



Recursosinformáticos

MySQL 5.7

Administración y optimización

<https://dogramcode.com/bloglibros>



Dogram

<https://dogramcode.com/bloglibros/libros-bases-de-datos>

Stéphane COMBAUDON

Archivos complementarios
para descarga



MySQL 5.7

Administración y Optimización

PRESENTACIÓN

Este libro acerca de **MySQL 5.7** se dirige a los **desarrolladores y administradores de MySQL** que deseen consolidar sus conocimientos sobre el SGBD Open Source más difundido del mercado.

El libro comienza con una **presentación de las bases** que serán necesarias para aprovechar al máximo todas las capacidades de MySQL: presentación de la **arquitectura del servidor** y de los principales **motores de almacenamiento**, métodos de **instalación de mono y multi-instancias**, buenas prácticas de **configuración**.

Después de estas bases que proporcionan una buena comprensión de las características del SGBD, aprenderemos cómo **administrar el servidor en el día a día**, teniendo en cuenta los principios esenciales de **seguridad**, poniendo en marcha estrategias eficaces para las **copias de seguridad y restauraciones**.

La última parte está dedicada a las técnicas avanzadas que darán las claves para resolver los problemas más complejos: **optimización de los índices** y las **consultas**, mejorar la **disponibilidad** y **rendimiento** con la creación de una solución de **replicación**, y **técnicas de supervisión** del estado del SGBD.

En el sitio www.ediciones-eni.com hay disponibles elementos complementarios para descarga.

<https://dogramcode.com/bloglibros/libros-bases-de-datos>

ÍNDICE

□ Prólogo

1. MySQL en pocas palabras
2. Objetivos del libro
3. Descripción

□ Aspectos generales de MySQL

1. Introducción
2. Arquitectura
 - 2.1 El servidor y los clientes
 - 2.2 Los protocolos de comunicación
3. Utilización de recursos de hardware
 - 3.1 Uso del disco
 - 3.2 Uso de la memoria

Presentación

Índice

- 3.3 Uso del procesador
- 3.4 Uso de la red
- 4. Variantes de MySQL
 - 4.1 MariaDB
 - 4.2 Percona Server
 - 4.3 Amazon RDS/Aurora
 - 4.4 WebScaleSQL
 - 4.5 Galera
- 5. Los motores de almacenamiento
 - 5.1 InnoDB
 - 5.1.1 Resumen de funcionamiento
 - 5.1.2 Funciones principales
 - 5.2 MyISAM
 - 5.3 Memory
 - 5.4 Archive
 - 5.5 XtraDB
 - 5.6 TokuDB
 - 5.7 RocksDB
 - 5.8 Otros motores
- 6. Bloqueos y transacciones
 - 6.1 Bloqueos implícitos
 - 6.1.1 Líneas generales
 - 6.1.2 Especificidades InnoDB
 - 6.2 Bloqueos explícitos
 - 6.2.1 Bloqueos de tabla
 - 6.2.2 Temas específicos de InnoDB
 - 6.3 Bloqueos cooperativos
 - 6.4 Transacciones
 - 6.4.1 Líneas generales
 - 6.4.2 InnoDB y las transacciones
 - 6.4.3 Mezclar motor transaccional y motor no transaccional
 - 6.4.4 Interbloqueos (deadlocks)
- Instalación del servidor
 - 1. Información general
 - 1.1 Estabilidad de las versiones
 - 1.2 Versión comunitaria y versión Enterprise
 - 1.3 Ciclo de desarrollo
 - 1.4 Elección del tipo de instalación
 - 2. Instalación en UNIX y derivados
 - 2.1 Instalación por gestor de paquetes
 - 2.2 Instalación con ejecutables precompilados
 - 2.3 Arranque del servidor
 - 2.3.1 Script mysql.server

- 2.3.2 Script `mysqld_safe`
 - 2.3.3 Invocación directa de `mysqld`
- 2.4 Parada del servidor
 - 2.4.1 Script `mysql.server`
 - 2.4.2 `mysqladmin`
 - 2.4.3 Comando `kill`
- 2.5 Resolución de problemas de instalación frecuentes
 - 2.5.1 Errores InnoDB
 - 2.5.2 Archivo `errmsg.sys` no encontrado
- 2.6 Securización de la instalación
- 2.7 Instalación de varias instancias
 - 2.7.1 Precauciones necesarias
 - 2.7.2 Instalación de versiones diferentes
 - 2.7.3 Uso del mismo ejecutable que otra instancia
 - 2.7.4 Arranque y parada de las instancias con `mysqld_multi`
- 2.8 MySQL Sandbox
 - 2.8.1 Presentación de MySQL Sandbox
 - 2.8.2 Instalación
 - 2.8.3 Creación de una instancia
 - 2.8.4 Creación de dos instancias independientes
 - 2.8.5 Otras posibilidades
- 3. Instalación en Windows
 - 3.1 Utilización del instalador
 - 3.2 Instalación con archivos ejecutables
 - 3.3 Arranque del servidor
 - 3.3.1 Servicio
 - 3.3.2 Invocación directa de `mysqld`
 - 3.4 Parada del servidor
 - 3.4.1 Servicio
 - 3.4.2 `mysqladmin`
 - 3.4.3 Administrador de tareas
 - 3.5 Resolución de problemas de instalación
 - 3.5.1 Permisos insuficientes
 - 3.5.2 Conflicto con una instalación existente
 - 3.5.3 Problemas en las rutas
 - 3.6 Securización de la instalación
 - 3.7 Instalación de varias instancias
 - 3.7.1 Precauciones necesarias
 - 3.7.2 Versiones diferentes
 - 3.7.3 Uso del mismo ejecutable
- 4. Actualización de MySQL
 - 4.1 Precauciones necesarias antes de la actualización
 - 4.1.1 Saltos de versión

- 4.1.2 Cambios introducidos por una versión
 - 4.1.3 Copia de seguridad de los datos
- 4.2 Proceso de actualización
 - 4.2.1 Posibles estrategias
 - 4.2.2 Actualización de los ejecutables
 - 4.2.3 Verificación de las tablas
- 4.3 Comprobaciones tras la actualización
- 5. Instalación de las herramientas utilizadas en el libro
 - 5.1 Instalación de la base world
 - 5.2 Instalación de la base sakila
- 6. Instalación de Percona Toolkit
- 7. Instalación de MySQL Utilities
- Configuración del servidor
 - 1. Introducción
 - 2. ¿Cómo configurar el servidor ?
 - 2.1 Configuración durante la compilación
 - 2.2 Configuración en el archivo de configuración
 - 2.2.1 Ubicación del archivo de configuración
 - 2.2.2 Estructura del archivo de configuración
 - 2.3 Configuración al arrancar mysqld
 - 2.4 Configuración dinámica del servidor
 - 2.4.1 Cambio para la sesión
 - 2.4.2 Cambio global
 - 3. Visualización de la configuración
 - 4. Configuración de InnoDB
 - 4.1 Parámetros fundamentales
 - 4.2 Aislamiento y durabilidad
 - 4.2.1 Ajuste del aislamiento
 - 4.2.2 Ajuste de la durabilidad
 - 4.3 Otros parámetros
 - 5. El registro
 - 5.1 El registro binario
 - 5.2 El registro de peticiones lentas
 - 5.3 El registro de errores
 - 5.4 El registro general
 - 5.5 Mejores prácticas
 - 5.5.1 Configuración
 - 5.5.2 Supervisar el uso del disco
 - 5.5.3 Impacto sobre el rendimiento
 - 6. El modo SQL
 - 6.1 Los modos habituales
 - 6.2 Las combinaciones de modos
 - 6.3 Modo SQL por defecto

7. Otros parámetros

7.1 Parámetros MyISAM

7.2 Caché de peticiones

7.2.1 Función de la caché

7.2.2 Activación de la caché

7.2.3 Peticiones excluidas de la caché

7.2.4 Llamada a un elemento de caché

7.2.5 Actualización de la caché

7.2.6 Fragmentación

7.2.7 Herramienta de caché

7.2.8 Parámetros asociados a la caché

7.2.9 Configuración del tamaño de la caché

7.2.10 Determinación de la eficiencia de la caché

7.2.11 Reducción de la fragmentación

7.3 Otras variables

7.3.1 Número de conexiones simultáneas

7.3.2 Cachés de tablas

7.3.3 Caché de threads

7.3.4 Parámetros que no deben modificarse

□ Seguridad y gestión de los usuarios

1. Introducción

2. Securitización del servidor MySQL

2.1 Securitización de la instalación

2.1.1 Controlar los permisos

2.1.2 Poner contraseña a la cuenta root

2.1.3 Eliminar las cuentas anónimas

2.1.4 Eliminar el esquema test

2.1.5 Securitizar la instalación con la herramienta `mysql_secure_installation`

2.2 Utilización de SSL

2.2.1 Las opciones

2.2.2 Las principales etapas

3. Cifrado de datos

4. Las opciones para reforzar la seguridad

4.1 `skip-networking`

4.2 `bind-address`

4.3 `skip-name-resolve`

4.4 `skip-show-database`

4.5 `secure-file-priv`

4.6 `chroot`

5. Gestión de usuarios y contraseñas

5.1 Conexión a cuentas de usuario

5.2 Gestión de cuentas de usuario

5.3 Roles

- 5.4 Plug-ins de autenticación
- 5.5 Plug-in de validación de las contraseñas
- 5.6 Expiración de contraseña
- 5.7 Utilidad de configuración de contraseñas
- 5.8 Asignación de los derechos
 - 5.8.1 Los derechos de administración
 - 5.8.2 Los derechos a nivel de los esquemas
 - 5.8.3 Los derechos a nivel de las tablas
 - 5.8.4 Los derechos a nivel de columnas
 - 5.8.5 Los derechos para las rutinas almacenadas
- 5.9 Limitación del uso de recursos
- 5.10 Visualización de los derechos
- 5.11 Entrada en vigor de los derechos
- 5.12 Eliminación de los derechos
- 5.13 Mejores prácticas de gestión de los derechos

6. Securización de las vistas y rutinas almacenadas

□ Respaldo y restauración

1. Líneas generales

- 1.1 Introducción
- 1.2 Respaldo lógico
- 1.3 Respaldo físico
- 1.4 Respaldo completo/incremental
- 1.5 Respaldo y replicación
- 1.6 Respaldo y motores de almacenamiento
 - 1.6.1 MyISAM
 - 1.6.2 InnoDB
 - 1.6.3 MyISAM e InnoDB

1.7 Restauración

2. En la práctica

- 2.1 Importación/exportación manual
- 2.2 mysqldump
- 2.3 Percona XtraBackup
- 2.4 Otras soluciones
 - 2.4.1 mylvmbbackup
 - 2.4.2 mysqlpump

□ Optimización

1. Hardware y sistema operativo

- 1.1 Procesador
- 1.2 Memoria RAM
- 1.3 Disco duro
 - 1.3.1 Elementos de elección
 - 1.3.2 RAID
 - 1.3.3 SSD

1.4 Sistema operativo

2. Optimización del esquema

2.1 Tipos de datos

2.1.1 Principios generales

2.1.2 Números

2.1.3 Cadenas de caracteres

2.1.4 Datos binarios

2.1.5 Fechas y horas

2.1.6 ENUM y SET

2.2 Normalización

2.2.1 Papel de la normalización

2.2.2 Primera forma normal

2.2.3 Segunda forma normal

2.2.4 Tercera forma normal

2.2.5 Resumen de las ventajas de la normalización

2.2.6 Inconvenientes de la normalización

2.3 Desnormalización

2.4 Modificación del esquema en producción

3. Indexación

3.1 Aspectos generales de los índices

3.1.1 Rol de un índice

3.1.2 Claves e índice

3.1.3 Columnas que pueden beneficiarse de un índice

3.1.4 Creación y eliminación de un índice

3.1.5 ¿ Qué columnas indexar ?

3.2 Tipos de índice

3.2.1 índices únicos

3.2.2 Claves primarias

3.2.3 índices no únicos

3.2.4 índice en varias columnas

3.2.5 índice sobre un prefijo de columna

3.2.6 índices redundantes

3.2.7 Claves externas

3.3 Conceptos avanzados

3.3.1 Index b-tree

3.3.2 índice hash

3.3.3 Otros algoritmos de indexación

3.3.4 Selectividad y distribución de valores

3.3.5 Index cluster InnoDB

3.3.6 índice de cobertura

3.4 Indexación FULLTEXT

3.4.1 Nociones básicas sobre la indexación FULLTEXT

3.4.2 Búsqueda en lenguaje natural

- 3.4.3 Búsqueda booleana
- 3.4.4 Búsqueda con expansión de la petición
- 3.4.5 Configuración de la búsqueda
- 3.4.6 Rendimiento
- 3.4.7 Limitaciones y puntos que se deben conocer

4. El comando EXPLAIN

- 4.1 Rol
- 4.2 Acceso a los datos
 - 4.2.1 Acceso secuencial o aleatorio
 - 4.2.2 Acceso en memoria o en disco
 - 4.2.3 En resumen
- 4.3 Leer el plan de ejecución
 - 4.3.1 Ejemplo simple
 - 4.3.2 Peticiones diferentes de SELECT
 - 4.3.3 Joins (uniones)
 - 4.3.4 Uniones
 - 4.3.5 Subconsultas
- 4.4 Columnas principales
 - 4.4.1 Tipos de acceso a los datos
 - 4.4.2 índices examinados
 - 4.4.3 Número de filas recorridas
 - 4.4.4 La columna Extra
- 4.5 EXPLAIN EXTENDED

5. Optimización de las peticiones

- 5.1 Aislamiento de las columnas
- 5.2 Joins
- 5.3 Filtrados
- 5.4 Ordenación
- 5.5 Agregaciones
- 5.6 Reescritura de peticiones
- 5.7 Utilización de varios índices
- 5.8 Otras características

6. Optimizaciones para MySQL 5.6/5.7

- 6.1 Index Condition Pushdown
- 6.2 Multi Range Read

7. Mantenimiento de las tablas

- 7.1 Actualización de las estadísticas del índice
- 7.2 Defragmentación de las tablas
- 7.3 Otros comandos

□ Replicación

1. Aspectos generales sobre la replicación

- 1.1 Utilidad de la replicación
- 1.2 Funcionamiento de la replicación

- 1.3 Formatos de replicación
- 2. Puesta en marcha de la replicación
 - 2.1 Replicación maestro-esclavo(s)
 - 2.1.1 Configuración
 - 2.1.2 Puntos fuertes y débiles de esta configuración
 - 2.2 Replicación maestro-maestro
 - 2.2.1 Configuración
 - 2.2.2 Puntos fuertes y débiles de esta configuración
 - 2.3 Replicación en varios niveles
 - 2.3.1 Configuración
 - 2.3.2 Puntos fuertes y débiles de esta configuración
 - 2.4 Principales variables
- 3. Resolución de problemas de operación frecuentes
 - 3.1 Impedir la replicación de algunas peticiones
 - 3.2 No-replicación de una petición
 - 3.3 Evitar el retraso de replicación
 - 3.4 Corregir un error de replicación
 - 3.5 Recuperar el espacio en disco de los registros binarios
 - 3.6 Eliminar la configuración de replicación
 - 3.7 Verificar la coherencia de los datos entre un maestro y un esclavo
 - 3.8 Algunos comandos útiles
 - 3.8.1 SHOW SLAVE STATUS
 - 3.8.2 START/STOP SLAVE {IO_THREAD|SQL_THREAD}
 - 3.8.3 RESET MASTER
- 4. Replicación y alta disponibilidad
 - 4.1 Promoción de un esclavo
 - 4.2 Automatización de la promoción
- 5. Replicación y escalabilidad
 - 5.1 Escalabilidad en lectura
 - 5.2 Escalabilidad en escritura
- 6. Funcionalidades avanzadas
 - 6.1 Identificadores de transacción
 - 6.2 Replicación paralela
 - 6.3 Replicación multifuente
 - 6.3.1 Introducción
 - 6.3.2 Implementación
 - 6.4 Replicación semisíncrona
 - 6.4.1 Introducción
 - 6.4.2 Implementación
 - 6.4.3 Novedades con MySQL 5.7
 - 6.5 Replicación retrasada

□ Otras funcionalidades

1. Particionado

- 1.1 Interés y limitaciones
 - 1.1.1 Gestión del incremento de carga
 - 1.1.2 Gestión de grandes volúmenes
 - 1.1.3 Partition pruning
 - 1.1.4 Eliminación rápida de un gran volumen de datos
 - 1.1.5 Limitaciones
- 1.2 Tipos de particiones
 - 1.2.1 El particionado de tipo RANGE
 - 1.2.2 El particionado de tipo RANGE COLUMNS
 - 1.2.3 El particionado de tipo LIST
 - 1.2.4 El particionado de tipo LIST COLUMNS
 - 1.2.5 El particionado de tipo HASH
 - 1.2.6 El particionado de tipo KEY
 - 1.2.7 Las variantes LINEAR HASH/KEY
 - 1.2.8 Selección explícita de una partición
 - 1.2.9 Subparticionado
 - 1.2.10 Funciones del particionado
 - 1.2.11 Import y export de una partición en una tabla
- 1.3 Gestión de particionado
- 1.4 Mantenimiento
- 2. Rutinas almacenadas
 - 2.1 Rol
 - 2.2 Sintaxis
 - 2.2.1 Procedimientos almacenados
 - 2.2.2 Funciones almacenadas
 - 2.3 Uso
 - 2.4 Metadatos
 - 2.5 Restricciones
- 3. Disparadores (triggers)
 - 3.1 Rol
 - 3.2 Sintaxis
 - 3.3 Restricciones
- 4. Eventos
 - 4.1 Rol
 - 4.2 Sintaxis
 - 4.3 Restricciones
- 5. Vistas
 - 5.1 Rol
 - 5.2 Sintaxis
- 6. Columnas generadas
 - 6.1 Introducción
 - 6.2 Columnas virtuales
 - 6.3 Columnas persistentes

7. Soporte JSON

7.1 El tipo de datos JSON

7.2 Ejemplo de operaciones con columnas JSON

7.3 Indexación

□ Herramientas de supervisión

1. Introducción

2. Acceso a los metadatos

2.1 Comandos específicos de MySQL

2.1.1 Comandos SHOW

2.1.2 Comando DESCRIBE

2.2 BBDD information_schema

3. Herramientas básicas para la supervisión

3.1 SHOW PROCESSLIST

3.2 SHOW GLOBAL STATUS

3.3 SHOW ENGINE INNODB STATUS

3.3.1 SEMAPHORES

3.3.2 LAST FOREIGN KEY ERROR

3.3.3 LAST DETECTED DEADLOCK

3.3.4 TRANSACTIONS

3.3.5 FILE I/O

3.3.6 INSERT BUFFER AND ADAPTATIVE HASH INDEX

3.3.7 LOG

3.3.8 BUFFER POOL AND MEMORY

3.3.9 ROW OPERATIONS

4. Performance Schema

4.1 Rol

4.2 Configuración

4.3 Esquema sys

5. Identificación de los problemas de las peticiones

5.1 Peticiones lentas

5.2 Deadlocks

6. Herramientas de supervisión del sistema

6.1 Cacti

6.2 Grafana

6.3 Nagios

6.4 Identificación de los problemas de sistema con Linux

6.4.1 vmstat

6.4.2 iostat

6.4.3 mpstat

índice

<https://dogramcode.com/bloglibros/libros-bases-de-datos>

MySQL 5.7

Administración y optimización

Descargado en: www.detodoprogramacion.org

Este libro acerca de **MySQL 5.7** se dirige a los **desarrolladores y administradores de MySQL** que deseen consolidar sus conocimientos sobre el SGBD Open Source más difundido del mercado.

El libro comienza con una **presentación de las bases** que serán necesarias para aprovechar al máximo todas las capacidades de MySQL: presentación de la **arquitectura del servidor** y de los principales **motores de almacenamiento**, métodos de **instalación de mono y multi-instancias**, buenas prácticas de **configuración**.

Después de estas bases que proporcionan una buena comprensión de las características del SGBD, aprenderemos cómo **administrar el servidor en el día a día**, teniendo en cuenta los principios esenciales de **seguridad**, poniendo en marcha estrategias eficaces para las **copias de seguridad y restauraciones**.

La última parte está dedicada a las técnicas avanzadas que darán las claves para resolver los problemas más complejos: **optimización de los índices** y las **consultas**, mejorar la **disponibilidad** y **rendimiento** con la creación de una solución de **replicación**, y **técnicas de supervisión** del estado del SGBD.

En esta página hay disponibles elementos complementarios para descarga.

Los capítulos del libro:

Prólogo – Aspectos generales de MySQL – Instalación del servidor – Configuración del servidor – Seguridad y gestión de los usuarios – Respaldo y restauración – Optimización – Replicación – Otras funcionalidades – Herramientas de supervisión

Autor(es)

Stéphane COMBAUDON

Stéphane COMBAUDON es Arquitecto MySQL/DBA y gestiona un parque de varios cientos de servidores MySQL. Está certificado en MySQL 5 DBA. Este experto MySQL administra diariamente y desde hace muchos años, el servidor de bases de datos en entornos profesionales muy variados. Hoy ofrece a los lectores un libro verdaderamente eficaz sobre la administración de MySQL y con muchos ejemplos que ilustran las explicaciones.

MySQL en Pocas Palabras

MySQL es el sistema de gestión de bases de datos Open Source más popular del mundo y es conocido por su rendimiento y fiabilidad. Tras una fase de difusión a comienzos de la década de 2000, MySQL se dedicaba principalmente a las aplicaciones personales o profesionales de gama baja. Los últimos años se han caracterizado por la adhesión de los grandes protagonistas de la Web a las características de MySQL. Así, hoy en día, la inmensa mayoría de los sitios de gran densidad de tráfico, como las redes sociales o muchos portales comunitarios, ha alcanzado su nivel de rendimiento mediante el uso intensivo de MySQL.

Los sucesivos rescates de MySQL AB, empresa editora de MySQL, por Sun en 2008 y luego de Sun por Oracle en 2009, confirman el interés cada vez más fuerte de la industria por el producto. La importantísima comunidad de usuarios está muy atenta a las directivas para el desarrollo de MySQL y participa activamente en la mejora del servidor.

Objetivos del libro

Este libro se dirige a todas las personas que deseen mejorar sus conocimientos sobre MySQL o que necesitan de experiencia adquirida para aprovechar al máximo las capacidades de sus bases de datos.

Los administradores de bases de datos acostumbrados a otros SGBD que comienzan con MySQL encontrarán toda la información que les permita garantizar el funcionamiento diario de sus instancias. Los administradores más experimentados también estarán interesados en la lectura de este libro, ya que los conocimientos teóricos sobre el funcionamiento de MySQL se complementan con las recomendaciones sobre las técnicas que es preciso utilizar o evitar.



Por último, los desarrolladores web que se ven obligados a trabajar con MySQL encontrarán mucha información que les permitirá optimizar su código y evitar las trampas convencionales a fin de obtener aplicaciones rápidas y seguras.

El lector sacará un máximo de beneficios de este libro si ya conoce el rol de un sistema de gestión de bases de datos. Se recomienda disponer de conocimientos de MySQL para abordar sus capítulos, aunque no es indispensable. Para un lector que nunca haya trabajado con MySQL, es recomendable comenzar por instalar el servidor como se describe en el segundo capítulo con el fin de poder ejecutar los ejemplos de código que se encuentran a lo largo de este libro.

Para los usuarios de UNIX/Linux, algunos conocimientos básicos sobre el shell permitirán al lector entender la descripción de algunas operaciones.

Descripción

En este libro no nos vamos a entretener en la sintaxis del lenguaje SQL (existen multitud de referencias para ello), sino que nos vamos a detener en dos aspectos de la utilización de MySQL: la administración del servidor y su optimización.

El lector, en primer lugar, recordará los conceptos fundamentales que le permitirán conocer MySQL y adquirir soltura en las operaciones. Muchos ejemplos prácticos apoyarán el discurso teórico de algunas partes.

Luego el libro abordará los aspectos avanzados relacionados con la optimización y puesta en marcha de funcionalidades interesantes en caso de volúmenes de datos importantes o de mucho tráfico.

Al terminar la lectura de este libro, el lector debe ser capaz de llevar a cabo el mantenimiento cotidiano de MySQL, poseer todos los conocimientos necesarios para resolver los problemas clásicos y conocer las posibilidades y técnicas para hacer frente a situaciones en las que el servidor se enfrenta a picos de carga.

En cada capítulo, el lector encontrará ejemplos de los conceptos descritos. Cuando las operaciones llaman a comandos del sistema operativo, hemos dado prioridad a los comandos compatibles UNIX/Linux para reflejar que los sistemas UNIX/Linux son mayoritarios en el mundo de la Web y de MySQL.

Se tratan las novedades aparecidas hasta la versión 5.7, como, por ejemplo, la replicación semisíncrona o las mejoras en la ejecución de las peticiones.

La primera parte del libro presenta las bases necesarias para una correcta comprensión del funcionamiento del servidor:

- El capítulo Aspectos generales de MySQL explica cuál es la arquitectura general del servidor MySQL, en particular cómo se utilizan el disco y la memoria. Este capítulo

también aborda el concepto de motor de almacenamiento.

