Guía MySQL II

CONTENIDO

- Proyecto: Programación avanzada de Base de Datos
- Motores de almacenamiento de MySQL y tipos de tablas
- El motor de almacenamiento InnoDB
- Procedimientos almacenados y funciones
- Disparadores (triggers)
- Vistas (Views)
- La base de datos de información INFORMATION_SCHEMA
- Matemáticas de precisión:
- Tipos de valores numéricos
- Cambios en el tipo de datos DECIMAL
- Manejo de expresiones
- MySQL Connector/ODBC
- MySQL Connector/NET
- MySQL Connector/J
- MySQL Connector/MXJ
- Connector/PHP
- Ejercicios avanzados

MOTORES DE ALMACENAMIENTO DE MYSQL Y TIPO DE TABLAS

Un motor de almacenamiento es una parte esencial de un SGDB puesto que se encarga de crear, recuperar, actualizar y borrar los datos de una base de datos.

¿Por qué son importantes los motores de almacenamiento?

Los datos en MySQL pueden ser almacenados de diversas formas, con distintas técnicas que nos aportan distintas funcionalidades. Aquí es donde encontramos la importancia de los motores de almacenamiento, ya que, dependiendo del motor que elijamos obtendremos distintas ventajas y dependerá de nosotros saber sacarles partido de forma acorde a nuestras necesidades.

MySQL nos provee por defecto de una serie de motores que podemos usa fácilmente (a parte de otros desarrollados por terceras partes que también podemos instalar), de hecho esta flexibilidad es una de las bazas que han jugado a favor de la popularidad de MySQL.

¿Cómo saber que motores están disponibles?

Es muy sencillo, basta con escribir en una consola MySQL la siguiente consulta:

```
show engines;
```

Y obtendremos algo como esto:

Engine	Support	Comment
MyISAM	DEFAULT	Default engine as of MySQL 3.23 with great performance
MEMORY	YES	Hash based, stored in memory, useful for temporary tables
InnoDB	YES	Supports transactions, row-level locking, and foreign keys
BerkeleyDB	NO	Supports transactions and page-level locking
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)
EXAMPLE	NO	Example storage engine
ARCHIVE	YES	Archive storage engine
CSV	YES	CSV storage engine
ndbcluster	DISABLED	Clustered, fault-tolerant, memory-based tables
FEDERATED	YES	Federated MySQL storage engine
MRG_MYISAM	YES	Collection of identical MyISAM tables
ISAM	NO	Obsolete storage engine

Motores y diferencias

Las comparativas entre motores se suelen hacer basándose en cuatro funcionalidades clave:

- Tipos de datos: aunque la mayoría son comunes hay algunos específicos que pueden ser decisivos bajo determinadas circunstancias.

- Bloqueo de datos: la forma en la que el motor protege un dato que está siendo modificado para evitar problemas de acceso concurrente a los datos y mantener la integridad referencial.
- Indexado: las diferentes técnicas de indexado pueden influir drásticamente es el rendimiento de una base de datos.
- Transacciones: dota de fiabilidad a los datos mientras se realizan operaciones, te permite utilizar los datos pero sólo te permite guardarlos cuando se comprueba que las otras condiciones que pudiesen requerirse se han cumplido.

MOTORES DE ALMACENAMIENTO

1. MyISAM

Se basa en el antiguo ISAM, al que añade muchas mejoras, es el motor que usa MySQL por defecto. Es una buena combinación entre funcionalidad y rendimiento aunque carece de algunas características interesantes.

Características más importantes:

● Límite de 2^32 registros
● Máximo de 64 índices por tabla
● Máximo de 16 columnas por índice
● Los datos son independientes de la máquina y el sistema operativo
● Permite campos índice como NULL
● BLOB y TEXT pueden ser índices
● Permite un gran tamaño en las tablas (hasta 256TB)
● No soporta transacciones
● Bloquea los datos a nivel de tabla
● No permite “claves ajenas”

Este motor pone especial empeño en la rapidez de las operaciones de lectura (predominio de SELECT), es una de las razones por las que MySQL es tan popular en la web, ya que la mayoría de las operaciones que se realizan son de este tipo. Que no tenga que hacer comprobaciones de integridad referencial también influye en su velocidad.

Ejemplo de creación de una tabla:

```
CREATE TABLE pruebaMyISAM (
codigo varchar(5) default NOT NULL,
descripcion varchar(255) default NULL,
PRIMARY KEY (codigo)
) ENGINE=MyISAM;
```

En general no hará falta indicar el uso de este motor pues es el que se usa por defecto.

2. MERGE

Permite combinar varias tablas de igual estructura en una única tabla, pudiendo así realizar consultas sobre una tabla que nos devuelve datos de varias.

Características más importantes:

- Límite de 2^32 registros
- Las tablas "base" deben ser MyISAM
- Bloqueo a nivel de tabla
- No tiene índices, usa los de las tablas "base" (salvo FULLTEXT)
- La lectura es más lenta al tener que ir consultando la clave en cada una de las tablas subyacentes
- No permite REPLACE
- No soporta transacciones
- En su creación no comprueba que las tablas que usa existan y tengan una estructura idéntica

Una de sus funcionalidades puede ser partir una tabla muy grande en otras más pequeñas y, al unirlas con MERGE, permitirnos trabajar con ellas como si fuesen una sola.

Ejemplo de creación de una tabla a partir de otras dos:

```
mysql> CREATE TABLE t1 (
    -> a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> message CHAR(20));
mysql> CREATE TABLE t2 (
    -> a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> message CHAR(20));
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
    -> a INT NOT NULL AUTO_INCREMENT,
    -> message CHAR(20), INDEX(a))
    -> TYPE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

3. MEMORY (HEAP)

Guarda todos los datos en memoria, de forma que si se cae el servidor o reiniciamos MySQL se pierden los datos, aunque la estructura de las tablas se guarda.

Características más importantes:

- Bloquea los datos a nivel de tabla
- Puede usar índices HASH

- No soporta BLOB ni TEXT
- No soporta transacciones
- Resulta extremadamente fácil perder los datos

Resultan útiles como tablas temporales para determinadas consultas ya que al estar en memoria y poder tener Hash como índice resultan extremadamente rápidas, es una buena elección cuando necesitamos realizar operaciones muy rápidas sobre conjuntos pequeños de datos.

4. FEDERATED

Este motor se incluyó en la versión 5.03. La novedad de este motor es que permite el acceso a una base de datos MySQL remota como si fuese local, en realidad tenemos una tabla local que representa a otra remota, ambas deben ser idénticas.

Características más importantes:

- Permite acceso a BBDD remotas
- MySQL no instala este motor por defecto
- No soporta transacciones
- No contempla el bloqueo de datos
- No permite ALTER

Ejemplo de tabla en la que se le indica la dirección de los datos:

```
CREATE TABLE federated_table (
    id     int(20) NOT NULL auto_increment,
    name   varchar(32) NOT NULL default '',
    other  int(20) NOT NULL default '0',
    PRIMARY KEY  (id),
    KEY name (name),
    KEY other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
COMMENT='mysql://root@remote_host:9306/federated/test_table';
```

5. ARCHIVE

Se utiliza básicamente para almacenar grandes cantidades de datos sin índices en muy poco espacio, ya que los comprime con zlib alcanzando un nivel de ahorro de espacio considerable.

Características más importantes:

- Gran compresión de los datos
- Sólo permite INSERTS y SELECTS
- Bloquea los datos a nivel de registro
- Almacena los datos en un buffer hasta que los comprime e inserta
- No soporta transacciones

Este motor resulta especialmente útil para el almacenamiento de históricos o logs ya que suelen ocupar gran cantidad de espacio y no es necesario modificarlos con posterioridad.

6. CSV

Almacena la información utilizando el formato de valores separados por comas (comma-separated values), de forma que cada tabla es un fichero que contiene los datos. No soporta indexado y su fin principal es permitir exportar las datos de forma que puedan ser importados fácilmente por algunas suites ofimáticas.

Características más importantes:

- Útil para exportar e importar datos
- No soporta indexación ni transacciones

7. BLACKHOLE

El sorprendente uso de este motor es no almacenar los datos sino crear un log con la consulta SQL utilizada. Como no almacena ningún dato lógicamente no soporta índices, ni transacciones.

Su principal utilidad es mantener un servidor esclavo que mantenga un log del sistema principal.

8. NDB

Es el motor de almacenamiento de los clúster de MySQL, las bases de datos se reparten por los diferentes nodos de un clúster.

Características más importantes:

- Proporciona alta disponibilidad mediante redundancia.
- Proporciona alto rendimiento mediante fragmentación de datos sobre los grupos de nodos.
- Proporciona alta escalabilidad mediante la combinación de las dos características anteriores.
- Los datos se guardan en memoria, pero los logs van a disco.

Es una buena elección cuando disponiendo de varios servidores necesitamos a la vez velocidad, transacciones y redundancia de datos; replicación síncrona; y resistencia a caídas de servidores.

9. BerkeleyDB (o BDB)

Este motor, independiente de MySQL, provee altas prestaciones. Posee un mecanismo de almacenamiento basado en hash de alta eficiencia, lo que facilita el rápido acceso a los datos de forma directa a costa de la lentitud en el acceso secuencial.

Características más importantes:

- MySQL no lo instala por defecto
- Máximo de 31 índices por tabla
- Máximo de 16 columnas por índice
- Hasta 256TB
- Sí soporta transacciones
- Usa índices HASH
- MySQL necesita una clave primaria por cada tabla BDB, en caso de no existir creará una oculta
- El bloqueo interno de las tablas se hace a nivel de página (8192 bytes)
- Abrir muchas tablas es bastante lento
- Cada tabla se almacena en la ruta de creación definida y no se puede cambiar de directorio salvo usando mysqldump

10. InnoDB

Está considerado como uno de los motores más avanzados para el almacenamiento de datos en MySQL. Provee un motor sólido con soporte completo de transacciones (es ACID compliant), permite el bloqueo de datos a nivel de registro permitiendo gran flexibilidad a la hora de utilizar las tablas, controla la integridad referencial, permite claves ajenas y tiene un sistema de recuperación de caídas.

No obstante la piedra de toque de InnoDB es su mecanismo de indexación y cache de los registros pues mantiene una caché de índices y datos en memoria y en disco proporcionando un muy alto rendimiento.

Características más importantes:

- ACID compliant
- Permite claves ajenas y transacciones, soporte de integridad referencial
- Bloqueo de datos a nivel de registro y no bloquea la lectura durante los selects (mejora la concurrencia)
- Sistema de recuperación de caídas
- Cambiar la ubicación de la base de datos/tabla es complicado

- Una tabla no puede tener más de 1000 columnas
- El tamaño de sus logs debe ser inferior a 4GB
- El tamaño máximo para una tabla es de 64TB
- No permite índices de FULLTEXT
- No mantiene una contador interno de registros (select count(*) from tabla lento al tener recorrer todo el índice)

Ejemplo de uso de transacciones:

```
mysql> CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A))
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM CUSTOMER;
+------+--------+
| A    | B      |
+------+--------+
|   10 | Heikki |
+------+--------+
1 row in set (0.00 sec)
```

11. FALCON

Es el futuro motor de almacenamiento para MySQL 6.0, actualmente se encuentra en estado alpha, y está diseñada para entornos de servidores Web de alto volumen.

Características más importantes:

- Cumple ACID
- Permite recuperación de estados / datos en caso de caídas
- Permite una alta concurrencia (bloqueo a nivel de registro)
- Caché de datos muy rápida y potente
- Incluye tablas para la monitorización de rendimiento y errores
- Permite una configuración simple

ACID

Se denomina ACID a la propiedad de una base de datos para realizar transacciones seguras. Así pues ACID compliant define a un sistema de gestión de bases de datos que puede realizar transacciones seguras.

En concreto ACID es un acrónimo de Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.

TIPOS DE TABLAS USADAS POR MYSQL

1. **ISAM**: es el formato de almacenaje mas antiguo, y posiblemente podria desaparecer en futuras versiones. Presentaba limitaciones importantes como la no exportación de ficheros entre maquinas de distintas arquitecturas o que no podia usar mayores de 4 GigaBytes.
2. **MYISAM**: es el tipo de tabla por defecto en MySQL desde la versión 3.23. Optimizada para sistemas operativos de 64 bits, permite ficheros de tamaños mayores que las ISAM. Los datos se almacenan en un formato independiente, lo que permite pasar tablas entre distintas plataformas. Los índices se almacenan en un archivo con la extensión ".MYI" y los datos en otro archivo con extensión ".MYD". Ofrece la posibilidad de indexar campos BLOB y TEXT. Además este tipo de tablas soportan el tipo de dato VARCHAR.

Un inconveniente es que las tablas pueden llegar a corromperse, almacenando datos incorrectos. Esto puede ser causado por:

- El proceso mysqld haya sido eliminado en el transcurso de una escritura
- problemas de hardware.
- Una caida del sistema durante su utilización.
- Un gusano en el código Mysql o MyISAM.

3. **INNODB**: InnoDB provee a MySQL con el soporte para trabajar con transacciones, además de hacer un mejor bloqueo de registros para las instrucciones SELECT muy parecido al usado por Oracle, con lo que incrementa el rendimiento y la concurrencia en ambientes multiusuario, por otro lado, InnoDB es el único formato que tiene MySQL para soportar llaves foráneas (FOREING KEY). Además de todo lo comentado, InnoDB ofrece unos rendimientos superiores a la anterior tecnología de tablas de MySQL (MyISAM).

InnoDB es un motor de bases de datos muy completo que ha sido integrado dentro de MySQL.

Otras de sus caractéristicas són:

- *Recuperación automática ante fallas*. Si MySQL se da de baja de una forma anormal, InnoDB automáticamente completará las transacciones que quedaron incompletas.

- *Integridad referencial*. Ahora se pueden definir llaves foráneas entre tablas InnoDB relacionadas para asegurarse de que un registro no puede ser eliminado de una tabla si aún está siendo referenciado por otra tabla.

- *Bloqueo a nivel de filas*. Al usar tablas MyISAM, y tener consultas muy grandes que requieren de mucho tiempo, simplemente no se podían ejecutar más consultas hasta que terminarán las consultas que estaban en ejecución. En cambio, las tablas InnoDB usan bloqueo a nivel de filas para mejorar de manera impresionante el rendimiento.

- *SELECTs sin bloqueo*. El motor InnoDB usa una técnica conocida como multi-versioning (similar a PostgreSQL) que elimina la necesidad de hacer bloqueos en consultas SELECT muy simples. Ya no será necesario molestarse porque una simple consulta de sólo lectura está siendo bloqueada por otra consulta que está haciendo cambios en una misma tabla.

4. **HEAP**: Tablas en memoria. Son temporales y desaparecen cuando el servidor se cierra, a diferencia de una tabla TEMPORARY, que solo puede ser accedida por el usuario que la crea, una tabla HEAP puede ser utilizada por diversos usuarios. No soportan columnas de autoincremento ni que haya valores nulos en los índices. Los datos son almacenados en pequeños bloques.
5. **BDB**: Base de datos Berkeley. TST(Transactions safe tables). Solo en MySQL
6. **MAX**: Este tipo de tablas permite la realización de transacciones (a partir de la versión 3.23.34), por lo que es posible la recuperación de datos (COMMIT y ROLLBACK). Estas tablas necesitan de una clave primaria en cada tabla, que ha de crear el administrador o de lo contrario Mysql creará una oculta. Otra de sus características es que pueden ser bloqueadas con el comando LOCK. Estas tablas són almacenadas en archivos ".DB".
7. **El TST:** *'Transactions safe tables'*, o tablas para transacciones seguras. Son menos rápidas y ocupan mas memoria, pero a cambio ofrecen mayor seguridad frente a fallos durante la consulta. Las tablas TST permiten ir introduciendo consultas y finalizar con un COMMIT (que las ejecuta) o ROLLBACK (que ignora los cambios). Disponbles a partir de la versión 4 de MySQL.

PROCEDIMIENTOS ALMACENADOS Y FUNCIONES

Los procedimientos almacenados son un conjunto de instrucciones SQL más una serie de estructuras de control que nos permiten dotar de cierta lógica al procedimiento. Estos procedimientos están guardados en el servidor y pueden ser accedidos a través de llamadas, como veremos más adelante.

Para crear un procedimiento, MySQL nos ofrece la directiva CREATE PROCEDURE. Al crearlo éste es ligado o relacionado con la base de datos que se está usando, tal como cuando creamos una tabla, por ejemplo.

Para llamar a un procedimiento lo hacemos mediante la instrucción CALL. Desde un procedimiento podemos invocar a su vez a otros procedimientos o funciones.

Un procedimiento almacenado, al igual cualquiera de los procedimientos que podamos programar en nuestras aplicaciones utilizando cualquier lenguaje, tiene:

- Un nombre.
- Puede tener una lista de parámetros.
- Tiene un contenido (sección también llamada definición del procedimiento: aquí
- se especifica qué es lo que va a hacer y cómo).
- Ese contenido puede estar compuesto por instrucciones sql, estructuras de control, declaración de variables locales, control de errores, etcétera.

MySQL sigue la sintaxis SQL:2003 para procedimientos almacenados, que también usa IBM DB2.

En resumen, la sintaxis de un procedimiento almacenado es la siguiente:

CREATE PROCEDURE nombre (parámetro) [características] definición

Puede haber más de un parámetro (se separan con comas) o puede no haber ninguno (en este caso deben seguir presentes los paréntesis, aunque no haya nada dentro).

Los parámetros tienen la siguiente estructura: modo nombre tipo

Donde:

- modo: es opcional y puede ser IN (el valor por defecto, son los parámetros que el procedimiento recibirá), OUT (son los parámetros que el procedimiento podrá modificar) INOUT (mezcla de los dos anteriores).
- nombre: es el nombre del parámetro.
- tipo: es cualquier tipo de dato de los provistos por MySQL.
- Dentro de características es posible incluir comentarios o definir si el procedimiento obtendrá los mismos resultados ante entradas iguales, entre otras cosas.
- definición: es el cuerpo del procedimiento y está compuesto por el procedimiento en sí: aquí se define qué hace, cómo lo hace y bajo qué circunstancias lo hace.

Así como existen los procedimientos, también existen las funciones. Para crear una función, MySQL nos ofrece la directiva CREATE FUNCTION.

La diferencia entre una función y un procedimiento es que la función devuelve valores. Estos valores pueden ser utilizados como argumentos para instrucciones SQL, tal

como lo hacemos normalmente con otras funciones como son, por ejemplo, MAX() o COUNT().

Utilizar la cláusula RETURNS es obligatorio al momento de definir una función y sirve para especificar el tipo de dato que será devuelto (sólo el tipo de dato, no el dato).

Su sintaxis es:

```
1.    CREATE FUNCTION nombre (parámetro)
2.    RETURNS tipo
3.    [características] definición
```

Puede haber más de un parámetro (se separan con comas) o puede no haber ninguno (en este caso deben seguir presentes los paréntesis, aunque no haya nada dentro). Los parámetros tienen la siguiente estructura:nombre tipo

Donde:

- nombre: es el nombre del parámetro.
- tipo: es cualquier tipo de dato de los provistos por MySQL.
- Dentro de características es posible incluir comentarios o definir si la función devolverá los mismos resultados ante entradas iguales, entre otras cosas.
- definición: es el cuerpo del procedimiento y está compuesto por el procedimiento en sí: aquí se define qué hace, cómo lo hace y cuándo lo hace.

Para llamar a una función lo hacemos simplemente invocando su nombre, como se hace en muchos lenguajes de programación.

Desde una función podemos invocar a su vez a otras funciones o procedimientos.

