigna automáticamente incrementando en uno el máximo valor registrado hasta ahora. Se aplica sólo a las columnas marcadas como clave primaria.
primary key	Señala al campo como clave primaria, implícitamente también lo declara como not null.

Veámoslo en un ejemplo:

```
mysql> create table personas(
    -> nombre varchar(40) not null,
    -> dirección varchar(50) null,
    -> estado_civil char(13) default 'Soltero',
    -> num_registro int primary key auto_increment
    -> );
Query OK, 0 rows affected (1.02 sec)
```

En este caso la tabla contiene cuatro columnas, de las cuales ***nombre*** y ***estado_civil*** permiten valores nulos, en ***estado_civil*** está implícito al no

declarar lo contrario. La columna ***num_registro*** no acepta valores nulos porque está definida como clave primaria.

Para lo que viene, vamos a crear una tabla de nombre proveedores y vamos a eliminar la tabla personas para luego crearla nuevamente utilizando las restricciones de tablas.

```
mysql> create table proveedores(
    -> nombre_proveedor varchar(40),
    -> id_proveedor int not null,
    ->
    -> primary key (id_proveedor)
    -> );
Query OK, 0 rows affected (0.72 sec)
```

```
mysql> drop table if exists personas;
Query OK, 0 rows affected (0.91 sec)
```

Aunque la creación de una clave primaria puede declararse como atributo de columna, es conveniente definirla como restricción de tabla, como se verá enseguida.

2.5.6. Restricciones de la tabla: Son las siguientes:

Restricción	Significado
primary key	Define la o las columnas que servirán como clave primaria. Las columnas que forman parte de la clave primaria deben de ser **not null**.
unique	Define las columnas en las que no pueden duplicarse valores. Serán las claves candidatas del modelo relacional.
foreign key (*columna*) **references** ***tabla*** (*columna2*)	Define que los valores de *columna* se permitirán sólo si existen en *tabla(columna2)*. Es decir, *columna* hace referencia a los registros de *tabla*, esto asegura que no se realicen referencias a registros que no existen.

También es posible indicar restricciones sobre la tabla y no sobre columnas específicas:

```
mysql> create table personas(
    -> nombre varchar(40) not null,
    -> fecha_nacimiento date not null,
    -> nombre_conyugue varchar(40),
    -> proveedor int not null,
    ->
    -> primary key (nombre, fecha_nacimiento),
    -> unique (nombre_conyugue),
    -> foreign key (proveedor) references proveedores (id_proveedor)
    -> );
Query OK, 0 rows affected (2.14 sec)
```

Se definen tres restricciones sobre la tabla después de la definición de cuatro columnas:

- La primera restricción se refiere a la clave primaria (PK), compuesta por las columnas ***nombre*** **y** ***fecha_nacimiento***: no puede haber dos personas que se llamen igual y que hayan nacido en la misma fecha. La clave primaria permite identificar de manera unívoca cada registro de la tabla. Las fechas se deben ingresar con el formato AAAA-MM-DD.

- La segunda restricción define que la pareja de una persona debe ser única: dos personas no pueden tener la misma pareja. Todo intento de insertar un nuevo registro donde el nombre de la pareja ya exista, será rechazado. Cuando se restringe una columna con ***unique***, los valores null reciben un trato especial, pues se permiten múltiples valores nulos.

- La tercera restricción afecta a la columna ***proveedor***, sólo puede tomar valores que existan en la clave primaria de la tabla ***proveedores***, cabe resaltar que no se puede definir un campo como llave foránea (FK) si no existe la referencia, ósea, no podemos definir ***proveedor*** como FK si la tabla ***proveedores*** y el campo ***id_proveedor*** no existen, por eso primero se creó la tabla ***proveedores***.

Las restricciones de tabla pueden definirse con un identificador útil para hacer referencias posteriores a la restricción:

```
mysql> create table personas(
    -> nombre varchar(40) not null,
    -> fecha_nacimiento date not null,
    -> nombre_conyugue varchar(40),
    -> proveedor int not null,
    ->
    -> constraint clave primary key (nombre, fecha_nacimiento),
    -> constraint monogamo unique (nombre_conyugue),
    -> constraint trabaja_en foreign key (proveedor) references proveedores (id_proveedor)
    -> );
Query OK, 0 rows affected (1.16 sec)
```

2.6. Tipos de datos:

MySQL cuenta con un rico conjunto de tipos de datos para las columnas, que es necesario conocer para elegir mejor cómo definir las tablas. Los tipos de datos se pueden clasificar en tres grupos:

- ✓ Numéricos.
- ✓ Cadenas de caracteres.
- ✓ Fechas y horas.

