Manual de Supervivencia del Administrador de MySQL

En gnu/Linux, por supuesto

6 de abril de 2007

Autor: Miguel Jaque Barbero

Índice de Contenidos

Presentación.....5
1.1. Contenido.....5
1.2. Licencia.....5
1.3. Responsabilidad.....6
1.4. Acerca del Autor.....6
Capítulo 1. Introducción.....7
1.1. La Historia de MySQL.....7
1.2. Características.....8
1.3. Instalación.....8
1.4. Configuración Básica.....9
Capítulo 2. Conceptos de Bases de Datos.....13
2.1. Vocabulario.....13
2.2. La Arquitectura de MySQL.....13
2.3. Concurrencia, Integridad y Rendimiento.....14
2.4. Transacciones.....17
2.5. Bloqueos.....19
2.6. Motores de Almacenamiento.....19
Capítulo 3. Benchmarking.....23
3.1. La Necesidad del Benchmarking.....23
3.2. Estrategias.....23
3.3. Herramientas.....24
Capítulo 4. Administración.....27
4.1. Comandos.....27
4.2. Sistemas de Administración Gráficos.....29
Capítulo 5. Seguridad.....37
5.1. Conceptos de Seguridad.....37
5.2. Sentencias GRANT y REVOKE.....40
5.3. Logs.....42
5.4. Seguridad en el Sistema Operativo.....43
5.5. Seguridad Externa.....44
5.6. Encriptación de Datos.....47
5.7. MySQL con chroot.....49
5.8. Copias de Seguridad.....50
Capítulo 6. Optimización.....55
6.1. Índices.....55
6.2. Consultas.....59
6.3. Rendimiento del Servidor.....65
Capítulo 7. Replicación.....71
7.1. Concepto.....71
7.2. Configuración.....72
7.3. Arquitecturas.....73
Capítulo 8. Bibliografía.....77

Presentación

1.1. Contenido

Este documento no es un manual completo de MySQL. Tampoco es una guía para programadores SQL ni, mucho menos, para quienes deseen colaborar en el desarrollo de MySQL. Simplemente es una Guía Rápida de Referencia para quienes tengan la suerte de administrar un Servidor de Bases de Datos MySQL.

Naturalmente, este documento contiene errores (y supongo que muchos). Pero puedes ayudarme a mejorarlo enviándome un correo con las erratas, pifias y sugerencias (mjaque@ilkebenson.com). Te lo agradezco de antemano.

Vamos a lo técnico.

Este documento está basado en MySQL 5. Para todos los puntos, salvo el apartado de Instalación en Windows, utilizaré un sistema operativo gnu/Linux. En concreto, Debian 3.1. Tendrás que adaptar lo que hay a la distribución o el sistema operativo que utilices.

1.2. Licencia

1.3. Responsabilidad

No se asume ninguna responsabilidad por los contenidos de este documento. El lector asume el riesgo derivado del uso de los conceptos, ejemplos y cualquier otro contenido. Al tratarse de una nueva edición, este documento puede contener errores e imprecisiones.

1.4. Acerca del Autor

Miguel Jaque Barbero nació en Barcelona en 1968. Es Ingeniero Superior de Telecomunicación por la Universidad Politécnica de Madrid y Máster en Administración de Empresas por el Instituto de Empresa de Madrid.

Ha desarrollado toda su carrera profesional en el sector de la ingeniería de software y, desde 1999 centrado exclusivamente en tecnologías de software libre a través de Ilke Benson (www.ilkebenson.com).

Su actividad profesional se centra desde entonces en el desarrollo de proyectos, formación y consultoría, utilizando exclusivamente estas tecnologías.

Para contactar con el autor: mjaque@ilkebenson.com

Capítulo 1. Introducción

1.1. La Historia de MySQL

Hoy, MySQL es uno de los gestores de bases de datos relacionales (SGBDR) más utilizado en el mundo, con más de 10 millones de instalaciones.

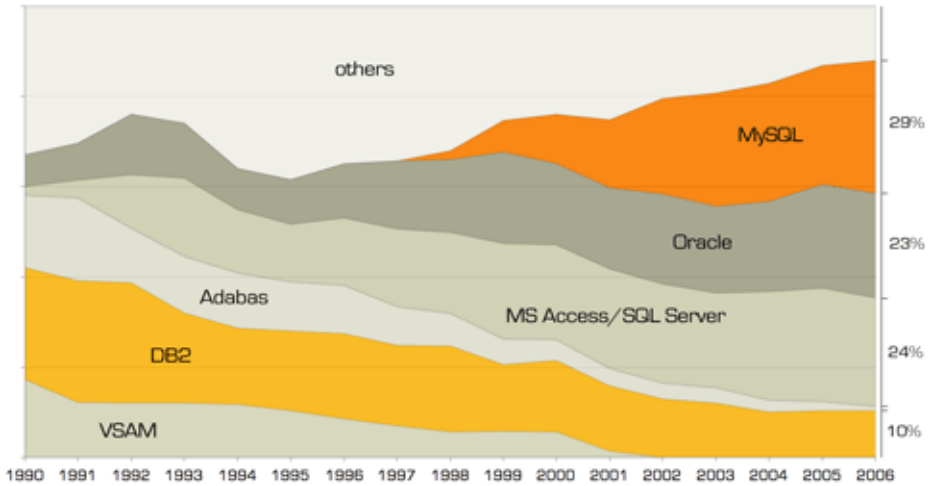
MySQL tuvo su origen en en el intento de mejorar la rapidez y flexibilidad de otra base de datos: mSQL.

mSQL era (y es) propiedad de Hughes Technologies, y no se distribuye como software libre. Sin embargo, cubría bastante bien el hueco de mercado entre base de datos pequeñas y con pocas características relacionales (como Access o DBF) y las bases de datos relacionales profesionales (como Oracle e Informix). MySQL heredó ese posicionamiento.

En 1995, dos suecos y un finés (Michael Widenius, David Axmark y Allan Larsson) fundan en Suecia la empresa MySQL AB para aprovechar comercialmente este desarrollo.

El origen del nombre no está muy claro. Por una parte, gran parte del código original tenía el prefijo “my” (mio en inglés) como una forma de distinguirlo del código de mSQL. Pero también es cierto que uno de lo cofundadores de MySQL (Monty Widenius) tiene una hija llamada “My”.

La primera versión de MySQL vio la luz en mayo de 1995 y ya desde entonces se distribuyó bajo los términos de la licencia GPL. Hoy, se encuentra en la versión 5 y tiene una cuota de mercado (según JoinVision – Julio de 2006) superior al de cualquier otra base de datos.



Además, MySQL es una pieza fundamental en las arquitecturas LAMP de servidores de aplicaciones bajo gnuLinux.

Actualmente, MySQL es propiedad de la empresa Sueca MySQL AB. "Es propiedad" significa que esta empresa es dueña del copyright de la mayoría del código, quien lo distribuye bajo licencia GPL.

El negocio de MySQL AB, como el de otras muchas compañías de software libre, se basa en una combinación de licencia dual y servicios asociados. Así, mientras MySQL AB distribuye gratuitamente su versión GPL del gestor de base de datos, también comercializa versiones bajo licencia tradicional y ofrece servicios de soporte técnico, formación y certificación.

El logo de MySQL es un delfín, de nombre "Sakila".

1.2. Características

Estas son las principales características de MySQL:

- Escrita en C/C++.
- **Funciona en múltiples plataformas: gnuLinux y Windows,** pero también en AIX, Amiga, FreeBSD, ...
- Es multithread y puede aprovechar la disponibilidad de multiprocesadores.
- **Es software libre (licencia GPL)** y también está disponible bajo licencia comercial (por ejemplo, para integración).
- **Tiene distintos motores de almacenamiento** (tipos de tablas), algunos con soporte transaccional y otro no.
- **Es muy, muy rápida...** si se utilizan motores de almacenamiento no transaccionales.
- Gran soporte del estándar SQL.
- **Numerosas opciones de conexión** (PHP, ODBC, sockets TCP/IP, JDBC, API de C/C++...)
- Soporte multilenguaje.
- **Fácil de instalar, usar y administrar**.

1.3. Instalación

La instalación de MySQL es, como para casi todos los programas hoy en día, realmente sencilla. Podemos conseguir la versión de MySQL para nuestro sistema operativo en http://dev.mysql.com

1.3.1. En gnuLinux

Si estás utilizando Debian, ni siquiera necesitarás visitar la página de mysql.com. Puedes instalarte MySQL utilizando el sistema apt de Debian.

Para ello, ejecuta el siguiente comando como root:

```
#> apt-get install mysql-server
```

Con este comando instalarás el gestor de base de datos relacionales.

También conviene que instales las herramientas de cliente. Puedes hacerlo con el siguiente comando:

```
#> apt-get install mysql-client
```

Si utilizas otras distribuciones de gnuLinux, podrás descargarte de la página de dev.mysql.com el código en paquete RPM, bien genérico o en versiones específicas para SuSE, Red Hat y Ubuntu. También podrás encontrar allí el código fuente de MySQL.

1.3.2. En Windows

La instalación en Windows también es sencilla. Basta con bajarse MySQL Installation Wizard y ejecutarlo. Las opciones por defecto son válidas para la mayoría de usos de MySQL.

El proceso de instalación es el siguiente (cuidado, puede cambiar en futuras versiones):

1. Descargar MySQL Community Server desde http://dev.mysql.com/downloads/
2. Escoger el ejecutable “Window Essentials (x86)”
3. Aceptar todas las opciones por defecto excepto:
   - Sign Up: Haz lo que quieras. Puedes registrarte o no.
   - Conjunto de Caracteres: Elegir “Best Support for Multilinguism”, que nos permitirá utilizar los caracteres “tipical Spanish” (ñ, á, é, í, ç...) en los nombres de tablas y columnas.
   - Modo de Ejecución: Elegir también “Include Bin Directory in Windows Path" además de la ejecución como Servicio Windows. Así podremos utilizar la consola SQL.

Tras la instalación, revisa que el servicio MySQL esté funcionando.

1.4. Configuración Básica

Al arrancar, MySQL lee el fichero `my.cnf` para establecer distintas variables del sistema.

Normalmente no es necesario modificar este fichero, pero sí conviene saber qué es y para qué se usa. Solo en caso de tener que optimizar el rendimiento de nuestro servidor necesitaremos meternos en este berenjenal.

1.4.1. Situación

El sistemas Debian gnu/Linux, este fichero se encuentra en `/etc/mysql`.

En sistemas Windows, se encuentra en el directorio de instalación, Pero su situación puede modificarse con el programa de configuración.

1.4.2. Variables del Sistema

Cada variable del sistema tiene un valor por defecto. Y, aunque su valor se establece al arrancar el servidor, muchas de ellas pueden modificarse en tiempo de ejecución mediante el comando `SET`.

El comando `SHOW VARIABLES`, muestra el valor de estas.

Algunas de las variables son:

- **auto_increment_increment:** establece el incremento para columnas autoincrementadas. Por defecto tiene el valor 1.
- **auto_increment_offset:** establece el valor inicial para las columnas autoincrementadas.
- **back_log:** tamaño de la cola de espera de conexiones TCP/IP.
- **basedir:** directorio de instalación del servidor.
- **binlog_cache_size:** tamaño de la caché binaria para sentencias SQL de transacciones.
- **character_set_client:** conjunto de caracteres para sentencias de cliente.
- **character_set_database:** conjunto de caracteres por defecto para las bases de datos.
- **character_set_results:** conjunto de caracteres para los conjuntos de resultados.
- **character_set_server:** conjunto de caracteres por defecto para el servidor.
- **connect_timeout:** tiempo de espera en segundos de los paquetes de conexión.
- **data_dir:** directorio de datos del servidor.
- **expire_logs_days:** Número de días para la eliminación automática de logs binarios.
- **ft_boolean_syntax:** operadores lógicos permitidos en las búsquedas de texto completo.
- **ft_max_word_length:** longitud máxima de las palabras indexadas por texto completo.
- **ft_min_word_length:** longitud mínima de las palabras indexadas por texto completo.
- **have_bdb:** indica si el servidor dispone de soporte para motores de almacenamiento BDB.
- **have_archive:** indica si el servidor dispone de soporte para motores de almacenamiento ARCHIVE.
- **have_openssl:** indica si el servidor dispone de soporte para conexiones SSL.
- **init_connect:** comando a ejecutar en las conexiones de clientes (salvo SUPER). También puede ser un comando o un fichero de comandos SQL.

- **init_file:** fichero a ejecutar en el arranque del servidor.
- **innodb_xxx:** variables del motor de almacenamiento InnoDB.
- **interactive_tiemout:** tiempo de expiración de las conexiones interactivas.
- **join_buffer_size:** tamaño del buffer para las operaciones de join que no utilicen índices.
- etc.

Todas las variables están descritas en el Manual de MySQL, en la sección "System Variables".

1.4.3. Otras Variables y Secciones

Las variables anteriores corresponden con la sección `[mysqld]` del fichero `my.cnf` y son las que pueden verse con el comando `SHOW VARIABLES`.

Pero el fichero de configuración incluye otras secciones e incluso otras variables no accesibles por estos métodos.

Entre las variables de la sección `[mysqld]` que no podremos consultar ni acceder mediante SET y SHOW, están:

- **user:** usuario del sistema con el que se ejecutará MySQL.
- **pid-file:** fichero del sistema que almacenará el número del proceso del servidor.
- **socket:** fichero del sistema que almacenará el número del socket TCP/IP del servidor.
- **port:** puerto TCP/IP en el que escuchará el servidor.
- **bind-address:** dirección IP en la que escuchará el el servidor.

Además, el fichero incluye las siguientes secciones de configuración:

- **[client]** – Sección para la configuración de acceso de los clientes.
- **[mysqld_safe]** – Sección para la ejecución del servidor en modo seguro.
- **[mysqldump]** – Configuración para el volcado de bases de datos.
- **[isamchk]** – Configuración para la comprobación de tablas MyISAM.
- **[MYSQL_CLUSTER]** – Configuración de Clusters de MySQL.

Además, el fichero de configuración suele hacer referencia a un directorio de configuración en el que podrían encontrarse configuraciones específicas requeridas por otras aplicaciones.

Capítulo 2. Conceptos de Bases de Datos

2.1. Vocabulario

Solo para aclarar conceptos. El término "Base de Datos" se utiliza para hacer referencia a dos realidades distintas. Se trata de las siguientes:

- **Sistema Gestor de Bases de Datos Relacionales (SGBDR)**: Con este término me refiero al programa encargado de gestionar los datos, procesar las consultas, etc. Hay muchos: PosgreSQL, Oracle, Informix, Adabas, SQL Server y, naturalmente, MySQL.
- **Base de Datos Relacional (BDR):** Este término hace referencia a los datos, a cada uno de los conjuntos de tablas (entidades y relaciones) asociados a un problema. Son creadas por los usuarios quienes les asignan sus nombres (test, banco, usuarios, nóminas, etc).

Normalmente el contexto explicará de qué significado se trata.

2.2. La Arquitectura de MySQL

Podemos imaginar la arquitectura interna de MySQL dividida en tres capas. Se trata de una división lógica, que no coincide necesariamente con la división interna del código, pero nos ayudará a entender los conceptos.

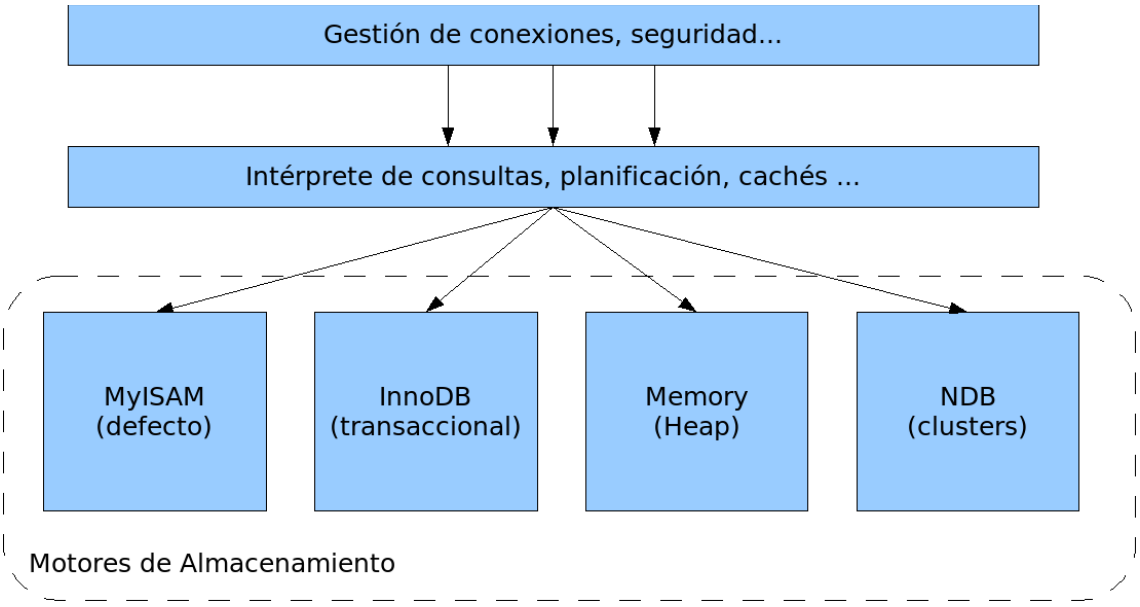
Las tres capas son:

1. **Capa de Conexión**: En la que reside la funcionalidad que conecta MySQL con otros sistemas y lenguajes (APIs, sockets, ODBC, etc.)
2. **Capa de Lógica**: En la que reside la lógica para procesar consultas SQL (parseo de sentencias, planificación, ejecución, cachés, etc.)
3. **Capa de Almacenamiento**: En la que reside la lógica para almacenar y acceder a los datos de las tablas.

Una característica de MySQL es que puede utilizar distintos *motores de almacenamiento*[1]. Es decir, que la capa de almacenamiento dispone de varios subniveles, uno por cada tipo de motor soportado.

La siguiente figura nos aclarará un poco más las cosas:

1 Antes, los motores de almacenamiento se denominaban "tablas". Pero ahora se cambiado la terminología para dar más énfasis a su lógica de almacenamiento y evitar considerarlos como elementos pasivos.



