Xestión de usuarios e Índices

Índice

1. Xestión de usuarios dunha base de datos

Unha das tarefas principais dun administrador de bases de datos é a xestión de usuarios.

Tódolos accesos a unha base de datos requiren a conexión mediante un usuario. Os usuarios teñen asignados unha serie de privilexios que son os que lles dan permiso de uso de certos obxectos da base de datos. Desta maneira todo usuario terá dereito a utilizar certos obxectos da base de datos e terá restrinxido o uso doutros.

Para unha maior simplicidade, a maioría de Sistemas Xestores de Bases de Datos permiten agrupar os permisos que se lles poden aplicar aos usuarios nunhas estruturas chamadas perfís e roles, que en definitiva son un conxunto de permisos.

Deste xeito, cando un usuario quere conectarse a unha base de datos, primeiro debe autentificarse, é dicir, probar que é quen di ser (normalmente mediante un contrasinal). Dita autentificación terá asociados uns privilexios (uns dereitos) e unhas restricións.

Polo tanto, será responsabilidade do administrador a creación de usuarios e a asignación aos mesmos dos distintos roles e privilexios que lles permitan desenvolver a súa actividade sen poñer en compromiso a seguridade da base de datos.

De forma xeral, non é unha boa práctica deixar que todos os usuario con acceso ao servidor teñan todos os privilexios. Para conservar a integridade dos datos e das estruturas, será conveniente que só algúns usuarios poidan realizar determinadas tarefas, e que outras, que requiran maior coñecemento sobre as estruturas de bases de datos e táboas, só poidan realizarse por un número limitado e controlado de usuarios.

1.1 Xestión de usuarios en MySQL

1.1.1 Niveis de privilexios

En MySQL existen cinco niveis distintos de privilexios:

- **Globais**: aplícanse ao conxunto de todas as bases de datos nun servidor. É o nivel máis alto de privilexios, no sentido de que o seu ámbito é o máis xeral.
- **De base de datos**: refírense a bases de datos individuais, e por extensión, a todos os obxectos que contén cada base de datos.
- **De táboa**: aplícanse a táboas individuais, e polo tanto, a todas as columnas desas táboa.
- **De columna**: aplícanse a unha columna nunha táboa concreta.
- **De rutina**: aplícanse aos procedementos almacenados.

1.1.2 Creación de usuarios

Aínda que a partir da versión 5.0.2 de MySQL existe unha sentenza para crear usuarios, CREATE USER, en versións anteriores úsase exclusivamente a sentenza GRANT para crealos.

A sintaxe de CREATE USER é:

```
CREATE USER usuario [IDENTIFIED BY [PASSWORD] 'contrasinal']
                    [, usuario [IDENTIFIED BY [PASSWORD] 'contrasinal']] ...
```

Usando GRANT pódese crear un usuario e ao mesmo tempo concederlle tamén os privilexios que terá, aínda que **nas últimas versións de MySQL a recomendación é crear primeiro o novo usuario con CREATE USER e logo usar GRANT para darlle privilexios**.

A sintaxe simplificada para usar GRANT é:

```
GRANT    tipo_privilexio [(lista_columnas)] [, tipo_privilexio [(lista_columnas)]] ...
      ON  obxecto  TO   nome_usuario[ @equipo ] [ IDENTIFIED BY 'contrasinal' ]
      [WITH GRANT OPTION]
```

- **tipo_privilexio**, representa os privilexios que se lle poden conceder aos usuarios, e dicir, o que se lle vai a permitir facer cos obxectos do servidor. A orden **show privileges** permite ver todos os tipos de privilexios posibles. ***Algúns dos máis utilizados son***:
  - ALL [PRIVILEGES] — Todos os privilexios, excepto GRANT OPTION.
  - ALTER — Modificar obxectos coa orden ALTER (táboas).
  - CREATE — Crear obxectos coa orden CREATE (bases de datos ou táboas).
  - CREATE VIEW — Crear vistas.
  - DROP — Borrar táboas con DROP TABLE.
  - EXECUTE — Executar procedementos almacenados.
  - SELECT — Facer consultas con SELECT.
  - UPDATE — Modificar datos das táboas.
  - USAGE — Sinónimo de 'sen privilexios'.
  - GRANT OPTION — Conceder privilexios a outros usuarios.

- No caso de utilizar a opción WITH GRANT OPTION, se lle está dando ao usuario a posibilidade de ceder a outros usuarios os privilexios que se lle conceden a el.

- **obxecto**, representa sobre que cousas se conceden, ou retiran, os privilexios. Os máis utilizados son:
  - *.* — Todas as táboas de todas a bases de datos
  - * — Todas as táboas da base de datos activa
  - nome _bd.* — Todas as táboas da base de datos nome_bd
  - nome_db.nome_táboa — A táboa especificada da base de datos nome_bd

  Tamén se poden conceder ou retirar privilexios sobre funcións ou procedementos almacenados.

- **nome_usuario**, representa o usuario ao que se conceden os permisos. O nome do usuario é unha cadea de carácteres que só debe levar letras, números, e o guión baixo (_).

- **equipo**, pode ser o nome dun equipo, unha dirección IP, ou ben, o símbolo %, que representa calquera ordenador (excepto a máquina local). O símbolo % tamén se pode utilizar como un carácter comodín en combinación co nome do equipo, ou a dirección IP:

  Exemplos de usuarios:

  - **'administrador'@'localhost'** — Usuario administrador cando se conecte desde o equipo local.

- **'administrador'@'%'** Usuario administrador cando se conecte desde calquera equipo da rede, excepto o equipo local (localhost).

- **'julio'@'ordenador124'** Usuario julio cando se conecta desde o equipo co nome ordenador124.

- **'andres'@ '192.68.123.50'** Usuario andres cando se conecta desde o equipo coa IP 192.68.123.50.

- **'luis'@'192.68.%.%'** Usuario luis cando se conecta desde un equipo cunha IP que empeza por 192.68.

- **contrasinal**, A cláusula IDENTIFIED BY permite asignarlle unha contrasinal ao usuario no momento en que se lle conceden os permisos de acceso. Se pode cambiar a contrasinal dun usuario usando algunha das seguintes sentencias SQL:

```
GRANT USAGE ON *.* TO  nome_usuario[ @equipo ]  IDENTIFIED BY 'contrasinal';
SET PASSWORD [FOR usuario]  =  PASSWORD('contrasinal');
```

  A función PASSWORD cifra o contrasinal, utilizando o sistema de cifrado de MySQL, que converte calquera cadea de texto nunha cadea de 41 carácteres en hexadecimal. Existen outros sistemas de cifrado que se poden utilizar facendo uso doutras funcións de cifrado, como por exemplo SHA1, ou MD5.