El valor ***null*** es un caso especial de dato, ya que al significar ausencia de valor se aplica a todos los tipos de columna. Los siguientes símbolos se utilizan en la definición y descripción de los tipos de datos en MySQL:

- ✓ M - El ancho de la columna en número de caracteres.
- ✓ D - Número de decimales que hay que mostrar.
- ✓ L - Longitud o tamaño real de una cadena.
- ✓ [ ] - Lo que se escriba entre ellos es opcional.

2.6.1. Datos numéricos:

Los tipos de datos numéricos comprenden dos categorías, los enteros y los números con punto flotante.

Datos enteros: La principal diferencia entre cada uno de los tipos de enteros es su tamaño, que va desde 1 byte de almacenamiento hasta los 8 bytes. Las columnas de tipo entero pueden recibir dos atributos adicionales, que deben especificarse inmediatamente después del nombre del tipo:

- ✓ **Unsigned:** Indica que el entero no podrá almacenar valores negativos. Es responsabilidad del usuario verificar, en este caso, que los resultados de las restas no sean negativos, porque MySQL los convierte en positivos.

- ✓ **Zerofill**: Indica que la columna, al ser mostrada, rellenará con ceros a la izquierda los espacios vacíos. Esto de acuerdo al valor especificado por M en la declaración del tipo. Una columna con el atributo zerofill es al mismo tiempo **unsigned** aunque no se especifique.

```
create table números (
x int(4) zerofill not null,
y int(5) unsigned
);
```

Datos con punto flotante: MySQL cuenta con los tipos float y double, de 4 y 8 bytes de almacenamiento. Además incluye el tipo decimal, que se almacena como una cadena de caracteres y no en formato binario.

Tipo	Espacio de almacenamiento	Significado
float	4 bytes	Simple precisión
double	8 bytes	Doble precisión
decimal	M + 2 bytes	Cadena de caracteres representando un número flotante

2.6.2. Cadena de caracteres: son:

Tipo	Equivalente	Tamaño máximo	Espacio de almacenamiento
char[(M)]		M bytes	M bytes
varchar[(M)]		M bytes	L+1 bytes
tinytext	tinyblob	2^8–1 bytes	L+1 bytes
text	blob	2^{16}–1 bytes	L+2 bytes
mediumtext	mediumblob	2^{24}–1 bytes	L+3 bytes
longtext	longblob	2^{32}–1 bytes	L+4 bytes
enum('v1','v2',. . .)		65535 valores	1 o 2 bytes
set('v1','v2',. . .)		64 valores	1 a 8 bytes

Si observamos la tabla, vemos que el único tipo de dato que siempre utiliza el tamaño especificado por M es el tipo ***char***. Por este motivo, se ofrece el tipo ***varchar*** que ocupa sólo el espacio requerido por el valor de la columna.

```
create table persona(
comentario char(250),
recado varchar(250)
);
```

2.6.3. Fechas y horas: son:

Tipo	Espacio de almacenamiento	Rango
date	3 bytes	'1000-01-01' al '9999-12-31'
time	3 bytes	'-838:59:59' a '838:59:59'
datetime	8 bytes	'1000-01-01 00:00:00' a '9999-12-31 23:59:59'
timestamp[(M)]	4 bytes	19700101000000 al año 2037
year[(M)]	1 byte	1901 a 2155

2.7. Modificar tablas:

2.7.1. Agregar o eliminar columnas:

Alterar la estructura de una tabla es una tarea más frecuente de lo que uno puede imaginar en un principio. La sentencia ***alter table*** permite una amplia gama de formas de modificar una tabla. La siguiente sentencia nos recuerda un poco a la estructura de la sentencia ***create table***, en donde modificamos la tabla ***personal*** creada en la sección anterior.

```
mysql> alter table personas add(
    -> mascota char(30) default 'perro',
    -> pasatiempo char(20) not null
    -> );
Query OK, 0 rows affected (0.72 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Después de ejecutar la sentencia anterior, aparecen dos nuevas columnas en la tabla.