- El capítulo Instalación del servidor describe los procedimientos de instalación posibles para entornos UNIX/Linux y Windows. La implementación de varias instancias sobre el mismo servidor anfitrión también forma parte de los temas abordados, así como la forma de actualizar el servidor cuando están disponibles nuevas versiones.
- El capítulo Configuración del servidor ofrece una visión resumida de las posibilidades de configuración del servidor para saber cuáles son los parámetros principales que hay que ajustar y cómo transmitirlos a una instancia. También se aborda la configuración de los registros.

La segunda parte se refiere a los puntos principales de la administración diaria del servidor:

- El capítulo Seguridad y gestión de los usuarios da las claves para proteger las instalaciones: gestión de derechos, SSL... También se describen aquí buenas prácticas para la gestión de los usuarios, que le evitarán crear cuentas con derechos innecesarios.
- El capítulo Respaldo y restauración expone los métodos de protección aplicables con MySQL y explica cuáles son las ventajas e inconvenientes. Aprenderá, pues, no solo cuáles son los comandos necesarios para guardar y restaurar sus bases, sino también qué métodos elegir.

La última parte se centra en los aspectos avanzados de la administración de MySQL:

- El capítulo Optimización ofrece un largo resumen de las técnicas de optimización para ganar en rendimiento: la elección del equipo, colocación de índices, normalización, reescritura de los comandos, configuración del servidor, mantenimiento de las tablas.
- El capítulo Replicación tiene como objetivo enseñarle a utilizar el mecanismo de replicación para construir una arquitectura de aplicación sólida y eficiente. Se describen diferentes configuraciones en este capítulo (maestro-esclavo, maestro-maestro) con sus ventajas e inconvenientes, así como los medios que permiten comprobar el buen funcionamiento de la replicación.
- El capítulo Otras funcionalidades aclara los conceptos de las particiones, procedimientos almacenados, disparadores, planificador de eventos y vistas, y aborda dos funcionalidades añadidas en MySQL 5.7: las columnas calculadas y el tipo de

datos JSON.

- El capítulo Herramientas de supervisión se centra en las herramientas para supervisar sus instancias y obtener información sobre su funcionamiento.

Introducción

MySQL se dio a conocer a finales de la década de 1990 como un sistema de gestión de bases de datos relacional especialmente adecuado para pequeños proyectos web, aprovechando sobre todo de su carácter gratuito y su rapidez. Después, en el transcurso de los años 2000, los gigantes de la Web se pusieron a utilizar MySQL de forma masiva. Lo siguiente fue más difícil, ya que en la segunda mitad de la década de 2000 todos estos grandes actores tuvieron que hacer frente a las limitaciones de MySQL en cuanto a su escalabilidad. Fue en ese momento cuando surgieron muchas soluciones NoSQL. Pero, finalmente, MySQL ha evolucionado de forma rápida en los últimos años, mientras que los problemas de juventud de las soluciones NoSQL se han hecho cada vez más evidentes.

Hoy en día, MySQL sigue siendo una opción muy extendida para proyectos web, mucho menos para proyectos más tradicionales. ¿Cuáles son las razones? En primer lugar, MySQL es capaz de ofrecer buenos rendimientos incluso con los servidores menos potentes. Además, su estabilidad es excelente y, en una instancia configurada de forma correcta, es muy raro que MySQL se cuelgue o pierda los datos. Por último, su carácter gratuito permite contemplar despliegues con cientos o miles de instancias en caso necesario sin gastar una fortuna en licencias.

Estas son las razones por las que MySQL es tan popular en el mundo de la Web. A pesar de todo, los responsables de los sectores más tradicionales siguen mostrándose reticentes a utilizar MySQL. Para ellos, el producto experimenta desarrollos curiosos: la gestión de errores es extraña, lo que puede dar la impresión de que MySQL cambia a veces de forma silenciosa los datos, y las funcionalidades como los procedimientos almacenados o los triggers han sido descuidadas totalmente y no ofrecen la flexibilidad o las prestaciones esperadas.

Todos estos errores del pasado están en vías de corregirse y Oracle promete excelentes sorpresas para las versiones futuras.

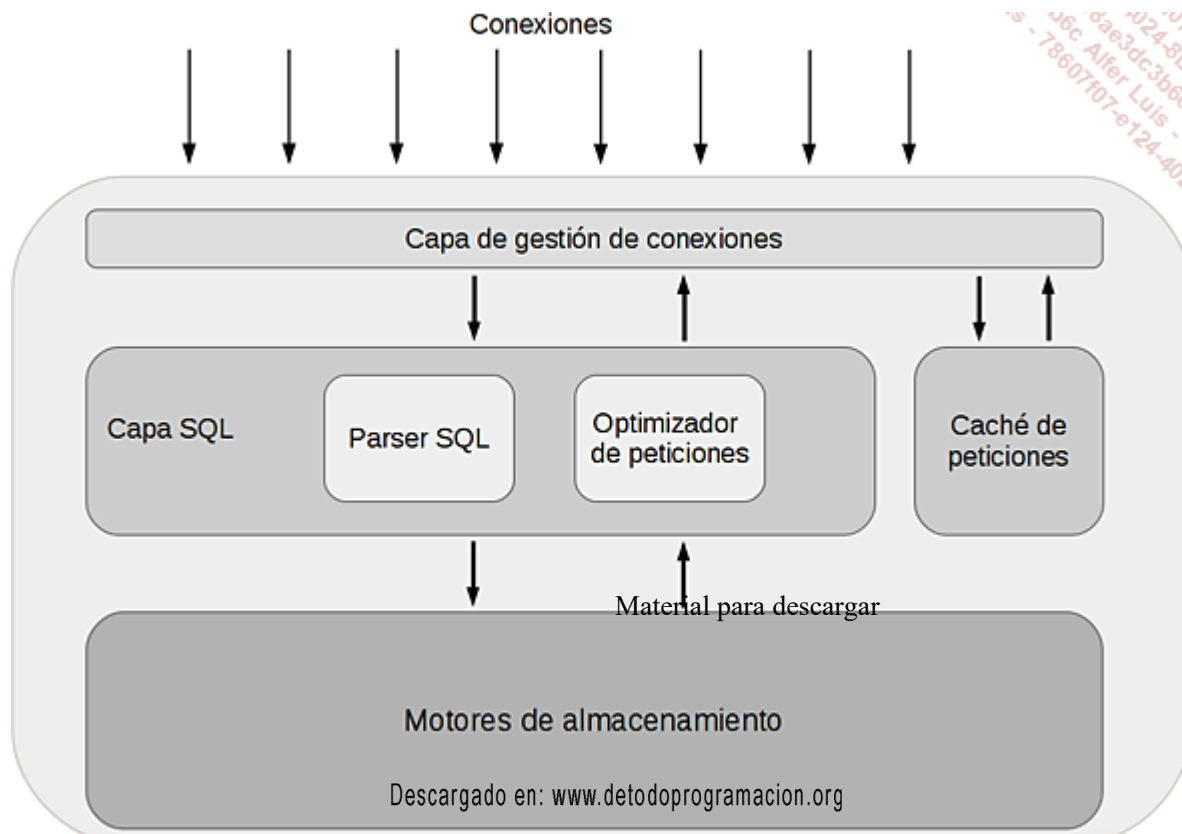
Arquitectura

1. El servidor y los clientes

El servidor MySQL (`mysqld`) intercepta las peticiones formuladas por los clientes, transforma estas peticiones en un plan de ejecución, recupera los datos según el plan de ejecución ofrecido y, por último, devuelve el resultado al cliente. Se compone de varios módulos de gestión:

- Los protocolos de comunicación con los clientes (TCP/IP, socket SSL...).
- Los permisos de acceso a los diferentes recursos disponibles (ver el capítulo Seguridad y gestión de los usuarios).
- Las cachés para minimizar el acceso a disco.
- Los diferentes tipos de registros de servidor (binarios, peticiones lentas, etc.).
- El análisis, la optimización y la ejecución de las peticiones.
- El almacenamiento de los datos.

El diagrama siguiente resume los distintos módulos del servidor:



Cada distribución de MySQL contiene varios clientes en línea de comando para interactuar con el servidor. Los más utilizados son:

- `mysql`: para ejecutar las peticiones.
- `mysqldump`: para realizar las copias de seguridad lógicas.
- `mysqladmin`: para realizar las operaciones de administración (por ejemplo, consultar las estadísticas del servidor, interrumpir peticiones que duran mucho tiempo, cambiar contraseñas).

2. Los protocolos de comunicación

MySQL ofrece diferentes protocolos de comunicación entre el cliente y el servidor. Bajo UNIX, el protocolo por defecto para una comunicación en local es `UNIX socket`. Sin embargo, podemos utilizar el protocolo `TCP/IP` en local. Este último es el único disponible para un cliente remoto. Si nos encontramos en un entorno MS Windows, en local podemos elegir entre `TCP/IP`, que es el protocolo por defecto, `shared memory` (una zona de memoria compartida entre el cliente y el servidor) o `named pipes` (un archivo que permite comunicar los procesos sin vínculos de parentesco). Para una conexión remota, solo podemos utilizar el protocolo `TCP/IP`.

Protocolo	Conexión	Sistema operativo
TCP/IP	Local, remoto	Todos
UNIX socket	Solo de forma local	UNIX
shared memory	Solo de forma local	Windows
named pipes	Solo de forma local	Windows

Utilización de recursos de hardware

1. Uso del disco

Los datos se almacenan en disco para garantizar su persistencia. Encontrará en su servidor MySQL:

- Los esquemas (o bases de datos), que aparecen en el disco como carpetas de igual nombre: por ejemplo, el nombre de la carpeta del esquema `world` es `world`.
- Archivos que tienen la extensión `.frm`. Contienen la estructura de las tablas (`frm` de formato). Desde un punto de vista de SQL, una tabla pertenece a un esquema, lo que en el disco se traduce en al menos un archivo (que contiene la estructura de la tabla) dentro de un directorio. Por ejemplo, la tabla `City`, que pertenece al esquema `world`, está representada en el disco por el archivo `City.frm`, localizado en el directorio `world`. Los datos y los índices se encuentran en uno o varios archivos o solo en la memoria; todo depende del motor de almacenamiento. Esta parte de la arquitectura se detalla en la sección Los motores de almacenamiento.
- Los registros del servidor (registro binario, registro de errores...), los de los motores de almacenamiento (`ib_logfile0` de InnoDB, por ejemplo), de la replicación, el archivo de configuración...
- Los disparadores (triggers).

Por defecto, todos estos archivos se almacenan en un único directorio llamado directorio de datos. La ubicación de este directorio puede configurarse con el parámetro `datadir`.

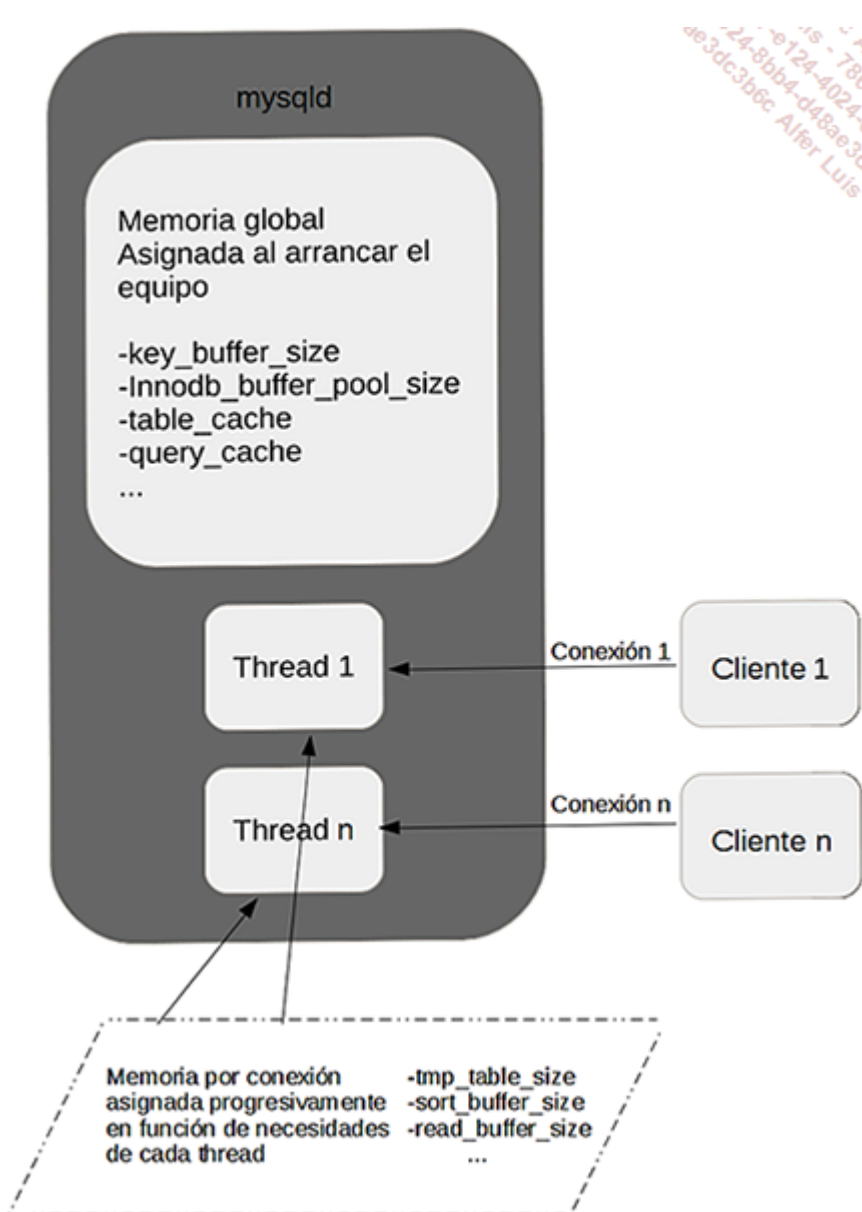
Podemos modificar la configuración para separar el almacenamiento de los registros y el almacenamiento de datos. Esta elección puede ser útil en el caso de que algunas particiones del sistema de archivos tengan un tamaño limitado. Por ejemplo, podemos colocar los datos

de forma preferente en discos rápidos mientras que los registros pueden, sin problema, encontrarse en discos más lentos.

- ☐ Atención a las mayúsculas y minúsculas: un error común de los usuarios de sistemas operativos no sensibles a mayúsculas y minúsculas como MS Windows es escribir el nombre de los esquemas y tablas sin preocuparse de ello. En efecto, en tal sistema, las peticiones `SELECT * FROM mi_tabla` y `SELECT * FROM Mi_Tabla` son equivalentes. Por el contrario, si su sistema operativo es sensible a mayúsculas y minúsculas, como la mayoría de los sistemas UNIX (lo que no es el caso de Mac OS X), el servidor intentará acceder respectivamente a los archivos que comienzan por `mi_tabla` y `Mi_Tabla`, que no son los mismos. En el mejor de los casos, tendremos un error si una de las dos tablas no existe. En el peor de los casos, no se ejecutará la petición deseada. Para evitar este tipo de errores, ¡debemos establecer una norma y mantenerla!

- ☐ Una alternativa consiste en configurar la opción `lower_case_table_names` en 1, lo que tendrá como consecuencia forzar el servidor a escribir los nombres de los esquemas y las tablas en minúsculas en el disco.

2. Uso de la memoria



Descargado en: www.detodoprogramacion.org

Estructura de la memoria

La asignación de memoria del servidor puede dividirse en dos categorías. La primera es la asignación por instancia. Se trata de zonas de memoria que pueden ser compartidas o utilizadas por todas las conexiones. El objetivo principal es evitar al máximo los accesos a disco, ya que los discos siguen siendo todavía la parte más lenta de un sistema. Los discos SSD y Flash han contribuido a disminuir los problemas de rendimiento relacionados con los discos, pero es evidente que MySQL puede ejecutar peticiones aún más rápido cuando gran parte de los datos está disponible en memoria.

La segunda categoría es la asignación por threads. Esta memoria se asigna y desasigna de forma dinámica en función de las necesidades de cada cliente. Está compuesta en particular por el espacio de memoria que ocupan los datos contenidos en las cachés utilizadas por los clientes como `sort_buffer_size` o `read_buffer_size`. La podemos encontrar a su vez en el espacio de memoria de las tablas temporales

(tmp_table_size), en el espacio de memoria de las tablas Memory (max_heap_table_size), etc.

3. Uso del procesador

El procesador funciona a todos los niveles para el tratamiento de la información: validación sintáctica de las peticiones, búsqueda del plan de ejecución óptimo, tratamiento de datos en memoria.

Cuando la mayoría de los datos están en caché, el acceso a ellos es muy rápido y el procesador pasa a ser el punto crítico: su velocidad tiene un impacto directo en la rapidez de ejecución de las peticiones y, por tanto, en el rendimiento experimentado por el usuario final.

Al contrario, cuando los accesos a disco son frecuentes, la velocidad del procesador tiene mucha menos importancia, ya que el factor principal para el tiempo de ejecución de las peticiones será el acceso al disco. Observe que, debido al aumento de la memoria disponible en los servidores y al predominio de discos rápidos, es cada vez más importante elegir procesadores rápidos.

4. Uso de la red

La red se utiliza en todas las comunicaciones entre cliente y servidor (excepto cuando el cliente y el servidor se encuentran en el mismo equipo) y para todas las comunicaciones entre servidores, en el caso de servidores conectados por replicación.

Sin embargo, la velocidad de la red rara vez limita la ejecución de las peticiones, ya que los resultados devueltos suelen ser compactos. La única excepción se da cuando se devuelven grandes columnas de tipo TEXT o BLOB: en este caso, grandes cantidades de datos van a tener que circular por la red.

En el caso de la replicación, una red lenta puede tener un impacto en los servidores esclavos. En efecto, pueden tener dificultades para transferir las peticiones del maestro y van, por lo tanto, a encontrarse con retraso respecto a este. Sin embargo, el maestro no sufrirá ningún impacto, ya que la replicación MySQL es asíncrona: el maestro no se preocupa nunca del estado de sus esclavos.

Variantes de MySQL

1. MariaDB

Monty Widenius, el creador de MySQL, lanzó MariaDB en 2009 como alternativa a la versión oficial de Oracle.

MariaDB incorpora mejoras procedentes de la Comunidad (motor XtraDB de Percona, muchos motores de almacenamiento), así como las funcionalidades diseñadas de forma directa por los desarrolladores de MariaDB, como cambios en el optimizador de peticiones.

MariaDB tenía como objetivo ser 100 % compatible con la versión MySQL de Oracle, pero esta compatibilidad se ha roto, ya que algunas funcionalidades se han desarrollado de manera independiente de MySQL (por ejemplo: columnas virtuales, identificadores de transacción para la replicación, JSON).

La numeración de las versiones de MariaDB es a su vez totalmente independiente de la de MySQL. Por consiguiente, es difícil saber qué versión de MariaDB es más o menos equivalente a la de MySQL.

En resumen, MariaDB es interesante porque dispone de funcionalidades que MySQL no ofrece, pero las incompatibilidades con MySQL pueden complicar las migraciones.

Para más información, consulte la página del proyecto: <http://www.mariadb.com>

2. Percona Server

Percona empezó a desarrollar correctores para sus clientes en 2007 y, poco a poco, estos parches se unieron para formar una versión derivada de MySQL llamada Percona Server.

Percona Server es una versión mejorada de MySQL. Uno de sus objetivos es permanecer

100 % compatible con la versión oficial. Algunos parámetros no permiten la compatibilidad, pero están desactivados por defecto.

Entre las funcionalidades adicionales, podemos encontrar:

- El Motor XtraDB, que ofrece en algunos casos mejores resultados que InnoDB (ver la sección sobre los motores de almacenamiento).
- Mejor instrumentación, con un registro de peticiones lentas mejorado y tablas adicionales en la BBDD `information_schema`.

En resumen, Percona Server es interesante si está buscando una versión mejorada de MySQL con la que no tenga que preocuparse de problemas de migración. Sin embargo, en comparación con MySQL las mejoras de rendimiento se ven minimizadas en la mayoría de las situaciones con la versión 5.7. Existen diferencias significativas de rendimiento para las versiones 5.0 y 5.1, pero con las versiones 5.5 y 5.6 Oracle ha conseguido corregir la mayoría de los problemas que afectaban a un gran número de usuarios.

Las versiones de Percona Server siguen a las versiones oficiales procedentes de Oracle con cerca de un mes de desfase. Existe más información disponible en la página del proyecto:

<https://www.percona.com/software/mysql-database/percona-server>

3. Amazon RDS/Aurora

En lugar de crear instancias EC2 vírgenes y luego instalar MySQL, configurar las copias de seguridad, la replicación o una solución de alta disponibilidad, Amazon propone utilizar sistemas preconfigurados.

El RDS es el más antiguo de estos sistemas: las operaciones de creación de una instancia o de un esclavo de replicación son automatizadas y realizables desde la consola gráfica de Amazon. El RDS es especialmente interesante para las entidades que no tengan competencias específicas en MySQL: el usuario elegirá las especificaciones de su servidor (cantidad de memoria, número de procesadores) y, al cabo de unos pocos minutos, estará disponible un servidor listo para utilizarse en producción.

Lo que es ideal para los usuarios novatos se convierte, sin embargo, en una desventaja para los usuarios expertos: no se puede acceder a la máquina física que hay detrás de los servidores RDS y, por lo tanto, es imposible optimizar los parámetros del sistema. El

usuario no es administrador de la base de datos, no dispone de todos los derechos y no es posible saber cómo funciona exactamente el mecanismo de alta disponibilidad.

Observe que RDS ha avanzado mucho desde su lanzamiento. Ahora se pueden cambiar casi todos los parámetros de configuración, mientras que antes algunas directivas esenciales no podían modificarse. La migración en caliente es también más fácil, ya que se puede configurar RDS como esclavo de otro servidor.

Si RDS tiende más bien a pequeños y medianos proyectos MySQL, Aurora se orienta, por el contrario, a proyectos para los cuales la base de datos será muy solicitada. Aurora pone énfasis en la facilidad de obtener a la vez escalabilidad y alta disponibilidad. Esto ha sido posible modificando MySQL de forma muy significativa.

Cabe señalar que, en el momento de escribir este libro, Aurora es todavía un proyecto muy joven con bastantes limitaciones. Por ejemplo, por ahora, Aurora solo ofrece MySQL 5.6 y no hay aún una fecha disponible para MySQL 5.7. Más grave aún, las migraciones son en extremo difíciles, ya que la única posibilidad es hacer una copia de seguridad del antiguo servidor y restaurar los datos en Aurora. Dicha migración supone muchas horas de indisponibilidad del servicio, lo que no es asumible en muchos casos.

Para más información sobre RDS o Aurora, se pueden consultar las siguientes páginas:

- Para RDS: <https://aws.amazon.com/es/rds/>
- Para Aurora: <https://aws.amazon.com/es/rds/aurora/details/>

4. WebScaleSQL

Ingenieros de varias empresas que utilizan MySQL a gran escala descubrieron que estaban todos intentando resolver los mismos problemas de MySQL. Decidieron que, en lugar de trabajar de manera aislada, iban a unir sus contribuciones en el mismo repositorio de datos.

Los principales contribuyentes de WebScale son Facebook, Google, Twitter, LinkedIn y Alibaba. Oracle integra a veces algunas modificaciones en MySQL. Es el caso por ejemplo del `super_read_only`, que permite poner un servidor MySQL en modo de solo lectura (a diferencia del `read_only`, que funciona solo para los usuarios que no tengan el derecho SUPER).

WebScaleSQL está basado en MySQL 5.6 y no encontraremos su código fuente en el sitio

del proyecto: a diferencia de MariaDB o Percona, que aportan paquetes listos para usarse y probarse, WebScaleSQL no fue concebido para ser utilizado de forma directa, sino para servir como base para desarrollos específicos adicionales. No se recomienda instalar WebScaleSQL para su uso en producción.

Se puede acceder al sitio del proyecto desde la siguiente dirección: <http://webscalesql.org/>

5. Galera

La empresa Codership publica desde hace varios años una librería de replicación de MySQL que presenta características únicas: la replicación es síncrona y sustituye a la replicación nativa de MySQL, y es posible escribir sobre cualquier servidor en un clúster Galera (mientras que todas las escrituras deben llegar al maestro en un esquema de replicación estándar).

Galera es interesante sobre todo para quienes se interesan por la alta disponibilidad: todos los servidores de un clúster Galera son idénticos, y pueden recibir lecturas y escrituras. Por lo tanto, en caso de error de un servidor, cualquier otro servidor del clúster podrá sustituir de forma inmediata al servidor que falle sin necesidad de recurrir a operaciones complejas. En contrapartida, Galera impone una serie de limitaciones, como la presencia de claves primarias en todas las tablas y la utilización del motor InnoDB. Además, la replicación síncrona retrasa un poco las escrituras. Por lo tanto, no todas las aplicaciones pueden usar Galera de forma sencilla.