```
1.    mysql> delimiter //
2.    mysql> CREATE PROCEDURE procedimiento (IN cod INT)
3.        -> BEGIN
4.        ->    SELECT * FROM tabla WHERE cod_t = cod;
5.        -> END
6.        -> //
7.    Query OK, 0 rows affected (0.00 sec)
8.    mysql> delimiter ;
9.    mysql> CALL procedimento(4);
```

En el código anterior lo primero que hacemos es fijar un delimitador. Al utilizar la línea de comandos de MySQL vimos que el delimitador por defecto es el punto y coma (;): en los procedimientos almacenados podemos definirlo nosotros.

Lo interesante de esto es que podemos escribir el delimitador anterior; sin que el procedimiento termine. Más adelante, en este mismo código volveremos al delimitador clásico. Luego creamos el procedimiento con la sintaxis vista anteriormente y ubicamos el contenido entre las palabras reservadas BEGIN y END.

El procedimiento recibe un parámetro para luego trabajar con él, por eso ese parámetro es de tipo IN. Definimos el parámetro como OUT cuando en él se va aguardar la salida del procedimiento. Si el parámetro hubiera sido de entrada y salida a la vez, sería de tipo denominado INOUT.

El procedimiento termina y es llamado luego mediante la siguiente instrucción:

mysql> CALL procedimento(4);

Otro ejemplo:

```
1.  CREATE PROCEDURE procedimiento2 (IN a INTEGER)
2.  BEGIN
3.  DECLARE variable CHAR(20);
4.  IF a > 10 THEN
5.  SET variable = ‘mayor a 10’;
6.  ELSE
7.  SET variable = ‘menor o igual a 10’;
8.  END IF;
9.  INSERT INTO tabla VALUES (variable);
10. END
```

• El procedimiento recibe un parámetro llamado a que es de tipo entero.
• Se declara una variable para uso interno que se llama variable y es de tipo char.
• Se implementa una estructura de control y si a es mayor a 10 se asigna a variable un valor. Si no lo es se le asigna otro.
• Se utiliza el valor final de variable en una instrucción SQL.

Observemos ahora un ejemplo de funciones:

```
1.  mysql> delimiter //
2.  mysql> CREATE FUNCTION cuadrado (s SMALLINT) RETURNS SM
ALLINT
3.      -> RETURN s*s;
4.      -> //
5.  Query OK, 0 rows affected (0.00 sec)
6.  mysql> delimiter ;
7.  mysql> SELECT cuadrado(2);
```

DISPARADORES O TRIGGERS EN UNA BASE DE DATOS

Un Disparador o Trigger es una rutina autónoma asociada con una tabla o vista que automáticamente realiza una acción cuando una fila en la tabla o la vista se inserta (INSERT), se actualiza (UPDATE), o borra (DELETE). Un Disparador nunca se llama directamente, en cambio, cuando una aplicación o usuario intenta insertar, actualizar, o anular una fila en una tabla, la acción definida en el disparador se ejecuta automáticamente (se dispara).

Las ventajas de usar los Disparadores son:

- La entrada en vigor automática de restricciones de los datos, hace que los usuarios entren sólo valores válidos.
- El mantenimiento de la aplicación se reduce, los cambios a un disparador se refleja automáticamente en todas las aplicaciones que tienen que ver con la tabla sin la necesidad de recompilar o relinquear.
- Logs automáticos de cambios a las tablas. Una aplicación puede guardar un registro corriente de cambios, creando un disparador que se active siempre que una tabla se modifique.
- La notificación automática de cambios a la Base de Datos con alertas de evento en los disparadores.

Los Dispararores tienen dos palabras clave, OLD y NEW que se refieren a los valores que tienen las columnas antes y después de la modificación. Los INSERT permiten NEW, los DELETE sólo OLD y los UPDATE ambas.

Un ejemplo de un disparador seria uno asociado a la sentencia DELETE en una tabla de clientes, para impedir que se elimine uno que tenga un saldo distinto de cero. Otro disparador seria guardar los datos que se modifican de un cliente en otra base de datos que serviria de auditoría.

Cómo crear trigger en MySQL Server 5, disparadores para auditoría de tablas

Explicamos cómo crear triggers o disparadores en MySQL Server 5.1. Mostramos qué es un trigger y cómo usarlo en MySQL. Como ejemplo creamos un trigger para simular un deshacer de una tabla, un trigger que guarda de forma automática los valores de los campos anteriores y los nuevos valores en caso de modificación de un registro. Cómo crear, eliminar y mostrar triggers en MySQL.

TRIGGER (DISPARADOR)

Un trigger o disparador en una Base de datos , es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación. Dependiendo de la base de datos, los triggers pueden ser de inserción (INSERT), actualización (UPDATE) o borrado (DELETE). Algunas bases de datos pueden ejecutar triggers al crear, borrar o editar usuarios, tablas, bases de datos u otros objetos.

Los triggers son usados para mejorar la administración de la Base de datos, sin necesidad de contar con que el usuario ejecute sentencias de SQL determinadas para tal efecto. Además, pueden generar valores de columnas, pueden prevenir errores de datos, sincronizar tablas, modificar valores de una vista, auditorías de seguridad, etc.

La estructura básica de un trigger es:

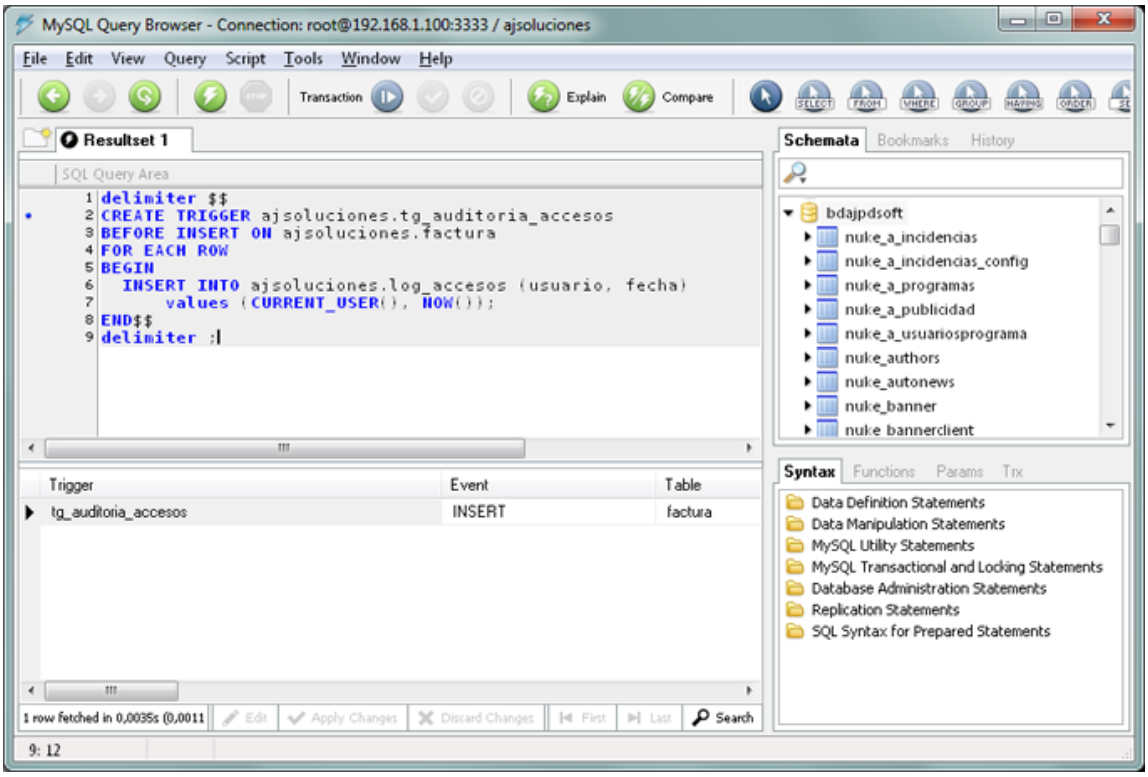
- **Llamada de activación**: es la sentencia que permite "disparar" el código a ejecutar.
- **Restricción**: es la condición necesaria para realizar el código. Esta restricción puede ser de tipo condicional o de tipo nulidad.
- **Acción a ejecutar**: es la secuencia de instrucciones a ejecutar una vez que se han cumplido las condiciones iniciales.

Existen dos tipos de disparadores que se clasifican según la cantidad de ejecuciones a realizar:

- ***Row Triggers*** (o disparadores de fila): son aquellos que se ejecutaran n-veces si se llaman n-veces desde la tabla asociada al trigger.
- ***Statement Triggers*** (o disparadores de secuencia): son áquellos que sin importar la cantidad de veces que se cumpla con la condición, su ejecución es única.

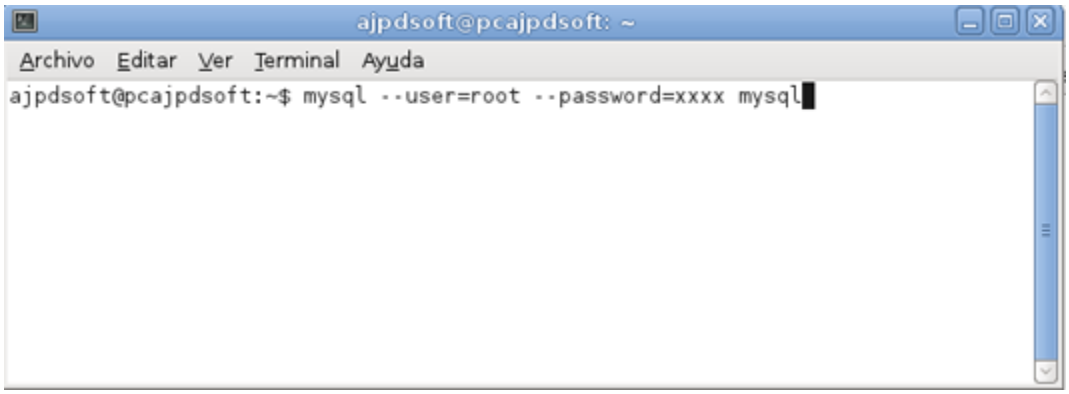
Cómo crear un trigger en mysql server 5.1

Para crear un trigger o disparador en MySQL Server deberemos usar alguna aplicación que permita ejecutar sentencias SQL, por ejemplo MySQL Administrator con MySQL Query Browser:



O desde una terminal (MS-DOS de Windows o Linux) ejecutando el comando "mysql":

mysql --user=nombre_usuario --pasword=contraseña_usuario base_datos



En cuanto dispongamos de la aplicación para ejecutar sentencias SQL y un usuario de MySQL Server con permisos suficientes para crear triggers o disparadores en la base de datos donde queramos, a continuación deberemos analizar para qué vamos a usar el trigger, dependiendo de la tarea a realizar necesitaremos, por ejemplo, una tabla auxiliar.

Para obtener el usuario actual de MySQL Server hemos usado la función *CURRENT_USER()* y para obtener la fecha y la hora actuales hemos usado la función *NOW()*.

La **sintaxis** para crear un trigger en MySQL Server:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name trigger_time trigger_event
    ON tbl_name FOR EACH ROW trigger_body
```

Otras acciones o tareas a realizar con los triggers o disparadores de mysql

- Para consultar los triggers o disparadores creados en una base de datos ejecutaremos el comando SQL:

show create triggers;

Nos mostrará un registro por cada trigger creado con los campos: trigger, event, table, statement, timing, created, sql_mode, definer, character_set_client, collation_connection, database_collation. Las importantes son:

- **Trigger**: almacena el nombre del disparador.
- **Event**: indica el tipo de trigger (insert, update, delete).
- **Table**: tabla de la base de datos a la que se asocia el trigger.
- **Statement**: código SQL del trigger.
- **Timing**: tiempo en que se ejecutará el trigger: before (antes), after (después).
- Para **eliminar** un trigger o disparador existente ejecutaremos la siguiente consulta:

drop trigger nombre_trigger;

- Para **mostrar la consulta SQL** completa de **creación** de un trigger ejecutaremos el comando:

show create trigger nombre_trigger;

- **Almacenamiento** de los trigger en MySQL: los triggers o disparadores se almacenan en la tabla *TRIGGERS* del catálogo del sistema *information_schema*, para verlos:

*select * from information_schema.triggers*;

Otras opciones y funcionalidades de los trigger en mysql

- **Seguridad**: si queremos que un usuario, además del superusuario "root" tenga permisos para crear triggers o disparadores en una tabla, ejecutaremos el comando SQL:

GRANT CREATE TRIGGER ON nombre_tabla TO nombre_usuario

- **Seguridad**: para dar permisos de creación de triggers para un usuario para todas las tablas ejecutaremos el comando SQL:

*GRANT CREATE TRIGGER ON *.* TO nombre_usuario*

Vistas (Views)

Una vista es un objecto de la base de datos que se define mediante una SELECT que agrupa o selecciona un conjunto de datos. Vamos a ver cómo usarlas.