Podemos ver que MySQL soporta cuatro motores de almacenamiento básicos. Hay que destacar que una base de datos puede almacenar sus datos utilizando una combinación de distintos motores. Es decir, parte de los datos pueden estar en una tabla MyISAM y otra parte en una tabla InnoDB.

Enseguida veremos las diferencias entre los distintos motores, pero antes tenemos que repasar algunos conceptos básicos.

2.3. Concurrencia, Integridad y Rendimiento

El dilema de todo almacenamiento de datos, ya sea una base de datos relacional o un simple fichero de disco, es la elección entre concurrencia e integridad.

Por un lado, nos gustaría ofrecer la máxima concurrencia, permitiendo que múltiples usuarios accedieran a los datos simultáneamente, ya sea para leerlos, escribirlos o modificarlos sin sufrir penalizaciones de tiempo.

Pero también somos conscientes que hay que garantizar que los datos leídos por cada usuario sean íntegros. Es decir, que no hayan sido corrompidos en el proceso de acceso.

Ambos conceptos son incompatibles entre sí.

Esto no le ocurre sólo a MySQL, como veremos, es un problema general, de pura lógica.

Si permitimos la máxima concurrencia, dos usuarios pueden intentar escribir el mismo dato al mismo tiempo y, como consecuencia, tendríamos un galimatías incongruente. Por ejemplo, si se tratara de un buzón de correo electrónico, podríamos acabar con una mezcla de dos mensajes recibidos por distintas vías.

Al mismo tiempo, ¿qué ocurriría si un usuario leyera un mensaje mientras otro lo está modificando? Podría empezar leyendo la primera parte del mensaje original y la última parte del mensaje modificado... Incongruente.

La solución está en la implementación de semáforos[2]

Tenemos que garantizar que un usuario no es "molestado" por otros usuarios cuando acceda a los datos y para ello, implementamos un sistema de semáforos que regulará el acceso de los usuarios a los datos.

Afortunadamente no es necesario impedir todas las operaciones. Las únicas que causan problemas son las operaciones que modifican los datos. Los lectores, como no alteran los datos, pueden compartir acceso con otros lectores sin molestarse.

2.3.1. Tipos de Semáforos

Podemos así implementar dos tipos de semáforos, de lectura y de escritura.

- **Semáforos de Lectura (o Compartidos):** Son los semáforos que obtienen los usuarios cuando pretenden leer datos. Este semáforo garantiza que ningún usuario modificará los datos durante la lectura. O, dicho de otra manera, que ningún usuario obtendrá un semáforo de escritura sobre esos datos. Pero otros usuarios sí podrán obtener semáforos de lectura sobre los mismos datos, incrementando así la concurrencia al permitir operaciones de lectura simultáneas. El semáforo no se retira hasta que el último lector ha terminado su operación.
- **Semáforos de Escritura (o Exclusivos):** Son los que obtienen los usuarios al modificar los datos. Este semáforo impide a cualquier otro usuario (lector o escritor) acceder a los datos mientras dura la operación.

El problema de los semáforos vuelve a ser el rendimiento.

Mientras un usuario obtiene un semáforo sobre los datos, el resto (salvo que se trate de dos lectores) deberán esperar, reduciendo así el rendimiento de todo el sistema.

Sin embargo, este problema puede mitigarse modificando la granularidad de los semáforos.

2.3.2. Granularidad

Entendemos por Granularidad el tamaño de los datos que se ven afectados por el semáforo.

No es lo mismo controlar el acceso a toda la base de datos cuando queremos escribir un único registro, que limitar el acceso exclusivamente al registro afectado.

Si limitamos toda la tabla, el rendimiento empeorará, pues otros usuarios cuyas operaciones no estuvieran dirigidas sobre datos a los accedidos por el primer usuario pero pertenecieran a la misma tabla, tendrían aun así que esperar a que éste terminase porque el semáforo afecta a los datos que intentan acceder.

2 Llamados "locks" en inglés.

Sin embargo, aunque en principio, parecería que un semáforo por filas mejoraría el rendimiento, no siempre es así. Una estrategia de semáforos de "baja granularidad" consume mucho tiempo de gestión. La comprobación, obtención y liberación de semáforos, cuando son por filas, consume muchos recursos del servidor y puede llegar a dar al traste con el rendimiento.

Así que, es necesario alcanzar el equilibrio.

Afortunadamente, MySQL es muy bueno para eso. Puedes elegir entre distintos motores de almacenamiento y tener distintos niveles de semáforos según las necesidades de cada caso.

La siguiente tabla muestra las estrategias de semáforos de cada motor de almacenamiento de MySQL y sus características:

Semáforos	**Concurrencia**	**Sobrecarga de Gestión**	**Motores**
Por Tabla	La más baja	La más baja	MyISAM, Heap y Merge
Por Página	Media	Media	BDB
Por Fila (Con Multiversioning)	La más alta	La más alta	InnoDB

Control de Concurrencia por Multiversión (multiversioning)

Este sistema, también llamado (MVCC – Multi-Version Concurrency Control) es utilizado en varias bases de datos, como Oracle y PostgreSQL. Supone una mejora a los semáforos por fila mejorando su concurrencia. Veamos como funciona.

Conceptualmente, cualquier transacción debe ver los datos de las tablas con la misma información que tenían cuando la transacción se inició, aunque la transacción tarde horas en completarse. Este es un requerimiento para garantizar la integridad de las transacciones.

En un sistema de versiones, a cada fila de la tabla se le añaden dos datos ocultos: un identificador de creación y otro de borrado. Los identificadores se toman de un contador incremental, que crece cada vez que se inicia una transacción.

Cuando se crea una fila, su identificador de creación toma el valor del "contador".

Cuando una fila se borra, su identificador de borrado toma el valor del "contador". De hecho la fila no es borrada realmente hasta que no queda abierta ninguna transacción con identificador menor o igual que el identificador de borrado.

Ahora, es fácil entender la lógica de las operaciones SQL:

- **SELECT:** Al realizar un select, el servidor devuelve sólo las filas que:
  a) Tengan un identificador de creación menor o igual que el contador de la transacción y que no pertenezca a ninguna transacción en ejecución.
  b) Si tienen identificador de borrado no nulo, sea mayor que el contador de la transacción.
- **INSERT:** Al realizar un INSERT, el servidor pone el identificador de creación de la fila al valor del contador de la transacción.
- **DELETE:** Al borrar una fila, el servidor pone el identificador de borrado al valor del contador de la transacción.
- **UPDATE:** Al modificar una fila, el servidor pone como identificador de borrado el valor actual del identificador de creación y pone como identificador de creación el contador de la transacción.

Como resultado, las consultas de lectura nunca bloquean los datos. Pero el servidor debe almacenar más información y realizar cálculos adicionales para todas las operaciones.

2.4. Transacciones

Se denomina transacción a un conjunto de operaciones SQL que, automáticamente, se ejecutan como una única unidad de trabajo.

El ejemplo típico es el de la cuenta bancaria. Supongamos que un usuario quiere realizar una transferencia bancaria entre dos de sus cuentas. El conjunto de operaciones SQL a ejecutar podría ser algo como:

```
SELECT saldo FROM cuenta WHERE numero_cuenta = Nº cuenta origen;

(comprobar que tiene saldo suficiente)

SELECT saldo FROM cuenta WHERE numero_cuenta = Nº cuenta destino;

(sumar el saldo de destino con la cantidad transferida)

UPDATE cuenta SET saldo = Nuevo Saldo WHERE numero_cuenta = Nº cuenta
destino;

(restar el saldo de destino con la cantidad transferida)

UPDATE cuenta SET saldo = Nuevo Saldo WHERE numero_cuenta = Nº cuenta
origen;
```

Imagínate el caos si esta operación se detuviera antes de realizar el último INSERT. Se habría hecho un abono sin realizar el cargo. También podría ocurrir que, entre la primera lectura de saldo y su actualización, otra operación hubiera liquidado el saldo de la cuenta.

Naturalmente, los bancos desean que esta operación se ejecute completamente o no se ejecute en absoluto, pero nunca que se quede a medias (Y el mismo interés tenemos los usuarios si se cambia el orden de las actualizaciones de saldo).

Para evitar esto existen las transacciones.

2.4.1. Propiedades ACID

Es habitual exigir que los gestores de bases de datos relacionales superen lo que se conoce como "Test ACID", que consta de las siguientes propiedades:

- **Atomicidad:** Las transacciones son indivisibles, o bien se ejecutan completamente o no se ejecutan en absoluto. No hay términos medios.
- **Consistencia:** La base de datos evoluciona de un estado consistente a otro. En ningún momento permanece en estados inconsistentes (con transacciones a medias). Si una transacción no se completa, sus cambios nunca se reflejan en la base de datos.
- **Independencia[3]:** Las operaciones de una transacción son "normalmente" invisibles para el resto de transacciones hasta que esta se completa.
- **Durabilidad:** Una vez completada una transacción, los cambios en la base de datos son permanentes y no pueden perderse (bueno, salvo si el disco falla, obviamente).

Debemos de nuevo señalar que las transacciones no son gratuitas. Aportan seguridad a la programación evitando tener que añadir código para el control de errores, recuperación de caídas, etc. pero a cambio requieren un gran consumo de CPU, memoria y disco en el servidor. Todo esto, debe tenerse en cuenta al diseñar nuestro sistema y evitarlo cuando nuestra aplicación no lo requiera.

Niveles de Independencia

La independencia de transacciones es algo más complicada. El estándar SQL define cuatro niveles distintos, según los datos que resulten visibles desde fuera y desde dentro de una transacción. Son los siguientes:

- **Read Uncommitted (dirty reads)**: En este nivel, las transacciones pueden ver datos modificados por transacciones sin terminar. Es muy peligroso por la inconsistencia de los datos y apenas se utiliza.
- **Read Committed**: Es el nivel por defecto en muchos servidores. Las transacciones sólo ven los datos de otras transacciones que hayan finalizado antes de su inicio. El problema es que este nivel no garantiza la integridad de lecturas repetidas, y eso, puede dar algún problema[4].

3 En inglés, el término es "Isolation" (aislamiento). Lo he traducido como Independencia para mantener la I de ACID.

4 Por ejemplo, si realizamos una transacción que lea valores de una tabla y los inserte en otra por su clave primaria, puede ocurrir que, entre ambas operaciones, otra transacción modifique las claves (o incluso elimine algún elemento), convirtiendo en absurda el resultado de la inserción de la primera.

- **Repeatable Read**: En este nivel, las filas leídas en una transacción son bloqueadas hasta la finalización de la misma. Aunque, poniéndonos paranoicos, todavía tiene el problema de las *filas fantasmas*[5].
- **Serializable**: En este nivel, las transacciones se ejecutan de una en una, evitando cualquier problema de integridad, pero reduciendo al mínimo la concurrencia y el rendimiento del servidor.

2.5. Bloqueos

Sin embargo, es posible que si varias transacciones obtienen semáforos sobre los mismos datos, se produzca un bloqueo.

Supongamos las siguientes transacciones:

```
#Transacción 1
BEGIN
UPDATE cuenta SET saldo = A WHERE numero_cuenta = N1;
UPDATE cuenta SET saldo = B WHERE numero_cuenta = N2;
COMMIT

#Transacción 2
BEGIN
UPDATE cuenta SET saldo = C WHERE numero_cuenta = N2;
UPDATE cuenta SET saldo = D WHERE numero_cuenta = N1;
COMMIT
```

Con un poco de mala suerte, la primera transacción bloqueará los datos de la cuenta N1 y la segunda los de la N2, con lo que ninguna podrá completar su segunda operación.

Normalmente los gestores de base de datos implementan sistemas de detección de bloqueos y timeouts. Este es el caso del motor de almacenamiento InnoDB.

2.6. Motores de Almacenamiento

Al diseñar una base de datos con MySQL será necesario decidir qué motores de almacenamiento vamos a utilizar.

MySQL dispone, actualmente, de los siguientes motores de almacenamiento:

- **MyISAM:** Es el motor por defecto. Es muy rápido pero no transaccional.
- **InnoDB:** Es transaccional, incluyendo integridad referencial.
- **Memory (Heap):** Es una tabla MyISAM, pero almacenada en memoria, no en disco. Es todavía más rápida.
- **Archive:** Es una tabla MyISAM, pero comprimida y de sólo lectura.
- **MRG_MyISAM:** Es una agregación de tablas MyISAM. Las tablas agregadas deben ser exactamente iguales.

5 Supongamos que una transacción realiza dos consultas sobre los mismos datos (por ejemplo, WHERE a>10). Las filas leídas por la primera consulta son bloqueadas. Pero si antes de la segunda consulta, otra transacción insertara en la tabla nuevas filas que cumplieran la condición, estas no habrían sido bloqueadas en la primera consulta (porque no existían) y el resultado de la segunda consulta sería distinto del de la primera. A estas filas "que aparecen repentinamente" se les denomina *filas fantasma*. InnoDB resuelve este problema bloqueando el siguiente valor de la clave primaria de la tabla.

- **CSV:** Es una tabla que se almacena en un fichero de valores separados por comas.
- **FEDERATED:** Se trata de una tabla que, realmente, reside en otro servidor MySQL.
- **Blackhole**: Esta es una base de datos en la que todo lo que metes, desaparece.

Naturalmente es posible utilizar distintos motores en una misma base de datos, incluso pueden ser utilizados en una misma transacción (con las consideraciones que se exponen más adelante).

Desgraciadamente, la decisión correcta depende en gran parte del uso futuro que tendrá la base de datos. Y eso, suele ser difícil saberlo con exactitud durante el diseño.

De todas formas, estos son algunos de los criterios a considerar:

2.6.1. Transacciones y Concurrencia

Si vamos a necesitar transacciones y alta concurrencia de usuarios, probablemente InnoDB será la mejor opción. Es decir, que obtendremos mejores condiciones de rendimiento si las tablas que se vayan a ver involucradas frecuentemente en consultas transacciones y que sufran una alta concurrencia de usuarios se basan en el motor InnoDB.

Si la concurrencia no es tan alta, pero seguimos necesitando transacciones, tanto InnoDB como BDB pueden ser la solución.

El motor MyISAM no soporta transacciones. Pero si no las necesitamos, MyISAM será con toda probabilidad la mejor opción.

2.6.2. Copias de Seguridad

La realización de copias de seguridad también condiciona la elección de tablas.

Si podemos detener periódicamente la base de datos para realizarlas, cualquier motor servirá. Pero si debemos realizarlas “en caliente” la elección será mucho más complicada.

Abordaremos este asunto en el capítulo de Copias de Seguridad.

2.6.3. Características Especiales

Por último, las características especiales de nuestra aplicación pueden obligarnos a recurrir a un tipo determinado de motor de almacenamiento.

En concreto, si nuestra aplicación depende de la capacidad para contar rápidamente el número de registros de una tabla (`SELECT COUNT(*) FROM`) recurriremos a MyISAM, que siempre sabe cuantos registros tiene sin necesidad de contarlos. Sin embargo, InnoDB debe realizar la cuenta.

Así mismo, recurriremos a MyISAM si necesitamos realizar eficientemente búsquedas por texto completo (`WHERE notas LIKE '%...`).

Pero sin embargo, InnoDB será la solución en los casos en los que necesitemos disponer de integridad referencial.

2.6.4. Resumen de Características

La siguiente tabla resume las características de los motores de almacenamiento:

Característica	MyISAM	Memory	BDB	InnoDB
Transacciones	No	No	Sí	Sí
Granularidad	Tabla	Tabla	Página (8KB)	Fila
Almacenamiento	Ficheros Separados	En Memoria	Un Fichero por Tabla	Tablespace[6]
Niveles de Aislamiento	Ninguno	Ninguno	Read Commited	Todos
Formato Portable	Sí	N/A	No	Sí
Integridad Referencial	No	No	No	Sí
Clave Primaria con Datos	No	No	Sí	Sí
Caché	No	Sí	Sí	Sí

6 Un “tablespace” (espacio de tabla en inglés) es un área continua de disco que es gestionada directamente por el gestor de base de datos para almacenar sus datos.

Capítulo 3. Benchmarking

3.1. La Necesidad del Benchmarking

Realmente, como administrador de bases de datos, sólo tienes dos opciones: o esperas a que algo vaya mal para lanzarte a arreglarlo a toda prisa, o te preparas analizando tus bases de datos para conocer sus límites y estar preparado.

La primera opción es la más cómoda. La segunda, aunque ideal, requiere demasiado tiempo y no garantiza resultados... ¿o sí?

La diferencia está en si quieres dormir tranquilo por las noches o no. Veamos como puedes conseguir conciliar el sueno y vivir tranquilo sabiendo que tus bases de datos no estallarán de la noche a la mañana.

3.1.1. ¿Qué puedo esperar?

Con una buena estrategia de Benchmarking podrás saber cosas como:

- ¿Qué tiempo de respuesta tendrán mis consultas cuando el número de registros se multiplique por 10?
- ¿Cómo mejorará el sistema si incremento la memoria RAM?
- Este nuevo servidor... ¿es más rápido que el anterior? ¿Cuánto?
- ¿Qué pasaría si deshabilito la caché? ¿Y si la incremento?
- ¿Qué es más rápido, utilizo un select anidado o dos subconsultas?

El benchmarking consiste en plantearse preguntas, comparar situaciones y sacar conclusiones. Ahora, el objetivo es hacerlo de forma sencilla.

3.2. Estrategias

En primer lugar, debemos distinguir entre dos tipos de test: tests de rendimiento y tests de estrés.

3.2.1. Tests de Rendimiento

Su objetivo es comparar entre dos opciones. ¿Es mejor la configuración A o la B? ¿Obtendremos mejor rendimiento con la consulta A o con la B?

Con este tipo de tests, no nos importa tanto el resultado (X segundos) como la comparación (la opción A es 20 veces más rápida que la B).

3.2.2. Tests de Estrés

Pero los tests de rendimiento no nos permiten saber cómo se comportará el sistema en situaciones reales (o hipotéticas). Para eso necesitamos los tests de estrés.