Encontrará más información disponible en el sitio del proyecto:

<http://www.galeracluster.com>

Asimismo, existen paquetes listos para utilizarse en los sitios de Codership (MySQL + Galera), MariaDB (MariaDB + Galera) y Percona (Percona XtraDB Cluster).

Observe que Oracle está trabajando en una herramienta llamada Group Replication, que ofrece funcionalidades similares a Galera.

Los motores de almacenamiento

Una de las características únicas de MySQL es el concepto de motores de almacenamiento. Cada motor deberá ofrecer un denominador común de funcionalidad, pero es posible añadir las funcionalidades que faltan en el servidor. Un ejemplo clásico es el de las claves externas: aunque el servidor reconozca la sintaxis, las claves externas no provocan ninguna acción específica por parte del servidor. No obstante, el motor InnoDB ofrece soporte de claves externas, lo que no es el caso del motor MyISAM, por ejemplo.

Esta arquitectura tiene, sin embargo, inconvenientes. Los motores no son todos equivalentes en términos de rendimiento: si el motor X posee una funcionalidad que no está disponible con nuestro motor actual, no está garantizado que el motor X sea más rápido.

Los motores tampoco son todos equivalentes en términos de seguridad de datos: InnoDB, por ejemplo, tiene la capacidad de recuperar de forma automática los datos en caso de fallo inesperado, mientras que un fallo similar con MyISAM conllevará inevitablemente una pérdida de datos. Por último, aunque pueda ser tentador elegir un motor de almacenamiento para cada una de las tablas, no es fácil administrar un servidor en el que se emplean varios motores de almacenamiento: la memoria física del servidor se debe compartir entre los diferentes motores, los respaldos son más difíciles y diagnosticar los problemas es más complicado. Por estas razones, se recomienda utilizar un solo motor de almacenamiento.

La elección del motor se realiza durante la creación de la tabla, con el comando SQL `CREATE TABLE` incluyendo la cláusula `ENGINE`.

```
mysql> CREATE TABLE mi_tabla (i INT) ENGINE=InnoDB;  
Query OK, 0 rows affected (0,24 sec)
```

Para conocer el motor de almacenamiento de una tabla, podemos usar el comando `SHOW`

CREATE TABLE:

```
mysql> SHOW CREATE TABLE mi_tabla \G
***** 1. row *****
Table: mi_tabla
Create Table: CREATE TABLE `mi_tabla` (
  `i` int(11) DEFAULT NULL
) ENGINE=InnoDB;
```

o la base INFORMATION_SCHEMA:

```
mysql> SELECT TABLE_NAME, ENGINE FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_NAME = 'actor' AND TABLE_SCHEMA = 'sakila';
+-----+-----+
| TABLE_NAME | ENGINE |
+-----+-----+
| actor       | InnoDB |
+-----+-----+
```

La opción del motor es reversible, se puede cambiar con una petición ALTER TABLE:

```
mysql> ALTER TABLE mi_tabla ENGINE = MyISAM;
Query OK, 0 rows affected (0,24 sec)
Records: 0 Duplicates: 0 Warnings: 0

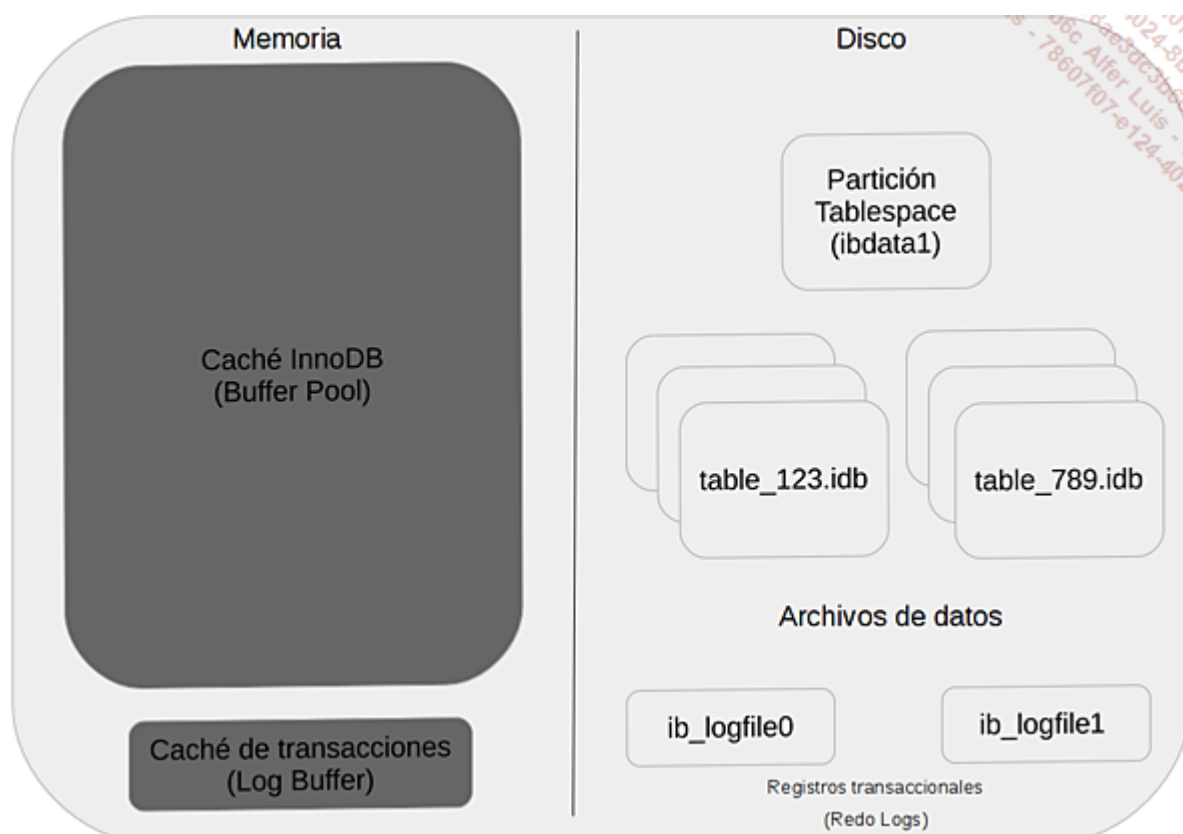
mysql> SHOW CREATE TABLE mi_tabla \G
***** 1. row *****
Table: mi_tabla
Create Table: CREATE TABLE `mi_tabla` (
  `i` int(11) DEFAULT NULL
) ENGINE=MyISAM
```


1. InnoDB

a. Resumen de funcionamiento

El motor InnoDB es el motor de almacenamiento predeterminado de MySQL desde la versión 5.5. De tipo transaccional ACID (Atomicidad, Coherencia, Aislamiento y Durabilidad), implementa el MVCC (*Multi Version Concurrency Control*), que permite tener lecturas que no bloquean a las escrituras y viceversa, y un bloqueo a nivel de registro. Esto permite ser eficaz en entornos con muchas escrituras y lecturas.

El siguiente esquema resume el funcionamiento de InnoDB:



El buffer pool es una caché que contiene los últimos datos e índices a los que se haya accedido. Cuando ejecutamos una petición de lectura, InnoDB comenzará por examinar el buffer pool. Si los datos necesarios están disponibles, se devuelven de forma directa sin requerir ningún acceso a disco. Si algunos datos no están disponibles en memoria, se leen los archivos de datos disponibles en el disco y se copian los datos que faltan en el buffer pool y luego se devuelven.

Cuando ejecutamos una escritura, InnoDB buscará primero la página de datos correspondiente. Si la página de datos está disponible en el buffer pool, esta se modifica y

se informa al usuario del éxito de la operación. Si la página de datos no está disponible en el buffer pool, InnoDB la lee del disco, la escribe en el buffer pool y la modifica en el buffer pool. Los cambios se codifican a su vez en la caché de transacción (log buffer).

Cuando se valida la transacción (COMMIT), los cambios escritos en la caché de transacción se escriben en el registro de transacciones (*redo logs*). Observe que, cuando se valida la transacción, los datos se modifican en la memoria (en el buffer pool), pero no en el disco. Se trata de una optimización que permite evitar costosas escrituras síncronas en el disco.

¿Qué ocurre en caso de un fallo inesperado del servidor? ¿Se pueden perder los datos? En realidad, si los archivos de datos no son modificados en el momento de la escritura, la información contenida en los registros de transacción bastará para reconstruir los datos. En caso de fallo inesperado, las modificaciones en memoria se pierden, pero InnoDB recuperará de forma automática todas las transacciones perdidas volviendo a leer sus registros de transacción. De esta forma, tenemos la garantía de que cualquier transacción validada podrá ser recuperada.

Observe que el mecanismo implementado para garantizar que no se pierda ningún dato es costoso: cada vez que se valida una transacción, los cambios se escriben en el registro de transacción y se sincronizan en el disco. En algunos casos, preferimos cambiar la configuración de InnoDB para escribir las transacciones validadas en los registros de transacción, pero no la sincronización de los registros en un disco, salvo una vez por segundo para favorecer los resultados. Debemos aceptar algunas pérdidas de datos en caso de fallo sin previo aviso. Este será, por ejemplo, el caso en un esclavo de replicación: como los datos ya están en el servidor maestro, la pérdida de datos en el esclavo no es muy importante (el esclavo siempre puede reconstruir los datos a partir del maestro).

Por supuesto, aunque no se produzca un fallo inesperado del servidor, los datos terminan siendo escritos en disco. Pero esta operación se efectúa en segundo plano. Para el usuario, todo ocurre como si la petición de escritura fuera ejecutada solo en memoria: los rendimientos son, pues, muy buenos, al tiempo que la seguridad de los datos se garantiza por el registro de transacciones.

Cabe preguntarse de qué sirve escribir un registro de transacciones durante la ejecución de una petición de escritura y luego tener un proceso en segundo plano para sincronizar los datos en disco, en lugar de solo sincronizar los datos en disco al ejecutar la petición. La razón es la diferencia de rendimiento entre un acceso de disco secuencial y un acceso

aleatorio.

El registro de transacciones es un conjunto de archivos en los que las escrituras se realizan siempre al final del archivo (modo `append-only`). En este caso, todos los accesos a disco son secuenciales y todos los tipos de disco ofrecen buenos rendimientos para este tipo de acceso.

Una escritura en un archivo de datos se efectúa, sin embargo, en una ubicación que no puede preverse. Aquí hablamos de acceso aleatorio, y este es el tipo de acceso más lento.

La función de InnoDB es favorecer las escrituras rápidas para las interacciones con el usuario final y reservar las escrituras lentas a los procesos en segundo plano. Observe, a su vez, que los algoritmos de sincronización de InnoDB son capaces de agrupar varias escrituras en una misma página y una única operación de escritura en disco. Esta optimización permite limitar al máximo las costosas escrituras aleatorias.

Si observamos el contenido de la raíz de la carpeta de datos, veremos los archivos `ib_logfile0` e `ib_logfile1`: estos son los archivos del registro de transacciones. También encontraremos un archivo `ibdata1`, que es el tablespace del sistema. Este archivo contiene un cierto número de metadatos globales para InnoDB, tales como los datos que permiten anular las transacciones (*rollback segment* o *undo logs*).

Si examinamos ahora los archivos que contiene un subdirectorio (cada subdirectorio corresponde a un esquema), encontraremos los archivos `.frm` y probablemente los archivos `.ibd`. Los `.ibd` son los archivos de datos y de índices. Solo existen si la opción `innodb_file_per_table` está activada, lo que es el valor por defecto a partir de MySQL 5.6. De lo contrario, los datos e índices se almacenan en el tablespace de sistema, el famoso archivo `ibdata1`.

El funcionamiento de InnoDB es en realidad más complejo que el descrito en los párrafos anteriores. Si le interesa InnoDB, no deje de consultar la documentación en línea, que contiene numerosos detalles (<https://dev.mysql.com/doc/internals/en/innodb.html>).

b. Funciones principales

Sin entrar en todos los detalles, InnoDB ofrece las siguientes funcionalidades:

- Soporte de las transacciones (recordemos que el servidor MySQL no soporta esta

funcionalidad de forma nativa; los motores de almacenamiento son libres de implementarla o no).

- Recuperación automática en caso de fallo inesperado: mediante el registro de transacciones, InnoDB garantiza que toda transacción validada por el servidor (COMMIT) puede recuperarse incluso si el servidor se detiene de forma brusca y si los archivos de datos no han sido todavía actualizados.
- Soporte de claves externas.
- Bloqueo a nivel de filas.
- Cambios de esquema sin bloqueo: antes, todos los comandos ALTER TABLE bloqueaban las concurrentes en la tabla modificada, causando importantes problemas de operación para los cambios de esquema en tablas de gran volumen. A partir de MySQL 5.6, la mayoría de las modificaciones de esquema pueden realizarse aceptando escrituras en la tabla en curso de modificación. Encontraremos con frecuencia el término *Online DDL*.
- Compresión transparente: transparente significa aquí que los clientes no necesitan saber si el servidor comprime los datos o no. Observe que MySQL 5.7 introdujo un nuevo tipo de compresión (*punch-hole compression*) con el fin de resolver los múltiples problemas surgidos con la implementación utilizada hasta la versión 5.6, que no está libre de objeciones.
- Respaldo y restauración del buffer pool: hemos comprendido que la mayor parte de los resultados de InnoDB proviene de su uso del buffer pool. Sin embargo, ya que se trata de una caché, su contenido desaparecerá si MySQL debe reiniciarse. Los resultados se verán degradados de forma importante si el buffer pool no contiene suficientes datos relevantes, lo que puede tomar un tiempo considerable. Desde MySQL 5.6, InnoDB puede escribir una huella del buffer pool antes de la parada del servidor y utilizarla para recargar el contenido del buffer pool de forma automática en el momento del arranque.

Los parámetros correspondientes

```
son:innodb_buffer_pool_dump_at_shutdown=ON
```

```
e innodb_buffer_pool_load_at_startup=ON
```

- Soporte de la indexación fulltext: aunque InnoDB nunca podrá hacerle la competencia a software especializado en la búsqueda fulltext, como ElasticSearch

2. MyISAM

Este es el motor de almacenamiento histórico del servidor MySQL. Todavía se utiliza por defecto para los datos de la base de sistema creada al inicializar el servidor (`base mysql`).

En el disco duro, una tabla MyISAM está representada por tres archivos:

- `nombre_tabla.frm`, que contiene la estructura de la tabla.
- `nombre_tabla.MYD`, que contiene los datos.
- `nombre_tabla.MYI`, que contiene los índices.

Comparada con InnoDB, su arquitectura es más simple y su huella en disco y memoria menos importante. Implementa un bloqueo a nivel de tabla, lo que supone graves problemas de rendimiento de forma muy rápida. El motor de almacenamiento MyISAM no es transaccional, no cuenta con soporte para claves externas y no posee ningún sistema de recuperación de datos automático después de un crash (*crash recovery*). Por último, no es posible efectuar copias de seguridad en caliente. Rendimiento inferior a InnoDB y baja seguridad de datos: estas dos características explican por qué este motor ha caído en desuso. No se han realizado cambios importantes en su código desde hace diez años.

Se recomienda no utilizar nunca MyISAM, excepto si es un experto en MySQL y entiende perfectamente las consecuencias de su elección. La sola idea de que un fallo inesperado del servidor nos hará perder los datos debe desalentar el uso de este motor de búsqueda.

A menudo, encontraremos en Internet artículos que sugieren que MyISAM es más rápido que InnoDB para lecturas y se recomienda optar por MyISAM si nuestra aplicación genera de forma principal lecturas. Sepa que esta afirmación es totalmente falsa. Incluso en el caso más favorable para MyISAM (sin escritura), InnoDB mantiene una ventaja, ya que puede almacenar en memoria datos e índices, mientras que MyISAM no almacena en memoria los índices. Los datos MyISAM están en el mejor de los casos disponibles en la caché del sistema de archivos, pero se trata de un modo de almacenamiento mucho menos eficaz que la memoria gestionada de forma directa por el motor de almacenamiento.

3. Memory

El motor de almacenamiento `Memory` permite almacenar los datos y los índices de la tabla solo en la memoria. En caso de parada y reinicio, esta información se pierde. En contraposición, la estructura de la tabla es persistente, ya que está contenida en un archivo de formato `nombre_tabla.frm` que se genera durante la creación de la tabla. Los registros son de longitud fija; en otras palabras, las columnas de tipo `VARCHAR` se almacenan y tratan como columnas de tipo `CHAR`. Con nuestras tablas `Memory`, debemos tener cuidado con no consumir toda la memoria.

Para esto, MySQL ofrece dos opciones: `MAX_ROWS`, que permite fijar un límite máximo local en cada una de las tablas, y `max_heap_table_size`, opción del servidor que permite fijar el límite máximo para todas las tablas `Memory` del servidor.

Otra particularidad de este motor es que permite implementar por defecto un algoritmo de control para los índices. La distribución es muy eficaz para las peticiones de igualdad (`WHERE columna_con_idx = 10`), sobre todo en las columnas sin duplicación. Por consiguiente, es especialmente adecuado para claves primarias e índices únicos. Por el contrario, para las peticiones de desigualdades (`<`, `<=`, `>`...) o si hay mucha redundancia, debemos utilizar preferentemente el algoritmo `BTREE`.

Índice en una tabla `Memory`

El comando `SHOW INDEX FROM` permite obtener información sobre los índices de una tabla.

```
mysql> CREATE TABLE memoria (id int PRIMARY KEY)ENGINE=Memory;
Query OK, 0 rows affected (0,09 sec)
```

```
mysql> SHOW INDEX FROM memoria \G
***** 1. row *****

Table: memoria
Non_unique: 0
Key_name: PRIMARY
Seq_in_index: 1
Column_name: id
```

Collation: NULL

Cardinality: 0

Sub_part: NULL

Packed: NULL

Null:

Index_type: HASH

Comment:

1 row in set (0,00 sec)

```
mysql> CREATE UNIQUE INDEX idx_id_btree USING BTREE ON memoria (id);
```

Query OK, 0 rows affected (0,07 sec)

Records: 0 Duplicates: 0 Warnings: 0

```
mysql> SHOW INDEX FROM memoria\G
```

***** 1. row *****

Table: memoria

Non_unique: 0

Key_name: PRIMARY

Seq_in_index: 1

Column_name: id

Collation: NULL

Cardinality: 0

Sub_part: NULL

Packed: NULL

Null:

Index_type: HASH

Comment:

***** 2. row *****

Table: memoria

Non_unique: 0

Key_name: idx_id_btree

Seq_in_index: 1

Column_name: id

Collation: A

Cardinality: NULL

Sub_part: NULL

Packed: NULL

Null:

Index_type: BTREE

Comment:

2 rows in set (0,00 sec)

Las principales características del motor `Memory` son:

- La velocidad de lectura y escritura, porque los datos están en la memoria.
- La implementación de algoritmos de índice b-tree y hash.
- El bloqueo a nivel de tabla.
- La falta de persistencia de datos e índices.
- La falta de soporte de los tipos `BLOB` y `TEXT`.
- El funcionamiento no transaccional, la ausencia de crash recovery.
- La falta de copias de seguridad en caliente.
- La falta de soporte de claves externas.

Debe saber, por último, que un error típico es utilizar el motor `Memory` para pequeñas tablas a las que se accede con mucha frecuencia o tablas que solo contienen datos temporales. Es mucho más seguro utilizar `InnoDB` como motor: nos beneficiaremos de un bloqueo a nivel de filas y no a nivel de tabla, los datos no corren el riesgo de desaparecer en caso de fallo inesperado del servidor y la velocidad de acceso se preservará, ya que los datos estarán en la caché de `InnoDB`.

La principal función de este motor es almacenar temporalmente tablas implícitas creadas por peticiones.

- ☐ Al arrancar el servidor, las tablas `Memory` están vacías. Podemos rellenarlas de forma automática al arrancar el servidor MySQL con la opción `init-file`, que hace referencia a un archivo que contiene comandos SQL de tipo `LOAD DATA INFILE` o `INSERT INTO ... SELECT`.

4. Archive

El motor `Archive` comprime los datos al almacenarlos. La ganancia de espacio en comparación con una tabla `MyISAM` puede alcanzar el 70 %. El motor solo permite las peticiones `INSERT` y `SELECT`.

Sus principales características son:

- El almacenamiento y la compresión de los datos.
- La falta de índices.
- El bloqueo a nivel de registro.
- El funcionamiento no transaccional, la ausencia de crash recovery.
- La falta de copias de seguridad en caliente.
- La falta de soporte de claves externas.

5. XtraDB

`XtraDB` es un motor creado por Percona. Está basado en `InnoDB` y tiene como objetivo ser más eficiente y tener una mayor capacidad para soportar la carga en arquitecturas multiprocesador. Este motor es totalmente compatible con `InnoDB` y puede sustituirse sin tocar la aplicación.

En el resto de este libro, y salvo excepciones, no haremos ninguna distinción entre `InnoDB` y `XtraDB`.

6. TokuDB

Este motor ha sido creado por Tokutek, empresa posteriormente adquirida por Percona. `TokuDB` es transaccional, soporta claves externas y puede ser respaldado en caliente, lo que lo convierte en realidad en un competidor de `InnoDB`. La originalidad del motor reside en el almacenamiento de los datos: en lugar de utilizar b-trees como casi todos los motores, `TokuDB` utiliza Fractal Trees.

Este tipo de árbol se caracteriza fundamentalmente por su eficacia en la escritura: mientras que todos los motores basados en b-trees ven sus rendimientos en escritura caer de forma drástica a medida que los datos ocupan cada vez más memoria, `TokuDB` es capaz de

mantener un alto nivel de rendimiento de escritura con grandes volúmenes de datos.

El método de almacenamiento tiene, sin embargo, un sobrecoste para las lecturas en comparación con el tradicional b-tree. Además, TokuDB tiene varios defectos que lo hacen a veces difícil de utilizar. Por ejemplo, todos los archivos de datos se encuentran en la raíz del directorio de datos, lo que es muy molesto para los DBA o administradores de sistemas. Un problema aún más molesto es la pérdida de rendimiento que se produce cada 60 segundos en el momento del checkpointing.

TokuDB no está disponible en MySQL de forma directa, es recomendable usar Percona Server o MariaDB (ver con anterioridad en este capítulo la descripción de estas dos variantes de MySQL).

7. RocksDB

RocksDB es otro competidor de InnoDB. Al igual que TokuDB, se desarrolló a partir de un algoritmo de almacenamiento muy eficaz para las escrituras: el LSM Tree (*Log-Structured Merge Tree*).

RocksDB es desarrollado por Facebook y no está disponible en ninguna de las variantes principales de MySQL: la única manera de utilizar RocksDB es la compilación a partir de las fuentes con la variante Facebook de MySQL. No se podía utilizar en producción en el momento de escribir este libro.

8. Otros motores

Existen muchos otros motores más:

- CSV: almacenamiento de datos en formato CSV, como su nombre indica.
- Aria: un sustituto de MyISAM en MariaDB, nunca desarrollado por completo.
- Federated: un motor para acceder de forma transparente a tablas almacenadas en servidores remotos.
- Spider: un motor que permite el *sharding* automático.

Todos estos motores son exóticos y muy poco utilizados. Lea con atención la

documentación correspondiente (si existe) para entender sus numerosas limitaciones y no los utilice en producción hasta haberlos probado exhaustivamente.

Para terminar, he aquí un cuadro sinóptico de las principales características de los motores mencionados:

Motor	Tipo de bloqueos	Transacciones	Resistente a las paradas imprevistas	Respaldo en caliente
InnoDB	Fila	Sí	Sí	Sí
MyISAM	Tabla	No	No	No
	Tabla	No	No	No
Archive	Tabla	No	No	No
XtraDB	Fila	Sí	Sí	Sí
TokuDB	Fila	Sí	Sí	Sí
RocksDB	Fila	Sí	Sí	Sí

Bloqueos y transacciones

Imaginemos que abrimos un archivo para leer su contenido y que, al mismo tiempo, otro usuario abre el archivo para modificarlo. Corremos el riesgo de encontrar contenido incoherente, ya que una parte puede haber sido modificada mientras no llegan los datos. Para asegurarnos de poder leer el archivo sin que nadie lo cambie, debemos informar a los demás usuarios.

Para MySQL, la situación es idéntica: los accesos concurrentes estarán autorizados. Por lo tanto, es posible que una petición lea un conjunto de filas, mientras que otra petición cambia algunas de estas filas. Por ello, es indispensable que cada conexión MySQL indique a las demás conexiones cuáles son los recursos que no deben ser modificados. Esta notificación se efectúa mediante la incorporación de bloqueos.

Existen dos tipos de bloqueos: los bloqueos de lectura, que permiten a otras conexiones leer los mismos datos pero no modificarlos, y los bloqueos de escritura, que prohíben a todas las demás conexiones leer o escribir.

La diferencia entre estos dos tipos de bloqueos es fácil de comprender si recordamos el ejemplo del archivo que queremos abrir. Si solo queremos leer este archivo, otros usuarios también pueden abrir el archivo para leerlo. Pero nadie tiene derecho a modificar, si no corremos el riesgo de leer datos incoherentes. Y, si deseamos actualizar el contenido del archivo, no queremos que otros usuarios puedan leer o escribir en este para evitar que el contenido sea incoherente y por lo tanto inservible.