Vamos a suponer la tabla valias:

```
mysql> desc valias;
+-------------+----------+------+-----+---------+-------+
| Field       | Type     | Null | Key | Default | Extra |
+-------------+----------+------+-----+---------+-------+
| alias       | char(32) | NO   | MUL | NULL    |       |
| domain      | char(64) | NO   |     | NULL    |       |
| valias_line | text     | NO   |     | NULL    |       |
+-------------+----------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

En esta tabla podemos consultar los alias de correo para todos los dominios del servidor. Para poder dar acceso a un usuario que solo tenga acceso a sus dominios lo podemos hacer mediante vistas. Por ejemplo, podemos crear una vista para un determinado dominio mediante la siguiente vista:

```
mysql> create view valias_systemadmin_es as select alias, valias_line from valias where domain="systemadmin.es";
Query OK, 0 rows affected (0.04 sec)
```

A continuación mediante la vista podremos acceder solo a los datos que la vista selecciona:

```
mysql> select * from valias_systemadmin_es;
+--------------+----------------------------+
| alias        | valias_line                |
+--------------+----------------------------+
| helpdesk     | &diwit@systemadmin.es      |
(...)
```

Una de las confusiones más comunes entre los programadores es suponer un aumento del rendimiento por usar vistas. Vamos a ver cómo funcionan realmente.
Mediante la palabra clave “ALGORITHM” podemos indicar como deseamos que funcione:

- **UNDEFINED**: Dejamos que sea MySQL quien decida el algoritmo por si mismo, es el caso por defecto.
- **MERGE**: Se refiere a que junte la query que se hace sobre la vista con la query de la vista y se ejecute la query resultante. De esta manera vemos como la query que se ejecuta sobre

una vista es tan complicada como la suma de las dos queries. Esto lo tenemos que tener muy en cuenta, ya que estamos ocultando la query que realmente ejecuta MySQL. Un ejemplo sería:

```
mysql> create ALGORITHM=MERGE view valias_systemadmin_es as select alias, valias_line from valias where domain="systemadmin.es";
Query OK, 0 rows affected (0.00 sec)
```

El **EXPLAIN** resultante sería:

```
mysql> explain select * from valias_systemadmin_es;
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table  | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | valias | ALL  | NULL          | NULL | NULL    | NULL |   27 | Using where |
+----+-------------+--------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.03 sec)
```

- **TEMPTABLE**: En el momento de hacer una consulta sobre la vista se crea una tabla temporal. Un ejemplo sería:

```
mysql> create ALGORITHM=TEMPTABLE view valias_systemadmin_es as select alias, valias_line from valias where domain="systemadmin.es";
Query OK, 0 rows affected (0.00 sec)
```

Con su correspondiente **EXPLAIN**:

```
mysql> explain select * from valias_systemadmin_es;
+----+-------------+------------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table      | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+------------+------+---------------+------+---------+------+------+-------------+
|  1 | PRIMARY     | <derived2> | ALL  | NULL          | NULL | NULL    | NULL |   15 |             |
|  2 | DERIVED     | valias     | ALL  | NULL          | NULL | NULL    | NULL |   27 | Using where |
+----+-------------+------------+------+---------------+------+---------+------+------+-------------+
2 rows in set (0.00 sec)
```

El caso de **TEMPTABLE** tiene una gran ventaja y una gran desventaja:

- **Desventaja**: La vista no es actualizable, por lo que cualquier cambio se deberá hacer en la tabla original.

- **Ventaja**: Los bloqueos se liberan antes, ya que la consulta de la vista se hace a partir de la tabla temporal. Esto permite que otros threads accedan antes a la tabla que ejecutando una consulta mucho más pesada usando el algoritmo MERGE.

La base de datos de información INFORMATION_SCHEMA

INFORMATION_SCHEMA es la base de datos de información, que almacena información acerca de todas las otras bases de datos que mantiene el servidor mysql. Dentro del INFORMATION_SCHEMA hay varias tablas de sólo lectura. En realidad son vistas, no tablas, así que no puede ver ningún fichero asociado con ellas. Cada usuario mysql tiene derecho a acceder a estas tablas, pero sólo a los registros que se corresponden a los objetos a los que tiene permiso de acceso.

Matemáticas de precisión

La matemática de precisión se basa en dos cambios de implementación:

- La introducción de nuevos modos SQL en MySQL 5.0.2 que controlan lo estricto que es el servidor para aceptar o rechazar datos inválidos.
- La introducción en MySQL 5.0.3 de una biblioteca para aritmética de punto fijo.

Estos cambios tienen varias implicaciones para operaciones numéricas:

- Cálculos más precisos.

Para números exactos, los cálculos no introducen error en coma flotante. En su lugar, se usa precisión exacta. Por ejemplo, un número tal como .0001 se trata como un valor exacto en lugar de valor aproximado, y sumarlo 10,000 veces produce un resultado de 1, no un valor "cercano" a 1.

- Comportamiento bien definido para el redondeo.

Para números exactos, el resultado de ROUND() depende de sus argumentos, no de factores tales como el comportamiento de la biblioteca C subyacente.

- Independencia de plataforma mejorada.

Las operaciones con números exactos son los mismos entre distintas plataformas tales como Windows y Unix.

- Control sobre tratamiento de datos inválidos.

Desbordamiento y división por cero pueden detectarse y tratarse como errores. Por ejemplo, puede tratar un valor que es demasiado grande para una columna como un error en lugar de truncarlo para adaptarlo al rango del tipo de datos. Similarmenet, puede tratar la división por cero como un error en lugar que como una operación que produce un resultado de NULL. La elección de qué aproximación seguir se determina mediante la variable de sistema sql_mode .

Tipos de valores numéricos

El ámbito de matemáticas de precisón para operaciones de valores exactos incluyen tipos de datos precisos (**DECIMAL** y tipos interos) y literales de valores numéricos exactos. Los tipos de datos aproximados y literales numéricos se tratan como valores en coma flotante.

Literales numéricos de valores exactos tienen una parte entera o fraccional, o ambas. Pueden tener signo. Ejemplos: **1**, **.2**, **3.4**, **-5**, **-6.78**, **+9.10**.

Literales de valores numéricos aproximados se representan en notación científica con una mantisa y exponente. Una o ambas partes pueden tener signo.
Ejemplos: **1.2E3**, **1.2E-3**, **-1.2E3**, **-1.2E-3**.

Números que parecen similares no necesitan ser ambos valores exactos o aproximados. Por ejemplo, **2.34** es un valor exacto (punto fijo), mientras que **2.34E0** es un valor aproximado (coma flotante).

El tipo de datos **DECIMAL** es un tipo de punto fijo y los cálculos son exactos. En MySQL, el tipo **DECIMAL** tiene varios sinónimos: **NUMERIC**, **DEC**, **FIXED**. El tipo entero también es un tipo de valor exacto.

Los tipos de datos **FLOAT** y **DOUBLE** son tipos de coma flotante y los cálculos son aproximados. En MySQL, los tipos sinónimos de **FLOAT** o **DOUBLE** son **DOUBLE PRECISION** y **REAL**.

Manejo de expresiones

Con matemáticas de precisión, los números con valores exactos se usan tal y como se dan cuando es posible. Por ejemplo, números en comparaciones se usan exactamente como se dan sin cambiar su valor. En modo SQL estricto, para un **INSERT** en una columna con un tipo exacto (**DECIMAL** o entero), se inserta un número con su valor exacto si está dentro del rango de la columna. Cuando se recibe, el valor debe ser el mismo que se insertó. (Sin modo estricto, se permite truncar para **INSERT**.)

El tratamiento de expresiones numéricas depende de qué clase de valores contiene la expresión:

- Si hay presente algún valor aproximado, la expresión es aproximada y se evalúa usando aritmética de punto flotante.
- Si no hay presente ningún valor aproximado, la expresión contiene sólo valores exactos. Si algún valor exacto contiene una parte fraccional (un valor a continuación del punto decimal), la expresión se evalúa usando aritmética exacta **DECIMAL** y una precisión de 64 dígitos. ("Exacto" esta sujeto a los límites de lo que puede representarse en binario. **1.0/3.0** puede representarse como **.333...** con un número finito de dígitos, no como "exactamente un tercio", así que **(1.0/3.0)*3.0** no se evalúa como "exactamente 1.0.")

- En otro caso, la expresión contiene sólo valores enteros. La expresión es exacta y evaluada usando aritmética entera y tiene la misma precisión que **BIGINT** (64 bits).

Si una expresión numérica contiene cualquier cadena de caracteres, se convierten a valores de coma flotante y doble precisión y la expresión es aproximada.

Las inserciones en columnas numéricas están afectadas por el modo SQL, controlada por la varible de sistema sql_mode. (Consulte Sección 1.7.2, “Selección de modos SQL”.)

La siguiente discusión menciona el modo estricto (seleccionado por los valores de modo **STRICT_ALL_TABLES** o **STRICT_TRANS_TABLES**) y**ERROR_FOR_DIVISION_BY_ZERO**.

Para activar todas las restricciones, puede usar el modo **TRADITIONAL**, que incluye tanto el modo estricto como **ERROR_FOR_DIVISION_BY_ZERO**:

```
mysql> SET sql_mode='TRADITIONAL';
```

Si se inserta un número en una columna de tipo exacto (**DECIMAL** o entero), debe insertarse con su valor exacto si está dentro del rango de la columna.

Si el valor tiene demasiados dígitos en la parte fraccional, se redondea y se genera una advertencia. El redondeo se hace como se describe en "Comportamiento del redondeo".

Si el valor tiene demasiados dígitos en la parte entera, es demasiado grande y se trata como se explica a continuación:

- Si el modo estricto no está activado, el valor se trunca al valor legal más cercano y se genera una advertencia.
- Si el modo estricto está activo, se genera un error de desbordamiento.

Desbordamiento inferior no se detecta, así que su tratamiento no está definido.

Por defecto, la división por cero produce un resultado de **NULL** y ninguna advertencia. Con el modo SQL**ERROR_FOR_DIVISION_BY_ZERO** activado, MySQL trata la división por cero de forma distinta:

- Si el modo estricto no está activo, aparece una advertencia.
- Si el modo estricto está activo, las inserciones y actualizaciones con divisiones por cero están prohibidas y ocurre un error.

En otras palabras, inserciones y actualizaciones que impliquen expresiones que realizan divisiones por cero pueden tratarse como errores, pero esto requiere **ERROR_FOR_DIVISION_BY_ZERO** además del modo estricto.

Suponga que tenemos este comando:

```
INSERT INTO t SET i = 1/0;
```

Esto es lo que ocurre al combinar modo estricto y **ERROR_FOR_DIVISION_BY_ZERO** :

sql_mode Valor	**Resultado**
''	No advertencia, no error, i es **NULL**
strict	No advertencia, no error, i es **NULL**
ERROR_FOR_DIVISION_BY_ZERO	Advertencia, no error, i es **NULL**
strict,**ERROR_FOR_DIVISION_BY_ZERO**	Error, no se inserta el registro

Para inserciones de cadenas de caracteres en columnas numéricas, las conversiones de cadenas a números se tratan como se muestra si la cadena tiene contenido no numérico:

- Una cadena que no comienza con un número no puede usarse como número y produce un error en modo estricto, o una advertencia en otro caso. Esto incluye la cadena vacía.
- Una cadena que comienza con un número puede convertirse, pero se trunca la parte no numérica final. Esto produce un error en modo estricto, o una advertencia en otro caso.

Cómo se redondea

Esta sección discute el redondeo de la matemática precisa para la función **ROUND()** y para inserciones en columnas **DECIMAL** .

La función **ROUND()** redondea de forma distinta dependiendo de si su argumento es exacto o aproximada:

- Para valores exactos, **ROUND()** usa la regla "redondeo al alza": Un valor con parte fraccional de .5 o superior se redondea al siguiente entero si es positivo o al anterior entero si es negativo. (En otras palabras, siempre se redondea alejándose del cero.) Un valor con una parte fraccional menor que .5 se redondea al anterior valor entero si es positivo o al siguiente entero si es negativo.
- Para números aproximados, el resultado depende de la biblioteca C. En muchos sistemas, esto significa que**ROUND()** usa la regla "redondeo al número par más próximo": Un valor con un parte fraccional se redondea al siguiente entero par.

El siguiente ejemplo muestra cómo difiere el redondeo para valores exactos y aproximados:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

Para inserciones en una columna **DECIMAL**, el objetivo es un tipo de datos exacto, así que el redondea usa "redondeo al alza" independientemente de si el valor a ser insertado es exacto o aproximado:

```
mysql> CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 2

mysql> SELECT d FROM t;
+------+
| d    |
+------+
| 3    |
| 3    |
+------+
```

Cambios en el tipo de datos DECIMAL

En MySQL 5.0.3, se hicieron varios cambios en distintos aspectos del tipo de datos **DECIMAL** (y sus sinónimos):

- Número máximo de dígitos
- Formato de almacenamiento
- Requerimientos de almacenamiento
- Las extensiones MySQL no estándar al rango superior de columnas **DECIMAL**

Algunos de los cambios provocan posibles incompatibilidades para aplicaciones escritas en versiones antiguas de MySQL. Estas incompatibilidades se muestran durante esta sección.

La sintaxis de declaración para columnas **DECIMAL** sigue siendo **DECIMAL(*M*,*D*)**, aunque el rango de valores para los argumentos ha cambiado algo:

- ***M*** es el número máximo de dígitos (la precisión). Tiene un rango de 1 a 64. Introduce una posible incompatibilidad para aplicaciones antiguas, ya que versiones prévias de MySQL permiten el rango de 1 a 254.
- ***D*** es el número de dígitos a la derecha del punto decimal (la escala). Tiene el rango de 0 a 30 y no debe ser mayor que ***M***.

El valor máximo de 64 para ***M*** significa que los cálculos con valores **DECIMAL** son precisos hasta 64 dígitos. Este límite de 64 dígitos de precisión también se aplica a literales con valor exacto, así que el rango máximo de tales literales es diferente al anterior. (Antes de MySQL 5.0.3, los valores decimales podían tener hasta 254 dígitos. Sin embargo, los cálculos se hacían usando coma flotanto y por lo tanto eran aproximados, no exactos.) Este cambio en

el rango de valores literales es otra posible fuente de incompatibilidades para aplicaciones antiguas.

Los valores para columnas **DECIMAL** no se representan como cadenas que requieren un byte por dígito o carácter de signo. En su lugar, se usa un formato binario que empaqueta nueve dítigos decimales en cuatro bytes. Este cambio del formato de almacenamiento de **DECIMAL** cambia los requerimientos de almacenamiento también. El almacenamiento para las partes enteras y fraccionales de cada valor se determinan por separado. Cada múltiple de nueve dígitos necesita cuatro bytes, y los dígitos restantes necesitan una fracción de cuatro bytes. Por ejemplo, una columna **DECIMAL(18,9)** tiene nueve dígitos en cada parte del punto decimal, así que la parte entera y fraccional necesitan cuatro bytes cada una. Una columna **DECIMAL(20,10)** tiene 10 dígitos en cada lado del punto decimal. Cada parte requiere cuatro bytes para nueve de los dígitos, y un byte para el dígito restante.

El almacenamiento requerido para los dígitos restantes lo da la siguiente tabla:

Dígitos Restantes	**Número de Bytes**
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4
9	4

Como resultado del cambio de cadena de caracteres a formato numérico para almacenamiento **DECIMAL**, las columnas **DECIMAL** no necesitan un carácter '**+**' o dígito '**0**' precedente. Antes de MySQL 5.0.3, si insertaba**'+0003.1'** en una columna **DECIMAL(5,1)** , se almacenaría como **+0003.1**. Desde MySQL 5.0.3, se almacena como **3.1**. Aplicaciones que confían en el antiguo comportamiento deben modificarse teniendo en cuenta este cambio.

El cambio de formato de almacenamiento también significa que las columnas **DECIMAL** no soportan la extensión no estándar que permitía valores mayores que el rango implicado por la definición de la columna. Antiguamente, se reservaba un byte para almacenar el carácter de signo. Para valores positivos que no necesitaban byte de signo, MySQL permitía almacenar un byte extra. Por ejemplo, una columna **DECIMAL(3,0)** debe soportar un rango de al menos -999 a **999**, pero MySQL debería permetir almacenar valores de **1000** a **9999** también, usando el byte de signo para almacenar un dígito extra. Esta extensión del rango superior de las columnas**DECIMAL** no se permite. En MySQL 5.0.3 y

posteriores, una columna **DECIMAL(*M*,*D*)** permite como mucho ***M*−*D***dígitos a la izquierda del punto decimal. Esto puede resultar en una incompatibilidad si una aplicación tiene confianza en que MySQL permita valores "demasiado grandes".

El estándar SQL requiere que la precisión de **NUMERIC(*M*,*D*)** sean exactamente ***M*** dígitos.

Para **DECIMAL(*M*,*D*)**, requiere una precisión de al menos ***M*** dígitos, pero permite más. En MySQL, **DECIMAL(*M*,*D*)** y **NUMERIC(*M*,*D*)**son los mismo y ambos tienen una precisión de exactamente ***M*** dígitos.

Resumen de incompatibilidades:

La siguiente lista resume las incompatibilidades resultantes de cambios de la columna **DECIMAL** y tratamiento de valores. Puede usarla como guía cuando al portar aplicaciones antiguas para usar con MySQL 5.0.3 y posteriores.

- Para **DECIMAL(*M*,*D*)**, el máximo ***M*** es 64, no 254.
- Los cálculos que implican valores decimales con valores exactos son precisos hasta 64 dígitos. Esto es menor que el número máximo de dígitos permitidos antes de MySQL 5.0.3 (254 dígitos), pero la precisión exacta es mayor. Los cálculos anteriormente se hacían con punto flotante de doble precisión, que tiene una precisión de 52 bits (acerca de 15 dígitos decimales).
- La extensión no estándar MySQL del rango superior de columnas **DECIMAL** no se soporta.
- Los caracteres precedentes '**+**' y '**0**' no se almacenan.

MySQL Connector

Los Conectores MySQL, controladores (drivers) que proporcionan a los programas cliente conectividad con el servidor MySQL. Existen actualmente cinco conectores MySQL:

- Connector/ODBC proporciona soporte a nivel de controlador para la conexión con un servidor MySQL usando la API de Conectividad de Bases de datos Abierta (ODBC por sus siglas en inglés). Con este controlador la conexión ODBC es posible desde las plataformas Windows, Unix y Mac OS X.
- Connector/NET permite a los desarrolladores crear aplicaciones .NET usando los datos almacenados en una base de datos MySQL. Connector/NET implementa una interfaz ADO.NET totalmente funcional y proporciona soporte para su uso con herramientas compatibles con ADO.NET. Las aplicaciones que se desee usen Connector/NET pueden escribirse en cualquier lenguaje .NET soportado.
- El Plugin Visual Studio MySQL trabaja con Connector/NET y Visual Studio 2005. Este plugin es un proveedor DDEX, lo que significa que se pueden usar herramientas de manipulación de esquemas y datos dentro de Visual Studio para crear y editar objetos dentro de una base de datos MySQL.

- Connector/J proporciona soporte de controlador para conectar con MySQL desde una aplicación Java usando la API de Conectividad con Bases de Datos Java estándar (JDBC).
- Connector/MXJ es una herramienta que permite poner en marcha y administrar fácilmente el servidor y la base de datos MySQL a través de una aplicación Java
- Connector/PHP es un controlador para conectar Windows con PHP. Proporciona las extensiones mysql y mysqlipara su uso con MySQL 5.0.18 y posteriores.