Aínda que sempre se deben conceder os mínimos privilexios necesarios, existen algúns privilexios que son especialmente perigosos. Por exemplo, nunca se debe conceder acceso de carácter global. Os seguintes privilexios poden resultar unha ameaza para a seguridade da base de datos:

Calquera privilexio sobre a base de datos mysql	Nesta base de datos se almacena información de todo o sistema de seguridade do servidor.
ALTER	Un usuario podería modificar as táboas de privilexios, e inutilizalas.
DROP	Un usuario podería borrar as táboas de privilexios, perdéndose a información das contas, o que impediría o acceso dos usuarios.
FILE	Os usuarios poderían crear un ficheiro con información das contas de usuario, que todo o mundo poda ler.
GRANT	Permite que un usuario poda ceder os seus privilexios a outros usuarios, que poden non ser tan fiables como el
SHUTDOWN	Os usuarios con este privilexio poden parar o servidor, e deixar ao resto dos usuarios sen servizo

Por exemplo, para crear un usuario sen privilexios:

```
GRANT USAGE ON *.* TO anonimo IDENTIFIED BY 'clave';
```

Hai que ter en conta que **o contrasinal débese introducir entre comiñas de forma obrigatoria**. O usuario 'anonimo' poderá abrir unha sesión MySQL, pero non poderá facer

moito máis, xa que non ten privilexios. Non terá, por exemplo, oportunidade de facer seleccións de datos, de crear bases de datos ou táboas, inserir datos, etc.

Máis exemplos:

- Conceder permiso para executar os comandos *insert* e *delete* na táboa *titles* ao usuario *Mary* :

```
grant insert, delete
on titles
to mary
```

- Conceder permiso para executar o comando *update* nas columnas *price* e *advance* da táboa *titles* ao usuario *public*.

```
grant update (price, advance)
on titles
to public
```

- Conceder permiso a Mary e John para utilizar os comandos create database e create table.

```
grant create database, create table
to mary, john
```

- Conceder todos os permisos de acceso á táboa *titles* a todos os usuarios.

```
grant all on titles
to public
```

- Conceder permiso a *Mary* para utilizar o comando *update* na táboa *authors* e para conceder ese permiso a outros.

```
grant update on authors
to mary
with grant option
```

1.1.3 Revogar privilexios

Para revogar privilexios úsase a sentenza REVOKE.

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
ON
FROM user [, user] ...
```

A sintaxe é similar á de GRANT, por exemplo, para revogar o privilexio SELECT da táboa *xente* da base de datos *proba* ao usuario *anonimo*, usarase a sentenza:

```
REVOKE SELECT ON proba.xente FROM anonimo;
```

1.1.4 Mostrar os privilexios dun usuario

Pódense ver os privilexios que se lle concederon a un usuario mediante a cláusula SHOW GRANTS. A saída desta sentenza é unha lista de sentenzas GRANT que se deben executar para conceder os privilexios que ten o usuario. Por exemplo:

```
mysql> SHOW GRANTS FOR anonimo;
-------------------------------------------------------------------
| Grants for anonimo@% |
-------------------------------------------------------------------
| GRANT USAGE ON *.* TO 'anonimo'@'%' IDENTIFIED BY PASSWORD '*5...' |
| GRANT SELECT ON `proba`.`xente` TO 'anonimo'@'%' |
-------------------------------------------------------------------
```

1.1.5 Borrar usuarios

Para eliminar usuarios úsase a sentenza DROP USER.

Por exemplo:

```
mysql> DROP USER anonimo;
Query OK, 0 rows affected (0.00 sec)
```

2. Índices

Facer que unha consulta traballe é unha cousa, pero obter unha consulta que traballe o máis rapidamente posible é outra moi diferente. Pódense acelerar as consultas de dous xeitos basicamente, unha delas é afinando o servidor para que responda o mellor posible, e a outra é facendo uso de índices.

Para resolver unha consulta sen un índice, un SXBDs ten que ler todos os rexistros das táboas de forma secuencial para atopar os rexistros relevantes. Isto é algo que se debe evitar. En particular, debemos evitar os escaneos completos de táboas por razóns de sobrecarga de CPU, de sobrecarga de disco e concorrencia de acceso (xa que normalmente o SXBD cando está lendo os datos dunha táboa, bloquéaa, de tal xeito que ninguén máis pode escribir nela, aínda que se poida ler).

Os índices son usados polo SXBDs para atopar máis rapidamente os rexistros que teñan un determinado valor nalgunha das súas columnas. De xeito simple, un índice permítelle ao SXBDs determinar se un valor dado coincide con calquera fila nunha táboa. Cando se indexa unha columna en particular, créase outra estrutura de datos (un índice) que se usa para almacenar información extra acerca dos valores na columna indexada. Os valores indexados son chamados frecuentemente claves.

A mellora na obtención de información utilizando índices é máis significativa cando as táboas teñen gran cantidade de datos.

2.1 Indices en MySQL

2.1.1 Creación de índices

Existen catro tipos de índices que se poden utilizar en MySQL: de clave primaria, únicos, de texto completo, e ordinarios.

Índices de clave primaria

Unha clave primaria é un índice sobre un ou máis campos onde cada valor é único e ningún dos valores é NULL.

Para crear un índice de clave primaria existen basicamente dúas opcións:

- Crear o índice de clave primaria no momento de crear a táboa. Utilizando a opción PRIMARY KEY ao final da definición dos campos, cunha lista dos campos que serán parte do índice.