En los tests de estrés buscamos el resultado absoluto. ¿Cuál es el tiempo de

respuesta con 100 veces más registros que los actuales? ¿Y si tuviéramos picos de 25 usuarios simultáneos?

3.2.3. Consideraciones

Ambos tipos de tests son necesarios. Pero antes, todavía tenemos que establecer algunas consideraciones importantes:

- **Realiza los Cambios de Uno en Uno**. De lo contrario, no sabrás que es lo que ha ido bien o mal.
- **Prueba Gradualmente**. No incrementes los valores bruscamente. Haz pequeños cambios y observa las tendencias. En lugar de triplicar el número de registros, sube un 10% cada vez. En lugar de ajustar los buffers en un 200%, hazlo de 10% en 10%.
- **Repite Siempre los Tests**. Independientemente de lo bien o mal que hayan salido los resultados, repítelos. Puede que olvidaras desactivar el cron, o que algún proceso iniciara una tarea intenso trabajo de disco... asegúrate repitiendo los tests al menos cuatro veces.
- **Utiliza Datos Reales**. De lo contrario, los resultados no serán relevantes. Si tu base de datos espera que tenga 2 millones de registros, ¿de qué te sirve probar con 25.000? Quizás tu servidor pueda cachear 25.000 registros y ofrecer magníficos tiempos de repuesta, pero puede colapsarse con 500.000.
- **No te Pases con los Clientes**. Quizás mole mucho saber que tu servidor puede aguantar 100 clientes simultáneos. Pero será una pérdida de tiempo si no prevés tener más de 10.
- **Separa el Cliente del Servidor**. Si utilizas la misma máquina para lanzar los tests y albergar el servidor, puede que los resultados se vean influenciados por el consumo de recursos del cliente.

3.3. Herramientas

Existen varias herramientas que te ayudarán al realizar benchmarking.

3.3.1. MySQL Benchmark Suite

Todas las distribuciones de MySQL vienen con un conjunto estándar de tests. Están escritos en perl y puedes encontrarlos en el subdirectorio sql-bench del directorio de instalación[7].

Esta suite no está diseñada para comparar distintas configuraciones de un mismo servidor, sino para comparar el rendimiento de distintos servidores. Por ejemplo, te ayudará a decidir cuál es la arquitectura hardware más conveniente para tu servidor. Y como también puede ejecutarse para otros servidores de base de datos no MySQL, como Adabas, Oracle, DB2, MS SQL, etc. te puede servir para comprobar las diferencias.

Estos tests no aprovechan las posibilidades del multiproceso.

7 En Debian gnu/Linux este es /usr/share/mysql/sql-bench

La forma más cómoda de ejecutarlos es mediante el comando:

```
perl run-all-tests --server=MySQL --user=root --password=secreta
```

Pero, si lo prefieres, puedes ejecutar sólo los tests que te interesen (conexión, insert, select, transacciones, etc.).

La ejecución consume mucha CPU y, en algunos de ellos, también es muy intensiva en disco.

Los resultados se guardan en el subdirectorio `output` y dispone de herramientas para automatizar la comparación de resultados.

3.3.2. Super-smack

Se trata de una herramienta mucho más complicada, pero que permite someter al servidor a tests de carga con varios usuarios.

Funciona para MySQL y para PostgreSQL. Incluso tiene un soporte mínimo para Oracle.

Fue inicialmente desarrollada por un empleado de MySQL, A.B. (Sasha Pachev). Después pasó a manos de Jeremy Zawodny, un consultor y escritor especializado en MySQL y, actualmente, la mantiene Tony Bourke, un administrador de sistemas neoyorkino.

En definitiva, que hoy el código de Super-smack está en http://vegan.net/tony/supersmack/ pero puede que mañana tengas que buscarlo algo más.

Instalación

Tras bajar el código, la instalación es muy sencilla, utiliza autoconf de GNU.

```
$> tar -xvzf super-smack-versión.tar.gz
$> cd super-smack-versión
$> ./configure -with-mysql
$> make
$> su
#> make install
```

Ejecución

Ejecutar los tests de super-smack es sencillo. Por ejemplo, el comando

```
$> super-smack update-select.smack 30 10000
```

simula 30 usuarios simultáneos cada uno ejecutando 10.000 veces el test update-select.

La escritura de los tests requiere algo más de habilidad. Conviene leer el Tutorial incluido y consultar los tests de ejemplo.

Super-smack también incluye la herramienta gen-data, que facilita mucho la creación de datos de ejemplo para cargar las bases de datos.

3.3.3. Soluciones A Mano

Pero si realmente queremos probar el rendimiento de nuestra base de datos con intensidad, tendremos que pensar en desarrollar nuestro propio conjunto de pruebas.

Super-smack puede ser una solución, pero es habitual que los administradores de bases de datos programen y ejecuten sus propios scripts de prueba.

Capítulo 4. Administración

Para la administración de un servidor MySQL disponemos de dos tipos de herramientas.

Por un lado están las herramientas de consola, que se ejecutan como comandos del sistema operativo y tienen la gran ventaja de que pueden ejecutarse a través de una simple conexión ssh. Sin embargo, puede resultar complicado recordar sus opciones y su sintaxis.

Por otro lado están las herramientas gráficas, mucho más intuitivas y vistosas, pero que pueden resultar inútiles cuando no disponemos de nuestro entorno de trabajo habitual o fallan otros elementos (entorno gráfico, comunicaciones, etc.). Claro, que suele ser en estas situaciones cuando un administrador de sistemas tiene más trabajo.

Así que, necesitamos aprender a manejar los dos tipos.

4.1. Comandos

Los comandos de administración de MySQL están disponibles tras instalar el paquete mysql-client. No olvides que todas ellas vienen con su correspondiente página de Manual, accesible mediante `man nombre_comando`. Prácticamente ningún administrador, por avanzado que sea, recuerda la sintaxis exacta de cada comando ni, mucho menos, todas sus opciones.

Así que, ¡no te cortes! consulta el manual antes de ejecutar el comando para asegurarte.

Los principales comandos de administración de MySQL son:

4.1.1. mysql

Inicia una sesión de consola en MySQL.

Este comando ofrece un intérprete de SQL en línea. Permite lanzar consultas, modificar tablas, crear bases de datos, cambiar permisos... prácticamente todo, desde una sencilla línea de comandos.

Al iniciar, se le puede indicar tanto la base de datos a la que queremos conectarnos como el usuario y su contraseña. Por ejemplo,

```
mysql -u root -p banco
```

abrirá la sesión conectándonos como root a la base de datos “banco”. El parámetro p indica que el acceso está protegido por contraseña y el comando nos la pedirá antes de intentar conectar.

El comando mysql se utiliza frecuentemente para realizar scripts que trabajen con la base de datos. Para ello, se añade como entrada al comando un fichero que contenga sentencias SQL. Por ejemplo, el comando

```
mysql -u root --password=difícil banco < banco.sql
```

abrirá la sesión conectándonos a la base de datos banco como root y utilizando la clave "difícil" (en este caso no nos la pedirá). El comando tomará como entrada el fichero banco.sql que podría ser un conjunto de sentencias SQL para crear las tablas e insertar los datos en esa base de datos.

4.1.2. mysqladmin

Hay algunas tareas que resultan incómodas de realizar en SQL. E incluso otras, pocas, que no pueden hacerse directamente con SQL.

Para realizarlas disponemos de mysqladmin.

Este comando permite crear y eliminar bases de datos (también puede hacerse con CREATE/DROP DATABASE), ver el estado del servidor (esto no puede hacerse con SQL), cambiar la contraseña del administrador (root), o incluso arrancarlo y cerrarlo.

4.1.3. myisamchk

Proporciona información sobre las tablas de tipo MyISAM y permite comprobarlas, repararlas y optimizarlas.

myisamchk se utiliza con el servidor detenido, pues realiza sus funciones sobre los ficheros que sirven de almacenamiento lógico de las tablas e índices. Estos ficheros, que están en el directorio /var/lib/mysql/nombre_base_datos (en Debian), tienen extensiones .MYD para las tablas y .MYI para los índices).

myisamchk utiliza los nombres de ficheros (opcionalmente con *) como parámetro, no utiliza el nombre de la base de datos ni de las tablas.

4.1.4. mysqlcheck

Este comando también permite comprobar, reparar y optimizar las tablas. Pero lo hace con el servidor en marcha.

En este caso, los parámetros que utiliza sí son los nombres de las bases de datos y las tablas del servidor, y no los ficheros de almacenamiento que los soportan.

mysqlcheck funciona con distintos tipos de motores de almacenamiento, siempre que estos soporten los comandos CHECK TABLE, REPAIR TABLE, ANALIZE TABLE y OPTIMIZE TABLE que están realmente detrás del comando.

4.1.5. mysqldump y mysqlimport

Estos son dos de los comandos más utilizados.

mysqldump permite realizar copias de seguridad de una base de datos o de un conjunto de tablas generando un fichero de texto SQL. Dispone de múltiples opciones para añadir o eliminar sentencias de creación de tablas y base de datos, añadir sentencias de drop previas, bloquear las tablas, etc.

La otra cara de la moneda es mysqlimport que, naturalmente, carga una fichero

de texto SQL. Realmente mysqlimport no es más que un interfaz al comando SQL LOAD DATA INFILE.

Además de los descritos aquí, hay otros muchos comandos de administración que pueden facilitar las tareas de administración en un momento dado.

4.2. Sistemas de Administración Gráficos

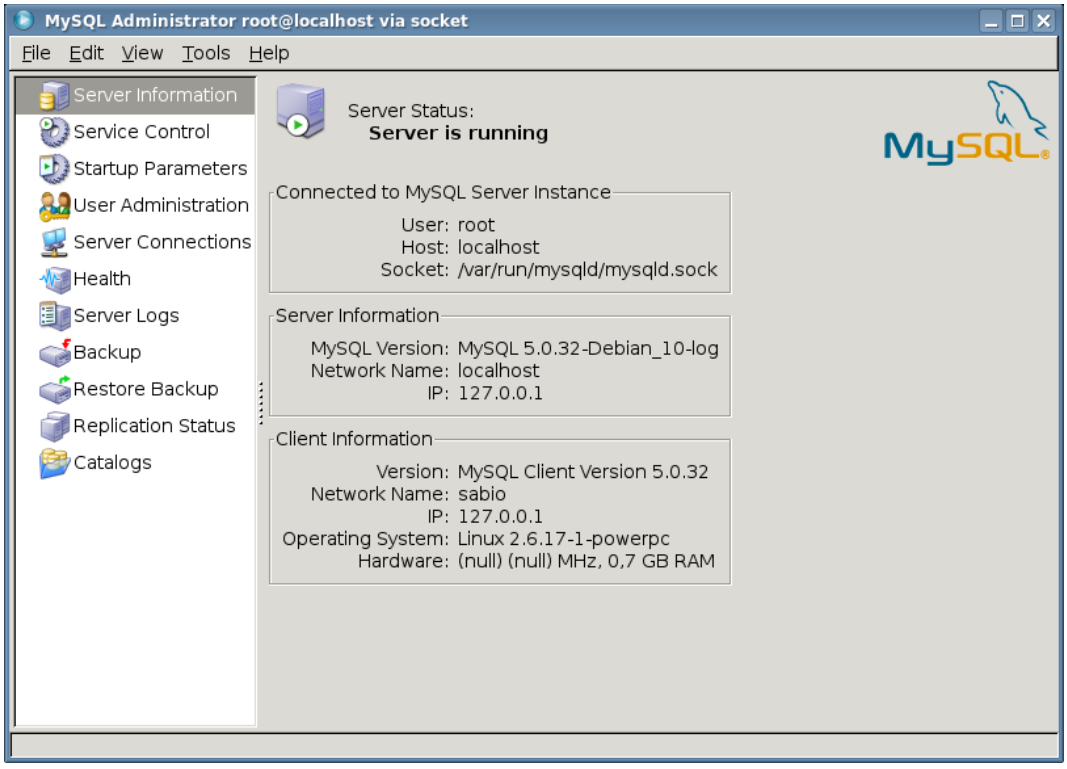
Las herramientas de administración gráfica nos facilitan la labor de administración haciéndola algo más humana.

Vamos algunas de ellas y las principales funciones que desempeñan.

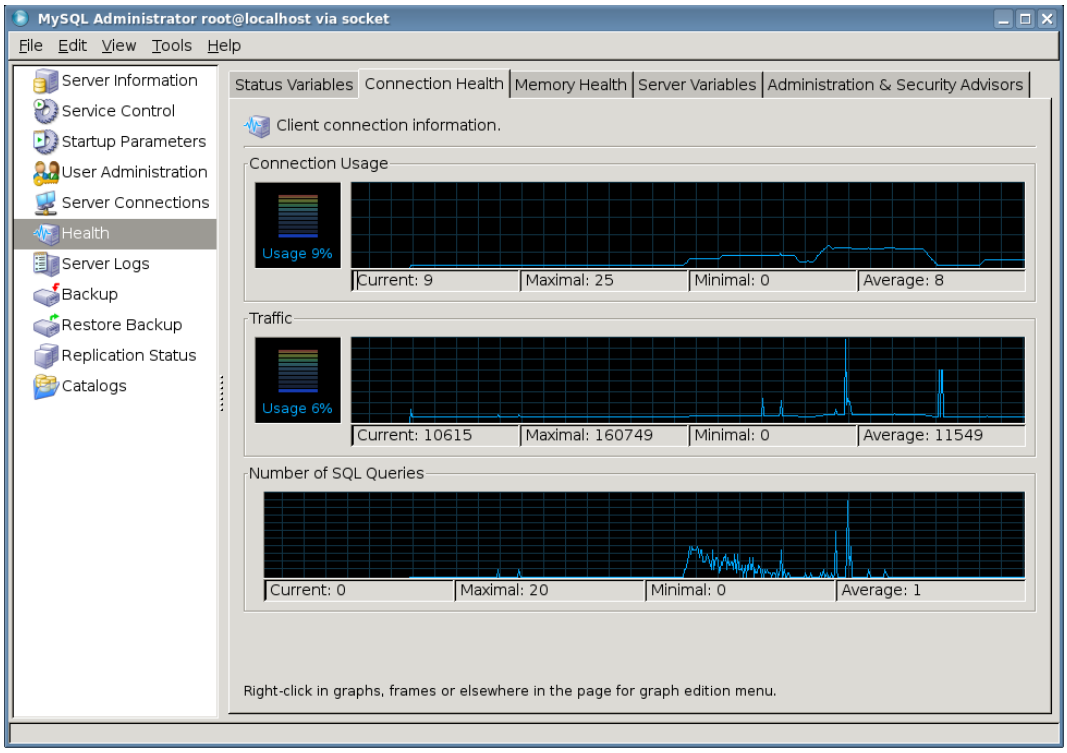
4.2.1. MySQL Administrator

Se trata de una aplicación que nos permite ver el estado de nuestro servidor en tiempo de ejecución. Puede resultar muy útil para identificar problemas de rendimiento.

La aplicación se lanza con el comando mysql-admin y presenta un interfaz gráfico con distintas secciones.



A parte de funciones de administración (arrancar/parar el servidor, realizar copias de seguridad y gestionar usuarios), la característica más apreciada de MySQL Administrator es la posibilidad de ver y crear gráficos del estado del servidor.



Creación de Gráficos

Sin lugar a dudas, una de las opciones más potentes de MySQL Administrator es la posibilidad de crear gráficos de rendimiento a medida.

Vamos a ver cómo se hace,

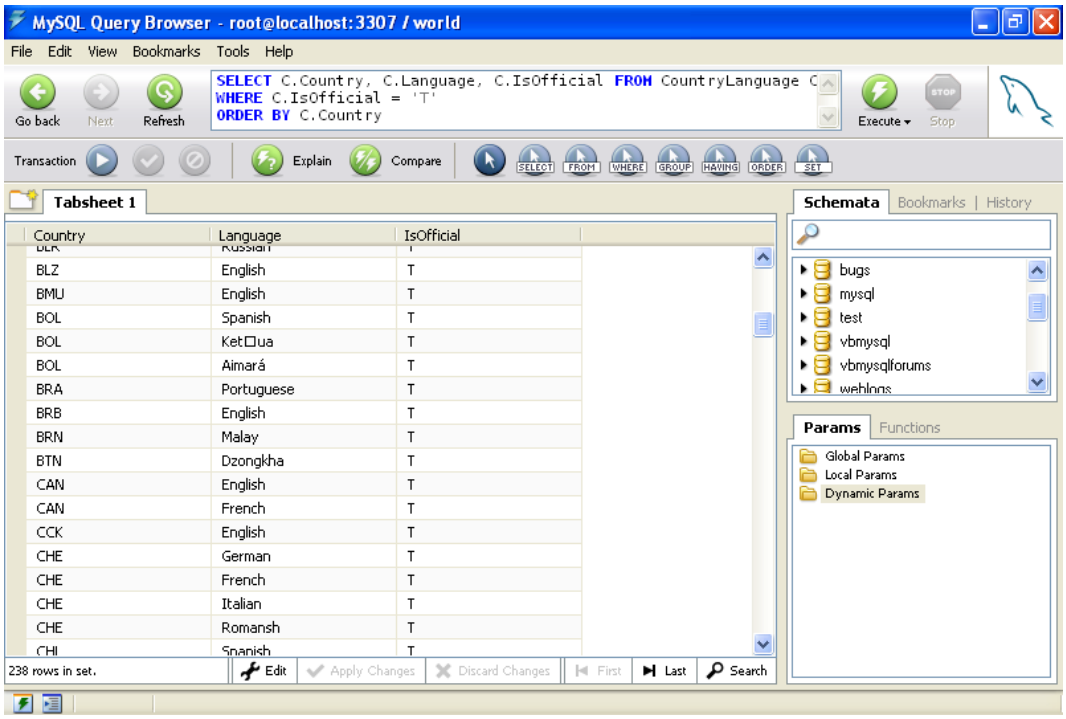
1. Seleccionamos las variables que queremos monitorizar. Podemos escoger cualquiera de la pestaña de variables de estado ("Status Variables") o de la pestaña de variables del servidor ("Server Variables"). Anota los nombres de las variables, te harán falta.
2. Creamos un grupo. Para ello vamos a la pestaña en la que queremos insertar el gráfico (por ejemplo "Memory Health") y, pulsando con el botón de la derecha, seleccionamos "Add Group". Opcionalmente, podemos crear nuestras propias páginas de gráficos.
3. Creamos un gráfico. Para ello, pulsamos con el botón derecho sobre el grupo que hemos creado. Tendremos que elegir el tipo de gráfico, su nombre y, sobre todo, la fórmula que queremos visualizar.
4. Creación de la Fórmula. La fórmula se crea escribiendo en el correspondiente cuadro de texto. Los nombres de las variables de MySQL se ponen entre corchetes. Si queremos el valor relativo a la última evaluación (por segundo), antepondremos a la fórmula el acento circunflejo (^). Podemos utilizar también números y operadores matemáticos.