Siguiendo la analogía del archivo abierto para lectura o escritura, si pensamos que el archivo cuenta con varias secciones, es posible que permitamos a otro usuario modificar las secciones que no estamos leyendo. En este caso, lo que leamos será siempre coherente, ya que nadie modificará el contenido, y haremos esperar menos tiempo (por no decir ninguno) a los usuarios que quieren realizar modificaciones en el archivo.

Con MySQL, el mismo principio existe: podemos bien bloquear una tabla entera, bien bloquear una o varias filas. El bloqueo de filas es mucho más potente porque permite a menudo, en cada conexión, acceder a las filas necesarias sin tener que esperar a que otras conexiones hayan terminado su trabajo. Este tipo de bloqueo está, sin embargo, disponible solo en algunos motores de almacenamiento, por ejemplo InnoDB.

1. Bloqueos implícitos

a. Líneas generales

Las peticiones de lectura y escritura (SELECT, INSERT, UPDATE...) plantean un bloqueo implícito cuyo nivel (en principio tabla o fila) está gestionado por el motor de almacenamiento. Los motores más antiguos, como MyISAM o Memory, bloquean la tabla completa, mientras que los motores más recientes, como InnoDB o TokuDB, solo bloquean las filas necesarias.

En caso de mucha concurrencia, las mismas peticiones realizadas en tablas que contengan datos idénticos, pero que no tengan los mismos motores, pueden tener resultados completamente diferentes, solo a causa del nivel de bloqueo. Es además una de las razones principales por las cuales el motor MyISAM ofrece resultados tan mediocres. Imaginemos una tabla con cien millones de filas y que ejecutamos una petición que va a leer una fila: a pesar de todo, los cien millones de filas quedarán inaccesibles para su escritura.

b. Especificidades InnoDB

En muchos casos, este motor de almacenamiento no tiene necesidad de bloquear las filas para las lecturas. InnoDB dispone de un mecanismo de multiversión, llamado MVCC (*multiversion concurrency control*), que permite consultar una instantánea de datos y que, por norma, requiere menos trabajo por parte del servidor que el bloqueo de filas.

InnoDB utiliza el bloqueo de filas si es necesario, sin escalar nunca al bloqueo de toda la tabla. En algunos casos, esta estrategia es poco eficaz, por ejemplo cuando InnoDB deba recorrer todas las filas de la tabla y, por lo tanto, bloquearlas de forma consecutiva.

InnoDB lleva a cabo los bloqueos necesarios a medida que considera que se deben realizar. Según el plan de ejecución de la petición, de esta forma se podrá conducir a

InnoDB a bloquear muchas más filas de las que podríamos imaginar. Proporcionaremos un ejemplo un poco más adelante en este capítulo.

La asociación de lecturas potenciales sin bloqueo usando la multiversión y bloqueos de filas hace a InnoDB muy eficaz cuando el servidor recibe una mezcla de peticiones de lectura y escritura.

En resumen, con InnoDB, es posible que, de forma simultánea en la misma tabla, varios clientes:

- lean las filas, idénticas o diferentes,
- lean las filas y escriban otras filas,
- escriban en filas diferentes.

Sin embargo, no es posible que algunos clientes escriban de forma simultánea en una misma fila de una tabla.

- ☐ Observe también que algunas peticiones, como `ALTER TABLE`, provocarán un bloqueo total de la tabla. La situación está cambiando, ya que desde MySQL 5.6 la mayoría de los cambios de esquema con `ALTER TABLE` no bloquean la tabla (ver el capítulo Optimización para más detalles).

2. Bloqueos explícitos

a. Bloqueos de tabla

MySQL ofrece la posibilidad de bloquear una o varias tablas para una instrucción. El objetivo no es corregir los errores de bloqueo por parte del servidor, sino indicar al servidor que se necesita bloquear una o varias tablas al realizar ciertas operaciones.

De forma general, un usuario puede pedir al servidor ubicar un bloqueo compartido, que permite a otros clientes acceder en modo solo lectura a las tablas bloqueadas, o un bloqueo exclusivo, que impide el acceso a las tablas bloqueadas. El comando que se debe utilizar es `LOCK TABLES`, con varias palabras clave posibles.

- `READ` impone un bloqueo compartido.

- WRITE impone un bloqueo exclusivo.

Ejemplo de uso

```
mysql> LOCK TABLES t1 READ, t2 READ, t3 WRITE;
```

- ☐ Tenga en cuenta que es fundamental bloquear en una sola instrucción el conjunto de las tablas que desea bloquear. De hecho, la instrucción LOCK TABLES tiene como efecto implícito el desbloqueo de todas las tablas que hayan sido bloqueadas antes.
- ☐ Un bloqueo READ impide todas las escrituras en los recursos que hayan sido bloqueados, incluso para el cliente que colocó el bloqueo.

El comando UNLOCK TABLES desbloquea todas las tablas bloqueadas por el usuario.

Ejemplo de uso

```
mysql> UNLOCK TABLES;
```

- ☐ La instrucción UNLOCK TABLES no permite desbloquear las tablas bloqueadas por otro cliente.

En la práctica, es bastante raro usar el comando LOCK TABLES en producción, en especial si el servidor está sujeto a una fuerte carga. De forma histórica, LOCK TABLES era utilizado en principio con el motor MyISAM para emular las transacciones. Desde que InnoDB se convirtió en el motor más corriente, LOCK TABLES casi ha desaparecido.

Existe otro comando que permite ubicar bloqueos: FLUSH TABLES WITH READ

LOCK. Esta instrucción cierra todas las tablas de todas las bases e implanta un bloqueo compartido sobre todas estas tablas. El bloqueo no es de la misma naturaleza que el planteado por LOCK TABLES: se trata en efecto de un bloqueo global, y no de un bloqueo sobre cada una de las tablas.

FLUSH TABLES WITH READ LOCK es útil para hacer un servidor accesible en solo lectura mientras dure una operación que solicite la interrupción de las escrituras, por ejemplo, con la replicación para compartir un maestro y un esclavo o justo antes de comenzar una copia de seguridad para guardar la posición actual en los registros binarios (para más detalles, consulte el capítulo Respaldo y restauración).

Ya que FLUSH TABLES WITH READ LOCK es casi indispensable para los respaldos, su empleo es mucho más frecuente que LOCK TABLES. Tenga en mente, sin embargo, que FLUSH TABLES WITH READ LOCK es particularmente intrusivo en un servidor de producción, ya que todas las grabaciones se cortan mientras el bloqueo está activo.

- ☐ No existe ninguna instrucción equivalente para obtener un bloqueo exclusivo sobre el conjunto de las tablas de todas las bases.
- ☐ La instrucción UNLOCK TABLES desbloquea las tablas bloqueadas con FLUSH TABLES WITH READ LOCK.

b. Temas específicos de InnoDB

Como se mencionó antes, InnoDB ejecuta las peticiones SELECT sin ningún bloqueo. Sin embargo, existen palabras clave para indicar al motor de almacenamiento que hay que bloquear las filas leídas.

- SELECT . . . LOCK IN SHARE MODE implementa un bloqueo compartido para las lecturas, pero no las escrituras.
- SELECT . . . FOR UPDATE plantea un bloqueo exclusivo, impidiendo todas las lecturas hechas con un SELECT . . . LOCK IN SHARE MODE. Las lecturas realizadas por los SELECT habituales no están bloqueadas gracias al

Estos modificadores solo tienen sentido dentro de una transacción: se pueden usar de forma exclusiva dentro de un bloque `START TRANSACTION` o cuando el modo `AUTOCOMMIT` se encuentra desactivado (abordaremos las transacciones más adelante en este capítulo).

`SELECT ... LOCK IN SHARE MODE` es poco común, pero `SELECT ... FOR UPDATE` es habitual cuando se trata de coordinar varias instancias de un mismo script para tratar una cola.

El siguiente es un algoritmo clásico:

- Inicie una transacción (`START TRANSACTION`).
- Seleccione y bloquee un elemento de la pila (`SELECT * FROM mi_cola ORDER BY created ASC LIMIT 1 FOR UPDATE`).
- Efectúe el trabajo requerido en el elemento de la pila.
- Cambie el estado del elemento tratado (`UPDATE mi_cola SET status = 'tratado' WHERE id=123`).
- Valide la transacción (`COMMIT`).

Este algoritmo es muy lógico, pero tiene un inconveniente importante: los bloqueos planteados por `SELECT ... FOR UPDATE` provocan importantes problemas de rendimiento si se suele acceder a la tabla que representa la cola porque estos bloqueos se suelen conservar durante largos períodos. Un mejor algoritmo es el siguiente:

- Actualice el estado de un elemento indicando qué instancia del script va a tratar el elemento (`UPDATE mi_cola SET status = 'en curso', worker_id=456 order by created ASC LIMIT 1`).
- Seleccione la información del elemento que se ha de tratar (`SELECT * FROM mi_cola WHERE status = 'en curso' AND worker_id=456`).
- Efectúe el trabajo requerido en el elemento de la pila.
- Cambie el estado del elemento tratado (`UPDATE mi_cola SET status = 'tratado' WHERE id=123`).

La diferencia entre los dos algoritmos es solo que en el segundo caso no es necesario bloquear el elemento que se ha de tratar durante el paso del script de tratamiento, aunque este script puede durar varios segundos, mientras que en el caso de ejecutar una petición `update` de forma directa, el bloqueo durará, como mucho, unos pocos milisegundos solamente.

- En general, siempre es posible y preferible evitar `SELECT . . . LOCK IN SHARE MODE` o `SELECT . . . FOR UPDATE`.

3. Bloqueos cooperativos

MySQL dispone de instrucciones para gestionar los bloqueos cooperativos con las funciones `GET_LOCK()`, `RELEASE_LOCK()` e `IS_free_LOCK()`. Los bloqueos puestos o eliminados por estas funciones no son verdaderos bloqueos que impidan el acceso a los recursos, sino indicaciones para que los clientes que van a utilizar estos recursos puedan coordinarse (de ahí el término bloqueo cooperativo).

Por ejemplo, los bloqueos cooperativos permiten centralizar los bloqueos en el servidor de bases de datos cuando el acceso a un recurso debe administrarse entre varios frontales web.

Para adquirir un bloqueo cooperativo:

```
mysql> SELECT GET_LOCK('mi_bloqueo',10);
```

El primer argumento es el nombre del bloqueo, el segundo es el tiempo que se debe esperar si el bloqueo no puede ser obtenido de forma inmediata. La función devuelve 1 si el bloqueo se obtiene y 0 en caso de timeout.

Para eliminar el bloqueo así obtenido:

```
mysql> SELECT RELEASE_LOCK('mi_bloqueo');
```

La función devuelve 1 si el bloqueo ha sido eliminado.

Para informarse sobre el estado de un bloqueo:

```
mysql> SELECT IS_FREE_LOCK('mi_bloqueo');
```

La función devuelve 1 si el bloqueo ya no existe y 0 en caso de existir.

4. Transacciones

a. Líneas generales

En un primer momento, una transacción es un grupo de peticiones que son consideradas por el servidor como una sola unidad lógica: todas las peticiones se ejecutan con éxito y las modificaciones se actualizan de manera permanente en la base de datos (COMMIT), o bien se produce un error y todas las modificaciones introducidas en la transacción son anuladas (rollback).

El simple mecanismo de COMMIT y ROLLBACK no basta para que un sistema sea considerado transaccional. También se requieren funcionalidades adicionales agrupadas bajo el acrónimo ACID:

- Atomicidad: o todas las peticiones de la transacción se llevan a cabo con éxito, o todas están anuladas.
- Coherencia: si la base de datos se encuentra en un estado coherente al comienzo de una transacción, se mantiene en un estado coherente al final de la transacción, pase lo que pase durante la transacción.
- Aislamiento (en inglés «Isolation»): las peticiones de una transacción no afectan a otras transacciones.
- Durabilidad: cuando un COMMIT se ejecuta, el sistema garantiza que los datos están bien grabados y que los datos no se pierden.

El trabajo adicional que se solicita al servidor para asegurar las características ACID está lejos de ser insignificante. Por esta razón, los motores transaccionales son siempre muy

complejos, aunque esta complejidad no aparece siempre de forma inmediata durante su uso.

- ☐ Sin embargo, es aún más costoso tratar de emular las transacciones con un motor no transaccional y bloqueos explícitos.

Es posible modificar los parámetros de aislamiento y durabilidad del servidor para mejorar el rendimiento de los motores transaccionales, pero esto conlleva una pérdida de las características ACID y, por lo tanto, de la seguridad de los datos. Esta configuración se aborda en profundidad en el capítulo Configuración del servidor.

b. InnoDB y las transacciones

InnoDB respeta las características ACID, lo que lo convierte en un motor completamente transaccional. Por defecto, funciona en modo AUTOCOMMIT, lo que significa que cada petición se trata como parte de una transacción separada.

En modo AUTOCOMMIT, las peticiones siguientes:

```
mysql> INSERT INTO t1 (nombre,edad) VALUES ('Juan',30);  
mysql> SELECT * FROM t1;
```

Son equivalentes a:

```
mysql> START TRANSACTION;  
mysql> INSERT INTO t1 (nombre,edad) VALUES ('Juan',30);  
mysql> COMMIT;  
mysql> START TRANSACTION;  
mysql> SELECT * FROM t1;  
mysql> COMMIT;
```

c. Mezclar motor transaccional y motor no transaccional

La utilización en las transacciones de tablas con un motor no transaccional con tablas con

un motor transaccional es una fuente potencial de errores en caso de ROLLBACK. En efecto, las peticiones realizadas sobre una tabla no transaccional no se anularán. MySQL se limitará a emitir una advertencia.

Ejemplo de esa situación con las tablas siguientes:

```
CREATE TABLE t1 (  
    id INT,  
    nombre VARCHAR(30)  
)ENGINE=MyISAM;
```

```
CREATE TABLE t2 (  
    id INT,  
    total INT  
)ENGINE=InnoDB;
```

Las peticiones siguientes muestran lo que puede pasar en una transacción:

```
mysql> START TRANSACTION;  
mysql> INSERT INTO t1 (id,nombre) VALUES (1,'Juan');  
mysql> INSERT INTO t2 (id,total) VALUES (1,10);  
mysql> ROLLBACK;  
Query OK, 0 rows affected, 1 warning (0,02 sec)
```

```
mysql> SHOW WARNINGS\G
```

```
***** 1. row *****
```

```
Level: Warning
```

```
Code: 1196
```

```
Message: Some non-transactional changed tables couldn't be rolled  
back
```

```
mysql> SELECT * FROM t1;
```

```
+-----+-----+  
| id    | nombre |  
+-----+-----+
```

```
|      1 | Juan-- |  
+-----+-----+
```

```
mysql> SELECT * FROM t2;  
Empty set (0,00 sec)
```

La inserción en la tabla `t1` no se ha podido anular, pero la advertencia emitida por MySQL pasa desapercibida con facilidad. Como se indica en la parte de los motores de almacenamiento, preferimos utilizar siempre un solo motor de almacenamiento (y, salvo razón particular, InnoDB será la mejor opción).

d. Interbloqueos (deadlocks)

Los deadlocks (término traducido al español como «abrazo de la muerte» o interbloqueo) ocurren cuando varios clientes en curso de ejecución de una demanda están bloqueados e impiden de forma mutua su consecución. Esta situación se producirá, por ejemplo, cuando las transacciones intentan bloquear de forma simultánea recursos bloqueados por otra transacción.

He aquí un ejemplo de deadlock posible:

Transacción 1

```
mysql> START TRANSACTION;  
mysql> UPDATE persona SET nombre='Pedro' WHERE id=1;  
mysql> UPDATE persona SET nombre='Pablo' WHERE id=2;
```

Transacción 2

```
mysql> START TRANSACTION;  
mysql> UPDATE persona SET nombre='Jacobo' WHERE id=2;  
mysql> UPDATE persona SET nombre='Daniel' WHERE id=1;
```

Si cada transacción ejecuta sucesivamente una petición del listado anterior, la ejecución del segundo UPDATE será imposible para cada una de las transacciones, ya que cada una esperará que la otra libere el bloqueo exclusivo que se estableció: ¡deadlock!

InnoDB adquiere siempre los bloqueos, a medida que el motor lo considere necesario. Por lo tanto, dicha situación de deadlock puede ocurrir de vez en cuando. Observe que, a menudo, InnoDB es capaz de detectar por sí mismo los deadlocks.

En este caso, el motor provoca un ROLLBACK de la transacción que ha iniciado el menor número de bloqueos exclusivos. El objetivo es que cada vez que se plantee un bloqueo exclusivo, tenga lugar una escritura. Eligiendo la transacción que tiene menos bloqueos exclusivos, InnoDB tiene buenas posibilidades de elegir la transacción que llevó a cabo el menor número de escrituras y, por tanto, es más fácil de cancelar.

Si el deadlock no es detectado, las transacciones implicadas terminarán por rebasar el tiempo máximo de espera para las transacciones y sufrirán un ROLLBACK consecutivo hasta que la situación se desbloquee.

- ☐ Este escenario no se puede producir con MySQL SAM, ¡que no tiene soporte para transacciones y que bloquea a nivel de tablas y no de las filas!

Información General

1. Estabilidad de las versiones

Solo las versiones clasificadas como GA (*Generally Available*) se consideran estables y pueden ser instaladas sin riesgo en entornos de producción.

Salvo por una razón específica, debemos evitar las antiguas versiones estables, que solo benefician a lo sumo las correcciones de seguridad, pero no aportan nuevas funcionalidades.

Las versiones RC (*Release Candidate*) son versiones candidatas al título de versiones estables y son por lo general de buena calidad. Sin embargo, salvo en casos excepcionales, no están recomendadas para uso en producción.

En cuanto a las versiones alfa o beta, son perfectas para descubrir las últimas funcionalidades en un entorno de prueba o de desarrollo, pero no son adecuadas para un entorno de producción debido a su falta de madurez.

2. Versión comunitaria y versión Enterprise

La versión comunitaria está disponible bajo licencia GNU GPL V2.

La versión Enterprise, bajo licencia comercial, se suministra con soporte de Oracle y herramientas de ayuda sobre la administración: el Enterprise Monitor, para la vigilancia de los foros, el Query Analyzer, para la detección y análisis de las peticiones de bajo rendimiento o la herramienta de backup MySQL Enterprise Backup.

Para una versión determinada, el servidor MySQL es el mismo ya estemos utilizando la versión comunitaria o la versión Enterprise. Este dato es importante porque no siempre ha

sido el caso, y podremos encontrar en Internet páginas antiguas que llevan a pensar lo contrario.

Todos los ejemplos de este libro se basan en las versiones comunitarias.

3. Ciclo de desarrollo

Existen fundamentalmente tres tipos de versiones disponibles para descargar:

- Las versiones estables (GA) se pueden utilizar tanto en producción como en entornos de prueba.
- Las versiones DMR (*Development Milestone Release*) se publican tres o cuatro veces al año. El nivel de calidad es de tipo *Release Candidate*. Podemos esperar un producto estable de forma razonable. El objetivo de las versiones DMR es proporcionar un resumen de las características de la próxima versión estable y recoger el retorno de los usuarios.
- Las versiones *Lab Release* proponen una nueva funcionalidad importante, que aún está en fase de desarrollo. Se trata esencialmente de obtener un feedback por parte de los usuarios con el que valorar si la funcionalidad y sus opciones cumplen con las expectativas. Estas versiones solo pueden servir para pruebas, ya que no se garantiza ninguna estabilidad y la funcionalidad en cuestión puede experimentar aún cambios significativos.

Cabe señalar que, una vez que una versión mayor se publica, no se efectúa ningún cambio significativo. Solo las correcciones de problemas y las correcciones de seguridad se añaden a las distintas versiones de menor importancia. Esta lógica, sin embargo, se encuentra en fase de cambio con MySQL 5.7, ya que la funcionalidad de cifrado de datos se ha añadido tras la publicación de la primera versión estable y que Oracle ofrece ahora nuevas funciones no estabilizadas en forma de plugins.

4. Elección del tipo de instalación

Sin entrar en detalles que son el objeto de los párrafos siguientes, podrá elegir entre dos opciones para realizar la instalación: la compilación de fuentes y la utilización de un ejecutable precompilado.

La compilación, para llevarse a cabo de forma correcta, requiere competencias que exceden el marco de este libro y está reservada para usos específicos (como host, por ejemplo). Por lo tanto, las secciones siguientes describirán solo los métodos de instalación a partir de archivos ejecutables.

Instalación en UNIX y derivados

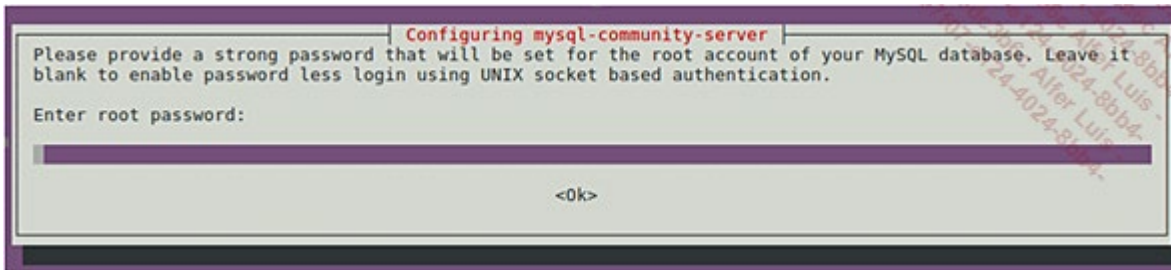
1. Instalación por gestor de paquetes

Se trata de la manera más fácil de instalar el servidor MySQL. Basta con conocer el nombre del paquete, que es a menudo `mysql-server` o `mysql-server-version`, y luego invocar al gestor de paquetes.

Ejemplo en Ubuntu en línea de comandos

```
sudo aptitude install mysql-server-5.7
```

El proceso de instalación se realiza de forma automática. El sistema solicitará la contraseña para la cuenta `root` durante la instalación.



El instalador podrá solicitar la configuración requerida para permitir el envío de e-mails (por defecto, seleccionamos **Ninguna configuración**).

- Algunas distribuciones proporcionan una versión oficial modificada. Por ejemplo, para Debian y sus derivados, como Ubuntu, el archivo principal de configuración se encuentra en `/etc/mysql/` y no en `/etc/`, como se ve

con frecuencia en la documentación oficial. Estos cambios pueden ser fuente de confusión.

Una vez finalizada la instalación, podemos arrancar el servidor con el comando:

```
# service mysql start
```

Los paquetes presentes por defecto en las distribuciones no ofrecen siempre la versión más reciente de MySQL 5.7. A veces, contaremos solo con MySQL 5.6 o 5.5. En este caso, puede ser muy interesante utilizar los repositorios oficiales de Oracle, que están disponibles para Debian y derivados, así como para Red Hat y derivados.

Para Ubuntu, los pasos de instalación son los siguientes:

- ❑ Descargue el archivo `.deb` desde la página siguiente:

<http://dev.mysql.com/downloads/repo/apt>

- ❑ Instale el archivo `.deb`.

```
# dpkg -i mysql-apt-config_version.deb
```

- ❑ Refresque los repositorios:

```
# apt-get update
```

- ❑ Instale el paquete `mysql-server`:

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

En todo momento, se puede especificar las versiones principales a las que se podrá acceder

a partir del repositorio de Oracle mediante el comando:

```
# dpkg-reconfigure mysql-apt-config
```

Los comandos para las distribuciones Red Hat/CentOS son similares, utilizando `yum` en lugar de `apt-get`.

2. Instalación con ejecutables precompilados

¿Por qué no usar el gestor de paquetes si se trata aparentemente de la manera más fácil de instalar el servidor? Son varias las razones por las que es preferible optar por ejecutables precompilados:

- Si deseamos ejecutar varias instancias de MySQL en el mismo servidor, el gestor de paquetes no lo permite.
- Si deseamos utilizar una versión que no está todavía disponible en el gestor de paquetes, los ejecutables precompilados son la única solución.
- Si no deseamos que el gestor de paquetes actualice de forma automática MySQL (sin embargo, es posible indicar al gestor de paquetes que no hay que actualizar MySQL).

En la página de descarga del sitio www.mysql.com, están disponibles los ejecutables precompilados bajo la denominación «Linux (no RPM packages)», y se presentan en forma de archivos en formato `tar.gz`.

Comience por descargar y descomprimir el archivo en el directorio deseado. Si selecciona la línea de comandos, tendrá que introducir los siguientes comandos:

```
wget archivo.tar.gz
```

```
tar xzf archivo.tar.gz
```

reemplazando *archivo* por el nombre del archivo descargado.

La instalación propiamente dicha consiste en resolver los permisos de los archivos y en ejecutar después el script `mysqld` con opciones para crear las tablas del sistema.

- La inicialización básica del sistema se hacía hasta la versión 5.6 con el script `mysql_install_db`. Si utilizamos una versión distinta a MySQL 5.7, deberemos utilizar este script, ya que las opciones de `mysqld` descritas a continuación no existen.