```
CREATE TABLE nombreTabla(campo1 tipoDato,
[campo2...,] PRIMARY KEY (campo1 [,campo2...]) );
```

As claves primarias non poden conter valores nulos polo que debemos poñer NOT NULL para un campo cando este vaia a formar parte dunha clave primaria.

- Crear unha clave primaria nunha táboa existente utilizando o comando ALTER TABLE:

```
ALTER TABLE nombreTabla ADD PRIMARY KEY(campo1 [,campo2...]);
```

Por exemplo, supoñendo que xa se ten no sistema unha táboa que foi creada do seguinte xeito (sen clave primaria, e co campo *ide* aceptando valores NUL):

```
CREATE TABLE usuarios(ide int, nome varchar(50), apelidos varchar(70));
```

Pódese crear unha clave primaria sobre o campo *ide* con esta sentenza:

```
ALTER TABLE usuarios MODIFY ide INT NOT NULL, ADD PRIMARY KEY(ide);
```

As claves primarias poden constar de máis dun campo xa que hai algunhas veces nas que un só campo non pode identificar de xeito único a un rexistro.

Índices ordinarios

Un índice que non é primario permite valores duplicados (a menos que os campos sexan especificados como UNIQUE).

Para crear un índice ordinario existen basicamente tres opcións:

- Pódese crear un índice ordinario ao mesmo tempo que creamos a táboa co uso da opción INDEX.

```
CREATE TABLE nombreTabla(campo1 tipoDato, campo2 tipoDato,..
INDEX [nombreIndice] (campo1 [,campo2...]));
```

- De igual xeito, pódese crear o índice co uso da sentenza ALTER TABLE se é que a táboa xa existe.

```
ALTER TABLE nombreTabla ADD INDEX [nombreIndice] (campo1 [,campo2...]);
```

- Tamén é posible usar a sentenza CREATE INDEX para crear un índice nunha táboa existente.

```
CREATE INDEX nombreIndice ON nombreTabla(campo1 [,campo2...]);
```

Por exemplo, para crear un índice na columna apelidos da táboa *usuarios* cunha sentenza ALTER TABLE:

```
ALTER TABLE usuarios ADD INDEX idx_apelidos (apelidos);
```

Ou ben, cunha sentenza CREATE INDEX:

```
CREATE INDEX idx_apelidos ON usuarios(apelidos);
```

Índices de texto completo

Os índices de texto completo son do tipo FULLTEXT, **úsanse en táboas do tipo MyISAM** e poden conter un ou máis campos do tipo CHAR, VARCHAR e TEXT. Un índice de texto completo está deseñado para facilitar e optimizar a procura de palabras clave en táboas que teñen grandes cantidades de información en campos de texto.

Para crear un índice de texto completo existen basicamente tres opcións:

- Crear o índice ao momento de crear a táboa.

```
CREATE TABLE nombreTabla( campo1 TIPO, campo2 TIPO,
FULLTEXT [nombreIndice] (campo1 [campo2,...]) );
```

- Crear o índice unha vez que foi creada a táboa.

```
ALTER TABLE nombreTabla ADD FULTEXT [nombreIndice] (campo1 [,campo2,...]);
```

- Usar a seguinte sentenza para crear un índice cando a táboa xa existe.

```
CREATE FULLTEXT INDEX nombreIndice ON nombreTabla(campo1 [,campo2,...]);
```

Por exemplo, para a táboa *usuarios* poderíase crear un índice FULLTEXT na columna nome, na columna apelidos, ou ben, un índice que ocupe ambos campos. A continuación móstranse os tres casos.

```
CREATE FULLTEXT INDEX idx_nome ON usuarios(nome);
CREATE FULLTEXT INDEX idx_apelidos ON usuarios(apelidos);
CREATE FULLTEXT INDEX idx_nomee_apelidos ON usuarios(nome,apelidos);
```

Cando se teñen grandes cantidades de datos, é moito máis rápido cargar os datos nunha táboa que non ten índices de texto completo e despois crear os índices necesarios, xa que a carga de datos nunha táboa que xa ten índices deste tipo é un proceso lento.

Índices únicos

Os índices únicos son como os índices ordinarios, agás que os valores duplicados non son permitidos.

Para crear un índice UNIQUE téñense basicamente tres opcións:

- Crear un índice único cando a táboa é creada co uso da opción UNIQUE.

```
CREATE TABLE nombreTabla(campo1 tipoDato, campo2 tipoDato,..
UNIQUE [nombreIndice] (campo1 [,campo2...]));
```

- Se a táboa xa existe, pódese usar a sentenza ALTER TABLE.

```
ALTER TABLE nombreTabla ADD UNIQUE [nombreIndice] (campo1, campo2) ...
```

- Usar a sentenza CREATE INDEX para crear un índice único nunha táboa existente.

```
CREATE UNIQUE INDEX nombreIndice ON nombreTabla(campo1 [,campo2...]);
```

Por exemplo, para a táboa *usuarios* poderíase crear un índice UNIQUE na columna nome, e un índice UNIQUE na columna apelidos.

```
ALTER TABLE usuarios ADD UNIQUE idx_nome (nome);
CREATE UNIQUE INDEX idx_apelidos ON usuarios(apelidos);
```

No primeiro caso, faise uso do comando ALTER TABLE, no segundo caso créase o índice coa sentenza CREATE INDEX

Índices compostos

Os índices compostos son aqueles que están baseados en múltiples columnas. MySQL unicamente usa un índice por táboa cando está procesando unha consulta. Isto significa que se existen varias columnas que frecuentemente aparecen xuntas nunha cláusula WHERE, pódense acelerar estas consultas ao crear un índice composto.

Por exemplo, se hai un índice composto por tres columnas (col1, col2, col3), teríase capacidade de procura en (col1), (col1, col2) e (col1, col2, col3).

MySQL non pode usar un índice parcial cando as columnas non forman un prefixo máis á esquerda do índice. Supóñase que se teñen unhas sentenzas SELECT como estas:

```
mysql> SELECT * FROM algunaTabla WHERE col1=valor1;
mysql> SELECT * FROM algunaTabla WHERE col2=valor2;
mysql> SELECT * FROM algunaTabla WHERE col2=valor2 AND col3=valor3;
```

Se está definido un índice con (col1, col2, col3), só a primeira destas consultas usará o índice. A segunda e a terceira involucran ás columnas no índice, pero (col2) e (col2, col3) non son os prefixos máis á esquerda de (col1, col2, col3).

Por exemplo, se frecuentemente se fan consultas na táboa *usuarios* baseadas tanto no nome como nos apelidos, poderíase facer un índice composto das columnas nome e apelidos.

```
ALTER TABLE usuarios ADD INDEX idx_nome(nome, apelidos);
```

Debido á forma en que MySQL constrúe os índices compostos, pódese usar o índice *idx_nome* para resolver consultas baseadas só no nome, ou no nome e os apelidos, con todo e non usará o índice nunha consulta que faga referencia unicamente á columna apelidos.

Por exemplo, das seguintes tres consultas, só as dúas primeiras farían uso do índice *idx_nome*.