Y listo.

4.2.2. MySQL Query Browser

Se trata de una aplicación que nos ayuda a definir, ejecutar sentencias SQL así como a analizar los resultados producidos.

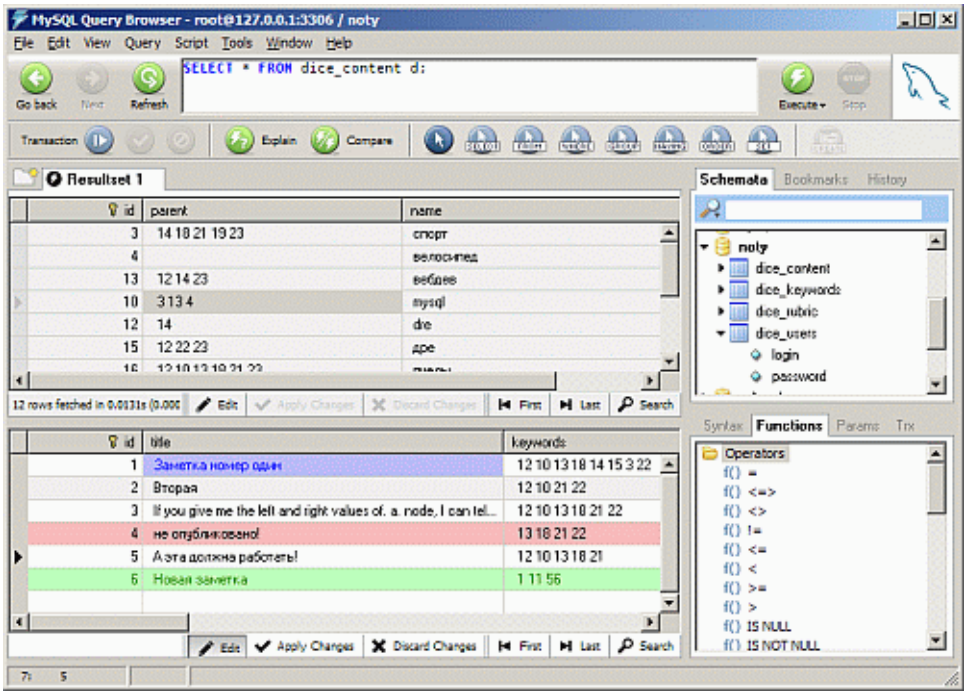
Vamos con algunas de las operaciones que podemos realizar.



Creación Visual de Consultas

Para crear visualmente una consulta realizamos los siguientes pasos:

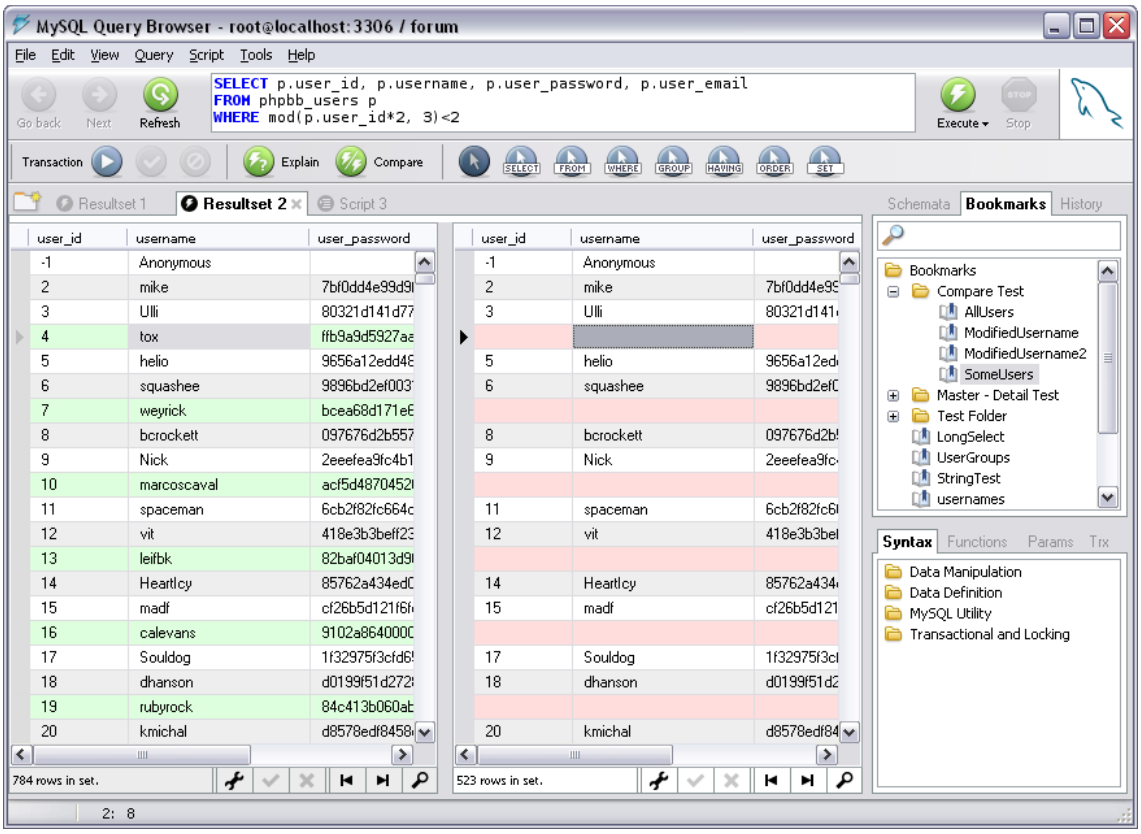
1. Elegimos la tabla. Para eso, podemos arrastrarla hasta el área de consulta o hacer doble click.
2. Añadir otras tablas. Podemos añadir otras tablas arrastrándolas y soltándolas en el área de consulta que nos interese. Observa como las opciones del área de consulta aparecen bajo esta al llevar el ratón.
3. Añadir columnas. También arrastrando, podemos añadir columnas a las distintas opciones de consulta: SELECT, WHERE, ORDER, HAVING, GROUP...
4. Escribe lo que necesites. Puedes escribir directamente en el área de consulta, por ejemplo, para poner los valores de comparación de la cláusula WHERE.
5. Ejecuta la consulta pulsando el botón “Execute”.
6. Para ver mejor los resultados, puedes añadir paneles de resultados y dividirlos vertical u horizontalmente.



Creación de Vistas Maestro-Detalle

Vamos a ver cómo crear la típica vista Maestro-Detalle con MySQL Query Browser.

1. Creamos la consulta Maestro. Tal y como lo hicimos en el ejemplo anterior, incluyendo las columnas que queramos visualizar en los resultados.
2. Ejecutamos la consulta.
3. Dividimos horizontalmente el panel de resultados (botón derecho).
4. Creamos la consulta de detalle en el área de consulta. Pero tenemos que relacionar ambas consultas añadiendo las columnas necesarias a la clausula WHERE. La única diferencia es que el valor a igualar lo seleccionamos del panel de parámetros ("Params" en la parte baja de la pantalla).
5. Ojo, para poder seleccionar parámetros, debemos tener seleccionado el panel de detalle. Así se mostrarán los parámetros "Dinámicos" disponibles en ese panel. Si no los pudieras añadir arrastrando, basta con que pongas su nombre precedido de dos puntos (:) en la clausula WHERE.
6. Vuelve a ejecutar la consulta y ya podrás navegar entre los resultados. Pulsando sobre un resultado de la consulta maestra, veremos los detalles en el panel de detalle.



Comparar Resultados

Podemos comparar resultados fácilmente. Esto nos permitirá comparar fácilmente consultas similares. Para ello, los resultados deben tener columnas con el mismo nombre y las tablas subyacentes deben tener claves primarias definidas.

1. Ejecutamos la primera consulta.
2. Dividimos verticalmente el panel de resultados y ejecutamos en él la segunda consulta.
3. Seleccionamos la operación “Compare” del menú “Query”.
4. Revisamos los resultados. MySQL Query Browser habrá añadido algunas ayudas:
5. Desplazamiento sincronizado en ambos paneles.
6. Los resultados que sólo estén presentes en un panel, se resaltarán en un color y se acompañarán de una fila vacía, resaltada en otro color en el panel contrario.
7. Cuando ambas filas sólo difieran en alguna columna, esta se resaltará.

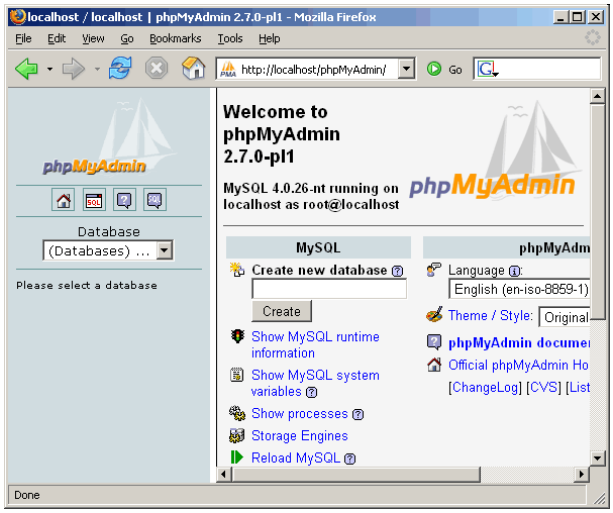
De esta forma, podremos navegar cómodamente por los resultados y ver las diferencias.

Editar Resultados

También podemos navegar por un resultado y modificar cómodamente sus valores. Para ello:

1. Ejecutamos la consulta.
2. Pulsamos el botón editar (en la barra de herramientas inferior). No estará disponible si cada fila del resultado no es identificable (pon clave primaria).
3. Navegamos por los resultados utilizando el ratón o las flechas de desplazamiento. Estas nos permitirán también llegar a cada columna.
4. Editamos los resultados pulsando la barra o haciendo doble click.
5. Puedes añadir filas (al final del resultado) o borrarlas con el menú de contexto (botón de la derecha).
6. Aplica o descarta los cambios con los botones de la barra inferior.

4.2.3. phpmyadmin

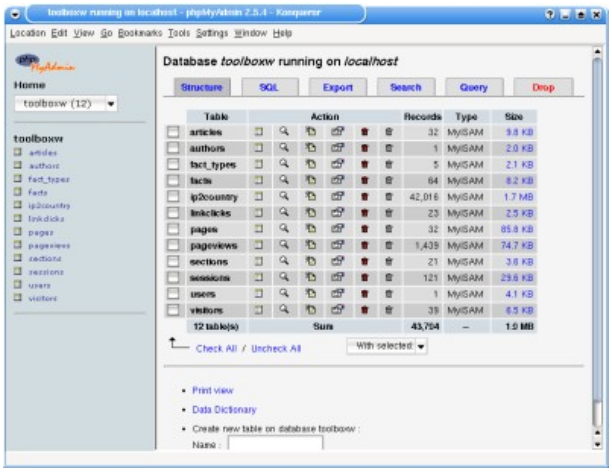


phpmyadmin es una herramienta distinta a las anteriores. La razón es que, en lugar de instalarse en el cliente (como las anteriores), se instala en el servidor.

phpmyadmin es una aplicación web que facilita la gestión de un servidor MySQL. Al instalarse en el servidor[8] permite acceder a ella desde cualquier ordenador utilizando un navegador web.

El uso es muy sencillo, y permite examinar la estructura y contenidos de la tablas, lanzar consultas, hacer cargas y descargas masivas, búsquedas en los datos, etc.

Su uso es muy intuitivo y por eso se ha hecho muy popular.

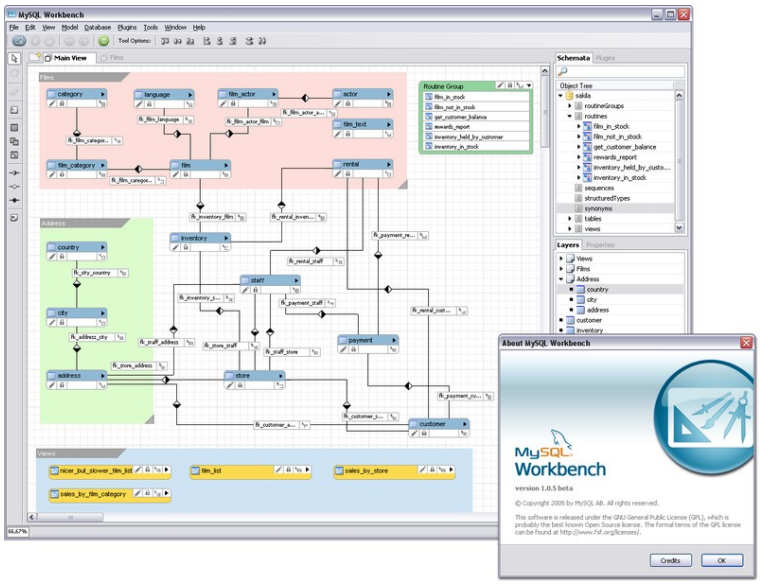


8 Técnicamente puede instalarse sobre otro servidor con acceso al servidor de base de datos, pero lo habitual es instalarlo en el mismo servidor que alberga la base de datos.

4.2.4. MySQL Workbench

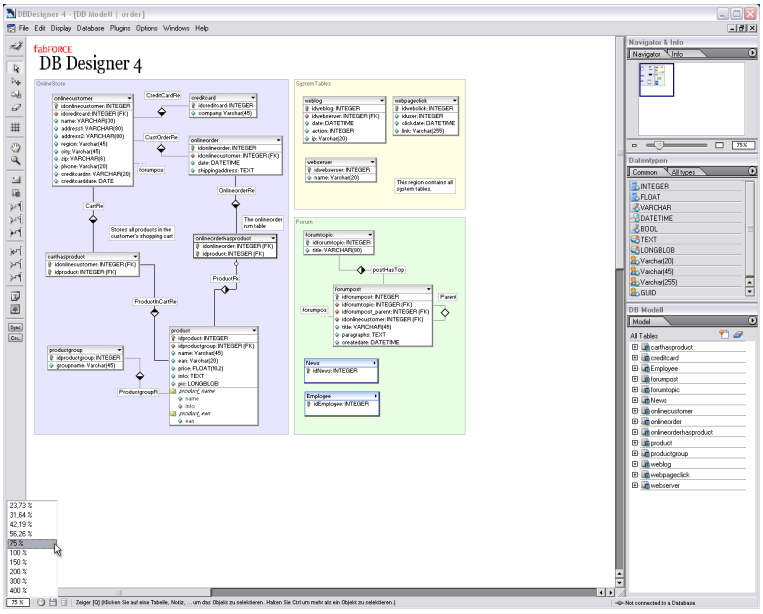
Es una aplicación para el diseño de bases de datos con MySQL. A fecha de hoy se encuentra todavía en fase beta, pero funciona bastante bien.

Tiene, más o menos, la misma funcionalidad que DBDesigner.



4.2.5. DBDesigner

Se trata de una aplicación desarrollada por FabForce (www.fabforce.net). Se trata de una aplicación visual para el modelado, diseño y creación de modelos de datos.



Permite por tanto la definición de tablas, columnas, relaciones, índices, etc. y la creación de bases de datos en el servidor partiendo de los modelos definidos. Pero también incluye un modo de "ingeniería inversa" que analiza una base de datos existente para crear su modelo.

Trabaja con MySQL (para la que está optimizado), Oracle y bases de datos ODBC.

Capítulo 5. Seguridad

5.1. Conceptos de Seguridad

5.1.1. Cuentas

Estamos acostumbrados a que una cuenta de acceso a un sistema se componga de usuario y contraseña. Pero en MySQL tendremos que acostumbrarnos a añadir también el "host", el ordenador desde el que se conecta el usuario.

Para MySQL, un usuario "juan" que se conecta desde la máquina "localhost", es totalmente diferente de otro usuario "juan" que se conecta desde la máquina "united_crakers.com". Y pueden tener distintos permisos de acceso.

¡OJO! Si una cuenta queda identificada por su usuario/clave/host, esto no incluye la base de datos. Es decir, que la cuenta juan/miclave/localhost tiene la misma clave de acceso ¡para todas las bases de datos del servidor! Pero que tenga la misma clave de acceso, no significa que tenga los mismos permisos.

5.1.2. Comprobaciones

Para cada conexión, MySQL realiza tres tipos de comprobaciones:

Autenticación

- "Y tú ¿quién eres?" -

Para cada nueva conexión, MySQL comprueba la cuenta en base al nombre de usuario, la clave y el host de origen. Esta información determinará los privilegios del usuario.

Autorización

- "¿Qué es lo que puedes hacer?" -

Para cada operación, MySQL comprueba que el usuario tiene permiso para realizarla. No es lo mismo tener permiso para cargar datos de un fichero (FILE) datos que para cerrar (SHUTDOWN) el servidor.

Control de Acceso

- "¿A qué datos puedes acceder?" -

Para cada operación, MySQL comprueba que el usuario tiene permiso para ver o modificar las tablas y columnas involucradas.

5.1.3. Privilegios

En MySQL podemos utilizar dos tipos de privilegios: Privilegios sobre objetos de la base de datos (relacionados con el control de acceso) y Privilegios Globales (relacionados con la autorización).

Privilegios sobre Objetos

Cada cuenta de acceso puede tener privilegios distintos para los diferentes objetos del servidor.