```
mysql> describe personas;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| nombre           | varchar(40) | NO   | PRI | NULL    |       |
| fecha_nacimiento | date        | NO   | PRI | NULL    |       |
| nombre_conyugue  | varchar(40) | YES  | UNI | NULL    |       |
| proveedor        | int(11)     | NO   | MUL | NULL    |       |
| mascota          | char(30)    | YES  |     | perro   |       |
| pasatiempo       | char(20)    | NO   |     | NULL    |       |
+------------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Si queremos agregar una sola columna, podemos usar la sintaxis siguiente:

```
mysql> alter table personas add capital int not null
    -> after nombre;
Query OK, 0 rows affected (2.32 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> describe personas;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| nombre           | varchar(40) | NO   | PRI | NULL    |       |
| capital          | int(11)     | NO   |     | NULL    |       |
| fecha_nacimiento | date        | NO   | PRI | NULL    |       |
| nombre_conyugue  | varchar(40) | YES  | UNI | NULL    |       |
| proveedor        | int(11)     | NO   | MUL | NULL    |       |
| mascota          | char(30)    | YES  |     | perro   |       |
| pasatiempo       | char(20)    | NO   |     | NULL    |       |
+------------------+-------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Este formato de alter table permite, además, insertar las columnas antes (before) o después (after) de una columna en cuestión.

Las columnas no deseadas pueden eliminarse con la opción drop.

```
mysql> alter table personas drop pasatiempo;
Query OK, 0 rows affected (1.87 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> describe personas;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| nombre           | varchar(40) | NO   | PRI | NULL    |       |
| capital          | int(11)     | NO   |     | NULL    |       |
| fecha_nacimiento | date        | NO   | PRI | NULL    |       |
| nombre_conyugue  | varchar(40) | YES  | UNI | NULL    |       |
| proveedor        | int(11)     | NO   | MUL | NULL    |       |
| mascota          | char(30)    | YES  |     | perro   |       |
+------------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

2.7.2. Modificar columnas: La modificación de una columna con la opción ***modify*** es parecida a volver a definirla.

```
mysql> alter table personas modify
    -> mascota char(14) default 'gato';
Query OK, 0 rows affected (2.42 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> describe personas;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| nombre           | varchar(40) | NO   | PRI | NULL    |       |
| capital          | int(11)     | NO   |     | NULL    |       |
| fecha_nacimiento | date        | NO   | PRI | NULL    |       |
| nombre_conyugue  | varchar(40) | YES  | UNI | NULL    |       |
| proveedor        | int(11)     | NO   | MUL | NULL    |       |
| mascota          | char(14)    | YES  |     | gato    |       |
+------------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Después de la sentencia anterior, los atributos y tipo de la columna han cambiado por los especificados. Lo que no se puede cambiar con esta sintaxis es el nombre de la columna. Para ello, se debe utilizar la opción ***change***:

```
mysql> alter table personas change nombre
    -> nombre_personas char(20);
Query OK, 0 rows affected (2.35 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> describe personas;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| nombre_personas  | char(20)    | NO   | PRI | NULL    |       |
| capital          | int(11)     | NO   |     | NULL    |       |
| fecha_nacimiento | date        | NO   | PRI | NULL    |       |
| nombre_conyugue  | varchar(40) | YES  | UNI | NULL    |       |
| proveedor        | int(11)     | NO   | MUL | NULL    |       |
| mascota          | char(14)    | YES  |     | gato    |       |
+------------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Finalmente, podemos cambiar de nombre la tabla:

```
mysql> alter table personas rename gente;
Query OK, 0 rows affected (0.64 sec)
```

```
mysql> describe personas;
ERROR 1146 (42S02): Table 'prueba.personas' doesn't exist
mysql>
```

```
mysql> describe gente;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| nombre_personas  | char(20)    | NO   | PRI | NULL    |       |
| capital          | int(11)     | NO   |     | NULL    |       |
| fecha_nacimiento | date        | NO   | PRI | NULL    |       |
| nombre_conyugue  | varchar(40) | YES  | UNI | NULL    |       |
| proveedor        | int(11)     | NO   | MUL | NULL    |       |
| mascota          | char(14)    | YES  |     | gato    |       |
+------------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

LENGUAJE DE MANIPULACIÓN DE DATOS DML (DATA MANIPULATION LANGUAGE)

Un lenguaje de manipulación de datos (*Data Manipulation Language*, o *DML* en inglés) es un lenguaje proporcionado por el SGBD que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.

El lenguaje de manipulación de datos más popular hoy día es SQL, usado para recuperar y manipular datos en una base de datos relacional.

1. **INSERT:** Una sentencia *INSERT* de SQL agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

Insertar información a la tabla proveedores:

```
insert into proveedores(nombre_proveedor, id_proveedor)
values ('Redsis','1234');
insert into proveedores(nombre_proveedor, id_proveedor)
```

values ('Macrocomputo','1235');
insert into proveedores(nombre_proveedor, id_proveedor)
values ('Megaprinter','1236');
insert into proveedores(nombre_proveedor, id_proveedor)
values ('Arco Ink','1237');
insert into proveedores(nombre_proveedor, id_proveedor)
values ('Print Toner','1238');
insert into proveedores(nombre_proveedor, id_proveedor)
values ('Exito','1239');

```
mysql> insert into proveedores(nombre_proveedor, id_proveedor)
    -> values ('Macrocomputo','1235');
Query OK, 1 row affected (0.23 sec)

mysql> insert into proveedores(nombre_proveedor, id_proveedor)
    -> values ('Megaprinter','1236');
Query OK, 1 row affected (0.19 sec)

mysql> insert into proveedores(nombre_proveedor, id_proveedor)
    -> values ('Arco Ink','1237');
Query OK, 1 row affected (0.12 sec)

mysql> insert into proveedores(nombre_proveedor, id_proveedor)
    -> values ('Print Toner','1238');
Query OK, 1 row affected (0.10 sec)

mysql> insert into proveedores(nombre_proveedor, id_proveedor)
    -> values ('Exito','1239');
Query OK, 1 row affected (0.14 sec)
```

Insertar registros a la tabla gente:

insert into gente (nombre_personas, capital, fecha_nacimiento, nombre_conyugue, proveedor, mascota) values ('Irwing Fontalvo', '0311', '1987-12-28', 'Olga Garcia', '1234', 'Vaca');

insert into gente (nombre_personas, capital, fecha_nacimiento, nombre_conyugue, proveedor, mascota) values ('Alberto Fabregas', '0312', '1987-12-12', 'Silvia Adarraga', '1236', 'Leon');

insert into gente (nombre_personas, capital, fecha_nacimiento, nombre_conyugue, proveedor, mascota) values ('Paulo Miranda', '0314', '1985-10-17', 'Liliana Cervantes', '1238', 'Mono');

```
mysql> insert into gente (nombre_personas, capital, fecha_nacimiento,
    -> nombre_conyugue, proveedor, mascota)  values ('Irwing Fontalvo',
    -> '0311', '1987-12-28', 'Olga Garcia', '1234', 'Vaca');
Query OK, 1 row affected (0.30 sec)

mysql> insert into gente (nombre_personas, capital, fecha_nacimiento,
    -> nombre_conyugue, proveedor, mascota)  values ('Alberto Fabregas',
    -> '0312', '1987-12-12', 'Silvia Adarraga', '1236', 'Leon');
Query OK, 1 row affected (0.26 sec)

mysql> insert into gente (nombre_personas, capital, fecha_nacimiento,
    -> nombre_conyugue, proveedor, mascota)  values ('Paulo Miranda',
    -> '0314', '1985-10-17', 'Liliana Cervantes', '1238', 'Mono');
Query OK, 1 row affected (0.15 sec)
```

2. **SELECT:** La sentencia SELECT nos permite consultar y listar los datos almacenados en una tabla de la base de datos. Cuando se acompaña la sentencia con * indica que traerá como resultado toda la información contenida en todas las columnas de la tabla. **Select * from <nombre_tabla>**

select * from gente;