```
SELECT * FROM usuarios WHERE nome='Eduardo';
SELECT * FROM usuarios WHERE nome='Eduardo' AND apelidos='Zarate M';
SELECT * FROM usuarios WHERE apelidos='Zarate M';
```

A idea é que os índices compostos poden usarse frecuentemente para acelerar algunhas consultas complexas, pero necesítase entender as súas limitacións e débese executar algún tipo de proba no canto de asumir que estes índices sempre nos van a axudar.

Índices de parte de campos

Nas columnas de tipo CHAR e VARCHAR pódese crear un índice que non use o campo por completo.

Retomando o exemplo anterior da táboa *usuarios*. Malia que o nome dunha persoa pode ser de ata 50 carácteres, é moi común que os nomes das persoas sexan diferentes nos primeiros 10 carácteres. Ao usar un índice de 10 carácteres en lugar de 50, o índice será máis pequeno, e permitirá que as consultas INSERT e UPDATE sexan máis rápidas, á vez que non se afecta a velocidade das consultas SELECT.

Para crear un índice como parte dun campo, só se ten que especificar o tamaño entre parénteses despois do nome da columna. Por exemplo, o índice *idx_nome* puido ser creado tamén do seguinte xeito:

```
ALTER TABLE usuarios ADD INDEX idx_nome(nome(10), apelidos(20));
```

2.1.2 Eliminar ou cambiar un índice

Algunhas veces tense a necesidade de cambiar ou eliminar un índice. Cando se fai algún cambio no índice, necesítase eliminar primeiro o índice e entón reconstruílo coa nova definición.

Para eliminar un índice de clave primaria pódese usar a seguinte sintaxe:

```
ALTER TABLE nombreTabla DROP PRIMARY KEY;
```

Para eliminar un índice ordinario, único, ou de texto completo, necesítase especificar o nome do índice e usar esta sintaxe:

```
ALTER TABLE nombreTabla DROP INDEX nombreIndice;
```

Tamén é válida estoutra sintaxe:

```
DROP INDEX nombreIndice ON nombreTabla;
```

Se non se está seguro de cal é o nome do índice que se desexa eliminar, pódese facer uso de sentenza SHOW KEYS:

```
SHOW KEYS FROM nombreTabla;
```

Este é un exemplo:

```
CREATE TABLE usuarios
(
ide INT NOT,
nome VARCHAR(50) NOT NULL,
apelidos VARCHAR(70) NOT NULL,
PRIMARY KEY (ide),
INDEX (nome, apelidos)
);
```

Para ver os índices que existen nesta táboa:

```
mysql> SHOW KEYS FROM usuarios;
---------- ------------ ---------- -------------- -------------
| Table | Non_unique | Key_name | Seq_in_index | Column_name |
---------- ------------ ---------- -------------- ------------- .
| usuarios | 0 | PRIMARY | 1 | ide | .
| usuarios | 1 | nome | 1 | nome | .
| usuarios | 1 | nome | 2 | apelidos |
---------- ------------ ---------- -------------- -------------

3 rows in set (0.00 sec)
```

A terceira columna é a que proporciona os nomes dos índices. Pódese observar que ao non especificar un nome ao índice ordinario en (nome, apelidos), asignóuselle o nome da primeira columna que forma o índice.

Para eliminar os dous índices que existen nesta táboa:

```
mysql> ALTER TABLE usuarios DROP PRIMARY KEY;
Query OK, 0 rows affected (0.01 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE usuarios DROP INDEX nome;
Query OK, 0 rows affected (0.00 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Para rematar, pódese verificar que estes índices xa non existen:

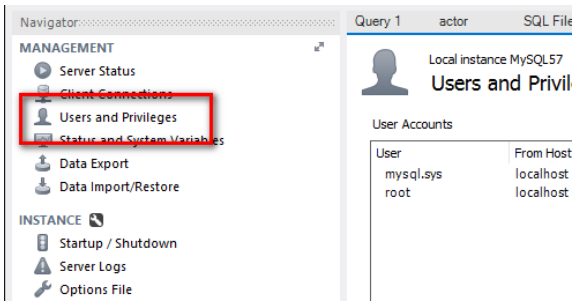
```
mysql> SHOW KEYS FROM usuarios;
Empty set (0.00 sec)
```

3. Xestión de usuarios e índices con MySQL Workbench

Mediante a ferramenta gráfica de administración *MySQL Workbench* tamén é posible xestionar tanto os usuarios da base de datos como os índices, tal e como se fixo nos apartados anteriores pero mediante unha interface gráfica.

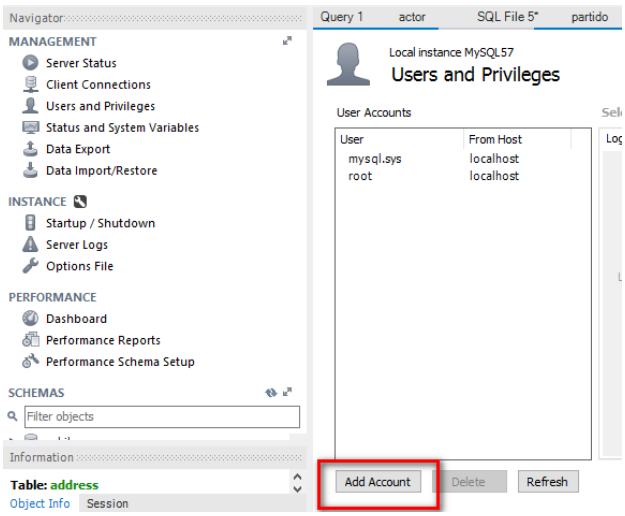
3.1 Xestión de usuarios con MySQL Workbench

A xestión de contas de usuarios realízase desde a opción *Users and Privileges* do menú esquerdo.