Los objetos disponibles son:

- Bases de Datos
- Tablas
- Columnas
- Índices

Y, para cada uno de ellos, el usuario puede tener concedidos o denegados privilegios. Algunos de estos son: Select, Insert, Update, Index, Delete, Drop, Index, Alter, Create, Grant, References... También hay algunos nombres de permisos que agrupan a otros. Este es el caso de All y Usage.

Ten en cuenta que los privilegios son booleanos. O se tienen o no se tienen. No hay valores intermedios (no sé si lo tiene), ni numéricos (tiene un permiso de 3).

Globales

Para comprobar las autorizaciones de operación MySQL dispone de privilegios globales. Frente a los anteriores, que se aplican a objetos del servidor (datos), los privilegios globales se aplican a todo el servidor. Son los siguientes:

- **Reload:** Muy inofensivo, permite al usuario realizar FLUSH.
- **Shutdown:** Para cerrar el servidor.
- **Process:** Permite ver la lista de procesos (SHOW PROCESSLIST) y matar (KILL) sus procesos. El acceso a la lista de procesos permite ver los datos de las mismas, incluyendo las claves de acceso.
- **File:** Permite cargar datos de ficheros (LOAD DATA FILE).
- **Super:** Permite matar cualquier proceso (del servidor de base de datos), incluso los que no pertenezcan al usuario.

5.1.4. Tablas Involucradas

MySQL guarda la información de permisos en la base de datos “mysql”, en distintas tablas.

Son las siguientes:

- **user:** Contiene los privilegios globales y las claves encriptadas. Es responsable de

determinar que hosts y usuarios pueden conectarse al servidor.

- **host:** Asigna privilegios por host, independientemente del usuario. No es modificada por los comandos de GRANT y REVOKE, así que hay que manejarla "a mano".
- **db:** Establece los privilegios para bases de datos.
- **tables_priv:** Establece los privilegios para cada tabla. No suele ser utilizada.
- **columns_priv:** Establece los privilegios para cada columna de cada tabla. No suele ser utilizada.

Aplicación

En primer lugar, MySQL comprueba la tabla user para decidir si acepta o no una determinada conexión.

Para cada consulta (de las conexiones aceptadas, obviamente), MySQL aplica los permisos "en cascada".

Es decir, en primer lugar consulta la tabla user. Los permisos que ahí encuentra los "pasa" a la tabla a la tabla host. El resultado de esta, es pasado a db, de ahí a tables_priv y de ahí a columns_priv.

Los criterios de aplicación (matching) son distintos para cada tabla:

1. En user se utilizan host, user y password.
2. En host se utilizan host y db.
3. En db se utilizan user, host y db.
4. En tables_priv se utilizan user, host, db y table.
5. En columns_priv se utilizan user, host, db, table y column.

La aplicación de hosts es más específica. Como es posible utilizar comodines para esta regla (%[9]), podría ocurrir que varias entradas de la tabla cumplieran los criterios y, por lo tanto, sus permisos fueran aplicables. MySQL resuelve el problema estableciendo una regla de aplicación: las entradas más genéricas se aplicarán primero (menor preferencia).

Mantenimiento

Técnicamente es posible dar, quitar y modificar permisos, tanto sobre objetos como globales, accediendo a estas tablas y lanzando consultas (INSERT, UPDATE...). De hecho, muchos administradores lo hacen así.

Pero no es la forma recomendada.

Podría ocurrir que, en un futuro, se modificara la estructura de estas tablas. O podrías cometer un pequeño error, pasar por alto una relación, etc. con consecuencias imprevisibles para la seguridad de tu servidor.

9 Los hosts pueden definirse por nombre (juan.midominio.com), por IP (192.168.2.145) y por subredes (192.168.2.0/255.255.255.0). En las definiciones por nombre pueden utilizarse comodines (_ para un carácter y % para cualquier número de caracteres). Por ejemplo %.midominio.com o incluso % para cualquier host.

En lugar de jugártela, puede ser preferible que conozcas los comandos GRANT y REVOKE de la siguiente sección.

5.2. Sentencias GRANT y REVOKE

Podemos abstraernos de las tablas de permisos de MySQL utilizando las sentencias GRANT y REVOKE. Estas se encargarán de introducir los valores adecuados en las tablas (salvo en host) con una sintaxis más humana.

5.2.1. Comando GRANT

Se utiliza para dar privilegios. Su sintaxis es la siguiente:

```
GRANT privilegios ON objeto TO usuario@host IDENTIFIED
BY clave WITH opciones
```

- "privilegios" es una lista de privilegios (Select, Shutdown, All...) separados por comas.
- "objeto" es cualquier objeto del servidor, pudiendo utilizar el * como comodín. No se indica ningún objeto para los privilegio globales.
- "usuario@host" es el nombre del usuario y el host. Si no se indica ningún host, los privilegios se concederán para cualquier host (%).
- "clave" es la clave en claro que se asigna al usuario. Si ya tiene clave asignada no es conveniente indicarla, porque quedaría modificada (para todas las bases de datos a las que tenga acceso el usuario).
- "opciones" es una lista separada por comas de las opciones posibles. Puede ser GRANT OPTION para permitir que el usuario asigne permisos, MAX_CONNECTIONS_PER_HOUR, MAX_QUERIES_PER_HOUR, etc.

Ejemplos

Veamos algunos ejemplos de sentencias GRANT:

```
GRANT ALL TO 'root'@'localhost' ON *.* IDENTIFIED BY 'miclave'
```

El usuario root podrá hacer lo que quiera desde localhost. Le hemos asignado clave.

```
GRANT INSERT, UPDATE ON banco.cliente TO @'caja.es'
```

Cualquier usuario podrá, desde el dominio 'caja.es', insertar y actualizar la tabla "cliente" de la base de datos "banco".

5.2.2. Comando REVOKE

Se utiliza para quitar privilegios previamente asignados. Su sintaxis es la siguiente:

```
REVOKE privilegios ON objeto FROM usuario@host
```

Los parámetros “privilegios”, “objeto” y “usuario@host” son idénticos a los del comando GRANT.

5.2.3. Puntos a Tener en Cuenta

MySQL está diseñada para ser fácil de usar. Y, aunque esta afirmación suene intrínsecamente positiva, en ocasiones puede suponer algún problema. Tal es el caso con los privilegios.

En concreto, MySQL tiene una política explícita de privilegios, pero no de revocaciones. Esto quiere decir que sólo pueden revocarse (GRANT) privilegios que se hubieran asignado previamente.

Así, si hacemos

```
GRANT ALL ON *.* TO 'juan'
REVOKE SELECT ON nominas.* FROM 'juan'
```

tendremos un error. MySQL considera que el privilegio SELECT no ha sido concedido a 'juan' y, por lo tanto, no se puede revocar.

El problema es que MySQL utiliza el sistema de comprobación que comentamos anteriormente, por tablas. Pero no tiene una tabla para recoger las “excepciones” a los permisos. Los privilegios pueden asignarse o no asignarse, pero no pueden negarse.

Siempre hay alguna solución, aunque sea mala. En este caso, habría que revocar el permiso genérico al usuario y concedérselos de uno en uno explícitamente (salvo para nóminas).

Lo mismo ocurre con los hosts. No es posible excluir un host determinado. Aunque aquí la solución puede ser algo más sencilla. Si queremos que un usuario no pueda conectarse desde un determinado host, podemos asignarle una clave imposible y seguir manteniendo el permiso global.

También hay que tener en cuenta que los permisos no desaparecen cuando desaparecen los objetos sobre los que se aplican. Esto tiene sus ventajas, si tiras (DROP) una tabla para volver a crearla, no tendrás que volver a asignar los permisos sobre ella a todos sus usuarios. Pero, sin embargo, si tiras una base de datos y meses después vuelves crearla, los usuarios antiguos seguirán teniendo sus permisos sobre el nuevo objeto... ¿es eso lo que querías hacer? Ten en cuenta que muchas contraseñas pueden quedar en viejas aplicaciones obsoletas.

Tendrás que acordarte de eliminar los permisos para cada objeto que destruyes... si eso es lo que quieres hacer.

5.3. Logs

MySQL dispone de cuatro ficheros de log. Todos ellos se definen y configuran en el fichero `my.cnf`.

5.3.1. Error Log

Este fichero registra el arranque y la parada del servidor MySQL, así como errores de integridad en las tablas y necesidad de chequeos. No registra ni operaciones SQL, ni mucho menos errores SQL.

En un sistema Debian gnu/Linux, la salida de este fichero está redirigida al syslog (`/var/log/syslog`).

5.3.2. Query Log

Este fichero de log registra todas las operaciones SQL que llegan al servidor. El registro se realiza según llegan las peticiones y antes de su ejecución. Así que el hecho de encontrar una sentencia SQL registrada no significa que esta se ejecutara con éxito.

Naturalmente, este log tiene un alto coste en rendimiento. En sistemas cargados, todas las operaciones deben escribirse en disco antes de realizarse, y eso, lleva su tiempo.

Para activar este log, hay que añadir la siguiente línea en la sección [mysqld] de my.cnf:

```
log = /var/log/mysql.log
```

Naturalmente la dirección del fichero puede ser otra. Pero debes asegurarte que el log se rota periódicamente para evitar la saturación. En Debian gnu/Linux la instalación de MySQL incluye la configuración de logrotate (en `/etc/logrotate.d/mysql-server`) para la rotación de este log.

5.3.3. Binary Log

Este fichero de log registra todas las operaciones que modifican (o podrían modificar) datos de la base de datos. Es decir, no se registran operaciones de consulta (SELECT, SHOW...) El registro se realiza tras la ejecución de cada sentencia, antes de liberar los semáforos.

Este log es imprescindible para la recuperación de bases de datos ante caídas del sistema y para la replicación.

Ciertamente, afecta al rendimiento del sistema, pero se estima que esta pérdida de rendimiento no supera el 1% aproximadamente.

Este log se configura en la sección [mysqld] de my.cnf con la siguiente directiva:

```
log_bin = /var/log/mysql/mysql-bin.log
```

Se puede establecer qué bases de datos utilizarán este log, y así evitar la pérdida de rendimiento, con las directivas binlog-do-db y bin-log-ignore-db.

Los ficheros del log binario se guardan permanentemente hasta que son eliminados mediante el comando:

```
PURGE MASTER LOGS TO nombre_fichero_log.número
```

Este comando borrará todos los ficheros de log anteriores al indicado.

5.3.4. Slow Queries Log

Por último, MySQL tiene la opción de activar un log especial para consultas lentas. Este fichero se configura en la sección [mysqld] del fichero `my.cnf` con las directivas:

```
log_slow_queries = /var/log/mysql/mysql-slow.log
long_query_time = 2
log-queries-not-using-indexes
```

Con estas directivas, se registrarán en el fichero indicado las consultas que tarden más de 2 segundos en ejecutarse. También se registrarán las consultas que no utilicen índices para su ejecución.

Ten en cuenta que el hecho de que una consulta, alguna vez tarde más de dos segundos no significa que sea necesariamente lenta. Puede que en esa ocasión ocurriera algo en el servidor (un proceso de backup al mismo tiempo que la consulta, un cliente lento...). Revisa las consultas registradas y céntrate en optimizar las que más se repitan.

5.4. Seguridad en el Sistema Operativo

De poco nos valdrá configurar perfectamente los privilegios de nuestros usuarios si algún intruso es capaz de acceder al sistema con privilegios especiales (root).

Un usuario con privilegios podría realizar copias de las bases de datos (las tablas MyISAM son portables) y llevárselos a otro sistema para acceder a ellas.

Como administradores del servidor, nos enfrentamos a dos riesgos principales:

a) Que alguien acceda a información a la que no debería acceder, violando así la privacidad de los datos.

b) Que alguien modifique datos que no debería modificar.

El segundo ataque puede ser muy dañino. Especialmente si pasa desapercibido y se prolonga en el tiempo... ¿cómo recuperaríamos unos datos modificados subrepticiamente desde hace seis meses?

5.4.1. Recomendaciones

Deberás hacerte un experto en seguridad. No solo de tu base de datos, sino también de tu sistema operativo y de todos los elementos con acceso a la base de datos (PHP, Java, etc.).

Busca libros y cursos sobre seguridad. En el caso de Debian, y en general para sistemas gnuLinux, es muy recomendable "Securing Debian Manual", que podrás encontrar en Internet (en inglés).

Pero, mientras te formas, vamos a repasar algunas recomendaciones básicas:

- **No ejecutes MySQL como root**. Cierto es que los procesos de instalación ya hacen esto por ti. Crean una cuenta de usuario (mysql) y configuran el sistema para que MySQL sea ejecutada por ese usuario. No lo cambies. De lo contrario, cualquier error en el código de MySQL (y los hay) podría dar acceso como root a un cracker avispado.
- **Mantén tu sistema actualizado**. Suscríbete a las listas de distribución no solo de MySQL, sino también de tu distribución gnuLinux, PHP, Java, etc. Y presta especial atención a los avisos sobre MySQL. Actualiza tu sistema periódicamente (por si algún aviso se te hubiera pasado).
- **Limita los usuarios en el host de la base de datos**. No todo el mundo necesita una cuenta de usuario para trabajar con MySQL. Limítate a darles permisos de acceso desde sus respectivas máquinas y evita crear cuentas de usuario innecesarias en la máquina del servidor. Divide claramente los procesos de trabajo de los desarrolladores del de los administradores de sistemas y bases de datos. Realmente, a tu servidor, sólo necesitarán acceso estos últimos. Al instalar aplicaciones, cambia las contraseñas de acceso de los programas.
- **Haz que te auditen**. Aunque todo el mundo tiende a ver las auditorías como una intromisión y una crítica al trabajo realizado, son en realidad oportunidades únicas de recibir ayuda para mejorar el sistema. Un buen auditor te puede enseñar mucho de seguridad. Si no tienes auditorías internas, busca un auditor externo.
- **Haz Buenas Copias de Seguridad**. Lo veremos más adelante, pero las copias de seguridad son críticas para poder recuperar un sistema tras un ataque. Traza una buena estrategia.

5.5. Seguridad Externa

Al hablar de seguridad externa, casi siempre nos viene a la mente la imagen de un cracker oscuro quien, escondido desde algún cubículo en un remoto país de la antigua Unión Soviética, realiza denodados esfuerzos por acceder ilegalmente a nuestro sistema.

Pero los ataques más habituales no son tan "de película". Curiosamente, los más peligrosos (según las estadísticas) son los usuarios de nuestra propia red.

De todas formas, tenemos que defender nuestro sistema de unos y otros.

5.5.1. Sistemas Monoservidor

Una de las configuraciones más habituales de MySQL es como base de datos de Apache para aplicaciones LAMP.

Bien, pues estamos de suerte.

Esta configuración es de las más seguras. Como Apache es un usuario de la misma máquina, no es necesario permitir el acceso a MySQL desde ningún otro servidor.

Además de configurar adecuadamente la tabla de host. Podemos indicarle a MySQL en el arranque que no acepte ninguna conexión externa vía TCP/IP. Para ello, basta con incluir la opción `skip-networking` en la sección `[mysqld]` del fichero `my.cnf`,

Y asunto terminado. El resto de opciones que expondremos a continuación son para instalaciones en las que se debe permitir el acceso a clientes de MySQL desde otros ordenadores.

5.5.2. Firewalls

Los firewalls constituyen un magnífico sistema de seguridad. Podemos poner auténticos cortafuegos en nuestra red o limitarnos a configurar el cortafuegos propio del sistema gnuLinux que alberga nuestro servidor.

Pero lo mejor es que hagamos ambas cosas. Así, si el firewall externo deja de funcionar (o es sustituido), nuestro servidor seguirá estando protegido.

Los firewall actúan igual que los comandos GRANT, estableciendo desde que hosts se admiten conexiones y de qué tipo (indicando, entre otras cosas, el puerto de conexión).

Normalmente los firewalls externos caen fuera de la responsabilidad del administrador de base de datos. Pero como muchas veces se trata de la misma persona o del mismo equipo, lo mejor que puedes hacer es configurarlos de la siguiente forma:

- Establece como política por defecto el rechazo a cualquier conexión.
- Acepta explícitamente conexiones al puerto 3306 para MySQL. Si es posible, indica desde que hosts se permitirán.
- Da acceso a los puertos de servicio que necesites (SSH), pero indicando los hosts.

5.5.3. Quita el Gateway

Normalmente, si el servidor sólo aloja MySQL, no tendrá que iniciar comunicaciones con nadie. Así que tener correctamente configurado el gateway para acceder a hosts remotos sólo será útil para potenciales crackers que accedan al sistema.

Sin gateway, el sistema seguirá respondiendo correctamente a las conexiones iniciadas por hosts remotos que hayan superado los controles previos. Y podrá comunicarse con cualquier host de su propia red.

5.5.4. Comunicaciones Encriptadas

Otro nivel adicional de seguridad se consigue encriptando las comunicaciones entre el cliente y el servidor.

¡OJO! En aplicaciones sujetas a la LOPD[10] con nivel de seguridad 3 (el máximo), es un requerimiento obligatorio que cualquier comunicación de datos se realice encriptada.

La encriptación impide que "mirones" externos puedan acceder a la información transmitida utilizando simplemente un sniffer. Además, añade la ventaja de comprimir los datos, con lo que ahorramos ancho de banda.

Tenemos tres opciones para crear estos "túneles" de comunicación:

Red Privada Virtual

Podemos crear una Red Privada Virtual (VPN) entre el cliente y el servidor. De esta forma, cualquier comunicación entre ellos estará encriptada, incluyendo las operaciones de MySQL.

Secure Sockets Layer (SSL)

MySQL incluye soporte nativo para comunicaciones SSL.

Desgraciadamente, no suele ser una de las opciones en las distribuciones más habituales de MySQL, por lo que tendremos que compilar nosotros mismos el código fuente[11].

Una vez hecho, podremos lanzar comandos GRANT con condiciones específicas de SSL:

- **REQUIRE SSL**: Obliga a que la comunicación sea encriptada por SSL para permitir la conexión de la cuenta (usuario/clave/host).
- **REQUIRE x509**: Obliga a que, además, el certificado del cliente sea verificable por una autoridad de certificación (CA) reconocida por el servidor.
- **REQUIRE SUBJECT dn**: Indica el certificado de cliente que adquirirá la autorización. Se indica mediante el nombre distinguido (dn).
- **REQUIRE ISSUER dn**: Indica qué emisor debe haber emitido el certificado de cliente. Muy útil si todos los certificados son de una misma CA (incluso la propia).
- **REQUIRE CIPHER tipo**: Requiere un tipo/s concreto de algoritmo de cifrado, teóricamente aquellos que merecen confianza por su robustez.