El conjunto de operaciones se realizan con la cuenta de `root` en el directorio raíz del archivo descargado. Estas son las operaciones necesarias:

- Creación del usuario y del grupo `mysql`:

```
groupadd mysql
useradd -g mysql mysql
```

- Configuración de los permisos:

```
chown -R mysql .
chgrp -R mysql .
```

- Inicialización de las tablas del sistema:

```
# ./bin/mysqld --initialize -user=mysql
[...]
2016-05-25T15:53:12.798942Z 1 [Note] A temporary password is
generated for root@localhost: usk+yK+gE1&T
```

Observe la última línea que se muestra al ejecutar el comando: esta contiene la contraseña del usuario `root@localhost`. Sin esta contraseña, no podrá contactar con su

instancia.

- Generar una contraseña de forma automática durante la instalación del servidor es beneficioso en términos de seguridad. Pero si la instalación se realiza mediante un script, debemos añadir código para recuperar la contraseña, lo que puede ser difícil. En este caso, puede ser útil utilizar la opción `--initialize-insecure` en lugar de `--initialize`. Así, el usuario `root@localhost` no tendrá contraseña.

Si ya tenemos un servidor MySQL funcionando en el mismo equipo, es preferible crear un archivo `my.cnf` y pasar el parámetro:

```
# ./bin/mysqld --defaults-file=ruta/a/my.cnf --initialize  
-user=mysql
```

- `--defaults-file` debe ser **siempre** la primera opción pasada a `mysqld`.

- Configuración final de los permisos:

```
chown -R root .  
chown -R mysql data
```

El servidor está listo para arrancar, pero todavía no es seguro. La seguridad de la instalación se abordará más adelante en el capítulo.

3. Arranque del servidor

Todos los métodos descritos para iniciar el servidor MySQL requieren estar conectado como `root`.

a. Script `mysql.server`

Cuando MySQL está instalado como servicio, como es el caso cuando se usa el gestor de paquetes, debemos arrancar el servidor con la acción `start` del script `mysql.server`. Si hemos instalado MySQL de forma manual, este script se encuentra en el directorio `soporte-files/`; mientras que si hemos utilizado el gestor de paquetes, el script habrá sido renombrado como `mysql` durante el registro de la instancia como servicio.

Según las distribuciones, la llamada a este script variará ligeramente. En Ubuntu, por ejemplo, usaremos el siguiente comando:

```
$ sudo /etc/init.d/mysql start
```

- ☐ El script `mysql.server` solo tiene previsto aceptar como parámetro una acción como `start` o `stop`. En el caso de instalar varias instancias de MySQL en el mismo host, es difícil utilizar este script para arrancar varias instancias. La única opción posible es modificarlo a mano.

b. Script `mysqld_safe`

`mysqld_safe` ejecuta `mysqld` y crea un registro de errores de forma automática. Se trata del método recomendado en la documentación oficial para iniciar el servidor. `mysqld_safe` tiene la particularidad de reiniciar `mysqld` de forma automática si este concluye de forma anormal.

- ☐ El script `mysql.server` llama a `mysqld_safe`.

Si, como consecuencia de una instalación con archivos ejecutables, `mysqld_safe` no

consigue localizar la ubicación del servidor, podemos indicarle al script la ruta hacia `mysqld` (opción `--ledir` descrita con más detalle en la parte sobre `mysqld_multi`) y la ruta al archivo de configuración `my.cnf` (opción `--defaults-file`).

- ☐ En las distribuciones en las que los servicios son controlados por `systemd`, `mysqld_safe` no es útil.

c. Invocación directa de `mysqld`

La última posibilidad es llamar de forma directa a `mysqld`:

```
/ruta/hacia/mysqld --defaults-file=/ruta/hacia/my.cnf  
--user=mysql &
```

Los mensajes de error o de advertencia aparecen de forma directa en la consola a menos que especifiquemos la opción `--log-error` en la línea de comandos. En principio, no tendremos que emplear este método para la depuración, salvo cuando necesitemos entender por qué el servidor no arranca de forma correcta.

- ☐ El parámetro `--defaults-file` debe colocarse forzosamente en primera posición en la lista de opciones.

4. Parada del servidor

a. Script `mysql.server`

Si MySQL está instalado como un servicio, debemos detener el servidor usando el comando `stop` del script `mysql.server`. Por ejemplo, en Ubuntu:

```
$ sudo /etc/init.d/mysql stop
```

b. mysqladmin

En todos los casos, `mysqladmin` puede detener el servidor. Necesitará proporcionar a `mysqladmin` el login y la contraseña de un usuario que tenga el derecho SHUTDOWN:

```
$ mysqladmin -uroot -p shutdown
```

- ☐ Atención, `mysqladmin stop` no detiene el servidor, sino solo los procesos de replicación si el servidor es un esclavo.
- ☐ Con la opción `-u`, podemos unir el nombre de usuario con la opción (`-uroot`) o dejar un espacio (`-u root`), pero, si le indicamos la contraseña en la línea de comandos, no debemos dejar un espacio tras la opción `-p` (`-pxxx`).
- ☐ La opción `--defaults-file` está disponible para `mysqladmin`. En Debian/Ubuntu, por ejemplo, podemos recurrir al archivo `debian.cnf` creado por defecto y utilizar el siguiente comando para detener el servidor:

```
shell> mysqladmin --defaults-file=/etc/mysql/debian.cnf shutdown
```

c. Comando kill

Si se diera un problema muy grave, puede que ninguno de los métodos descritos anteriormente funcionara. En tal caso, la única solución sería detener brutalmente el proceso `mysqld`. Atención, esta forma de proceder puede provocar la corrupción de

datos, en particular para las tablas MyISAM, o errores de replicación. `Kill -9` solo debe utilizarse como último recurso, cuando los otros mecanismos han fracasado.

- ☐ Si MySQL se ha iniciado con `mysqld_safe`, ¿debemos primero matar `mysqld_safe` antes de matar `mysqld` para evitar que `mysqld_safe` reinicie automáticamente `mysqld` !

5. Resolución de problemas de instalación frecuentes

En caso de problema, lo primero que debe hacerse es examinar el registro de errores, que se encuentra por defecto en el directorio de datos con una extensión `.err`, y comprobar los permisos de los ejecutables y el directorio de datos.

- ☐ Para determinar la ubicación del directorio de datos, puede buscar la variable `datadir` en el archivo de configuración `my.cnf`.

a. Errores InnoDB

InnoDB es el motor transaccional más utilizado en el universo MySQL. Los datos se guardan temporalmente en los registros especiales antes de ser transferidos a un espacio de almacenamiento llamado `tablespace`.

InnoDB es muy estricto con los registros y los archivos de su `tablespace`. Cualquier problema puede impedir, por lo tanto, el arranque del servidor. En este caso, como en principio en el momento de la instalación no hay nada todavía presente, lo más sencillo es eliminar todos los archivos InnoDB (por defecto, `ibdata1` para los datos, `ib_logfile0` e `ib_logfile1` para los logs) que se encuentran en la raíz del directorio de datos. La próxima vez que se ejecute el servidor, estos archivos se crearán de nuevo de forma automática.

- ☐ En las versiones anteriores, en caso de problema con InnoDB, el servidor se negaba a arrancar. En las últimas versiones, el servidor arranca, pero InnoDB

se desactiva: un error puede pasar desapercibido hasta que un usuario intenta utilizar InnoDB.

También puede ocurrir que, al ejecutar un servidor que ya funciona correctamente, aparezca una secuencia de errores similar a esta:

```
InnoDB: Unable to lock ./ibdata1, error: 11
InnoDB: Check that you do not already have another mysqld process
InnoDB: using the same InnoDB data or log files.
```

Estos mensajes indican que la instancia intenta usar el tablespace de otra instancia que se está ejecutando.

Esto se podría deber a que faltara la opción `--defaults-file` en la línea de comando que arranca el servidor, o a que el archivo `my.cnf` proporcionado fuera incorrecto.

b. Archivo `errmsg.sys` no encontrado

MySQL puede negarse a iniciar mostrando el error siguiente:

```
[ERROR] Can't find messagefile '...share/english/errmsg.sys'
```

Sin embargo, el archivo en cuestión se encuentra en la ubicación deseada...

Se trata de un problema conocido de algunas versiones y para resolverlo hay que añadir al archivo `my.cnf` la opción `language` o `lc-messages-dir` (según la versión de MySQL) con la ruta al archivo `errmsg.sys`.

6. Securitización de la instalación

La securización de un servidor MySQL consiste básicamente en asegurarse de que los

permisos de los archivos y directorios son correctos (en particular sobre los archivos de datos y los registros), de que las cuentas tienen una contraseña y no ofrecen derechos innecesarios y de que el servidor se ha iniciado con un usuario distinto al superusuario.

Sea cual fuere el modo de instalación, el script `mysql_secure_installation`, provisto con todas las distribuciones, puede ayudarnos a realizar algunas comprobaciones básicas.

Este script:

- Comprueba que el usuario `root` tiene una contraseña.
- Permite eliminar los usuarios anónimos.
- Elimina la cuenta `root` remota.
- Elimina la base `test`.
- Permite recargar los privilegios.

Este script es simple y limitado, pero permite identificar los errores más graves a nivel de seguridad.

- ☐ Las cuestiones relacionadas con la seguridad de MySQL se abordarán de manera más sistemática en el capítulo Seguridad y gestión de los usuarios.
- ☐ Este script ya no es útil con MySQL 5.7 porque los scripts de instalación predeterminados ya realizan todas las acciones. Solo es realmente útil usar `mysql_secure_installation` con MySQL 5.7 cuando se ha actualizado la versión de MySQL y queremos comprobar que los parámetros básicos de seguridad están operativos.

7. Instalación de varias instancias

a. Precauciones necesarias

Para cada nueva instalación en un equipo que ya cuente con una o varias instancias, necesitamos configurar una serie de parámetros para aislar por completo la nueva instancia.

Estos parámetros son los siguientes:

- `port` y `socket` para las conexiones de los clientes.
- Directorio de datos (`data-dir`).
- Directorio de registros.
- Tablespace para InnoDB (el tablespace se encuentra por defecto en el directorio de datos, la elección de un directorio de datos es por lo general suficiente).
- `server-id` (este parámetro puede escaparse, ya que no se convierte en crucial hasta la replicación. Para más información, ver el capítulo Replicación).

☐ Se aconseja detener todas las demás instancias corriendo en el mismo equipo para evitar efectos secundarios extraños en caso de un error en las operaciones.

b. Instalación de versiones diferentes

Imaginemos que ya hemos instalado una instancia MySQL de forma estándar con el gestor de paquetes. Desea ahora instalar en paralelo la última versión estable disponible para fines de prueba.

El gestor de paquetes no permite instalar una segunda instancia, por lo que debemos recurrir a una instalación con ejecutables tal y como se ha descrito antes.

c. Uso del mismo ejecutable que otra instancia

Cree nuevos directorios para los datos, los registros y los archivos de configuración:

```
$ mkdir /var/lib/mysql2
$ chown -R mysql:mysql /var/lib/mysql2/

$ mkdir /var/log/mysql2
$ chown -R mysql:mysql /var/log/mysql2

$ cp /ruta/para/antiguo/my.cnf /ruta/para/nuevo/my.cnf
$ chown root.root /ruta/para/nuevo/my.cnf
```

Luego edite el archivo `my.cnf` de la instancia que desee utilizar para iniciar la instancia con los parámetros correctos:

```
port = 3307
socket = /var/run/mysqld/mysqld2.sock
pid-file = /var/run/mysqld/mysqld2.pid
datadir = /var/lib/mysql2
server-id = 11
```

Compruebe que las rutas hacia los distintos registros apuntan hacia los destinos específicos de su instancia.

Después, arranque el script de instalación, pasando como parámetro la ruta del archivo de configuración que hemos elegido para esta instancia:

```
# mysqld --defaults-file=ruta/al/nuevo/my.cnf --user=mysql
```

- ☐ ¡Preste atención a los archivos `my.cnf` de las instancias instaladas por un gestor de paquetes! Las distribuciones efectúan a menudo algunas modificaciones con respecto a la versión estándar descargable en www.mysql.com, lo que puede traducirse en configuraciones adicionales en el archivo `my.cnf`, que habrá que modificar para la segunda instancia.

Inicie la instancia usando uno de los métodos antes descritos, teniendo cuidado de ejecutar los scripts que corresponden a esta instancia.

d. Arranque y parada de las instancias con `mysqld_multi`

`mysqld_multi` es un script Perl que permite iniciar, detener o conocer el estado de varias instancias que residen en el mismo equipo. Basta con crear un archivo de configuración que contenga la información adecuada para poder utilizar este script, pero

hay muchas sutilezas que es necesario conocer.

En primer lugar, se debe crear un usuario MySQL para detener todas las instancias. En efecto, `mysqld_multi` recurre a `mysqladmin` para detener. Es necesario que `mysqld_multi` conozca un usuario que tenga el derecho `shutdown` para cada una de las instancias. La cuenta `root` no es necesariamente utilizable (la contraseña no es siempre idéntica en todas las instancias y aparecerá visible en el archivo de configuración de `mysqld_multi`).

Cree un usuario llamado, por ejemplo, `multi_admin` en cada instancia gestionada por `mysqld_multi`:

```
mysql> GRANT SHUTDOWN ON *.* TO 'multi_admin'@'localhost'  
IDENTIFIED BY 'mi_cont';
```

El archivo de configuración de `mysqld_multi` tiene un formato parecido al del archivo `my.cnf` clásico, ya que está dividido en secciones y cada sección propone una o varias opciones. Introduzca primero una sección `[mysqld_multi]` que abarque tanto la información que se pase a `mysqld_multi` como las opciones por defecto de todas las instancias.

Ejemplo de dicha sección

```
[mysqld_multi]  
mysqladmin = /usr/bin/mysqladmin  
user        = multi_admin  
password    = mi_cont
```

La primera línea de la sección incluye la ruta a `mysqladmin`. Esto no es necesario si `mysqladmin` está instalado en un lugar estándar (lo que será el caso si se ha utilizado el gestor de paquetes para instalar MySQL).

La segunda y la tercera líneas de la sección proporcionan el usuario que

mysqladmin llamará para detener la instancia. Observe que la contraseña se almacena sin ocultar.

Después de esta sección `[mysqld_multi]`, escriba tantas secciones `[mysqldN]` como instancias deba gestionar. Cada sección deberá llevar el conjunto de datos para designar los recursos específicos de cada instancia y precisar los ejecutables o scripts que deban utilizarse.

Ejemplo de secciones `[mysqldN]`

```
[mysqld1]
socket      = /var/run/mysqld/mysqld.sock
port        = 3306
pid-file    = /var/run/mysqld/mysqld.pid
datadir     = /var/lib/mysql

[mysqld2]
defaults-file = /home/javier/mysql5549/my.cnf
mysqld       = /home/javier/mysql5549/bin/mysqld_safe
ledir        = /home/javier/mysql5549/bin
socket       = /tmp/mysql5549.sock
port         = 3307
pid-file     = /var/run/mysqld/mysqld5549.pid
datadir      = /home/javier/mysql5549/data
language     = /home/javier/mysql5549/share/english
```

La primera sección presenta aquí la información que señala una instancia instalada por el gestor de paquetes. No es necesario indicar las rutas a `mysqld` o `mysqld_safe` porque son las rutas por defecto.

La segunda sección permite configurar la instancia de una versión diferente en la que todos los archivos se encuentran en `/home/javier/mysql5549`. Como primera opción figura la ruta al archivo `my.cnf`. **¡Es esencial que esta opción sea la primera de la sección!** De no ser así, la opción no se tomará en cuenta y se iniciará una instancia equivocada o la instancia adecuada con parámetros erróneos.

Luego, la opción `mysqld` permite proporcionar la ruta a `mysqld`. Pero si queremos iniciar la instancia con `mysqld_safe`, como es el caso aquí, indicaremos la ruta hacia `mysqld_safe` en esta opción y la ruta a `mysqld` en la opción `ledir` (`ledir` de *library execution directory*).

- Atención: fíjese en las opciones que precisan que se especifique un directorio y las que precisan que se especifique un nombre de archivo.

Las opciones se escriben en el archivo, ¿qué nombre darle y dónde guardarlo? No existen limitaciones, pero sí que se deben tomar precauciones. En efecto, `mysqld_multi` espera que el archivo de configuración se llame `my.cnf` y se encuentre en un directorio estándar (`/etc/`, `/etc/mysql/`, `./` por ejemplo). Si este no es el caso, hay que pasar la ruta del archivo a `mysqld_multi`:

```
$ mysqld_multi --defaults-file=/etc/my.cnf.multi report
```

`mysqld_multi` reconoce tres acciones:

- `start`: inicia una o varias instancias.
- `stop`: detiene una o varias instancias.
- `report` : muestra el estado (arrancado o detenido) de una o varias instancias.

Para cada una de las acciones, como opción, podemos especificar números para indicar a `mysqld_multi` sobre qué instancia(s) se aplica la acción. El número corresponde con el número de la instancia en el archivo de configuración de `mysqld_multi` (por ejemplo, la instancia 2 es aquella cuya configuración está descrita en la sección `[mysqld2]`). Si deseamos aplicar una acción a varias instancias, podemos listar los números separados por comas, dar un rango de valor o mezclar los dos tipos de notación.

Si no se especifica un número, la acción se aplica a todas las instancias.

Ejemplos:

El comando `$ mysqld_multi start 1-3,6` inicia las instancias 1, 2, 3 y 6.

El comando `$ mysqld_multi stop 1,3` detiene las instancias 1 y 3.

El comando `$ mysqld_multi report` proporciona el estado de todas las instancias gestionadas por `mysqld_multi`.

- ☐ Encontrará un ejemplo completo y comentado de archivo de configuración en el archivo complementario de este libro, disponible en el sitio de Ediciones ENI.
- ☐ Con MySQL 5.0, apareció el ejecutable MySQL Instance Manager. Este pretendía ser una alternativa para `mysqld_multi` en un primer momento y terminar reemplazándolo. A día de hoy, está totalmente en desuso.

8. MySQL Sandbox

a. Presentación de MySQL Sandbox

Giuseppe Maxia, antiguo empleado de MySQL AB, escribió MySQL Sandbox: una herramienta que permite crear instalaciones desechables muy útiles para montar rápidamente entornos de pruebas.

A partir de algunas líneas de comandos y distribuciones en `.tar.gz` de las versiones que nos interesen, podemos crear una o varias instancias de MySQL, MariaDB o Percona Server. MySQL Sandbox también proporciona un conjunto de scripts para iniciar/detener nuestras instancias, conectarse para realizar peticiones y eliminar propiamente el conjunto de archivos cuando la instancia ya no es útil.

- ☐ También es posible utilizar Docker o cualquier otra tecnología de contenedores para crear fácilmente un entorno de prueba.

b. Instalación

Comience por descargar el archivo de la última versión de MySQL Sandbox a partir del sitio del proyecto (<https://github.com/datacharmer/mysql-sandbox>). Luego, como `root` en el directorio raíz del archivo descomprimido, basta con unas cuantas líneas de comando para realizar la instalación.

```
# perl Makefile.PL
# make
# make install
```

c. Creación de una instancia

Imaginemos que deseamos probar la versión 5.7, la más reciente.

Comenzamos por recuperar la distribución ejecutable, por ejemplo, en el directorio `/opt/`.

Luego, ejecutamos el comando `make_sandbox` con la cuenta de usuario habitual, que no sea `root`, indicando la ruta de los ejecutables MySQL recuperados:

```
$ make_sandbox /opt/mysql-5.7.12-linux-glibc2.5-x86_64/.tar.gz
```

- ☐ Si ejecutamos el comando `make_sandbox` como `root`, deberemos recargar la variable `SANDBOX_AS_ROOT`.

Después de algunos mensajes informativos, el script pide confirmación de una serie de parámetros que se aplicarán a la instalación. En principio, los parámetros propuestos deberían servirle.

La instalación termina entonces:

```
...
```

```
no_run                = 0
no_show               = 0
do you agree? ([Y],n) Y
loading grants
.. sandbox server started
Your sandbox server was installed in $HOME/sandboxes/msb_5_7_12
```

Para utilizar nuestro nuevo servidor MySQL, vaya al directorio indicado al final de la instalación (aquí `$HOME/sandboxes/msb_5_7_12`). Este directorio contiene los scripts para iniciar, detener, salvaguardar o conectarse a la instancia sin tener que introducir una línea de comando compleja. Es también en este lugar donde se almacenan los datos y el archivo de configuración de la instancia (`my.sandbox.cnf`).

Por ejemplo, para conectarse:

```
$ ./use
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.12 MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current
input statement.

mysql [localhost] {msandbox} ((none)) >
```

- ☐ Observe que las instancias instaladas con MySQL Sandbox modifican automáticamente el prompt, lo que permite localizarlas fácilmente.

Para detener el servidor, hay que utilizar el comando siguiente:

```
$ ./stop
```

y para reiniciar, este.

```
$ ./start
```

d. Creación de dos instancias independientes

Imaginemos ahora que deseamos crear una instancia en versión 5.5 y una instancia en versión 5.6.

Al igual que para la instalación de una única instancia, comenzamos por recuperar las dos distribuciones ejecutables que nos interesan.

Luego, basta con una sola línea de comando para iniciar las dos instalaciones:

```
$ make_multiple_custom_sandbox /opt/mysql-5.5.25a-linux2.6-i686.tar.gz  
/opt/mysql-5.6.6-m9-linux2.6-i686.tar.gz
```

Después de algunos mensajes, se nos confirma que la instalación se ha realizado correctamente.

```
group directory installed in $HOME/sandboxes/multi_cmsb_5_5_25-5_6_6
```

Y si nos dirigimos al directorio indicado, encontraremos una estructura de archivos similar a la que se reproduce a continuación:

```
$ ls -1F  
clear_all*  
n1*  
n2*  
node1/  
node2/
```

```
restart_all*
send_kill_all*
start_all*
stop_all*
use_all*
```

Los dos directorios `node1` y `node2` corresponden a las dos instancias instaladas. Cada directorio contiene todos los scripts de gestión de la instancia, los datos y el archivo `my.cnf`, como hemos visto en el apartado anterior.

La diferencia con respecto a la instalación de una sola instancia es que tenemos aquí más scripts para gestionar todas las instancias instaladas en una sola operación. Así, el script `start_all` permite arrancar las dos instancias, ahorrando de esta manera las llamadas a `node1/start` y `node2/start`.

Del mismo modo, es posible conectarse a una u otra de las instancias a través de los scripts `n1` y `n2`, que son equivalentes a `node1/use` y `node2/use`.

e. Otras posibilidades

MySQL Sandbox ofrece muchas opciones para crear la configuración que nos interesa. Podemos así crear con facilidad las configuraciones siguientes:

- La replicación clásica (1 maestro, N esclavos).
- La replicación circular con N nodos.
- La replicación piramidal (1 maestro, N esclavos, cada esclavo siendo él mismo maestro de X esclavos).
- La construcción de una instancia a partir de las fuentes.

Observe que el script es lo suficientemente robusto para detectar posibles puertos o posibles sockets ya en uso por otros servidores MySQL, instalados o no con MySQL Sandbox.

MySQL Sandbox contiene también `sbtool`, una herramienta que simplifica algunas operaciones en las instancias ya instaladas, como la eliminación o el desplazamiento.