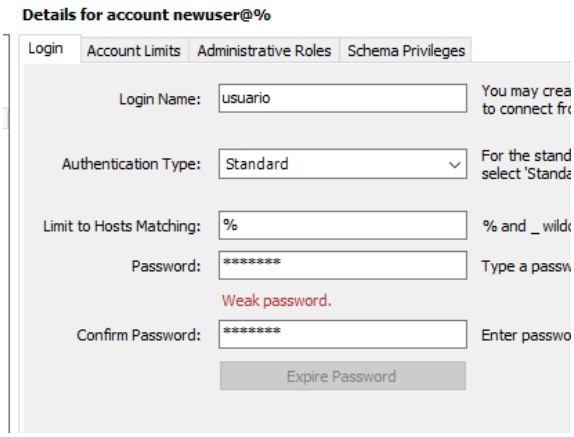


Crear usuarios

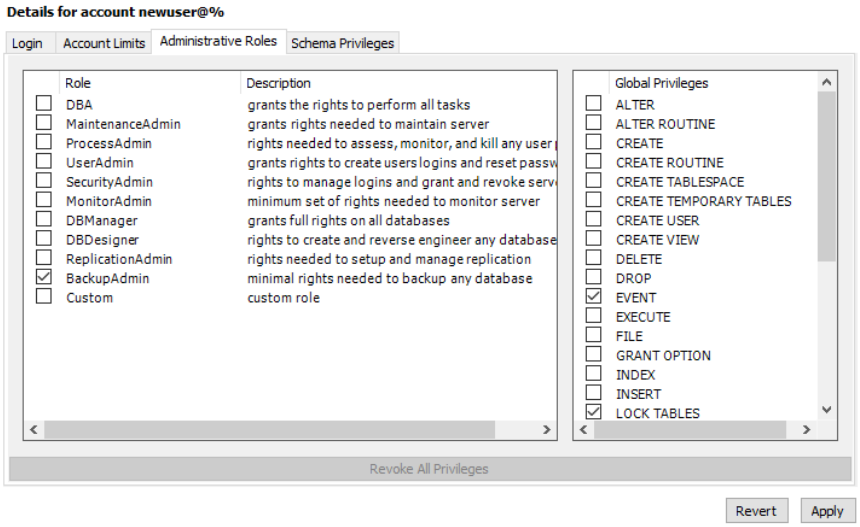
Para crear un novo usuario premerase na opción *Add Account*:



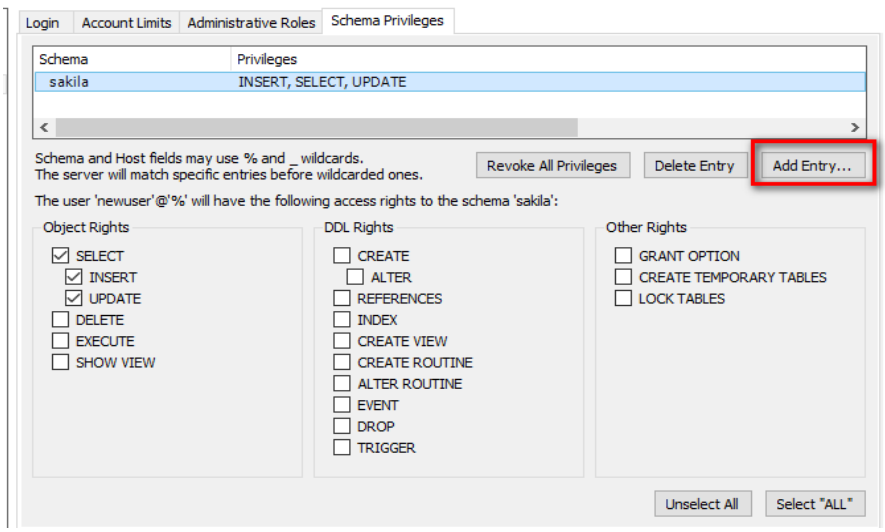
Na pestana *Login* introduciranse os datos do usuario:



Na pestana *Administrative Roles* poderanse indicar os privilexios globais do usuario. Para elo pódese seleccionar un rol xa establecido ou crear un novo e despois modificar individualmente os privilexios:

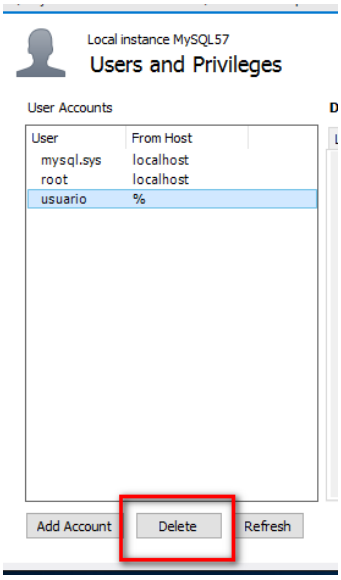


Na pestana *Schema Privileges* pódense seleccionar os privilexios que terá o usuario sobre un esquema da base de datos. O esquema pódese seleccionar na opción *Add Entry*...



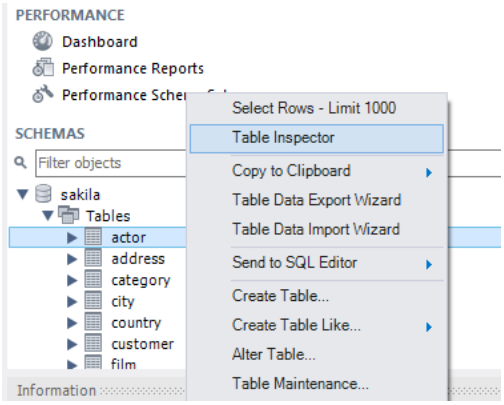
Borrar usuarios

Para borrar un usuario seleccionarase o usuario e premerase na opción *Delete*:

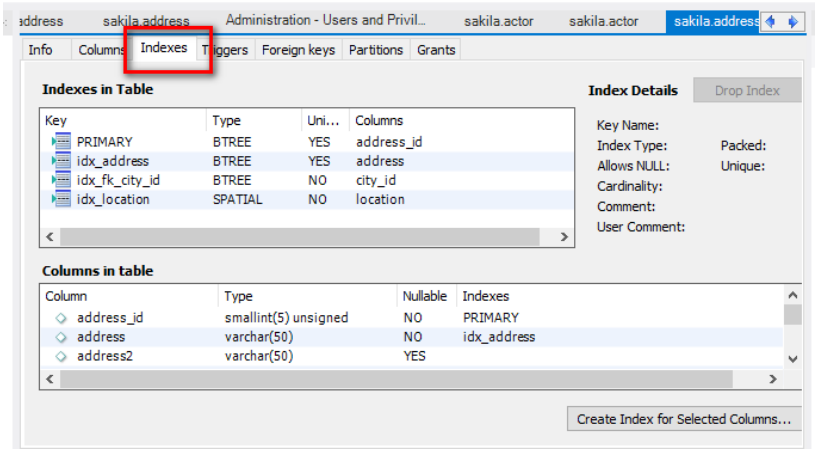


3.2 Xestión de índices con MySQL Workbench

A xestión de índices realízase desde a opción *Table Inspector* do menú contextual das táboas dunha base de datos:

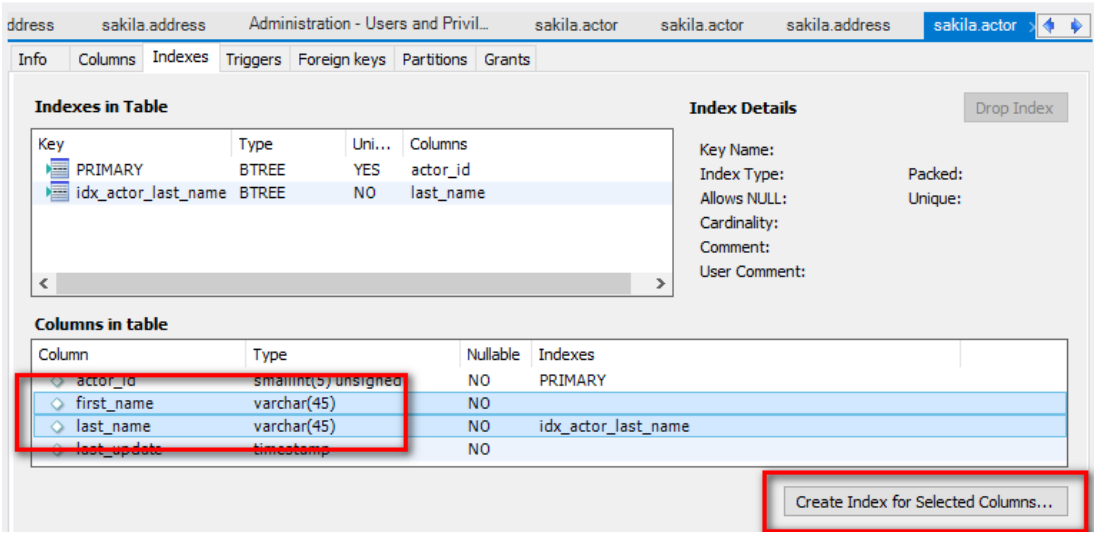


Na pestana *Indexes* pódense crear índices novos, así como borrar os índices existentes:

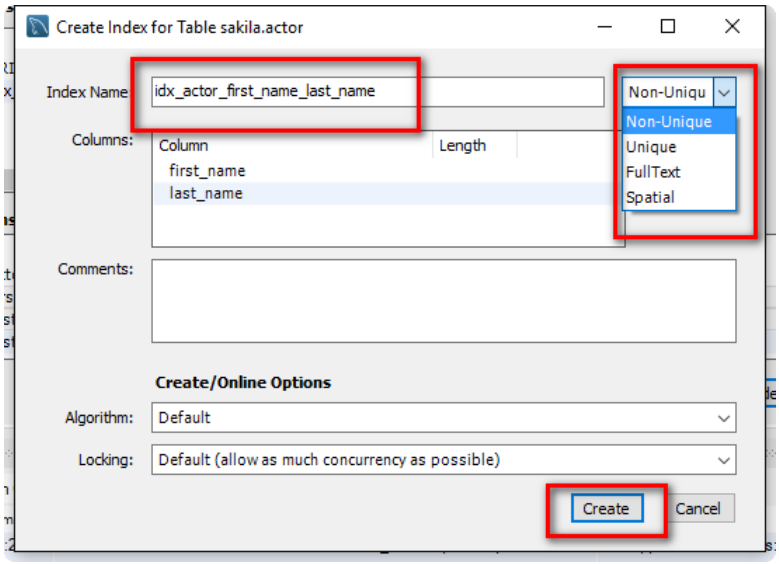


Crear índices

Para crear un novo índice, primeiro débense seleccionar as columnas sobre as cales se quere crear o índice e logo premer en *Create Index for Selected Columns*:

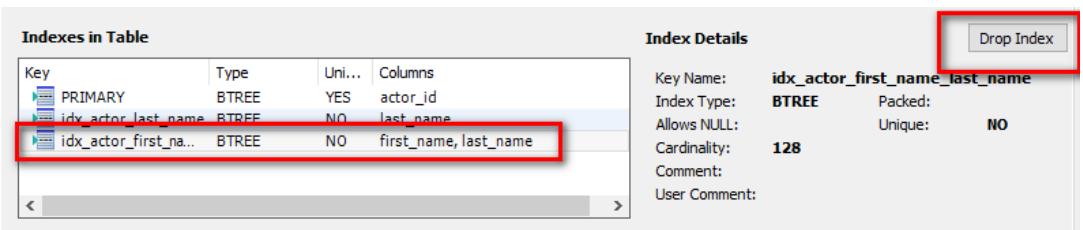


A continuación cubriranse os parámetros desexados e premerase en *Create* para finalmente crear o índice:



Borrar índices

Para borrar un índice xe existente, tan solo haberá que seleccionalo e premer en *Drop Index:*



4. Procesamento de consultas

As regras que usa MySQL para decidir como obter os datos dunha consulta poden chegar a ser difíciles de entender se ditas consultas son algo complexas. Afortunadamente hai unhas poucas regras e un comando que permiten ter un mellor entendemento de que é o que está facendo MySQL. Primeiro vanse comentar as regras:

- MySQL non usará un índice se decide que será moito máis rápido escanear completamente unha táboa. En xeral, se un índice lle fai saber a MySQL que indexará aproximadamente o 30 por cento das filas dunha táboa, MySQL abandona o índice e simplemente executa un escaneo completo da táboa.
- Se múltiples índices poden ser usados para satisfacer unha consulta, MySQL usará o que sexa máis restritivo, isto é, co que se obteñan o menor número de filas.
- Se as columnas que se están a seleccionar forman parte dun índice, MySQL pode ler todos os datos que se necesitan desde o índice e non tocar nunca a táboa.
- Cando se usan varias táboas nunha consulta, MySQL lerá primeiro os datos desde a táboa que regrese o menor número de filas. A orde no que se especifican as táboas pode non ser o mesmo que use MySQL. Isto afecta tamén á orde no que son devoltos finalmente os rexistros, así que hai que asegurarse de usar unha cláusula ORDER BY cando se necesita que os rexistros teñan unha orde en particular.