Varias de estas opciones se pueden combinar con AND.

10 Ley Orgánica de Protección de Datos Personales (España).

11 No es tan difícil como parece. La mayoría de administradores de MySQL lo hacen para elegir personalmente las opciones con que compilarán su servidor. Basta con lo de siempre "configure", "make" y "make install".

Secure Shell (SSH)

Si no quieres liarte con SSL, puedes recurrir fácilmente a SSH, que está habitualmente disponible en muchos sistemas.

SSH puede utilizarse para establecer un tunel de comunicación entre dos ordenadores.

Para ello, basta con abrir el túnel con el comando:

```
$ ssh -N -f -L 4406:dirección_del_destino:3306
```

Con este comando abriremos un túnel desde nuestro puerto 4406 al puerto 3306 del destino.

Ahora podemos conectarnos con el servidor del destino a través de nuestro propio puerto 4406 con el comando:

```
$mysql -h 127.0.0.1 -P 4406
```

5.5.5. TCP Wrappers

Antes de que existieran los firewalls, la seguridad perimetral de muchos sistemas se confiaba al sistema TCP/Wrappers. Ese sistema sigue activo y desempeñando un papel fundamental en la seguridad de nuestros sistemas.

Como además, MySQL puede compilarse con soporte para TCP/Wrappers, no hay razón para no utilizarlo y añadir otro nivel de seguridad a nuestro sistema.

Una vez disponible, podemos configurar los ficheros `/etc/hosts.deny`, `/etc/hosts.deny` y `/etc/services` para indicar desde qué hosts y subredes permitiremos el acceso al puerto de MySQL.

5.5.6. Bloqueo Automático de Hosts

Pero incluso aunque todo lo anterior fallara, MySQL incluye un último mecanismo para impedir el acceso, el bloqueo automático de hosts.

Si un cracker consigue llegar hasta nuestro servidor, todavía tendrá que reventar la contraseña de algún usuario de MySQL para tener acceso a los datos. El bloqueo automático le obliga a hacerlo sin superar un número máximo de intentos. Si lo supera, cualquier conexión desde ese host será rechazada.

El número máximo de intentos antes del bloqueo se indica en la variable de servidor max_conections_errors y tras superarse generará un mensaje de error en el log de MySQL.

Para liberar el bloqueo, naturalmente después de haber tomado las medidas de seguridad oportunas, bastará con ejecutar el comando FLUSH HOSTS. Este desbloqueará todos los servidores bloqueados. No hay forma de desbloquear uno concreto.

5.6. Encriptación de Datos

En ocasiones no es suficiente con encriptar las comunicaciones. Si nuestros datos son altamente sensibles, querremos encriptar el mismísimo almacenamiento físico.

5.6.1. Contraseñas

Normalmente las contraseñas de nuestros usuarios, tanto las de MySQL como las de cualquiera de las aplicaciones soportadas por el servidor, constituyen un dato crítico que no querremos almacenar "en claro".

No resulta conveniente que el administrador de base de datos tenga opción de conocer las contraseñas. Sobre todo, si podemos evitarlo.

Para ello, debemos guardar las contraseñas encriptadas. Pero, en lugar de utilziar un algoritmo de encriptación que permitiría, con tiempo y recursos, desencriptar las contraseñas, es preferible almacenar un hash[12].

Almacenando un hash de la contraseña, la única forma de romperlas será probando todas las entradas posibles hasta dar con una que genere el mismo hash. Y eso requiere mucho, muchísimo tiempo y recursos.

MySQL tiene varias funciones de hash: PASSWORD, OLD_PASSWORD, ENCRYPT, SHA1 y MD5.

Así, la inserción de una contraseña sería:

```
INSERT INTO usuario (nombre, clave) VALUES ('juan',
PASSWORD('miClave'));
```

De esta forma, la tabla almacena el hash, no la clave.

En el caso de aplicaciones informáticas (el más habitual), es preferible que el hash se realice en el cliente, evitando así que la clave en claro viaje por la red.

Para comprobar la clave de un usuario, la consulta sería:

```
SELECT * FROM usuario WHERE nombre = $nombre AND clave =
PASSWORD($clave);
```

De nuevo, es preferible que el hash se haga en el cliente.

5.6.2. Sistemas de Ficheros

Algo más radical es encriptar todo el sistema de ficheros que da soporte al almacenamiento de MySQL.

En este caso, no tendremos que preocuparnos de nada, porque el sistema de ficheros (VFS) se encargará de encriptar y desencriptar los datos según necesitemos, bastará con tener en cuenta el consumo de CPU que conllevan estas operaciones.

Sin embargo, esta opción puede ser un verdadero problema a la hora de realizar copias de seguridad. Ya no podemos simplemente copiar los ficheros de datos, porque sólo el sistema de ficheros sería capaz de desencriptarlos. Tendremos que realizar descargas de datos desencriptados y volver a encriptarlos sobre el soporte de copia de seguridad. Otra opción es copiar directamente toda la partición de disco encriptada... claro, que necesitaremos bastante espacio.

12 Un hash es una función que, partiendo de un dato de entrada, genera otro de salida. Pero no hay relación biunívoca, por lo que conociendo el dato de salida no es posible deducir el de entrada.

5.6.3. Encriptación a Nivel de Aplicación

Una opción habitual es dejar que sean las aplicaciones las que encripten sus propios datos críticos.

La ventaja principal es que ya no hay ningún problema con los backups. Pero en cambio, el acceso a los datos encriptados sólo podrá realizar a través de las aplicaciones que son capaces de desencriptarlos. Esto reduce mucho la reutilización.

Pero además hay otro problema. MySQL, y cualquier otra base de datos, opera mal con los datos encriptados.

En primer lugar, los índices no funcionarán tan bien al tener que realizarse sobre datos encriptados. En muchas ocasiones, la encriptación hace que valores de unos pocos bytes se transformen en valores mucho más largos, con los consiguientes problemas de almacenamiento e indexación.

Pero, al mismo tiempo, MySQL no puede utilizar los valores encriptados, ni para búsquedas (WHERE valor_encriptado > 1000) ni para funciones de agregación (SUM(valor_encriptado)).

Estas operaciones requieren que la aplicación descargue todos los datos, los desencripte y realice el trabajo que MySQL podría haber realizado fácilmente.

Lo mismo ocurre con la optimización de consultas. MySQL no puede elegir qué índices si hay valores encriptados en la consulta.

5.6.4. Encriptación a Medida

Tenemos una última opción. Sin duda complicada pero muy flexible.

Al ser MySQL software libre, podemos programar nuestro propio manejador de tablas (motor de almacenamiento), que encripte y desencripte como queramos.

El código de MySQL incluye un motor de ejemplo (EXAMPLE) del que se puede partir para facilitar esta labor.

Naturalmente, es necesario dominar C++ para hacerlo, pero no es difícil encontrar buenos programadores en este lenguaje con amplios conocimientos de bases de datos e incluso conocimientos específicos de MySQL.

5.7. MySQL con chroot

Avanzando en la seguridad de nuestro servidor, tenemos la opción de ejecutar MySQL en una “jaula chroot”.

Al ejecutar MySQL en una jaula chroot, limitamos el acceso del proceso a los ficheros contenidos en la jaula (un directorio y sus subdirectorios).

Con esto evitamos que algún bug pueda ser utilizado para acceder a ficheros sensibles del sistema, que siempre estarán fuera de la jaula chroot.

Para ejecutar MySQL con chroot tenemos que recompilarlo. Esto es necesario porque, en su formato habitual, MySQL accede a ficheros de otros directorios (/usr/bin, /var/lib, etc.) y debemos cambiar las opciones de compilación para que instale y busque todos sus

ficheros en el mismo directorio de ejecución.

Para ello, en el proceso de compilación indicaremos la opción

```
--prefix=/chroot/mysql
```

siendo /chroot/mysql el directorio en el instalaremos y ejecutaremos MySQL.

Podremos ejecutar MySQL con el comando:

```
# mysqld_safe --chroot=/chroot/mysql  --user=1001
```

Necesitamos indicar el usuario con su UID porque MySQL no podrá acceder a los programas de búsqueda por nombre.

5.8. Copias de Seguridad

Todos sabemos que es importantísimo hacer copias de seguridad... pero también es cierto que, normalmente, es la tarea más olvidada salvo que se automatice. E incluso en ese caso, sigue siendo una tarea olvidada, que no se comprueba ni se revisa.

Hacer copias de seguridad de nuestros datos es crítico, pero ¿porqué?

Hay dos motivos fundamentales para realizar copias de seguridad:

Catástrofes: Es posible que nuestros datos desaparezcan. Puede ser por un borrado inesperado, un error de hardware, un incendio o un terremoto, todo es posible e improbable. Pero aunque la probabilidad de una catástrofe sea baja (o no tan baja como imaginamos), ¿podemos asumir el riesgo?

Auditorías: En ocasiones necesitaremos auditar nuestros datos. Saber qué información teníamos en el pasado, como crece nuestra base de datos, corregir problemas que vienen de lejos... En algunos casos, disponer de los datos antiguos es una obligación legal. Este es el caso de la LOPD.

Pero incluso hay otras situaciones en las que dominar las técnicas de realización y recuperación de copias de seguridad nos resultará muy útil. Por ejemplo, para realizar pruebas de nuestro servidor querremos "poner y quitar" distintas copias de nuestros datos. También necesitaremos proveer a los desarrolladores de aplicaciones con copias de datos reales o casi reales para que puedan hacer bien su trabajo.

Todos esto y alguna cosa más requiere conocer las técnicas para realizar y restaurar copias de seguridad.

5.8.1. Consideraciones

Antes de empezar a hacer copias como locos y llenar cintas o DVDs, tendremos que tener en cuenta varias consideraciones:

Volcados o Copias Raw

Podemos optar por volcar nuestros datos a ficheros de texto en formato SQL, incluyendo sentencias CREATE e INSERT y disponer así de copias en texto plano tanto de la estructura de nuestra base de datos como de su

contenido.

O podemos realizar directamente copias de los ficheros de almacenamiento físico.

El volcado es más flexible. Nos permitirá editar los datos, consultarlos directamente sin necesidad de cargarlos, etc. Pero pueden requerir mucho espacio de disco incluso haciéndolos comprimidos. Además, requieren mucho consumo de CPU tanto para generarse como para cargarse. En este último caso, la base de datos deberá regenerar todos los índices. Además, durante el volcado tendremos que bloquear el acceso a las tablas.

La copia directa de ficheros es más rápida. Pero no permite editar los datos ni consultarlos si no los cargamos en el servidor. Además, requiere detenerlo para realizar la copia.

En línea o en Batch

¿Vamos a detener el servidor para realizar la copia de seguridad o debemos mantenerlo en marcha?

Si podemos detenerlo, la cosa es fácil. Podremos hacer copia directa de los ficheros o, si optamos por el volcado (cortando el acceso a los usuarios), podremos volcar las tablas sin preocuparnos de los bloqueos.

Mantener la Consistencia

En caso de realizar copias en línea, deberemos preocuparnos de la consistencia del volcado.

No basta con bloquear una tabla, volcarla y bloquear la siguiente. Si las tablas están relacionadas, y esto es lo más habitual en las bases de datos "relacionales", es posible que una operación de otro usuario haya insertado o borrado datos de otra tabla relacionada con la que estamos volcando. Por ejemplo, podría borrarse la ficha de un cliente mientras nosotros volcamos la lista de pedidos... incluyendo los del cliente desaparecido.

Si optamos por el volcado en línea, debemos trazar una delicada estrategia de bloqueo.

Espacio de Almacenamiento y Tiempo de Vida

Por último, debemos tener en cuenta qué espacio ocupan nuestras copias de seguridad y qué tiempo necesitamos mantenerlas.

Es conveniente planear una estrategia a largo plazo y no dejarnos sorprender por una falta de espacio no prevista. Por ejemplo, si nuestros datos van a crecer mucho y empezamos a realizar copias en CDRom, llegará un día en el que necesitemos varios para copia. Más vale que tengamos la circunstancia prevista.

5.8.2. Estrategias de Backup

Bien, dicho esto, tenemos varias estrategias posibles.

Podemos parar el Servidor

Si podemos detener el servidor, por ejemplo por la noche para realizar copias de seguridad, la cosa es tremendamente fácil.

Podemos optar por cortar el acceso a los usuarios (por ejemplo, deteniendo Apache) y volcar los datos a un fichero de texto. Conviene mandar el fichero a otra ubicación física. Los incendios no distinguen entre datos reales y copias de seguridad.

Si nuestra base de datos es muy grande, podremos optar por hacer copias de los ficheros de almacenamiento.

Incluso en el caso de que no necesitemos auditar los datos, nos bastará con mantener la última copia de estos.

No Podemos parar el Servidor

La cosa se complica.

La opción más adecuada es instalar un servidor de replicación. Muchos administradores montan servidores de replicación sólo para realizar copias de seguridad.

El sistema es sencillo. Llegado el momento de realizar la copia de seguridad, detienes el servidor de replicación y haces la copia de sus datos. Al volver a arrancar se sincronizará con el servidor maestro. No te olvides de copiar también los ficheros de replicación.

Tampoco Hay Servidor de Replicación

Si nuestra base de datos es muy, muy grande puede ser inviable económicamente disponer de un servidor de replicación.

En ese caso, deberemos trazar nuestra propia estrategia de copias de seguridad.

Lo más conveniente es diseñar un script que, conociendo las tablas y sus relaciones, vaya bloqueando grupos de tablas y volcando sus datos.

Uso de Aplicaciones de Backup

Las aplicaciones genéricas de backup suelen estar poco preparadas para realizar copias de seguridad de bases de datos.

Puede ocurrir que al copiar los ficheros de datos, como hacen estas aplicaciones, no copien inmediatamente otros ficheros relacionados (índices, cachés, logs...) y que cuando copien estos, ya hayan sido modificados. Acabaremos entonces con una copia inconsistente de nuestra información.

Si vas a utilizar aplicaciones de terceros para copias de seguridad,

revisa bien cual es su funcionamiento.

5.8.3. Herramientas y Técnicas

Tenemos varias herramientas que nos ayudarán en nuestra labor de realizar y recuperar copias de seguridad.

mysqldump

Es un comando que permite realizar volcados de tablas y bases de datos a ficheros de texto SQL.

Muy útil para bases de datos pequeñas.

La recuperación de volcados de mysqldump se realiza directamente con el comando mysql, utilizando como entrada al mismo el fichero de volcado.

mysqlhotcopy

Es un comando para realizar copias en línea de la base de datos.

mysqlhotcopy bloquea las tablas y genera copias de los ficheros de almacenamiento en el directorio indicado.

Si hay mucho tráfico o muchas tablas, el bloqueo de mysqlhotcopy puede influir en el rendimiento.

La recuperación es sencilla. Paras el servidor y copias los ficheros en el directorio.

Capítulo 6. Optimización

6.1. Índices

Los índices en base de datos nos van a permitir encontrar rápidamente los datos que buscamos. Técnicamente, son estructuras de datos que asocian un criterio de ordenación con la posición de una fila.

Podemos compararlos con el índice de términos utilizados en un libro. En él, podemos buscar un término (por ejemplo, "encriptación") y el índice nos dirá en qué páginas podemos encontrar esa palabra.

La diferencia fundamental entre el índice de un libro y el de una base de datos es que el primero no cambia, mientras que la base de datos debe luchar por mantener el índice actualizado mientras los usuarios insertan, borran y modifican la información indexada.

Pensemos por un momento en una tabla de clientes. Queremos buscar los datos de un cliente con el apellido "Stephanovski". Es un apellido raro y probablemente no habrá muchos. ¿Qué hace la base de datos para encontrar los datos?

Pues, sin otra ayuda, tendrá que leer cada registro de la tabla (por millones que sean) y comparar el campo apellido con la cadena de texto buscada. Esta operación, así descrita, puede suponer un alto tiempo de ejecución, gran consumo de disco y la contaminación de la caché de disco del sistema. Casi nada.

Sin embargo, esta misma operación apenas nos lleva unos segundos en la guía telefónica.

Para eso están los índices.

Compromiso

Los índices no salen gratis. Como casi todo en esta vida se trata de un equilibrio. Si definimos índices en nuestras tablas, ciertamente estaremos acelerando las consultas a nuestros datos. Pero, al mismo tiempo, estaremos penalizando las inserciones y actualizaciones porque, en cada una de estas operaciones, el servidor tendrá que actualizar la información del índice.

También, el uso de índices supone un aumento del espacio de disco consumido.

Piensa qué te conviene más.

6.1.1. Tipos de Índices

MySQL nos ofrece muchas alternativas para elegir los índices que queremos crear en nuestra base de datos.

Parciales

En lugar de indexar todo el contenido de un campo de la base de datos, podemos indicarle a MySQL que haga un índice parcial.

```
ALTER TABLE cliente ADD INDEX (apellidos(5))
```

Multicolumna

También podemos utilizar varias columnas para definir un índice:

```
ALTER TABLE cliente ADD INDEX (apellidos, nombre)
```

O incluso hacer índices multicolumna parciales:

```
ALTER TABLE cliente ADD INDEX (apellidos(4),
nombre(5))
```

Incrustados (Clustered)

Las tablas innoDB tienen una característica especial en el índice asociado a su clave primaria. En lugar de almacenar el índice en un área diferente de los datos (como hacen las tablas MyISAM), los datos se almacenan ordenados por clave primaria. Es decir, con el índice incrustado en los datos.

Así, mientras que en una tabla MyISAM, la búsqueda por índice requiere dos operaciones (leer el índice y leer los datos), con índices incrustados estas búsquedas se resuelven en una sola operación. Una vez encontrada la entrada del índice, ya se han encontrado los datos.

Textuales (Full-Text)

Por su parte, las tablas MyISAM también tienen su punto fuerte. Son los índices textuales.