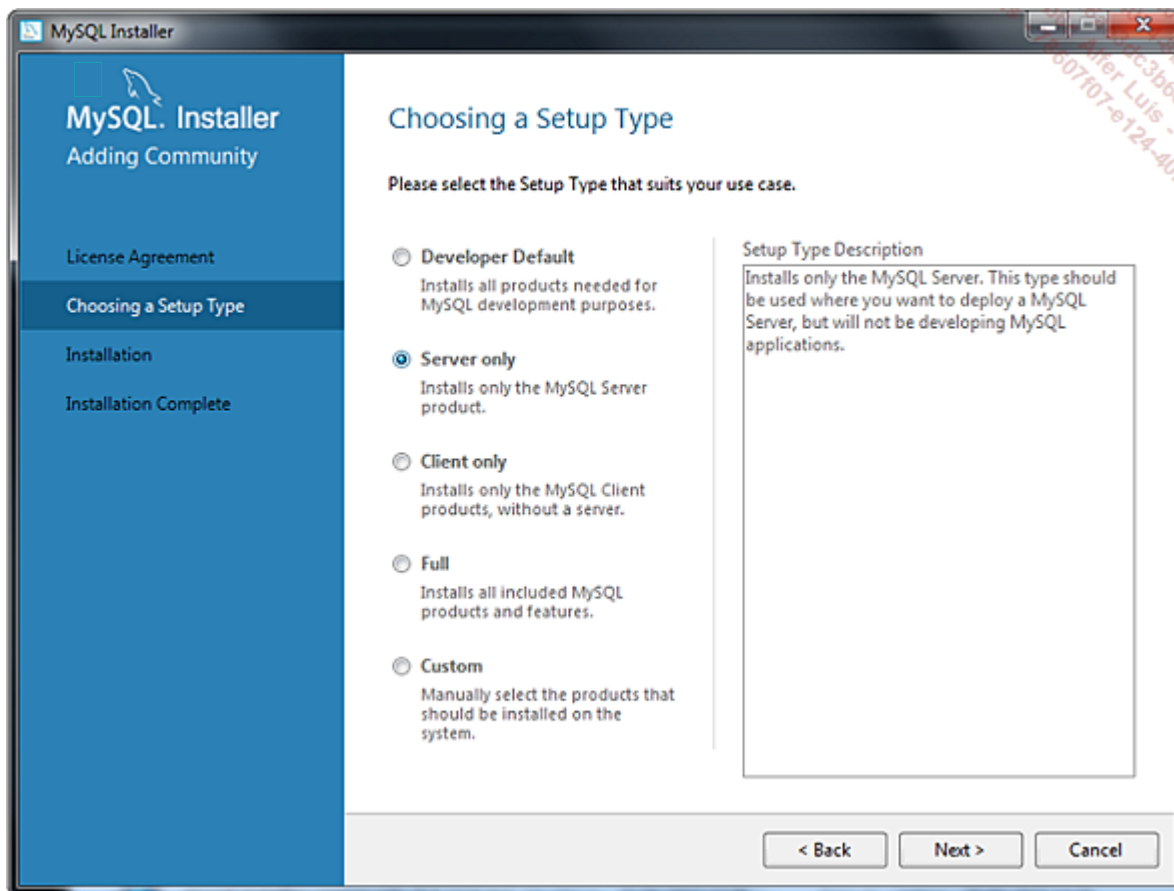
La documentación del proyecto se encuentra en la dirección <http://www.mysqlsandbox.net> y proporciona todos los detalles sobre las muchas posibilidades de uso de MySQL

Instalación en Windows

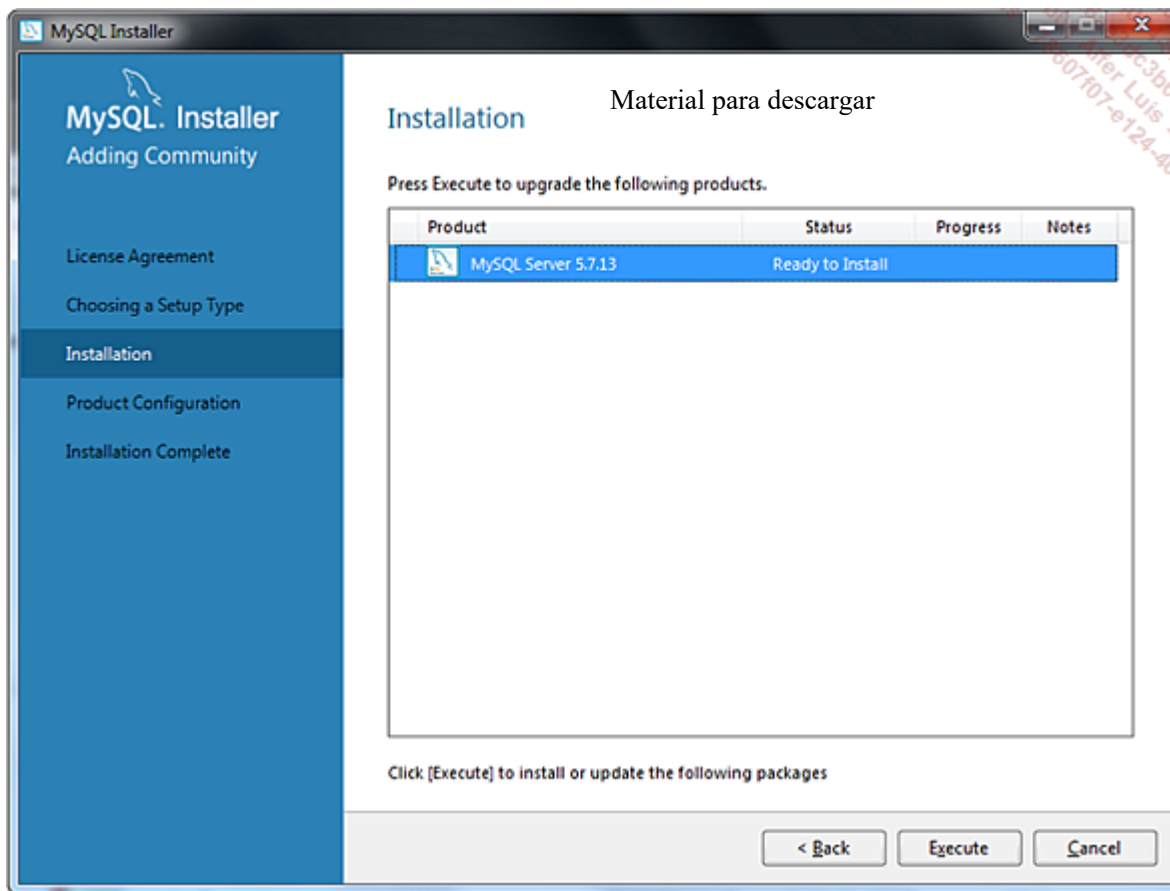
1. Utilización del instalador

A partir de la versión 5.5, el instalador de Windows es mucho más completo y permite, entre otras cosas, instalar productos complementarios o actualizaciones e incluso eliminar antiguas versiones.

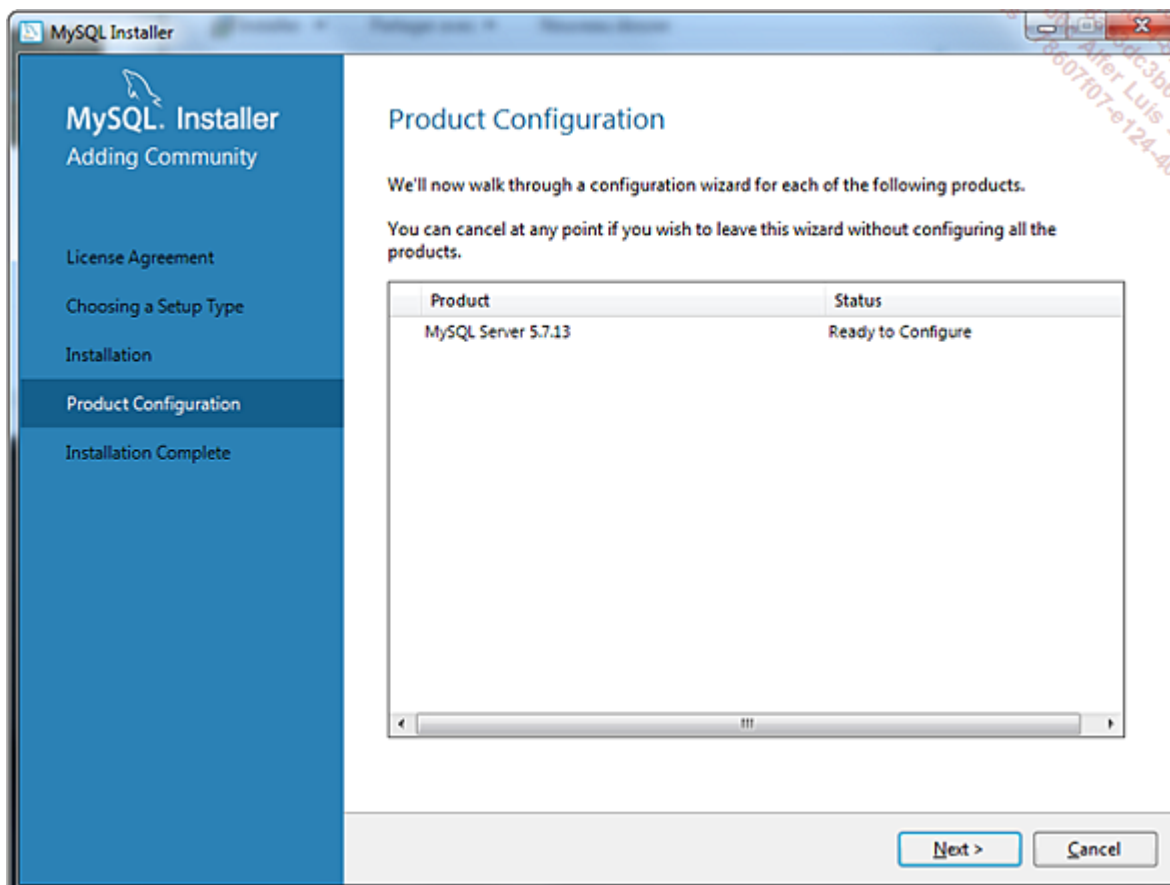
- El instalador nos preguntará en la primera pantalla el tipo de instalación que deseamos realizar.



- A continuación, seleccionamos la versión de MySQL que deseamos instalar:

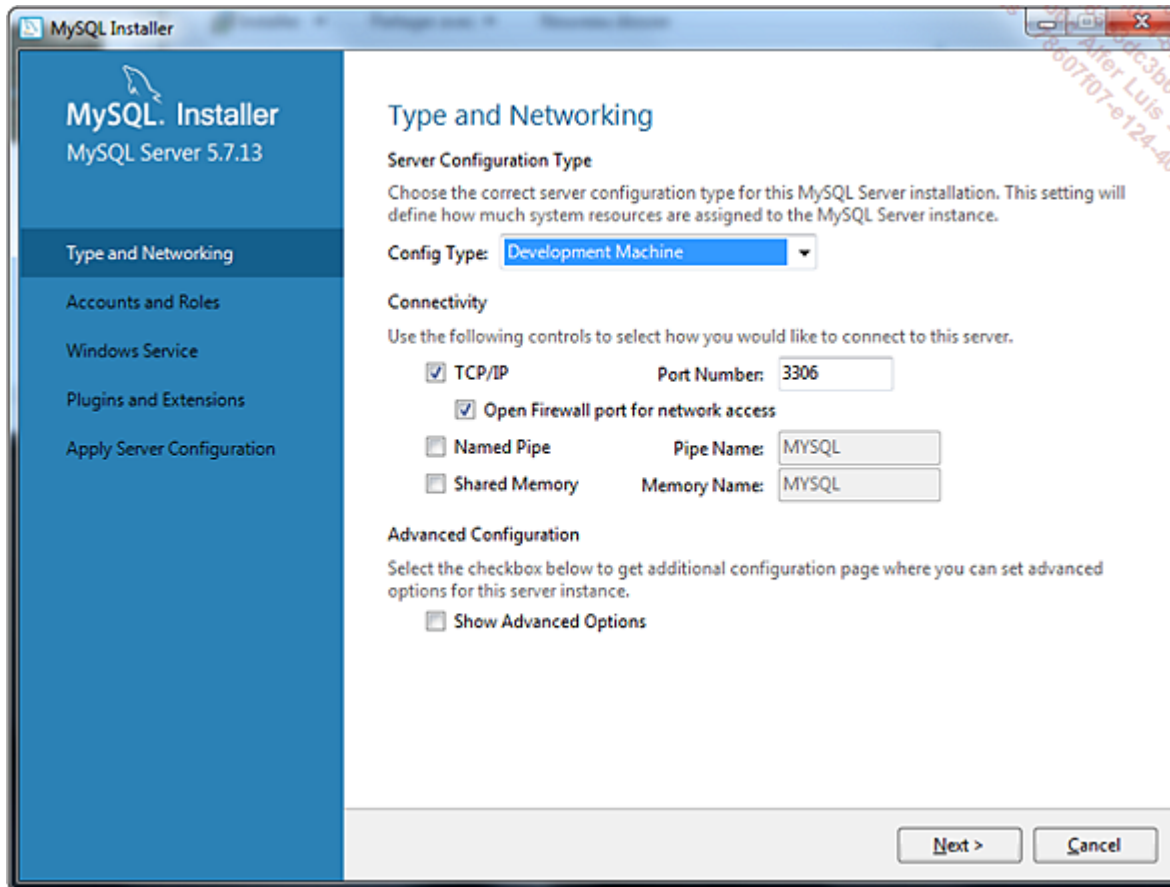


- Luego, hacemos clic en **Execute**. La instalación se lleva a cabo y, una vez concluida, aparecerá un mensaje indicando que la operación ha terminado:



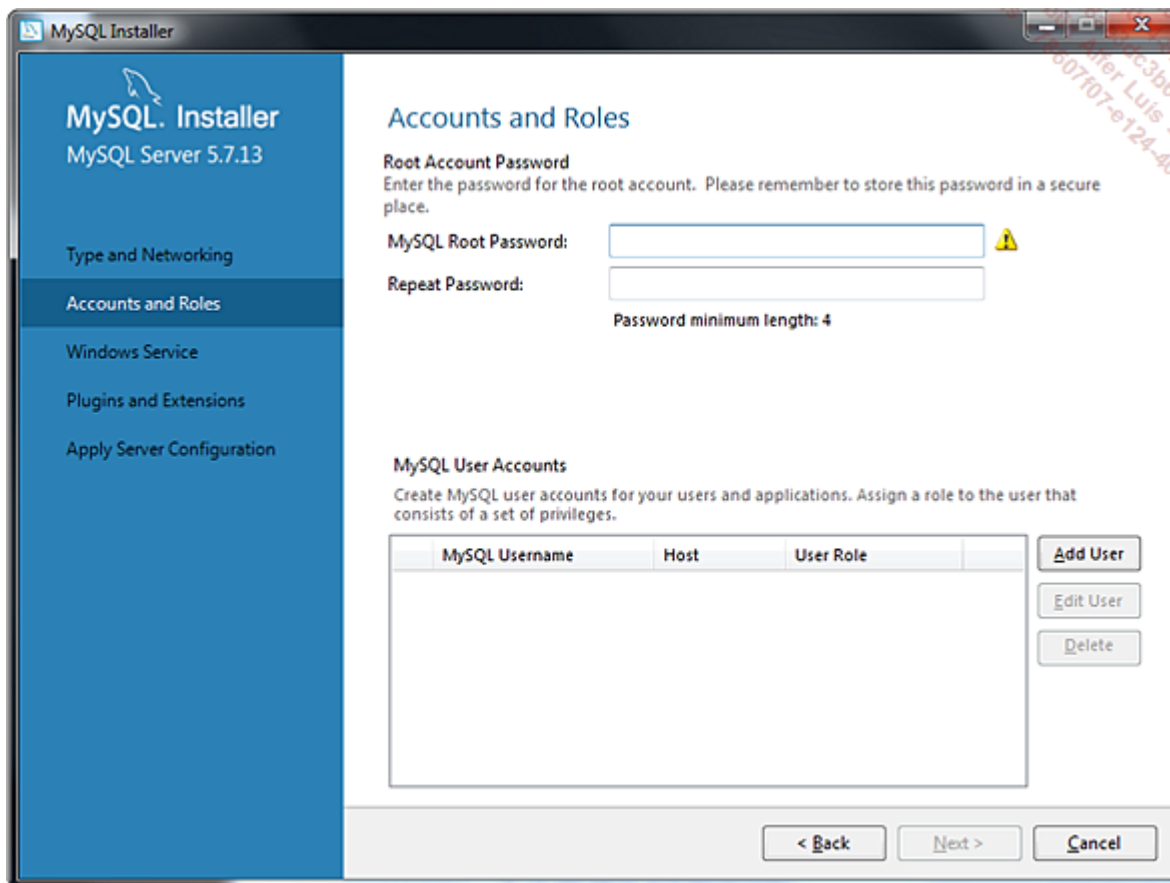
- Luego podemos escoger, si lo deseamos, algunos parámetros de configuración (si no

estamos seguros de nuestra elección, más adelante podremos cambiar cualquier parámetro en el archivo de configuración `my.ini`):

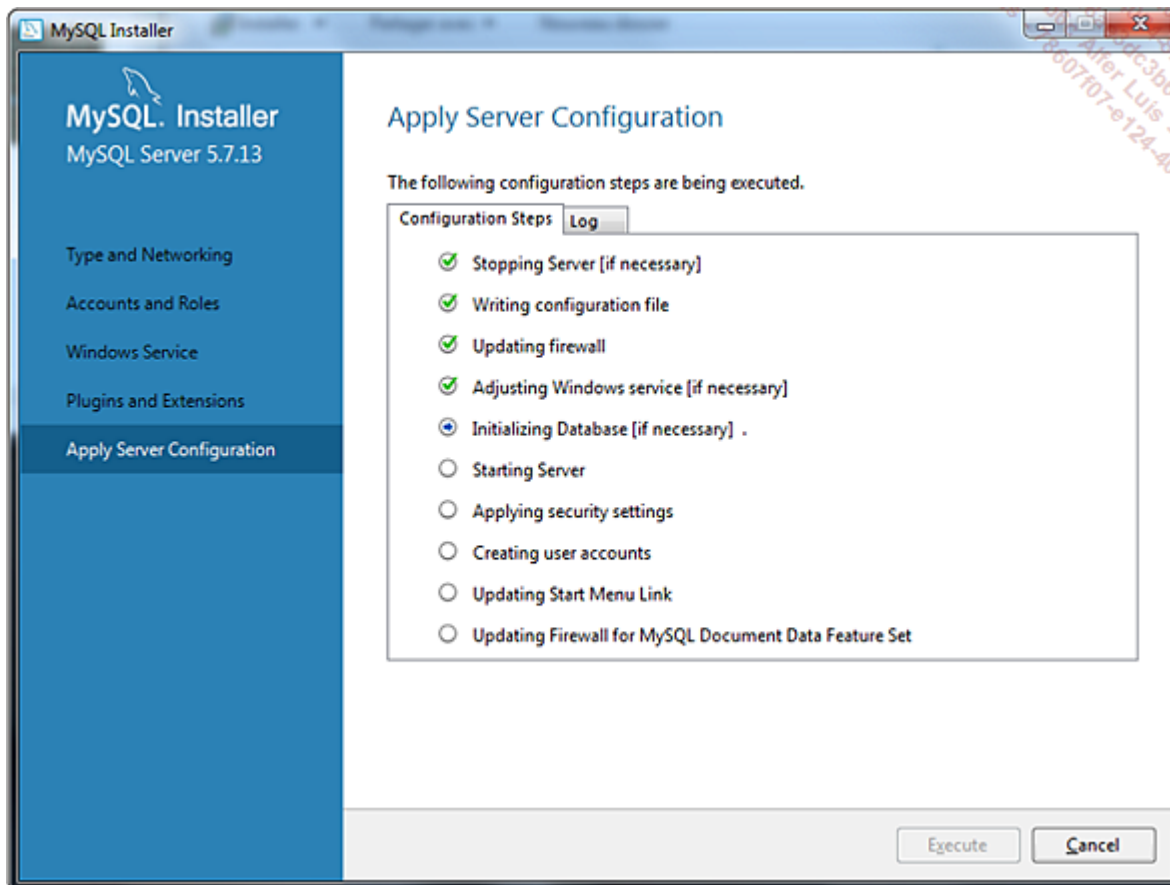


- El siguiente paso consiste en asignar una contraseña al usuario `root` si lo deseamos (esto es, naturalmente, muy recomendable), y podemos crear otros usuarios:

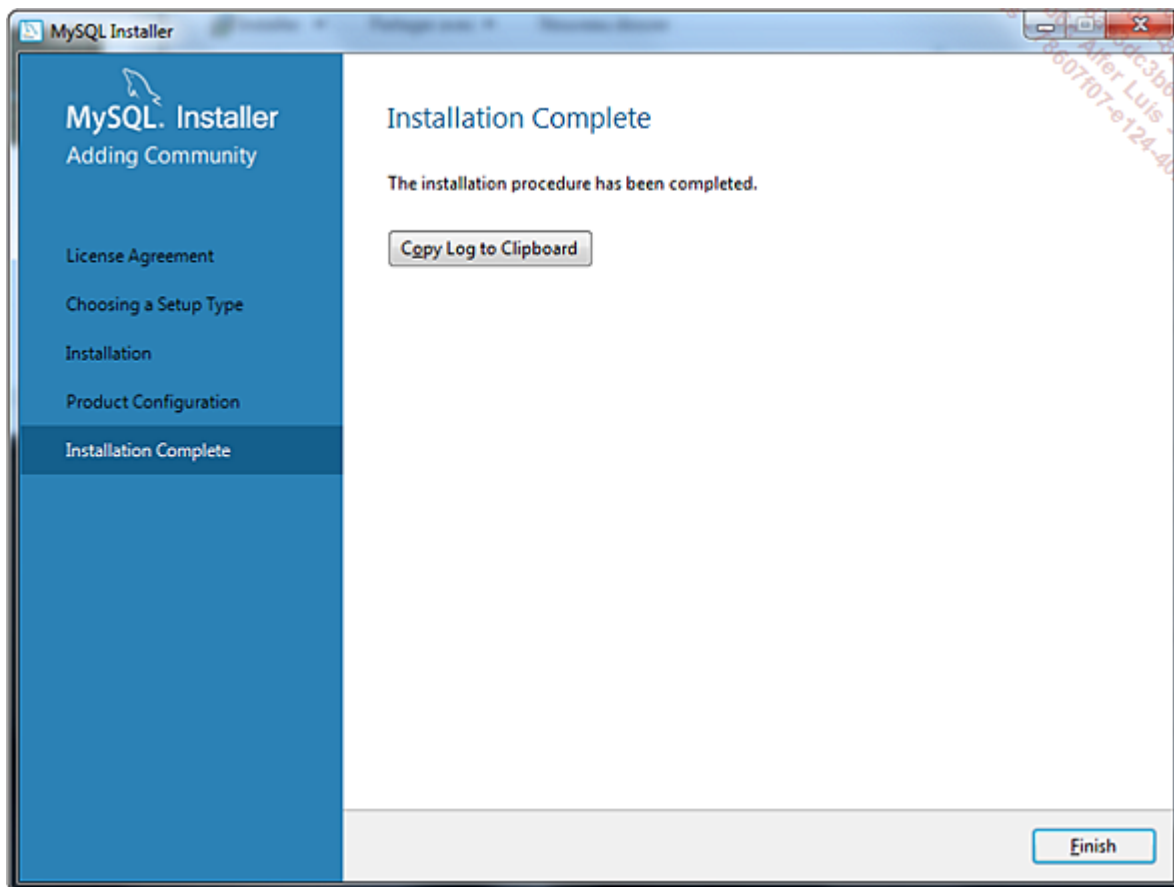




La configuración elegida se aplicará a continuación:



Por último, se mostrará un mensaje cuando la instalación haya terminado:



2. Instalación con archivos ejecutables

Los archivos ejecutables se pueden descargar como archivo `.zip` a partir de la categoría Windows y Windows x64 de la página de descargas del sitio www.mysql.com.

Después de haber descargado y descomprimido el archivo `.zip` en el directorio que hayamos elegido, no queda más que crear un archivo de configuración, tradicionalmente conocido en Windows como `my.ini` (en lugar de `my.cnf`). Lo más simple es copiar y pegar uno de los archivos de ejemplo presentes en la raíz del directorio descomprimido y modificar algunos parámetros.

Así, debemos pensar en:

- Cambiar el número del puerto en la sección `[client]` y la sección `[mysqld]` si otro servidor ya está a la escucha en el puerto 3306.
- Introducir al parámetro `basedir` en la sección `[mysqld]` con la ruta del directorio en el que hemos descomprimido el archivo.
- Cambiar el parámetro `server-id` en la sección `[mysqld]` si otro servidor ya posee este `server-id`.

He aquí cómo pueden aparecer estos parámetros:

```
[client]
port      = 3307
[mysqld]
port      = 3307
basedir   = C:/mysql5141
server-id = 11
```

El servidor se puede arrancar usando uno de los métodos que se presentan en el apartado siguiente.

3. Arranque del servidor

a. Servicio

Si el servidor se instala como un servicio, la instancia se puede iniciar usando el Panel de administración de servicios o el comando `net start XXX` (en el que `xxx` es el nombre del servicio).

Si este no es el caso, hay que empezar por instalar MySQL como servicio utilizando la opción `--install` de `mysqld`.

```
C:\>ruta\hacia\mysqld --install MySQL57
--defaults- file=ruta\hacia\my.ini
```

Se puede escoger cualquier nombre para el servicio a condición de que no se haya utilizado ya.

- ☐ Para eliminar el servicio asociado a una instancia MySQL, use la opción `--remove`:

```
C:\>ruta\hacia\mysqld --remove MySQL57
```

b. Invocación directa de mysqld

También podemos llamar a `mysqld` para iniciar el servidor:

```
C:\>ruta\hacia\mysqld --defaults-file=ruta\hacia\my.ini
```

Tenga en cuenta que existe la opción `--console`, que hace que los mensajes de error se muestren en la consola en vez de en un registro. Esta opción es más apreciable en caso de errores.

4. Parada del servidor

a. Servicio

Si el servidor está instalado como un servicio, la instancia puede detenerse usando el Panel de administración de servicios o el comando `net stop XXX` (en el que xxx es el nombre del servicio).

b. mysqladmin

Utilice el método que utilice para arrancar el servidor, el comando `shutdown` de `mysqladmin` le permitirá detener la instancia:

```
C:\>mysqladmin -uroot -p shutdown
```

El usuario pasado a `mysqladmin` debe tener el privilegio SHUTDOWN.