Habendo dito isto, é importante ter en conta que algunhas das decisións que toma MySQL están baseadas en suposicións, e poida que MySQL ocasionalmente faga algunha que sexa incorrecta. Para entender que é o que está facendo MySQL para procesar unha consulta, pódese usar o comando EXPLAIN.

EXPLAIN mostra (explica) como son procesadas as sentenzas SELECT por MySQL, como se usan os índices, e como se unen as táboas. Utilizar EXPLAIN pode axudar a seleccionar mellores índices e a escribir as consultas máis optimizadas. O único que hai que facer é agregar a palabra EXPLAIN ao comezo da consulta para que MySQL diga como a está executando. No canto de executar a consulta, MySQL reportará a lista de índices que se poderían usar na consulta e o que coñece acerca deles.

```
EXPLAIN SELECT nome, apelidos FROM usuarios WHERE ide = 1;

id        select_type    table   type    possible_keys  key     key_len        ref
   rows    Extra
1  SIMPLE  up      ALL     user_id_idx    NULL    NULL    NULL    13

1 row in set (0.00 sec)
```

A continuación explicarase o significado de cada unha das columnas.

- **Table**: Informa da táboa que se está a explicar.
- **Type**: O tipo de unión que se está usando. Desde a mellor ata a peor, os tipos de unións son system, const, eq_ref, ref, range, index, e ALL.
  - **System**: Táboa cunha única fila.
  - **Const**: Na táboa coincide unha única fila cos criterios indicados. Como só hai unha fila, o optimizador toma este valor como constante, por este motivo este tipo de táboas son moi rápidas.
  - **Eq_ref**: Unha fila da táboa 1 será lida por cada combinación de filas da táboa 2. Este tipo é usado cando todas as partes dun índice se usan na consulta e o índice é UNIQUE ou PRIMARY KEY.
  - **Ref**: Todas as filas con valores no índice que coincidan serán lidas desde esta táboa por cada combinación de filas das táboas previas. Similar a eq_ref, pero can-

do se usa só un prefixo máis á esquerda da clave ou se a clave non é UNIQUE ou PRIMARY KEY. Se a clave que é usada coincide só con poucas filas, esta unión é boa.

  - **Range**: Só serán recuperadas as filas que estean nun rango dado, usando un índice para seleccionar as filas. A columna key indica cal índice é usado, e o valor key_len contén a parte máis grande da clave que foi usada. A columna ref será NULL para este tipo.
  - **Index**: Escaneo completo da táboa para cada combinación de filas das táboas previas, revisando unicamente o índice.
  - **ALL**: Escaneo completo da táboa para cada combinación de filas. É o peor caso xa que revisará todas as filas para cada combinación.

- **Possible_keys**: Posibles índices que utilizará a consulta.
- **Key**: Índice utilizado para executar a consulta. Se indica o valor NULL, non se escolleu ningún índice.
- **Key_len**: Canto máis pequeno sexa este valor, máis rápida será a consulta, pois indica a lonxitude do índice usado.
- **Ref**: As columnas do índice que se está usando, ou unha constante se esta é posible.
- **Rows**: Número de filas que MySQL debe analizar para devolver os datos solicitados.
- **Extra**: Información complementaria sobre como MySQL executará a consulta. Os posibles valores neste campo poden ser:
  - **Distinct**: MySQL atopou unha fila coincidente cos filtros indicados e non necesita seguir analizando.
  - **Not exists**: MySQL foi capaz de facer unha optimización LEFT JOIN sobre a consulta e non examinará máis filas na táboa para a combinación de filas previa despois de que atope unha fila que coincida co criterio LEFT JOIN.
  - **Range checked for each record**: Non se atopou un índice válido. Para cada combinación de filas farase un chequeo para determinar que índice utilizar e en caso de atopar algún válido, utilizarao.
  - **Using filesort**: Este valor indica que MySQL necesita facer un paso extra para atopar a forma de ordenar as filas. Este tipo de consultas debe ser optimizada.
  - **Using index**: Recupera a información solicitada utilizando unicamente a información do índice. Isto sucede cando todas as columnas requiridas forman parte do índice.
  - **Using temporary**: Para resolver esta consulta, MySQL creará unha táboa temporal. Un dos casos típicos nos que devolve este valor é cando usamos un ORDER BY sobre un conxunto de columnas diferentes ás indicadas na cláusula GROUP BY. Este tipo de consultas debe ser optimizada.
  - **Where used**: Usarase unha cláusula WHERE para determinar que filas serán comparadas con outra táboa. Se non se desexa regresar todas as filas desde a táboa, e o join é do tipo ALL ou index, é moi probable que escribamos algo mal na consulta.

Para obter consultas que se executen o máis rápido posible, hai que ser coidadosos cando se vexa información extra do tipo **Using filesort** ou **Using temporary**.

Pódese obter unha boa indicación de o boa que é unha consulta ao multiplicar todos os valores da columna rows na saída de EXPLAIN. Isto di aproximadamente cantas filas debe examinar MySQL para executar unha consulta. Do que se trata é de ir mellorando unha consulta progresivamente usando a información proporcionada por EXPLAIN.

4.1 Como evitar escaneos completos de táboas

A saída de EXPLAIN mostrará ALL na columna type cando MySQL fai un escaneo de táboa para resolver unha consulta. Isto sucede usualmente baixo as seguinte condicións:

- A táboa é tan pequena que é máis rápido facer o escaneo da táboa que buscar un índice. Este é o caso común para táboas con menos de 10 filas.
- Non hai restriccións usables nas cláusulas ON ou WHERE para as columnas indexadas.
- Estanse comparando columnas indexadas con valores constantes e MySQL calculou que as constantes cobren unha gran parte da táboa e que o escaneo completo sería máis rápido.
- Estase usando unha clave con baixa cardinalidade (moitas filas que coinciden co valor clave). Neste caso, MySQL asume que ao usar o índice probablemente faranse unha gran cantidade de procuras adicionais de claves e que un escaneo da táboa será máis rápido.

Para táboas pequenas, un escaneo da táboa é frecuentemente apropiado. Para táboas moi grandes, pódense intentar as seguintes técnicas para evitar que o optimizador de consultas de MySQL escolla incorrectamente un escaneo completo.