Con esta opción, MySQL crea un índice sobre las palabras de un campo de la tabla. Esta opción proporciona soporte a búsquedas textuales.

```
ALTER TABLE cliente ADD FULLTEXT (apellidos)
```

Las sentencias de búsqueda que explícitamente hacen uso del índice son:

```
SELECT * FROM CLIENTE WHERE MATCH(apellidos)
AGAINST("Pérez")
```

6.1.2. Estructuras

Para el almacenamiento de los índices existen diversas estructuras

B-Tree

Como su nombre indica, se trata de un índice con una estructura de árbol. Es el índice más utilizado en bases de datos por su flexibilidad, rendimiento y uso de espacio.

Los árboles balanceados tienen la características de que cualquier búsqueda tiene el mismo tiempo de resolución. Frente a los árboles "binarios", los B-Trees tienen la ventaja de que cada nodo puede tener múltiples hijos, evitando que el árbol crezca demasiado en profundidad.

Los índices B-Tree facilitan la resolución de consultas por rango (BETWEEN) y las consultas relacionadas por > y <.

Hash

La utilidad de un índice depende de la distribución de sus valores. Una distribución uniforme garantiza que se podrán descartar filas rápidamente.

Esta es la estrategia de los índices Hash, también muy populares entre las bases de datos. Son índices planos, que almacenan el valor del hash de la clave asociándole el offset del registro o registros asociados.

Son muy rápidos para las búsquedas por clave pero, en las búsquedas por rango no pueden ser utilizados.

Tienen el problema de que, en algunas ocasiones, puede que el hash no produzca la distribución homogénea de resultados que buscamos. Sin embargo, suelen ocupar menos espacio que los B-Trees.

R-Tree

Este es un tipo de índice especial, utilizado para las tablas espaciales.

Mediante este índice, MySQL puede saber rápidamente si un conjunto de puntos están situados dentro o fuera de una determinada forma geométrica.

6.1.3. Motores de Almacenamiento e Índices

Cada motor de almacenamiento tiene su propio índice asociado. Desgraciadamente, es muy poco lo que el diseñador de bases de datos puede elegir. Pero sí es conveniente saberlo para tenerlo en cuenta.

MyISAM

Las tablas MyISAM utilizan índices B-Tree, full-text y R-Tree. No pueden utilizar índices hash.

Además, tienen algunas características para mejorar el rendimiento de los índices B-Tree:

- Compresión de prefijos en claves alfanuméricas. También llamado "compresión de prefijos". Si varias entradas de la columna indexada tienen el mismo prefijo (por ejemplo "http://") este será comprimido para ahorrar espacio de almacenamiento.
- Compresión de bytes altos en claves numéricas. También llamado "empaquetamiento de claves". Es la misma idea que con la claves alfanuméricas, pero aplicada al byte alto de las claves numéricas.

Los índices de las tablas MyISAM se guardan los ficheros con el mismo nombre que la tabla y extensión .MYI.

MySQL tiene la opción de posponer las actualizaciones del índice de una tabla para cargas masivas de datos.

HEAP

Las tablas HEAP, en memoria, pueden utilizar índices Hash (por defecto) o B-Tree. La opción de B-Tree y Heap ofrece magníficos tiempos de respuesta.

```
CREATE TABLE... INDEX USING BTREE(columna))...
```

InnoDB

Las tablas InnoDB utilizan índices B-Tree incrustados. Esto las hace muy rápidas en las búsquedas por clave primaria. Además, al tratarse del índice, es frecuente que la página con la clave y los datos ya se encuentre en la caché de disco.

6.1.4. Limitaciones en el Uso de Índices

Pero hay distintas situaciones en las que los índices no podrán ser utilizados por MySQL, y conviene tenerlas en cuenta:

Búsquedas con comodín

En una consulta del tipo

```
SELECT * FROM cliente WHERE apellidos LIKE '%er%'
```

un índice sobre apellidos resulta totalmente inútil. MySQL tendrá que escanear toda la tabla... salvo que utilicemos un índice textual y modifiquemos la consulta para utilizarlo.

Búsquedas con expresiones regulares

La situación es análoga si utilizamos búsquedas con expresiones regulares. En la consulta:

```
SELECT * FROM cliente WHERE apellidos RLIKE 'ez$'
```

Simple y llanamente, MySQL no intentará optimizar búsquedas basadas en expresiones regulares.

Problemas de Estadísticas

Por último, si MySQL prevé que una consulta tendrá que devolver más del 30% de las filas, considerará (y con razón) que utilizar el índice será más lento que realizar una búsqueda completa.

¿Y cómo lo sabe?

MySQL guarda estadísticas de cada tabla que le ayudan a prever el resultado que tendrá una determinada estrategia de resolución.

6.1.5. Mantenimiento

Tener los índices en buen estado puede ser crítico para el rendimiento de nuestro servidor. Afortunadamente hay pocas cosas que tengamos que hacer.

En primer lugar, si quieres conocer las características de un índice, debes utilizar el comando

```
SHOW INDEXES FROM tabla
```

Obtendrás información detallada del índice o índices definidos para la tabla.

Además, si ves que un índice está fallando (Ver Optimización) puedes optimizarlo refrescando sus estadísticas. Para ello, utiliza el comando

```
OPTIMIZE tabla
```

Así, MySQL releerá la tabla y recreará los índices actualizando sus estadísticas.

6.2. Consultas

Además del diseño del modelo de datos, con la creación de tablas y relaciones, la elección de los motores de almacenamiento y la creación de índices, hay otro punto en el que se puede optimizar el rendimiento de nuestra base de datos. Se trata del proceso de resolución de consultas.

Pero, para sacarle el máximo provecho, debemos saber primero cómo funciona:

6.2.1. Proceso de Consulta

Cada vez que MySQL recibe una consulta, ejecuta el siguiente proceso:

1.- La Caché de Consultas

MySQL ejecuta un algoritmo de hash sobre la consulta recibida y mira a ver si tiene alguna consulta idéntica en su caché de consultas.

Si es así, no se molesta en seguir, devuelve el resultado cacheado y, “a otra cosa”.

Este proceso de hash, aunque es muy rápido, hace que consultas

sintácticamente idénticas, pero escritas de forma diferente, no puedan ser identificadas y, por lo tanto, no aprovechen las ventajas de la caché. Así que, te cuidado con los espacios en blanco.

Afortunadamente, lo normal es que sean aplicaciones informáticas quienes lacen consultas contra la base de datos y no usuarios humanos. Así que el riesgo de que dos consultas iguales tengan diferente sintaxis es muy bajo.

MySQL hace alguna "trampa". En realidad sólo se molesta en "hashear" las consultas de tipo "SELECT", puesto que el resto no estarán cacheadas.

La caché de consultas se activa con la variable query_cache_type (en el fichero my.cnf).

Un valor a 1 hará que todas las consultas de tipo SELECT sean cacheadas, salvo que expresamente indiquen lo contrario mediante el modificador SQL_NO_CACHE.

Un valor de query_cache_type igual a 2 hará lo contrario. Las consultas no serán cacheadas salvo que expresamente lo soliciten con SQL_CACHE. Así es posible controlar y mejorar la calidad de la caché.

```
SELECT SQL_CACHE FROM cliente;
```

2. Parseado

Si la consulta no está cacheada, MySQL la parsea. El parseo tiene como objetivo comprobar la corrección sintáctica de la consulta y dividirla en sus elementos más básicos.

Para ello, identifica de qué tipo de consulta se trata (SELECT, INSERT, GRANT, ALTER...), qué tablas están involucradas, qué dice la clausula WHERE y si hay alguna pista para la resolución.

3. Planificación

Tras el parseo, MySQL debe decidir cómo va a resolver la consulta. Esta es la parte más interesante y a la que dedicaremos luego más espacio. Su objetivo es establecer el plan más conveniente para resolver la consulta.

4. Ejecución

Y esta es la parte más sosa. MySQL ejecuta el plan diseñado para resolver la consulta.

Vamos con el Planificador

6.2.2. El Planificador

El planificador de MySQL es un elemento crítico para el rendimiento. Está en constante evolución y recoge la experiencia y sabiduría acumulada por los programadores de MySQL.

Ahora bien, el planificador es genérico para cualquier consulta, así que no está específicamente diseñado para tu base de datos ni para tus problemas. Por eso es muy conveniente que conozcas cómo funciona y sepas resolver posibles problemas.

Sí, es posible aunque poco frecuente, que el Planificador de MySQL tome decisiones que no son las más adecuadas para alguna consulta concreta. Así es la vida.

Pero con lo que vamos a ver, podrás detectarlo y corregirlo.

Ten en cuenta que el planificador debe tomar una decisión muy rápida sobre el mejor camino a seguir. No puede probar todas las opciones porque pasaría más tiempo planificando que resolviendo. Y debe tomar sus decisiones con información parcial sobre, por ejemplo, la efectividad de un índice o el orden más conveniente de realizar una cadena de joins.

Afortunadamente podemos saber qué está pensando recurriendo al comando EXPLAIN.

El Comando EXPLAIN

El comando EXPLAIN nos va a mostrar el plan de ejecución que MySQL tiene para una determinada consulta.

```
EXPLAIN SELECT * FROM cliente WHERE id=20 \G
***************************1. row ***************************
id: 1
select_type: SIMPLE
table: cliente
type: const
possible_keys: PRIMARY
key: PRIMARY
key_len: 4
ref: const
rows: 1
Extra:
1 row in set (0.00 sec)
```

Como ves, EXPLAIN devuelve un registro por cada tabla involucrada en la consulta con una serie de campos que debemos interpretar. Vamos a ello:

- ***id:*** Identifica cada consulta de la consulta... si hay consultas anidadas.
- ***select_type:*** Indica el papel de la tabla en la consulta. Los valores posibles son: SIMPLE, PRIMARY, UNION, DEPENDENT UNION, SUBSELECT y DERIVED.

- ***table:*** Nombre de la tabla de la que MySQL leerá los registros.
- ***type:*** Indica el tipo de join que efectuará MySQL. Los valores posibles son: CONST, SYSTEM, EQ_REF, REF, RANGE, INDEX y ALL.
- ***possible_keys:*** Lista de índices que MySQL puede utilizar para resolver la consulta.
- ***key:*** Nombre del índice que MySQL ha elegido para resolver la consulta. MySQL sólo utilizará un índice por consulta.
- ***key_len:*** Tamaño del índice en bytes.
- ***ref:*** Las columnas o valores que se utilizarán en el índice.
- ***rows:*** Número estimado de filas que MySQL considera que tendrá que examinar para resolver la consulta.
- ***Extra:*** Información adicional.

El orden en el que aparecen los registros indica el orden en el que MySQL utilizará las tablas. Esto es muy relevante para el caso de joins.

```
   EXPLAIN SELECT cliente.id, cuenta.id FROM cliente JOIN cuenta
ON cuenta.id_cliente = cliente.id\G
*************************** 1. row ***************************
id: 1
select_type: SIMPLE
table: cuenta
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 1723
Extra:
*************************** 2. row ***************************
id: 1
select_type: SIMPLE
table: cliente
type: eq_ref
possible_keys: PRIMARY
key: PRIMARY
key_len: 4
ref: banco.cuenta.id_cliente
rows: 1
Extra: Using index
   2 rows in set (0.00 sec)
```

En este caso, vemos que la intención de MySQL es buscar primero en la tabla cuenta, sin utilizar ningún índice. Estima que procesará 1723 filas. Y, para cada fila de la primera tabla, tendrá que leer sólo una fila de la segunda tabla.

También nos dice que la relación entre ambas tablas es de equivalencia (eq_ref). Es decir, que realizará el join utilizando una relación de igualdad (`cuenta.id_cliente = cliente.id`) y nos indica sobre qué campo de la tabla anterior la realizará (ref). Por último, y este es un caso especial, como de la segunda tabla sólo necesita el campo id y este pertenece al índice que va a utilizar (PRIMARY), nos indica que sólo necesitará el índice (Using Index). Es decir, que no tendrá que leer la tabla de datos porque toda la información que necesita está en el índice que va a utilizar.

Podríamos obligar a MySQL a modificar su plan de ejecución si, por ejemplo, añadimos a la consulta una clausula WHERE sobre id_cliente que limite seriamente el número de resultados de esta tabla. MySQL modificará su plan de ejecución para utilizar primero cliente y restringir el número de filas antes de realizar el join.

En definitiva, podemos conocer perfectamente cómo va a resolver MySQL una consulta viendo la explicación de la misma (EXPLAIN).

Consideraciones

Hay algunas consideraciones importantes que influirán en la eficacia del planificador.

- **Falta de Diversidad:** Incluso teniendo millones de registros en una tabla, es posible que MySQL opte por no utilizar un índice si estima que no está suficientemente diversificado. Si MySQL estima que la consulta devolverá más del 30% de las filas de la tabla, considerará más conveniente escanearla completamente y no hacer dobles operaciones leyendo el índice y luego buscando los datos.
- **Ordenación por Índices:** Ordenar los resultados requiere tiempo, en ocasiones incluso más del que llevó recabar los datos. En ocasiones puedes evitarte el proceso de ordenación si utilizas un índice múltiple. Así, tus datos se recogerán ordenados.
- **Índices Textuales:** No te olvides de recurrir a los índices textuales en lugar de a los típicos LIKE.

Pistas

Podemos ayudar al planificador dándole algunas pistas sobre cómo actuar. Al fin y al cabo, nosotros conocemos nuestros datos.

Por ejemplo, podemos proponerle que considere determinados índices e ignore el resto con el modificador USE INDEX

```
SELECT * FROM tabla USE INDEX (índice1, índice2)
```

O podemos conseguir que ignore un índice determinado y elija entre el resto disponibles:

```
SELECT * FROM tabla IGNORE INDEX (índice)
```

O podemos ponernos serios y obligarle a que utilice un índice determinado, le guste o no le guste:

```
SELECT * FROM tabla FORCE INDEX (índice)
```

Si vemos que el orden de los joins que ha elegido MySQL no es el más conveniente, podemos obligarle a un orden determinado utilizando un STRAIGHT JOIN.

Si sabemos que al cliente le llevará tiempo recoger los datos de una consulta extensa, podemos pedirle a MySQL que guarde el resultado en una tabla temporal. De esta forma liberará antes los bloqueos. Para ello, utilizamos el modificador SQL_BUFFER_RESULT.

Así mismo, si prevemos que el resultado será muy grande, podemos avisar a MySQL para que lo tenga en cuenta y tome decisiones más arriesgadas. Para ello, utilizamos el modificador SQL_BIG_RESULT.

No te olvides tampoco de obligar a MySQL a cachear los resultados si sabes que se repetirá frecuentemente la misma consulta. Por ejemplo, la lista de poblaciones, países, provincias, usuarios...

Trucos Sencillos

Hay algunos trucos muy sencillos que te pueden ayudar a resolver consultas que, de otra forma, no serían tan eficientes.

Por ejemplo, hay ocasiones en que en lugar de realizar un join, es preferible dividir la consulta en dos y utilizar variables de servidor.

```
SELECT cuenta.id FROM cliente, cuenta WHERE
cliente.id = cuenta.id_cliente AND cliente.apellidos
= "López Pérez";
```

Pero también puede resolverse como:

```
SELECT @id := id FROM cliente WHERE apellidos =
"López Pérez";
SELECT id FROM cuenta WHERE id_cliente = @id;
```

Otro truco fácil es utilizar uniones en lugar de OR.

```
SELECT id_operación FROM movimiento WHERE importe
> 50000 OR importe < -50000;
```

Es lo mismo que:

```
(SELECT id_operación FROM movimiento WHERE importe
> 5000) UNION (SELECT  id_operación FROM movimiento
WHERE importe < -5000)
```

Los planes de ejecución para ambas pueden ser muy diferentes.

6.3. Rendimiento del Servidor

Sinceramente, ¡¡NO DEBES LEERTE ESTE APARTADO!!

Muchas veces, los administradores de bases de datos optan por la solución fácil... más hardware. Y es un error.

Las mejoras en el servidor, por muy efectivas que puedan ser, sólo podrán mejorar el rendimiento de la base de datos en un orden de magnitud. Muchas veces ni eso.

Sin embargo, las mejoras en el diseño del modelo de datos o las mejoras sobre las consultas... ¡producen mejoras de 3 órdenes de magnitud!

Así que si lo que buscas es mejorar el rendimiento de tu base de datos, ¡busca en los capítulos anteriores!

Pero, en el remoto caso de que ya hayas mejorado todo lo mejorable en el diseño de tus bases de datos y de tus consultas... y sólo en ese caso, echémosle un vistazo a lo que podemos mejorar en el servidor.

6.3.1. Factores de Rendimiento

En el rendimiento del servidor de base de datos, cada uno de los recursos hardware del ordenador sobre el que se ejecuta influye de manera diferente.

Discos

Sin duda las operaciones de E/S son las que más pueden limitar el rendimiento del servidor.

Comparado con otros elementos como la CPU o la memoria, los discos (por muy buenos que sean) son unas tortugas. Así que lo normal es que, para cualquier consulta que tenga que acudir a disco, el tiempo de las operaciones de E/S sea el mayor problema.

El tiempo de búsqueda de un disco depende de dos factores: el tiempo de posicionamiento de la cabeza y el tiempo de lectura de sector.

El primero es el tiempo necesario para desplazar la cabeza desde su posición actual hasta el sector del disco que contiene los datos. El segundo valor es el tiempo que tarda la cabeza en acceder al inicio de los datos y leerlos. Este último tiempo depende de la velocidad de rotación del disco.

Y aquí tenemos un punto de mejora. Si puedes optar por discos de 20.000

RPM, mejor que si son de 15.000. El dinero que inviertas te será recompensado.

Otra opción es utilizar RAIDs. Las distintas configuraciones de discos en RAID te permitirán mejorar en dos aspectos: fiabilidad y rendimiento.

Respecto a fiabilidad, algunas configuraciones garantizan el funcionamiento aun en el caso de fallo en uno de los discos. Son configuraciones con redundancia, como RAID 1 o RAID 10.

Otras configuraciones, permiten que una misma operación de lectura sea atendida por varios discos, haciendo que el tiempo de acceso sea menor, como es el caso de RAID 5 o RAID 10.