- ☐ Como se dijo en el apartado sobre la instalación en UNIX, `mysqladmin stop` no detiene el servidor, sino solo los procesos de replicación si el servidor es un esclavo.
- ☐ Como se explicó en el apartado sobre la instalación en UNIX, se puede o no dejar un espacio entre la opción `-U` y el nombre de usuario (`-uroot` o `-uroot`), pero no se puede dejar un espacio tras la opción `-p` si se especifica la contraseña en la línea de comandos (`-pxxx`).

c. Administrador de tareas

Cuando ninguno de los métodos anteriores funciona, a veces es necesario forzar la parada del servidor matando el proceso correspondiente a partir del Administrador de tareas.

Este método debe evitarse en la medida de lo posible porque una parada repentina del servidor puede provocar pérdidas o corrupción de datos.

5. Resolución de problemas de instalación

a. Permisos insuficientes

Para instalar MySQL como servicio, debemos tener una cuenta con derechos de administrador. De no ser así, Windows mostrará un mensaje de error y el servicio no podrá ser instalado.

b. Conflicto con una instalación existente

Como sucede con las instalaciones de tipo UNIX, existe una serie de recursos que no pueden ser compartidos entre varias instancias (puerto, directorio de datos, directorio de diarios, tablespace InnoDB, server-id).

En caso de conflicto, es necesario comprobar primero que se han introducido correctamente los parámetros en el archivo `my.ini`. Después, se debe comprobar que se llama a dicho archivo asegurándose de que el servidor MySQL se inició con la opción `--defaults-file`.

- ☐ Observe que, cuando se pasan varias opciones a `mysqld` para arrancar el servidor, `--defaults-file` siempre debe ser la primera opción.

c. Problemas en las rutas

Dependiendo de la versión de Windows y la versión de MySQL, puede haber problemas con el separador `/` o los nombres de directorios con espacios en la escritura de las rutas hacia los archivos o directorios. En este caso, es mejor usar siempre el separador `\` y escribir cada ruta entre `"` (comillas dobles).

6. Securización de la instalación

Los principios de seguridad son exactamente los mismos que se han descrito antes en el apartado sobre la seguridad de una instalación en UNIX y derivados.

El script `mysql_secure_installation` no existe en Windows, pero es posible (e incluso aconsejable) aplicar manualmente los mismos pasos que, recordemos, son:

- Comprobar que el usuario `root` tiene una contraseña.
- Eliminar los usuarios anónimos.
- Eliminar la cuenta `root` remota.
- Eliminar la base de datos `test`.

- ☐ Las cuestiones relacionadas con la seguridad de MySQL se abordarán con más detalle en el apartado Seguridad y gestión de los usuarios.

7. Instalación de varias instancias

a. Precauciones necesarias

Como se indicó en el apartado sobre los sistemas UNIX, instalar varias instancias en un mismo equipo es posible siempre que tengamos cuidado de aislar los recursos críticos, como el número de puerto, el directorio de datos, los registros, el tablespace `InnoDB` y el

`server-id`.

b. Versiones diferentes

El instalador no permitirá aislar completamente dos instancias. Tendremos que optar forzosamente por una instalación a partir de archivos ejecutables para la segunda instancia.

c. Uso del mismo ejecutable

También es posible utilizar el mismo ejecutable en varias instancias. En este caso, necesitaremos:

- Crear un nuevo directorio de datos.
- Copiar y pegar la base de datos de sistema `mysql` (es decir, el directorio `mysql/`) del antiguo directorio de datos en el nuevo.
- Crear un nuevo archivo `my.ini` con los parámetros `port`, `basedir`, `datadir` y `server-id` modificados.
- Ejecutar `mysqld` con la opción `--defaults-file` apuntando al nuevo archivo `my.ini`.

☐ Los registros y los archivos del tablespace InnoDB se crean de forma automática al ejecutar el servidor por primera vez.

Si lo desea, puede instalar esta nueva instancia como un servicio, siempre con el comando `mysqld --install nombre_servicio`.

Actualización de MySQL

1. Precauciones necesarias antes de la actualización

a. Saltos de versión

Si deseamos cambiar de versión menor, no es necesario pasar por todas las versiones menores intermedias. Por ejemplo, podemos pasar directamente sin grandes riesgos de una versión 5.6.17 a una versión 5.6.30. Se recomienda, sin embargo, conocer las diferencias entre las dos versiones (ver más arriba).

Al contrario, cuando se trata de versiones principales, se aconseja pasar por todas las versiones principales intermedias. En algunos casos, es posible saltar una o varias versiones principales, para, por ejemplo, actualizar de 5.1 a 5.6.

Consulte, por ejemplo, el siguiente artículo en el que se explica cómo pasar directamente de MySQL 5.0 a MySQL 5.7: <http://mysqlserverteam.com/upgrading-directly-from-mysql-5-0-to-5-7-using-an-in-place-upgrade/>

Observe que el procedimiento descrito en dicho artículo no permite realizar una actualización sin interrupción del servicio, lo que es un prerequisite en la mayoría de los casos.

b. Cambios introducidos por una versión

Se aconseja leer siempre los cambios introducidos por la versión a la que se desea migrar (así como los cambios de todas las versiones intermedias si se quiere saltar las versiones). En efecto, aunque parece lógico que una versión más reciente solo contenga mejoras, ocurre de vez en cuando que la corrección de un problema supone otro problema, o conlleva que se elimine una funcionalidad.

- ☐ A modo de ejemplo, el comando SQL `RENAME DATABASE` fue añadido en la versión 5.1.7 y eliminado en la versión 5.1.23.

c. Copia de seguridad de los datos

Una actualización de versión no es una operación anodina; por lo tanto, haga siempre una copia de seguridad de todos los datos antes de comenzar con la actualización.

- ☐ Las diferentes técnicas de respaldo (físicas y lógicas) se describen en el capítulo Respaldo y restauración.

2. Proceso de actualización

a. Posibles estrategias

Hay varias formas de actualizar una instancia. Todas ellas son más o menos arriesgadas y más o menos rápidas de llevar a cabo.

La más simple es realizar una copia de seguridad lógica (`mysqldump` o equivalente), eliminar todos los archivos de datos, actualizar los ejecutables y luego cargar la copia de seguridad. Este método le permite evitar cualquier problema relacionado con los cambios en el formato de almacenamiento de los datos. Sin embargo, para las bases de datos de varios cientos de GB, el tiempo de recarga de la base de datos es prohibitivo, lo que complica mucho la operación.

Otra manera de proceder es no tocar los archivos de datos y actualizar simplemente los ejecutables y reiniciar el servidor. La operación es, por supuesto, mucho más rápida, pero mucho más arriesgada: las tablas pueden quedar inaccesibles si utilizan un formato que no es válido. El servidor puede negarse a arrancar en algunos casos.

Sea cual sea el método elegido, la replicación puede permitirnos hacer una actualización con una interrupción de servicio mínima:

- ☐ Actualice los esclavos de replicación. Esta actualización no provoca la interrupción del servicio, aunque debe durar varias horas. Asegúrese solo de prohibir el tráfico a los esclavos durante la actualización.

- Cuando todos los esclavos utilicen la nueva versión, haga la promoción de uno de los esclavos. El antiguo maestro se convierte en un esclavo y puede actualizarse sin interrupción del servicio.
- Es posible hacer la promoción del antiguo maestro para recuperar la configuración de replicación original.

Esta estrategia limita las interrupciones de servicio durante la promoción de un esclavo, que puede durar unas decenas de segundos. Esta estrategia funciona bien, porque siempre es posible utilizar la replicación de una versión N hacia una versión N+1 (lo contrario no es siempre posible).

En cualquier caso, es necesario ejecutar el script `mysql_upgrade` para actualizar las tablas del sistema (ver más abajo).

b. Actualización de los ejecutables

Después de haber detenido el servidor, reemplace los antiguos ejecutables por los nuevos. Puede ser interesante crear enlaces simbólicos para poder volver a los antiguos ejecutables con facilidad si la actualización sale mal.

A continuación, se puede reiniciar el servidor.

c. Verificación de las tablas

Una vez que el servidor esté de nuevo en marcha, ejecute `mysql_upgrade`. Este ejecutable analizará todas las tablas de todas las bases de datos y tratará de repararlas en caso de problema. Después, aplicará los cambios a la base de sistema `mysql` si es necesario (añadir o modificar privilegios, por ejemplo).

- Con algunas distribuciones de Linux, `mysql_upgrade` se inicia de forma automática en los scripts de posinstalación.
- Observe la opción `--upgrade-system-tables`, que se limita a actualizar las tablas del sistema sin afectar a los datos. Esta opción puede ser muy práctica para probar una actualización de forma rápida.

3. Comprobaciones tras la actualización

Entre dos versiones principales, tienen lugar muchos cambios y, aunque en principio los cambios son concebidos para mejorar el rendimiento y la calidad del servidor, pueden ocurrir problemas. Para evitar sorpresas desagradables, es útil hacer las siguientes comprobaciones:

- Todas las peticiones se pueden ejecutar sin error con la nueva versión. Debido a la utilización de nuevas palabras clave o a cambios en la sintaxis, peticiones que eran válidas con la versión N podrían no serlo con la versión N+1.
- Todas las peticiones devuelven el mismo resultado. Los cambios de plan de ejecución con la nueva versión pueden provocar en algunos casos cambios en el resultado de una petición (clasificación implícita basada en la utilización de un índice y no en una cláusula `ORDER BY`, por ejemplo).
- Todas las peticiones tienen un tiempo de ejecución inferior o igual al de la nueva versión. Una vez más, pueden ocurrir cambios del plan de ejecución y modificar radicalmente el tiempo de respuesta de algunas peticiones.

Una manera de comprobar todos estos puntos es utilizar el registro de peticiones lentas con el parámetro `long_query_time` en 0, de manera que se capturen durante cierto tiempo (una hora, por ejemplo) todas las peticiones que se ejecutan en un servidor de producción con la versión antigua. Luego, la herramienta `pt-upgrade` del Percona Toolkit (ver más adelante en este capítulo) puede utilizar este registro para volver a ejecutar todas las peticiones en una instancia con la versión antigua y una instancia con la nueva versión para identificar los posibles problemas que podrían darse con la nueva versión.

Instalación de las herramientas utilizadas en el libro

Algunos de los ejemplos de este libro están basados en dos bases de datos de ejemplo que pueden descargarse desde el sitio www.mysql.com. Le será muy útil instalar estas bases de datos para poder reproducir directamente las operaciones que se describen en una instancia instalada en su equipo habitual.

1. Instalación de la base world

Encontrará un archivo comprimido de la base `world` en la siguiente dirección: <http://dev.mysql.com/doc/index-other.html>. Después de haber descargado y descomprimido el archivo, obtendrá un archivo `world.sql` que le permitirá restablecer los datos después de haber creado la base `world`:

```
mysql> CREATE DATABASE world;

shell> mysql -uroot -p world < world.sql
```

Si todo funciona, encontrará tres tablas en la base de datos `world`:

```
mysql> USE world;

mysql> SHOW TABLES;

+-----+
```

```
| Tables_in_world |  
+-----+  
| City            |  
| Country         |  
| CountryLanguage |  
+-----+
```

2. Instalación de la base sakila

El procedimiento es similar al descrito para la base `world`: después de haber descargado y descomprimido el archivo disponible en la misma página (<http://dev.mysql.com/doc/index-other.html>), obtendrá un directorio `sakila-db` que contiene tres archivos:

- `sakila-data.sql` contiene los datos de la base.
- `sakila-schema.sql` contiene el esquema de la base.
- `sakila.mwb` contiene una representación gráfica del modelo físico de la base, legible con el software MySQL Workbench (<http://www.mysql.com/products/workbench>).

Para reconstruir la base, bastará con inyectar los archivos `sakila-schema.sql` y luego `sakila-data.sql`:

```
shell> mysql -uroot -p < sakila-schema.sql  
  
shell> mysql -uroot -p < sakila-data.sql
```

Instalación de Percona Toolkit

El Percona Toolkit es un conjunto de scripts que facilita el trabajo de los administradores, haciendo que tareas por lo general engorrosas resulten simples. Estos scripts permiten, por ejemplo:

- Identificar de forma sencilla índices redundantes.
- Analizar un registro de peticiones con el fin de extraer las peticiones de mayor peso.
- Verificar la coherencia de los datos entre un servidor maestro y un servidor esclavo.
- Ejecutar comandos `ALTER TABLE` sin bloqueos.

Tendremos la oportunidad de estudiar el uso de varias de estas herramientas a lo largo de este libro.

Existen varios métodos para instalar Percona Toolkit:

- Usar el Gestor de paquetes: se trata del método más sencillo, pero hay muchas posibilidades de encontrarse con una versión antigua. El comando que se debe introducir en Ubuntu es:

```
shell> sudo aptitude install percona-toolkit
```

- Descargar una distribución en `tar.gz`:

```
shell> wget http://www.percona.com/get/percona-toolkit.tar.gz
```

- Descargar de forma individual las herramientas que nos interesen. Al ser cada

herramienta autónoma, podemos instalar solo algunas de ellas,

```
shell> wget http://www.percona.com/get/HERRAMIENTA
```

- donde HERRAMIENTA es el nombre de la herramienta que se desea instalar (por ejemplo, `pt-query-digest`).
- Podemos consultar la documentación de las herramientas en la página: <http://www.percona.com/software/percona-toolkit>

☐ Tenga en cuenta que las herramientas se han desarrollado inicialmente para operar bajo UNIX/Linux y que no son todas compatibles con Windows.

☐ Aunque la documentación es muy completa, debe saber que algunas herramientas son difíciles de utilizar. Debe saber también que algunas herramientas se limitan a leer los datos y no hay riesgos de que provoquen daños si se utilizan mal, pero otras sí que realizan cambios. Tenga especial cuidado y estudie bien el comportamiento de los diferentes scripts en un entorno de prueba.

Instalación de MySQL Utilities

Los MySQL Utilities son un conjunto de scripts suministrados por Oracle para facilitar el trabajo de los administradores de bases de datos. Esta breve descripción nos recordará sin duda a Percona Toolkit, al que se ha hecho referencia en la sección anterior. En efecto, las dos herramientas son muy similares en principio, pero existen diferencias notables:

- El Percona Toolkit está escrito en PERL, mientras que los MySQL Utilities están escritos en Python. Este detalle puede tener su importancia si necesitamos desarrollar extensiones.
- El Percona Toolkit se creó a mediados de los años 2000 y el código ha podido madurar y ser probado por muchos usuarios. Los MySQL Utilities datan del año 2010 y la base de usuarios es mucho más reducida.
- Los MySQL Utilities siguen la evolución de las funciones de MySQL, mientras que Percona Toolkit no sigue a menudo las funcionalidades disponibles a partir de MySQL 5.6 (por ejemplo, los GTID; ver el capítulo Replicación).
- Las herramientas de MySQL Utilities no siempre se pueden utilizar en producción porque a veces tienen requisitos previos inalcanzables, mientras que todas las herramientas de Percona Toolkit están diseñadas para utilizarse con el mínimo de riesgos en producción.

Existen varias formas de instalar MySQL Utilities, dependiendo del sistema operativo:

- Para Microsoft Windows, se puede usar el instalador de MySQL y seleccionar solo MySQL Utilities, o descargar el instalador específico de MySQL Utilities (<http://dev.mysql.com/downloads/utilities>).
- Para Linux, lo más sencillo es utilizar el gestor de paquetes, usar los repositorios de Oracle o descargar un paquete `.deb` o `.rpm` a partir de la página <http://dev.mysql.com/downloads/utilities>.

Introducción

Configurar de forma correcta MySQL es esencial para garantizar tanto el rendimiento como la estabilidad del servidor. Hay varios cientos de opciones disponibles, lo que dificulta la elección de los parámetros. Por otra parte, en muchos casos, no existe un solo valor correcto que podamos aplicar con los ojos cerrados: con frecuencia, varios valores son válidos para una opción y la mejor configuración dependerá del uso del servidor.

Sin embargo, obtener una configuración correcta de nuestro servidor MySQL no es tan complicado en realidad. En primer lugar, tenga en cuenta que no hay ninguna configuración ideal: si sus peticiones se realizan sin lentitud especial y el servidor no muestra nunca signos de sobrecarga, esto significa que los parámetros esenciales están correctamente ajustados. Por otra parte, debe saber que no basta con modificar la configuración para solucionar todos los problemas de rendimiento: otros factores, como los recursos de hardware, el esquema de las tablas y los índices también tienen un impacto importante en el rendimiento. Por último, tenga en cuenta que, salvo una veintena de parámetros que es fundamental conocer y saber ajustar, la mayoría de las opciones solo son útiles en casos muy concretos que puede que no se le presenten nunca.

He aquí algunos consejos antes de entrar en el meollo del tema:

- Cuando modifique los parámetros, intente siempre hacerlo uno por uno. Si cambia a la vez X e Y y se aprecia +15 % de rendimiento, es muy difícil saber si +10 % es por X y +5 % por Y o +25 %, gracias a X y -10 %, gracias a Y.
- Piense qué es más importante para usted, ¿el rendimiento o la seguridad de los datos? Se pueden tener las dos cosas, por supuesto; pero solo hasta cierto punto, ya que el aumento del rendimiento es siempre en detrimento de la seguridad de los datos. ¿Es esto aceptable? La documentación oficial no podrá ayudarle, solo usted puede responder a la pregunta.
- Desconfíe de los artículos que encuentre en Internet de tipo «cómo ganar X % de

rendimiento cambiando una sola variable». Aunque los autores de tales artículos suelen actuar con buena fe, se basan en general en casos muy particulares que tienen cuellos de botella importantes. Si no estamos en la misma situación, en el mejor de los casos, los cambios que hagamos no tendrán ningún impacto; y en el peor, será perjudicial.

- Cuando mueva una instancia MySQL en un servidor con un hardware más potente, no caiga en el error de «tengo dos veces más memoria RAM, por lo tanto se multiplica por dos el tamaño» de todas las cachés. Algunas cachés solo funcionan de forma correcta si siguen siendo reducidas, por lo que podemos degradar las prestaciones aumentando el tamaño de todas las cachés.

¿Cómo configurar el servidor?

Las opciones pueden definirse en diferentes niveles:

- Durante la compilación de las fuentes ejecutando el script `configure`.
- En el archivo de configuración `my.cnf` (`my.ini` en Microsoft Windows).
- Como parámetros del programa `mysqld`.
- De forma dinámica, es decir, durante la ejecución del servidor (en caliente), con el comando `SET`.

Si una opción se redefine en varios niveles, es su valor en el nivel más bajo de esta lista el que se tendrá en cuenta. Por ejemplo, si en el archivo de configuración la opción `long-query-time` es 10 y pasamos 2 al ejecutable `mysqld`, el valor para el servidor será 2.

El método más práctico, más seguro y que se recomienda para configurar el servidor es usar el archivo de configuración. Sin embargo, la modificación dinámica de las opciones se impone a veces.

1. Configuración durante la compilación

Una de las formas de instalar el servidor MySQL consiste en compilar sus fuentes. En Linux, al ejecutar el script `configure`, podemos personalizar la instalación pasando las opciones adecuadas. Esta solución se utiliza con frecuencia en las grandes empresas para que hacer que coincida la instalación del servidor con la norma en vigor en la empresa o bien por razones de rendimiento. Las directivas especificadas constituirán la configuración por defecto, salvo si se cambian los valores en uno de los otros niveles de configuración.

La compilación de MySQL sobre las diferentes plataformas se detalla en la documentación de MySQL: <http://dev.mysql.com/doc/refman/5.7/en/source->

2. Configuración en el archivo de configuración

a. Ubicación del archivo de configuración

El archivo `my.cnf` (o `my.ini` en MS Windows) es el archivo de configuración del servidor MySQL. Los programas proporcionados por MySQL (`mysqld`, `mysql`, `mysqldump`, `mysqlsamchk`...) lo consultan para buscar sus directivas. En UNIX, buscan de forma automática el archivo `my.cnf` en los siguientes directorios: `/etc/`, `/etc/mysql/`, `SYSCONFDIR/`, `$MYSQL_HOME/` y `~/`. `SYSCONFDIR` es el directorio especificado por la opción `sysconfdir` al ejecutar el script `configure` (`etc` por defecto), y `$MYSQL_HOME` es la variable de entorno de la ruta del directorio que contiene el archivo `my.cnf`.

Esta variable, en general, tiene como valor la ruta del directorio de datos o la ruta del directorio de instalación definido al ejecutar el script `configure`. Si existe más de un archivo `my.cnf`, las opciones que contienen serán leídas en ese orden y todas se tendrán en cuenta también. En caso de redefinición de una opción, es la última ocurrencia la que prevalece. Por ejemplo, si en `/etc/my.cnf` la opción `long-query-time` vale 10 y vale 2 en `/usr/local/mysql/etc/my.cnf`, su valor al arrancar el servidor será 2. Podemos ver la lista de directorios en los que el servidor buscará el archivo `my.cnf` con el comando `mysqld --help --verbose`.

```
$ mysqld --help --verbose | less
```

```
...
```

```
Default options are read from the following files in the given order:  
/etc/my.cnf /etc/mysql/my.cnf /usr/local/mysql/etc/my.cnf ~/.my.cnf
```

En MS Windows, el principio es similar. El servidor buscará el archivo `my.ini` en los siguientes directorios: `WINDIR\my.ini`, `WINDIR\my.cnf`, `C:\my.ini`, `C:\my.cnf`, `INSTALLDIR\my.ini` y `INSTALLDIR\my.cnf`. `WINDIR` es

el directorio de Windows (en general `C:\WINDOWS`) e `INSTALLDIR` el directorio en el que está instalado MySQL.

- ☐ Si detecta opciones extrañas en el servidor MySQL que no se corresponden con su archivo de configuración, asegúrese de que el servidor no tiene en cuenta otro archivo `my.cnf` (o `my.ini`).

Si el archivo no se encuentra en uno de estos directorios o si no se llama `my.cnf`, necesitará indicar a los programas (`mysqld`, `mysql`, `myisamchk`...) su ubicación poniéndola como parámetro en la llamada al programa. Use la opción `defaults-file` para indicar al programa la ubicación del archivo de configuración. Siempre debe figurar en primera posición. La opción `defaults-extra-file` permite leer un archivo de configuración secundario tras haber leído el principal.

```
$ mysql --defaults-file=/usr2/mysql/conf/mysql_client.cnf ...
$ mysqld --defaults-file=/usr2/mysql/conf/mysql_server.cnf ...
```

b. Estructura del archivo de configuración

El archivo de configuración está organizado en secciones (o grupos). Una sección está compuesta por un nombre (en general, el nombre del programa correspondiente) colocado entre dos corchetes, por opciones y termina por el comienzo de otra sección o el final del archivo. De esta forma, el nombre de la sección del servidor MySQL es `[mysqld]`. Sin embargo, es posible utilizar `[server]`. El nombre de la sección del cliente en texto por defecto es `[mysql]` (no confundir con `[mysqld]`). Siguiendo el mismo principio, podemos usar la sección `[mysqldump]`, que permite configurar el cliente texto para hacer respaldos `mysqldump`. También existe la sección `[client]`, que permite agrupar la información de configuraciones de todos los clientes (`mysql`, `mysqldump`, `mysqladmin`...).

La sintaxis de las opciones es la siguiente: `nombre_opción=valor`. Las opciones compuestas por varias palabras pueden separarse indistintamente por un guión «-» o un guión bajo «_».

Algunas opciones son ejecutables: la funcionalidad se activa con solo poner el nombre de la opción. Por ejemplo, la opción `enable-federated` permite activar el motor `FEDERATED`. Además, podemos poner comentarios en el archivo de configuración: basta con añadir el prefijo almohadilla «#» o un punto y coma «;».

Compruebe la sintaxis de las opciones escritas en el archivo de configuración, ya que el servidor se negará a arrancar si no comprende una opción de la sección `[mysqld]`. Se generará un error que se incluirá en el registro de errores.

Ejemplo de archivo de configuración

```
$ cat /etc/mysql/my.cnf

# Sección global a todos los clientes MySQL
[cliente]
socket = /tmp/mysql.sock # los clientes se conectan con el socket

# Sección del cliente texto mysql
[mysql]
prompt = \r:\m \U$\d> # la línea de comandos del cliente texto mysql
tiene el formato HH:MM user$>schema>

# Sección del servidor MySQL
[mysqld]
slow_query_log # activación del registro de peticiones lentas
slow_query_log_file = mysql-slow.log # Nombre del registro de
peticiones lentas
long-query-time = 2 # Las demandas cuya duración sea superior a
2 segundos son registradas
enable-federated # Activa el motor FEDERATED
```

La configuración del servidor se puede dividir en varios archivos. La instrucción `!include` permite incluir el contenido de un archivo en otro y la instrucción `!includedir` permite analizar un directorio en el que el archivo o archivos de configuración que tienen por extensión `.cnf` en UNIX (`.ini` y `.cnf` en MS Windows)

serán leídos. Podemos, por ejemplo, utilizarla para crear un archivo por grupos funcionales (`replication.cnf`, `log.cnf`, `innodb.cnf`...). Observe, sin embargo, que no se puede garantizar el orden de lectura de archivos con `!includedir`.