- Usar ANALIZE TABLE para actualizar a distribución de claves para a táboa escaneada.
- Usar FORCE INDEX na táboa escaneada para indicarlle a MySQL que use o índice dado.

Por exemplo:

```
SELECT * FROM táboa1, táboa2 FORCE INDEX (indiceParaColumna)
WHERE táboa1.nombreColumna=táboa2.nombreColumna;
```

4.2 Optimizando consultas SELECT

En xeral, cando se desexa facer que unha consulta SELECT ... WHERE se execute máis rápido, o primeiro que se debe mirar é se se pode agregar un índice. Todas as referencias entre táboas diferentes deben usualmente ser feitas con índices. Por suposto, se debe usar unha sentenza EXPLAIN para determinar cales índices están sendo usados para resolver a consulta.

4.3 Optimizando sentenzas INSERT

O tempo que lle toma a MySQL inserir un rexistro está determinado polos seguintes factores, onde os números indican unicamente valores aproximados:

- Establecer a conexión: (3)
- Enviar a consulta ao servidor: (2)
- Analizar a consulta: (2)
- Inserir o rexistro: (1 x tamaño do rexistro)
- Inserir índices: (1 x número de índices)
- Pechar: (1)

Isto non toma en consideración a sobrecarga inicial de abrir as táboas, o cal é feito unha vez por cada consulta en execución de xeito concorrente.

O tamaño da táboa fai máis lenta a inserción dos índices.

Polo tanto, pódense usar os seguintes métodos para lograr que os INSERTs se executen

máis rápido:

- No caso de inserir moitas filas desde o mesmo cliente ao mesmo tempo, usarase unha sentenza INSERT con múltiples listas de valores para inserir varias filas á vez. Isto é moito máis rápido que usar varias sentenzas INSERT de xeito separado. Por exemplo, a seguinte consulta:

```
INSERT INTO nombreTabla VALUES(rexistro1),(rexistro2),... (registroN);
```

  é moito máis rápida que esta alternativa:

```
INSERT INTO nombreTabla VALUES(rexistro1);
INSERT INTO nombreTabla VALUES(rexistro2);
...
INSERT INTO nombreTabla VALUES(registroN);
```

- Cando se está a inserir unha gran cantidade de filas desde diferentes clientes, pódese obter unha maior velocidade ao usar a sentenza INSERT DELAYED.
- Cando se está a cargar datos nunha táboa a partir dun arquivo de texto, o mellor é usar a sentenza LOADE DATA INFILE. Isto é ata 20 veces máis rápido que usar unha gran cantidade de sentenzas INSERT.
- É posible facer que LOAD DATA INFILE se execute aínda máis rápido cando a táboa ten moitos índices. O procedemento a seguir é:
  - Deshabilitar os índices co uso da sentenza ALTER TABLE nombreTabla DISABLE KEYS;
  - Inserir os datos na táboa con LOAD DATA INFILE. Neste momento non se actualizará ningún índice e polo tanto será moi rápido cargar os datos.
  - Iniciar a creación dos índices necesarios co uso de ALTER TABLE nombreTabla ENABLE KEYS. Isto creará os índices en memoria antes de escribilos en disco, o cal é moito máis rápido xa que se evita unha gran cantidade de lecturas e escrituras en disco.
- É un feito que non sempre se ten a posibilidade de inserir os datos a partir dun arquivo de texto, con todo, pódense acelerar as operacións INSERT que son feitas con múltiples sentenzas ao bloquear as nosas táboas:

```
LOCK TABLES nombreTabla WRITE;
INSERT INTO nombreTabla VALUES(rexistro1),(rexistro2),(rexistro3);
INSERT INTO nombreTabla VALUES(rexistro4),(rexistro5),(rexistro6);
...
INSERT INTO nombreTabla VALUES(registroN);
UNLOCK TABLES;
```

  O uso explícito das sentenzas de bloqueo (LOCK) non é necesario cando se poden inserir todos as filas cunha soa sentenza INSERT. Para as táboas transaccionais, débese usar BEGIN/COMMIT no canto de LOCK TABLE para obter o mesmo resultado.

  As sentenzas INSERT, UPDATE e DELETE son moi rápidas en MySQL, pero obterase unha mellor eficiencia ao agregar bloqueo cando se executen de media máis de 5 inserción ou actualizacións nunha fila.

4.4 Optimizando sentenzas UPDATE

As sentenzas UPDATE son optimizadas de xeito similar ás sentenzas SELECT coa sobrecarga adicional da escritura. Por exemplo, para efectos de optimización, o seguinte código:

```
UPDATE nombreCampo FROM nombreTabla WHERE algunaCondicion
```

É o mesmo que este:

```
SELECT nombreCampo FROM nombreTabla WHERE algunaCondicion
```

É dicir, pódese optimizar unha sentenza UPDATE da mesma forma que a súa equivalente sentenza SELECT.

A velocidade de escritura depende da cantidade de datos que están sendo actualizados e o número de índices que son actualizados, polo tanto débese ter coidado de crear índices que non sexan de verdade útiles, ou ben, facer que os campos da táboa sexan máis grandes do que realmente se necesita.

Tamén, outra forma de obter actualizacións rápidas é atrasar os UPDATEs e entón facer moitas actualizacións nunha fila posteriormente.

Débese notar que para unha táboa MyISAM que usa o formato de rexistro dinámico, o actualizar o rexistro a unha lonxitude total máis grande pode dividir o rexistro. Se isto chega a ocorrer, é moi importante usar o comando OPTIMIZE TABLE ocasionalmente.

4.5 Optimizando sentenzas DELETE

O tempo de eliminar rexistros individuais é exactamente proporcional ao número de índices. Cando se fan eliminacións, cada rexistro necesita ser eliminado desde calquera índice asociado, así como tamén do arquivo principal de datos.

Para eliminar todas as filas dunha táboa, é preferible usar o comando TRUNCATE TABLE en lugar de executar a tradicional sentenza DELETE, xa que se borra toda a táboa nunha soa operación, sen a necesidade de eliminar cada índice e cada rexistro de xeito individual.

Para obter maior velocidade nas táboas MyISAM, tanto para as sentenzas INSERT, como para as sentenzas DELETE, pódese facer máis grande a caché de claves ao incrementar a variable de sistema *key_buffer_size*.