Memoria

Después de los discos la memoria RAM es el siguiente factor limitante.

En los sistemas Linux, se utiliza la memoria RAM como caché, tanto de disco como de MySQL.

Si la memoria es suficientemente grande, y no hay otras aplicaciones usándola, es posible que al realizar una consulta parte de los datos ya estén cacheados. O, si no son los datos, es muy probable que una parte o incluso todo el índice de una tabla accedida frecuentemente esté disponible en la caché, ahorrando un tiempo valioso.

Por eso eso es crítico intentar evitar la polución de caché que se produce al realizar búsquedas en toda una tabla sin utilizar índices. O al menos, establecer una política para la caché de consultas de MySQL.

Por otra parte, si tu base de datos ocupa 512 Mbytes y el servidor tiene 1 Gbyte, añadir más memoria no servirá de nada.

CPU

Es raro que la CPU actúe como limitador del rendimiento de la base de datos. Comparada con la memoria y con los discos es mucho más rápida.

Sin embargo, hay algunas operaciones que sí consumen CPU. Este es el caso de la ordenación de resultados y de las operaciones sobre resultados (matemáticas, funciones de strings, etc.)

Al realizar operaciones como CHECK TABLE sí es posible que el consumo de CPU sea muy intensivo, igual que al realizar consultas sin índices.

Red

No es frecuente tener problemas de congestión de red con MySQL, salvo que esta esté mal configurada.

La única configuración algo delicada es la replicación. Si 50 esclavos intentan replicarse en línea con un único maestro, es posible que ese interfaz esté “algo congestionado”.

6.3.2. Identificación de Problemas

Conociendo la influencia de cada elemento, tenemos que ponernos manos a la obra para determinar qué le pasa a nuestro servidor.

Para eso, tenemos que utilizar las herramientas del sistema.

El Comando top

top nos permitirá saber qué procesos consumen más recursos del sistema y algunos datos generales.

Si vemos que el uso de la CPU es cercano al 100%, obviamente tendremos un cuello de botella ahí. Pero si los procesos que más CPU consumen (y aparecen primero en top) no son de mysql... el problema no es nuestro.

Si con top vemos que nuestra CPU está relativamente tranquila, tendremos que comprobar la memoria.

top también nos ofrece esta información, podemos ver el consumo de memoria de cada proceso y el consumo general. Si la memoria de swap está muy utilizada... tenemos un problema. Y tendremos que investigar qué proceso ha sido.

Por último, si tanto la CPU como la memoria están bien, tendremos que comprobar los discos. Para eso, podemos utilizar vmstat.

El Comando vmstat

"Virtual Memory Stattistics". Este comando nos da información no solo sobre la memoria virtual, sino sobre uso de discos, traps y actividad de la CPU.

Las columnas bi y bo nos señalarán el uso que está haciendo el sistema de los dispositivos de bloques.

Si el problema es de disco

Si el proceso mysql está realizando demasiadas operaciones de disco, el problema será, probablemente, de consultas ineficientes.

La mejor solución será activar el log de consultas lentas y de consultas sin índice para luego revisarlas de una en una viendo su plan de ejecución.

Otro problema puede ser la creación de tablas temporales en disco. Al resolver una consulta, MySQL puede necesitar escribir una tabla temporal. Esto queda reflejado en plan de ejecución con el extra "Using Temporary", aunque no indica si creará la tabla en memoria o en disco.

Esto depende del valor de tmp_table_size, que por defecto está en 32MB. Las tablas temporales en disco se crean en /tmp. Para comprobar qué está pasando, puedes monitorizar las variables created_tmp_disk_tables y created_temp_tables para ver las relaciones entre ellas (MySQL Administrator).

Si la relación es mala y se están creando demasiadas tablas temporales en

disco, puedes incrementar tmp_table_size. Pero cuidado, si te pasas y los threads de MySQL intentan crear demasiadas tablas en memoria, el sistema empezará a swapear.

Otra opción es montar un sistema de discos rápidos para /tmp.

Si el problema no tampoco en las tablas temporales, puedes concentrarte en utilizar al máximo la caché de consultas. O, como última opción, cambia a un RAID 0, RAID 5 o RAID 10 que dividirán las operaciones de disco entre distintos volúmenes.

Si el problema es de CPU

En este caso debemos coger las consultas lentas y someterlas a benchmarking. Aquellas que alcancen rápidamente el 100% de utilización de CPU debemos analizarlas con cuidado.

No es posible eliminar el trabajo de CPU. Si tienes que calcular el MD5 de 100.000 registros de resultados, es posible que la CPU esté algo cargada. Pero ¿qué podemos hacer?

Las soluciones son pocas:

a) Pasar el cálculo al servidor de aplicaciones o incluso al cliente... si se deja.

b) Poner más silicio. Bien cambiar a otro procesador mejor o poner varios procesadores.

c) Poner más servidores y distribuir la carga entre ellos.

Ciertamente, si hay que realizar millones de cálculos no hay mucho que podamos hacer.

Si el problema es de Memoria

Dimensionar correctamente el uso de memoria de MySQL requiere cierto equilibrio. MySQL tiene tanto buffers globales (para todos los procesos) como buffers asociados a cada proceso. Al menos deberíamos tener memoria suficiente para los buffers globales y para el valor de los de proceso multiplicados por el número de consultas simultaneas esperadas.

Los buffers por thread son: sort_buffer, myisam_sort_buffer, read_buffer, join_buffer y read_rnd_buffer.

Y los globales: key_buffer, innodb_buffer_pool, innodb_log_buffer, innodb_additional_mem_pol y net_buffer

El problema más común es dimensionar los buffers para la aplicación principal del servidor. Luego, como todo funciona tan bien, se van añadiendo otras aplicaciones. Pero si estas tienen muchos usuarios simultáneos, consumirán más memoria de la inicialmente prevista. Entonces el sistema swapeará y rendimiento caerá. Al caer, las consultas tardan más con lo que los procesos se acumularán. Cada vez habrá más, cada vez swapeará más y cada vez será más lento. Es necesario reconfigurar el tamaño de los buffers,

especialmente de los buffers por thread.

Las opciones son pocas:

a) Añadir más memoria. Esto mejorará mucho el rendimiento.

b) Reducir el número de conexiones permitidas (max_connections).

Capítulo 7. Replicación

Para poder disponer de una solución de base de datos completa, en ocasiones es necesario disponer de replicación de datos.

La replicación nos aporta varias ventajas:

- Nos permite disponer de los mismos datos en ubicaciones distintas.
- Facilita la realización de copias de seguridad.
- Aporta redundancia ante fallos.
- Permite distribuir la carga entre varios servidores.

Sin embargo, no es la panacea. En concreto, la replicación entre dos servidores no se realiza en tiempo real, por lo que este tipo de soluciones no pueden utilizarse para la transmisión de datos en tiempo real. Para eso, existen los clusters.

7.1. Concepto

MySQL implementa una arquitectura de replicación Maestro-Esclavo. Es decir, un servidor esclavo copia y replica la información de su maestro. Por lo tanto, ambos sistemas tendrán la misma información (las mismas bases de datos con los mismos datos).

Esto se consigue utilizando el log binario. Este log registra todas las consultas que modifican los datos del servidor, en este caso del maestro. El proceso es el siguiente:

1. El esclavo se conecta al servidor y recoge una copia del log binario. En concreto, de todos aquellos logs binarios que todavía no haya procesado.
2. El esclavo aplica el log binario del maestro sobre sus propios datos, adquiriendo así la misma información y estado que el maestro.

Es decir, se trata de una replicación unidireccional y asíncrona (frente a la replicación multidireccional y síncrona de un cluster).

Los cambios sobre los datos deben realizarse siempre en el maestro. Aunque las consultas pueden hacerse sobre cualquiera de ellos. Luego veremos arquitecturas más complicadas y flexibles.

Fíjate que en esta arquitectura, el maestro no sabe casi nada del esclavo. Entre otras cosas, no sabe ni cuantos esclavos tiene ni si estos están al día o no. Es responsabilidad de los esclavos acceder al log binario y actualizar sus datos convenientemente.

7.2. Configuración

La configuración es tan sencilla como el concepto.

7.2.1. Servidores Nuevos

Si tenemos dos servidores nuevos, sin datos, bastará con hacer el siguiente proceso con los servidores parados[13]:

1. Creamos en el servidor una cuenta para el usuario de replicación.

```
GRANT REPLICATION SLAVE ON *.* TO blade@IP_esclavo
IDENTIFIED BY 'runner'
```

2. Configura el maestro. Sólo necesitas establecer un valor de server-id en my.cnf diferente a cualquier otro. Y asegúrate de que el log binario está activo.

```
log-bin
server-id = 1
```

3. Configura el cliente. Tendrás que establecer las diferentes variables de replicación:

```
server-id = 2
master-host = ip_maestro
master-user = blade
master-password = runner
master-port = 3306
```

4. Arranca el maestro.
5. Arranca el esclavo.
6. Sincronízalos (Ver 7.2.3)

7.2.2. El Viejo Maestro

La situación anterior es la ideal. Un maestro sin información y un nuevo esclavo. Pero no es la habitual.

Lo habitual es que tengamos un servidor y, por motivos de rendimiento, fiabilidad o investigación, queramos crearle un esclavo. Seguimos necesitando un esclavo vacío, pero podemos partir de un "viejo maestro" si tenemos cuidado.

Con ambos servidores igualmente detenidos, haremos el mismo proceso de configuración descrito anteriormente.

Pero, antes de arrancar los servidores, tenemos que llevar al esclavo al mismo estado en que se encuentra su maestro, para lo que nos bastará con una simple copia de todos sus ficheros... incluyendo los logs binarios.

13 Técnicamente es posible, y no muy complicado, configurar un servidor esclavo sin detener el servidor. Pero vamos con un caso más sencillo.

Una vez listo, podremos arrancar ambos.

7.2.3. Sincronización

Por último, será necesario sincronizar el esclavo con el servidor para que sepa desde qué posición del maestro debe replicar.

Para eso, primero tenemos que saber en qué posición se encuentra el maestro. Ejecutamos, en el maestro, el comando:

```
SHOW MASTER STATUS;
```

Y nos dirá cual es el fichero de log actual y su offset. Ahora, en el esclavo, ejecutamos los comandos:

```
STOP SLAVE;
CHANGE MASTER TO MASTER_LOG_FILE =
'fichero_log_maestro', MASTER_LOG_POS =
pos_log_maestro;
START SLAVE;
```

Donde fichero_log_maestro y pos_log_maestro son los parámetros que hemos obtenido anteriormente del maestro.

Y listo. En el log de errores podrás ver si has tenido algún problema.

7.2.4. Monitorización

Por último, querremos saber si nuestros sistemas funcionan correctamente.

Respecto al maestro hay poco que saber. El comando SHOW MASTER nos indicará el fichero de log que está utilizando y su posición en él. Pero el maestro no sabe nada de sus esclavos, ni si replican o no.

Para monitorizar los esclavos, podemos recurrir a dos comandos:

```
SHOW SLAVE STATUS;
SHOW PROCESSLIST;
```

El primero nos ofrecerá información actualizada del estado de la replicación, incluyendo el tiempo de retraso que lleva el esclavo respecto al maestro.

El segundo comando nos mostrará la lista de procesos activos, entre los que deben encontrarse dos asociados a la replicación: Slave IO y Slave SQL.

7.3. Arquitecturas

La arquitectura que hemos visto (un maestro y un esclavo) es la más sencilla, pero no la única. De hecho, se pueden crear arquitecturas muy diversas siempre que se cumplan las siguientes reglas:

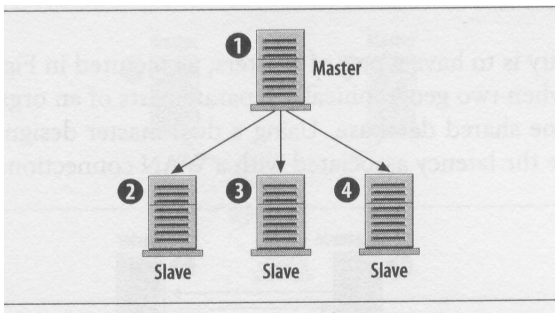
- Cada esclavo sólo podrá tener un server-id que será único.
- Cada esclavo sólo podrá tener un maestro.
- Un maestro puede tener varios esclavos.
- Un esclavo puede ser maestro de otros esclavos.

Con estas reglas en cuenta, revisemos algunas configuraciones habituales:

Maestro con Varios Esclavos

Esta es la configuración más habitual. Un maestro replica su información en varios esclavos.

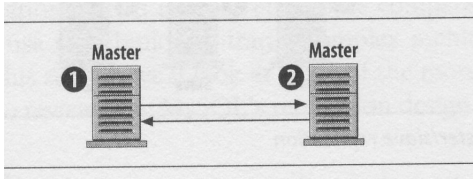
Si el maestro cae, cualquiera de ellos podrá asumir su puesto.



En esta configuración, y en muchas otras, resulta conveniente que el maestro también esté configurado para actuar como esclavo. De esta forma, tras recuperarse de su caída, podrá replicar la información y actualizarse.

Dos Maestros

Esta NO es la configuración recomendada... pero sí es una configuración posible con un poco de cuidado.

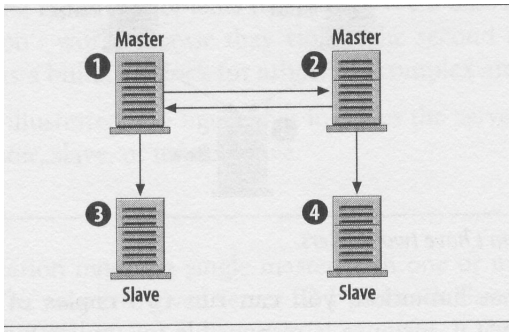


Si tenemos dos maestros, cada uno actuando como esclavo del otro, se producirán actualizaciones de información en ambos. El sistema de replicación de MySQL no incorpora ningún sistema para la resolución de conflictos. Así que si una misma fila es modificada en uno de los maestros y borrada en el otro, la replicación fallará. Lo mismo ocurrirá si se insertan filas con una misma clave primaria en ambos servidores.

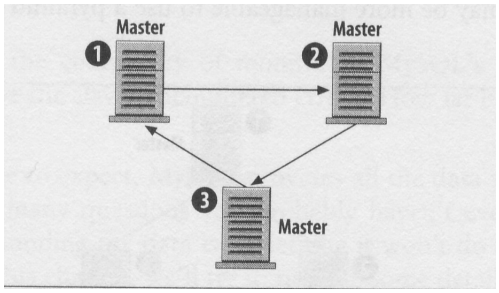
Pero, todo eso es controlable. Si podemos garantizar que estas situaciones no de conflicto no se producirán, podremos configurar dos o más maestros replicando entre sí.

Dos Maestros Replicados

Esta es la misma configuración que la anterior, pero cada maestro tiene un esclavo para replicar su información.

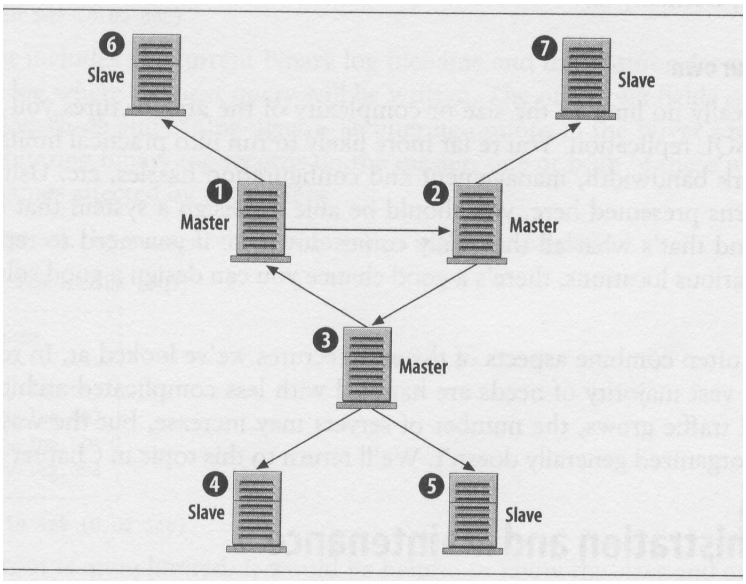


Anillo de Maestros



Esta es una configuración en la que tres maestros replican su información entre sí.

Sigue siendo crítico evitar los conflictos entre ellos.



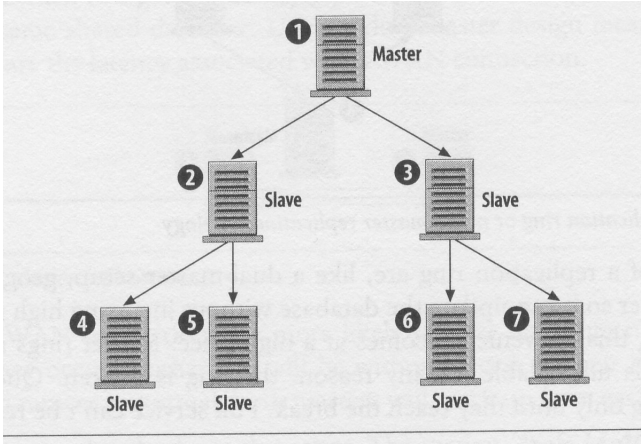
Anillo de Maestros Replicados

La misma arquitectura, pero añadiendo esclavos a cada maestro.

Jerarquía de Esclavos

Una de las arquitecturas más típicas. La información de un maestro es replicada en cascada a través de una pirámide de esclavos.

Esta configuración evita que se saturen los recursos del maestro al disponer de un elevado número de esclavos.


1
Master
2
Slave
3
Slave
4
Slave
5
Slave
6
Slave
7
Slave

Capítulo 8. Bibliografía

La mejor fuente de información técnica sobre MySQL es, naturalmente, *dev.mysql.com*. Además está siempre actualizada.

También te recomiendo el libro “High Performance MySQL”, escrito por Jeremy D. Zawodny y Derek J. Balling, de la editorial O'Reilly. Es mucho más legible que la documentación técnica de MySQL. Gran parte de la información de este documento está basada en ese libro.