```
mysql> select * from gente;
+------------------+---------+------------------+-------------------+-----------+---------+
| nombre_personas  | capital | fecha_nacimiento | nombre_conyugue   | proveedor | mascota |
+------------------+---------+------------------+-------------------+-----------+---------+
| Alberto Fabregas |     312 | 1987-12-12       | Silvia Adarraga   |      1236 | Leon    |
| Irwing Fontalvo  |     311 | 1987-12-28       | Olga Garcia       |      1234 | Vaca    |
| Paulo Miranda    |     314 | 1985-10-17       | Liliana Cervantes |      1238 | Mono    |
+------------------+---------+------------------+-------------------+-----------+---------+
3 rows in set (0.00 sec)
```

También podemos traer información de columnas específicas. **Select <nombre_columna1>, <nombre_columna2>… <nombre_columnaN> from <nombre_tabla>.**

```
mysql> select nombre_personas, fecha_nacimiento
    -> from gente;
+------------------+------------------+
| nombre_personas  | fecha_nacimiento |
+------------------+------------------+
| Irwing Fontalvo  | 1987-12-28       |
| Alberto Fabregas | 1987-12-12       |
| Paulo Miranda    | 1985-10-17       |
+------------------+------------------+
3 rows in set (0.00 sec)
```

Select * from proveedores**;**

```
mysql> select * from proveedores;
+------------------+--------------+
| nombre_proveedor | id_proveedor |
+------------------+--------------+
| Redsis           |         1234 |
| Macrocomputo     |         1235 |
| Megaprinter      |         1236 |
| Arco Ink         |         1237 |
| Print Toner      |         1238 |
| Exito            |         1239 |
+------------------+--------------+
6 rows in set (0.00 sec)
```

select nombre_proveedor
from proveedores
where id_proveedor = 1235;

```
mysql> select nombre_proveedor
    -> from proveedores
    -> where id_proveedor = 1235;
+------------------+
| nombre_proveedor |
+------------------+
| Macrocomputo     |
+------------------+
1 row in set (0.00 sec)
```

Donde en el ***where*** se define una condición.

3. **UPDATE:** Una sentencia *UPDATE* de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

update proveedores **set** nombre_proveedor = 'Exito Murillo' **where** id_proveedor = 1239;

```
mysql> select * from proveedores;
+------------------+--------------+
| nombre_proveedor | id_proveedor |
+------------------+--------------+
| Redsis           |         1234 |
| Macrocomputo     |         1235 |
| Megaprinter      |         1236 |
| Arco Ink         |         1237 |
| Print Toner      |         1238 |
| Exito            |         1239 |
+------------------+--------------+
6 rows in set (0.00 sec)

mysql> update proveedores set nombre_proveedor = 'Exito Murillo' where id_proveedor = 1239;
Query OK, 1 row affected (0.24 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from proveedores;
+------------------+--------------+
| nombre_proveedor | id_proveedor |
+------------------+--------------+
| Redsis           |         1234 |
| Macrocomputo     |         1235 |
| Megaprinter      |         1236 |
| Arco Ink         |         1237 |
| Print Toner      |         1238 |
| Exito Murillo    |         1239 |
+------------------+--------------+
6 rows in set (0.00 sec)
```

4. **DELETE:** Una sentencia *DELETE* de SQL borra uno o más registros existentes en una tabla.

delete from proveedores **where** id_proveedor=1239;

```
mysql> delete from proveedores where id_proveedor=1239;
Query OK, 1 row affected (0.22 sec)

mysql> select * from proveedores
    -> ;
+------------------+--------------+
| nombre_proveedor | id_proveedor |
+------------------+--------------+
| Redsis           |         1234 |
| Macrocomputo     |         1235 |
| Megaprinter      |         1236 |
| Arco Ink         |         1237 |
| Print Toner      |         1238 |
+------------------+--------------+
5 rows in set (0.00 sec)
```

BIBLIOGRAFÍA

SOFTWARE LIBRE – BASE DE DATOS
71Z799014MO
Primera edición: mayo 2005

Av. Tibidabo, 39-43, 08035 Barcelona
Material realizado por Eureca Media, SL

Depósito legal: B-15.562-2005
ISBN: 84-9788-269-5