3. Configuración al arrancar `mysqld`

Podemos configurar el servidor pasando las opciones de configuración al ejecutar el programa `mysqld`. Estas últimas son los mismas utilizadas en el archivo de configuración pero, precedidas por dos guiones «--».

```
$ ./mysqld --defaults-file=/home/daz/mysql/my.cnf
--basedir=/usr/local/mysql/ --datadir=/home/daz/mysql/data
--log-error=/home/daz/mysql/data/mysql.err
--pid-file=/home/daz/mysql/data/mysql.pid
--socket=/tmp/mysql.sock --port=3306 --console &

[1] 28871
$ 121102 22:20:57 [Warning] TIMESTAMP with implicit DEFAULT value
is deprecated. Please use --explicit_defaults_for_timestamp
server option (see documentation for more details).
```

Para conocer la lista completa de las opciones que puede tomar el servidor, hay que ejecutar el programa `mysqld` con las opciones `--help` y `--verbose`.

```
$ mysql --help --verbose
...
Starts the MySQL database server

Usage: mysql [OPTIONS]

Default options are read from the following files in the given
order:
/etc/my.cnf /etc/mysql/my.cnf /usr/local/mysql/etc/my.cnf
```

De manera general, para conocer las opciones de los ejecutables suministrados por MySQL, las ejecutamos con la opción `--help`. La opción `--verbose` solo es necesaria con el programa `mysqld`.

```
$ mysql --help
...
Usage: mysql [OPTIONS] [database]
-?, --help Display this help and exit.
-I, --help Synonym for -?
...
```

4. Configuración dinámica del servidor

Algunas opciones se pueden cambiar de forma dinámica (en caliente) mediante el comando `SET` siempre que el usuario tenga el derecho `SUPER` (ver el capítulo Seguridad y gestión de los usuarios). Esto es especialmente práctico para perfeccionar la configuración del servidor MySQL en caso de emergencia. El cambio puede realizarse de dos maneras, bien a nivel local en la conexión, es decir, solo para la sesión de un cliente en curso mediante la cláusula `SESSION`, o bien de forma global en el servidor, empleando la cláusula `GLOBAL`. La mayoría de las opciones tienen un alcance de sesión y global (como la opción `sql_mode` que veremos más adelante en este capítulo), pero este no es el caso de todas. Por ejemplo, la opción `sql_log_bin`, que permite desactivar el registro binario, solo tiene un alcance de sesión, mientras que la opción `query_cache_size`, que permite determinar el tamaño de la caché de peticiones, solo tiene alcance global.

a. Cambio para la sesión

Usando el comando `SET SESSION parámetro_sesion=valor`, el valor del parámetro cambiará solo para la sesión del cliente en la que se ejecutó el comando.

Cambio para la sesión de una variable con alcance de sesión y global

```
mysql> SHOW GLOBAL VARIABLES LIKE 'sort_buffer_size';
```

Variable_name	Value
sort_buffer_size	2097144

```
mysql> SHOW SESSION VARIABLES LIKE 'sort_buffer_size';
```

Variable_name	Value
sort_buffer_size	2097144

```
mysql> SET SESSION sort_buffer_size=1048576;
```

```
mysql> SHOW GLOBAL VARIABLES LIKE 'sort_buffer_size';
```

Variable_name	Value
sort_buffer_size	2097144

```
mysql> SHOW SESSION VARIABLES LIKE 'sort_buffer_size';
```

Variable_name	Value
sort_buffer_size	1048576

b. Cambio global

Con el comando `SET GLOBAL parámetro_global=valor`, el cambio es efectivo para todos los clientes que se conectan al servidor después de que se ejecute el comando. Sin embargo,

este nuevo valor no se tiene en cuenta para las sesiones de los clientes conectados antes del cambio (sesiones en curso), que conservan el valor inicial del parámetro. Lo mismo sucede con la sesión del cliente en la que se ha ejecutado el comando.

Una excepción a esta regla se refiere a los parámetros que solo tienen un alcance global. La actualización se llevará a cabo también en la sesión del cliente que realiza el comando SETGLOBAL.

Cambio global de una variable con alcance de sesión y global

```
mysql> SHOW GLOBAL VARIABLES LIKE 'sort_buffer_size';
```

+-----+-----+	
Variable_name	Value
+-----+-----+	
sort_buffer_size	2097144
+-----+-----+	

```
mysql> SHOW SESSION VARIABLES LIKE 'sort_buffer_size';
```

+-----+-----+	
Variable_name	Value
+-----+-----+	
sort_buffer_size	1048576
+-----+-----+	

```
mysql> SET GLOBAL sort_buffer_size=5242880;
```

```
mysql> SHOW GLOBAL VARIABLES LIKE 'sort_buffer_size';
```

+-----+-----+	
Variable_name	Value
+-----+-----+	
sort_buffer_size	5242880
+-----+-----+	

```
mysql> SHOW SESSION VARIABLES LIKE 'sort_buffer_size';
```

+-----+-----+	
Variable_name	Value
+-----+-----+	
sort_buffer_size	1048576

```
+-----+-----+
```

Cambio global de una variable con solo alcance global

```
mysql> SHOW VARIABLES LIKE 'query_cache_size';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| query_cache_size | 0 |
+-----+-----+
```

```
mysql> SET SESSION query_cache_size=122880;
ERROR 1229 (HY000): Variable 'query_cache_size' is a GLOBAL
variable and should be set with SET GLOBAL
$ SET GLOBAL query_cache_size=122880;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SHOW VARIABLES LIKE 'query_cache_size';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| query_cache_size | 122880 |
+-----+-----+
```

Es muy importante tener en cuenta que estos cambios en caliente no son persistentes. Están almacenados en el espacio de memoria utilizado por el servidor MySQL y no sobreviven, por lo tanto, a un reinicio del servidor. Los cambios realizados con el comando `SET SESSION` no resisten tampoco al reinicio del cliente, pues esto inicia una nueva sesión. Si quiere hacer que los cambios globales sean persistentes, debe ponerlos en el archivo de configuración (o también pasarlos como parámetros a `mysqld`).

Visualización de la configuración

El Administrador tiene varias posibilidades para ver la configuración del servidor MySQL. La manera más obvia es pasar por el archivo de configuración. Sin embargo, es posible que algunas de las opciones que contiene sean redefinidas al pasarlas por parámetro a `mysqld`. También debemos pensar en mirar el estado del proceso `mysqld`; por ejemplo, con el comando `ps -ef | grep mysqld` en Linux.

Pero estos dos primeros métodos tienen algunas limitaciones. En un servidor mal administrado, puede haber varios archivos de configuración, que el servidor no utilice necesariamente, y que además no tienen por qué llamarse `my.cnf` (o `my.ini`). Es más, los parámetros no suelen aparecer todos en el archivo de configuración y los que sí aparecen pueden redefinirse en caliente, como sucede con el parámetro de `mysqld`.

Para estar seguro de ver la configuración actual del servidor, debemos utilizar el comando `SHOW GLOBAL VARIABLES`, es decir, las tablas `GLOBAL_VARIABLES` del esquema virtual `INFORMATION_SCHEMA` o el comando `SELECT @@global.nombre_variable`.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES WHERE  
VARIABLE_NAME = 'datadir';
```

```
+-----+-----+  
| VARIABLE_NAME | VARIABLE_VALUE |  
+-----+-----+  
| DATADIR      | /home/daz/mysql/data/ |  
+-----+-----+
```

```
mysql> SHOW GLOBAL VARIABLES LIKE 'datadir';
```

```
+-----+-----+-----+  
| | | Índice |  
+-----+-----+-----+
```

Variable_name	Value
datadir	/home/daz/mysql/data/

```
mysql> SELECT @@global.datadir;
```

@@global.datadir	/home/daz/mysql/data/
------------------	-----------------------

Un error común es ejecutar el comando `SHOW VARIABLES` en lugar de `SHOW GLOBAL VARIABLES`. Si omitimos la cláusula `GLOBAL`, el comando se ejecuta implícitamente con la cláusula de `SESSION`. Lo que tiene el efecto de mostrar el estado de las variables para la sesión del cliente en la que se ejecuta el comando. En `INFORMATION_SCHEMA`, las opciones de configuración relacionadas con la sesión en curso se encuentran en la tabla `SESSION_VARIABLES`.

```
mysql> SHOW GLOBAL VARIABLES LIKE 'character_set_connection';
```

Variable_name	Value
character_set_connection	utf8

```
mysql> SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES WHERE  
VARIABLE_NAME LIKE 'character_set_connection';
```

VARIABLE_NAME	VARIABLE_VALUE
CHARACTER_SET_CONNECTION	utf8

```
mysql> SHOW VARIABLES LIKE 'character_set_connection';
```

```

+-----+-----+
| Variable_name          | Value |
+-----+-----+
| character_set_connection | utf8  |
+-----+-----+

```

```
mysql> SHOW SESSION VARIABLES LIKE 'character_set_connection';
```

```

+-----+-----+
| Variable_name          | Value |
+-----+-----+
| character_set_connection | utf8  |
+-----+-----+

```

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SESSION_VARIABLES WHERE
VARIABLE_NAME LIKE 'character_set_connection';
```

```

+-----+-----+
| VARIABLE_NAME          | VARIABLE_VALUE |
+-----+-----+
| CHARACTER_SET_CONNECTION | utf8          |
+-----+-----+

```

Otro comando que puede ser útil es STATUS. Proporciona, entre otras cosas, información sobre el estado del servidor e información sobre los juegos de caracteres utilizados:

```
mysql> STATUS
```

```
-----
```

```
mysql Ver 14.14 Distrib 5.6.7-rc, for linux2.6 (x86_64) using
EditLine wrapper
```

```
...
```

```
SSL: Not in use
```

```
Current pager: stdout
```

```
Using outfile: ''
```

```
Using delimiter: ;
```

```
...
```

```
Server version:          5.6.7-rc MySQL Community Server (GPL)
```

```
Protocol version:        10
```


Connection: Localhost via UNIX socket

Server charset: utf8

Db charset: utf8

Client charset: utf8

Conn. charset: utf8

UNIX socket: /tmp/mysql.sock

...

Configuración de InnoDB

1. Parámetros fundamentales

No es necesario modificar la configuración predeterminada de los parámetros que se mencionan en esta sección, pero sí que es importante pararse a pensar en cuál sería el valor adecuado para su aplicación.

- Tamaño de la caché de memoria (`innodb_buffer_pool_size`): se trata de la caché principal de InnoDB (buffer pool), en la que se almacenan los datos e índices a los que se accede con frecuencia. Para un servidor dedicado de MySQL, es corriente que se asigne la mayor parte de la memoria del servidor (por ejemplo, alrededor de 25 GB en un servidor con 32 GB de memoria física). La idea principal es que la caché permita evitar el acceso al disco: el tamaño de la caché es más importante que la velocidad del disco.

Si nuestra base de datos es pequeña (decenas de GB, por ejemplo), es bastante sencillo utilizar un servidor que tenga suficiente memoria para que todos los datos e índices InnoDB se mantengan en la caché. De lo contrario, hay que tratar de tener en caché la parte útil de los datos e índices, es decir, la parte de los datos e índices utilizada con frecuencia por la aplicación. Es bastante frecuente, en efecto, tener, por ejemplo, una base de 500 GB que contiene la historia de los datos sobre los últimos cinco años, pero solo se consultan a menudo los datos del último mes, lo que representa 10 GB. En este caso, es inútil tratar de asignar 500 GB al buffer pool; 10-15 GB serán suficientes.

- Tamaño del registro de transacciones (`innodb_log_file_size`): el registro de transacciones (*redo log*) le permite a InnoDB ofrecer un buen rendimiento de escritura, garantizando la integridad de datos en caso de fallo

inesperado (ver el capítulo Aspectos generales de MySQL). La idea principal es que InnoDB escriba de manera síncrona las modificaciones en su registro de transacciones (escrituras de bajo coste, ya que son secuenciales) y que un thread en segundo plano modifique los archivos de datos de forma asíncrona (escrituras más costosas al ser aleatorias). Sin embargo, el tamaño del registro de transacciones es limitado, así que si escribe más rápido en el registro de transacciones que espacio genera la purga en segundo plano en dicho registro, InnoDB podría impedir la escritura durante algunos segundos hasta que el proceso en segundo plano consiga espacio libre. Este comportamiento es catastrófico para el rendimiento de escritura; por tanto, es especialmente importante dimensionar de forma correcta el registro de transacciones. El valor por defecto de 48 MB es apropiado para aplicaciones que escriben poco, pero será muy insuficiente si el volumen de escritura es más elevado. Intente dimensionar el registro de transacciones para que pueda contener 1 hora de modificaciones en hora punta aproximadamente. Esto se hace simplemente comparando con 1 hora de diferencia el valor de la variable `Log sequence number` que aparece en la sección `log` de la salida de `SHOW ENGINE INNODB STATUS`. Los valores de 256/512/1024 MB son comunes.

- Sostenibilidad (`innodb_flush_log_at_trx_commit`): este parámetro se explica en la siguiente sección.
- Tablespace para cada tabla (`innodb_file_per_table`): en las antiguas versiones de MySQL, todos los datos e índices de todas las tablas estaban almacenados en el tablespace compartido (`ibdata1`). Esto no plantea ningún problema de rendimiento, pero muchos administradores no aprecian demasiado encontrarse con un único archivo enorme que contiene todos los datos. Este archivo puede superar el TB si la base es lo bastante voluminosa. Para evitar estas molestias, la solución ha sido siempre utilizar `innodb_file_per_table = 1`, lo que indica a InnoDB almacenar cada tabla en su propio archivo `.ibd`. Se trata del valor por defecto desde MySQL 5.6, y su principal ventaja es poder recuperar el espacio en disco de las tablas eliminadas.

Observe que `innodb_file_per_table` no ofrece ninguna ventaja de rendimiento. Por último, tenga en cuenta que hay un caso en el que se aconseja no utilizar esta opción: si tenemos muchas tablas en nuestro esquema, por ejemplo:

varias decenas de miles. En este caso, la mayoría de las tablas son pequeñas en general, e `innodb_file_per_table` pierde espacio en disco para cada tabla, lo que termina por convertirse en molesto cuando el número de tablas es muy elevado.

- Método de flush de datos (`innodb_flush_method`): por defecto, cuando las modificaciones se escriben en los archivos de datos, al menos una parte también se conserva en la caché del sistema de archivos. En general, el uso de esta caché es totalmente inútil, puesto que InnoDB tiene su propia caché (el buffer pool). A menudo, si los discos son rápidos, es preferible evitar esta doble caché especificando `innodb_flush_method = O_DIRECT`. Observe que, si los discos son lentos, `O_DIRECT` puede tener un verdadero impacto negativo sobre el rendimiento de todos los accesos a disco, ya que la caché del sistema de archivos se ignora.

2. Aislamiento y durabilidad

Aislamiento y durabilidad son la I y la D del acrónimo ACID al que nos hemos referido en el capítulo Aspectos generales de MySQL. Comprender estos dos conceptos es importante para una buena configuración de InnoDB: en efecto, es posible influir de forma significativa sobre el rendimiento del motor haciendo variar el nivel de aislamiento o el nivel de durabilidad, pero el impacto sobre la seguridad de los datos es también significativo. Tomemos, por lo tanto, el tiempo de entender las ventajas y desventajas de cada tipo de configuración.

a. Ajuste del aislamiento

En el acrónimo ACID, que describe las propiedades verificadas por los sistemas transaccionales, la I significa aislamiento (*isolation* en inglés), es decir, que las operaciones realizadas en una transacción no tienen efecto sobre las demás transacciones en curso. Sin embargo, podemos configurar el servidor para que este aislamiento se realice en mayor o menor medida.

Según el nivel de aislamiento, se pueden dar tres problemas en las transacciones:

- Lectura sucia (*dirty read*): este problema se produce cuando una transacción puede leer las líneas modificadas por otra transacción, pero que no han sido validadas aún

por un COMMIT.

- Lectura no repetible (*no-repeatable read*): este problema se produce cuando se ejecuta varias veces una misma petición SELECT y el servidor no devuelve el mismo resultado porque otras transacciones modificaron las filas y las han validado mediante un COMMIT.
- Lectura fantasma (*phantom read*): este problema se produce cuando los datos insertados por otras transacciones se hacen visibles.

Existen cuatro niveles de aislamiento, definidos en el estándar SQL y soportados por InnoDB.

- READ UNCOMMITTED: los cambios hechos por otras transacciones y no validados por un COMMIT son visibles. Las lecturas sucias, las lecturas no repetibles y las lecturas fantasmas son posibles.
- READ COMMITTED: los cambios hechos por otras transacciones y validados por un COMMIT son visibles. Las lecturas no repetibles y las lecturas fantasmas son posibles.
- REPEATABLE READ: en este nivel de aislamiento, si una misma petición se realiza varias veces en una misma transacción, el servidor devuelve siempre el mismo resultado. Solo son posibles las lecturas fantasmas. Es el nivel de aislamiento predeterminado de InnoDB.
- SERIALIZABLE: cada transacción se aísla por completo de las otras transacciones. Ninguno de los tres problemas mencionados es posible.

☐ El nivel de aislamiento SNAPSHOT de Oracle y SQL Server no existe con MySQL.

InnoDB mantiene el aislamiento entre las transacciones conservando siempre que sea necesario las antiguas versiones de las filas, de tal manera que varias transacciones diferentes puedan ver cada una la versión correcta de los datos. Mantener varias versiones de los datos sobre muchas de las filas tiene, sin embargo, un coste.

El nivel de aislamiento se modifica utilizando la variable `transaction-`

isolation. Los valores posibles son READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ y SERIALIZABLE.

Tenga en cuenta que, en general, el nivel predeterminado es el mejor compromiso entre rendimiento y buen aislamiento de las transacciones. Sin embargo, puede ser muy interesante utilizar el nivel READ COMMITTED si el servidor debe ejecutar de forma periódica peticiones muy lentas (como las peticiones de reporting): en este caso, READ COMMITTED permite mantener durante menos tiempo las antiguas versiones de las filas, lo que puede tener un impacto positivo significativo en el rendimiento.

b. Ajuste de la durabilidad

InnoDB almacena la información de cada transacción en una caché para evitar consultar el disco. Estos datos residen en memoria, por lo que no pueden resistir a un fallo catastrófico (crash) de MySQL y mucho menos a un crash del servidor entero. En el momento del COMMIT, esta información se copia en los registros de InnoDB para poder aplicarse a los archivos de datos correspondientes (tablas e índices) posteriormente.

Tan pronto como los datos se copian en un registro, se pueden considerar seguros. En efecto, incluso en caso de fallo inesperado, InnoDB puede leer el registro para efectuar las modificaciones solicitadas en las tablas. El mecanismo implementado por InnoDB permite, por tanto, garantizar la durabilidad de los cambios realizados en una transacción (la D del acrónimo ACID).

Se puede dar una dificultad adicional durante la escritura en el archivo de log: el sistema operativo puede decidir no escribir en el disco, sino en su caché. En ambos casos, InnoDB tendrá la impresión de que la escritura tuvo lugar en el disco, pero si los datos están en la caché del sistema operativo, están en realidad en la memoria y, por lo tanto, se producirán pérdidas en caso de fallo sin previo aviso.

Por ello, InnoDB pide al sistema operativo hacer un flush después de escribir para garantizar que los datos se encuentren en el disco. Esta operación tiene un gran coste en términos de rendimiento.

InnoDB permite escoger la calidad de la durabilidad con la opción `innodb_flush_log_at_trx_commit`. Los tres posibles valores son 0, 1 o 2:

1 es el valor predeterminado. Esto significa que InnoDB transfiere la información de la caché al registro y efectúa un flush del registro para cada COMMIT.

- 2 indica a InnoDB que debe transferir la información de la caché al registro en cada COMMIT, pero hace un flush del registro alrededor de una vez por segundo.
- 0 indica a InnoDB que debe transferir la información de la caché al registro y hacer un flush del registro alrededor de cada segundo.

Queda por explicar la diferencia entre los valores 0 y 2, que puede no parecer evidente.

Con el valor 2, con cada COMMIT, la información relativa a la transacción se almacenará, en el peor de los casos, en la caché del sistema operativo y, en el mejor, en el disco. Esto significa que si MySQL sufre un crash (pero no del servidor entero) entre el momento de COMMIT y el momento en que el archivo de registro se escribe en el disco (lo que se produce un máximo de un segundo más tarde), la información de la transacción se almacena en la caché del sistema operativo y puede ser recuperada. Al contrario, con el valor 0, la información de la transacción no se almacena en la caché y, por lo tanto, se pierde en caso de un crash de MySQL.

En contraposición, en caso de crash del equipo host, podemos perder a lo sumo un segundo de datos con los valores 0 o 2. Solo el valor 1 nos garantiza no perder datos.

En resumen, cuando necesitamos garantizar la durabilidad en el sentido ACID del término, el valor 1 es un imperativo. De lo contrario, el valor 2 es sin duda un buen compromiso entre rendimiento y seguridad de datos.

- ☐ A título comparativo, podemos considerar que el nivel de durabilidad aportado por MyISAM es equivalente al de InnoDB cuando `innodb_flush_log_at_trx_commit` se define en 0. Téngalo en cuenta si busca comparar el rendimiento de MyISAM e InnoDB.

3. Otros parámetros

He aquí una lista de parámetros que puede ser útil modificar:

- `Innodb_buffer_pool_instances`: si el buffer pool es muy grande

(varias decenas o centenares de GB), puede que algunas operaciones sufran contención. En tal caso, es aconsejable usar varias instancias del buffer pool para disminuir las contenciones. El valor predeterminado de 8 suele ser correcto.

- `innodb_buffer_pool_dump_at_shutdown`,
`innodb_buffer_pool_load_at_startup`: estas dos opciones permiten tener una huella del buffer pool antes de apagar el servidor y de recargar el buffer pool al arrancar. Esto sirve para preservar más o menos el buffer pool, que es el elemento más crítico de InnoDB para el rendimiento. Desde MySQL 5.7, estas opciones están activadas por defecto.
- `innodb_io_capacity`, `innodb_io_capacity_max`: los procesos en segundo plano de InnoDB limitan su actividad en disco para no penalizar las actividades en primer plano. Sin embargo, si contamos con discos rápidos y muchas escrituras, los procesos en segundo plano podrían no utilizar todas las capacidades de los discos. Esto indica que hay que aumentar el valor de estas dos variables. Elegir un buen valor es bastante difícil, ya que un valor demasiado elevado llevará a InnoDB a utilizar el disco para los procesos en segundo plano, lo que también puede causar problemas de rendimiento. La unidad para estas dos opciones es el número de operaciones en disco por segundo. Se debe saber cuántas operaciones puede realizar el disco para configurar este parámetro. Por ejemplo, si el disco puede alcanzar 2000 operaciones por segundo, `innodb_io_capacity = 1000` y `innodb_io_capacity_max = 1800` podrían ser valores adecuados.

El registro

El servidor MySQL utiliza cuatro tipos de registros, cada uno con sus especialidades: el registro binario (*binary log* o *binlog*), el registro de peticiones lentas (*slow query log*), el registro general (*general query log*) y el registro de errores (*error log*), el único de los cuatro que se activará por defecto.

- ☐ El relay-log, otro tipo de registro, es creado por el servidor durante la replicación. Se aborda en el capítulo Replicación.

1. ☐ El registro binario

El registro binario o *binlog* se encarga de almacenar en un formato binario todas las peticiones que modifican los objetos de la base de datos (INSERT, UPDATE, DELETE, DROP, CREATE, ALTER...). Es el elemento central de la replicación MySQL (ver el capítulo Replicación). También es muy útil para la restauración de los datos (ver el capítulo Respaldo y restauración). Para activar el registro binario, use la opción `log-bin`, y para definir su archivo de índice, configure la opción `log-bin-index`. Este archivo, que contiene la lista de todos los registros binarios desde la última purga, le permite al servidor conocer el nombre del archivo actual que se obtiene con el comando `SHOW MASTER STATUS` y también mostrar la lista de todos los registros binarios presentes en el servidor con `SHOW BINARY LOGS`. El registro binario se puede desactivar en caliente, pero solo para la sesión de un cliente, con la opción `SQL_LOG_BIN`.

- ☐ No confunda los registros binarios, que registran todas las escrituras, con los registros de transacción de InnoDB (redo logs), que solo registran las

escrituras en las tablas InnoDB y cuyo único objetivo es garantizar la restauración automática de InnoDB en caso de fallo inesperado.

Una configuración básica podría ser:

```
[mysqld]  
log-bin = /var/lib/mysql/mysql-bin
```

- ☐ Observe que `log-bin-index` no está definido. Por defecto, el archivo se creará en el mismo directorio que los registros binarios (`/var/lib/mysql` aquí).
- ☐ Observe también que puede especificar una ruta relativa para la opción `log-bin`. En este caso, la ruta será relativa en relación con el directorio de datos. Para evitar confusiones, utilice siempre una ruta absoluta.

El formato utilizado para el registro es un formato binario; es significa que las entradas del registro no pueden leerse de forma directa. Por lo tanto, necesitará recurrir a `mysqlbinlog` cada vez que tenga que leer el contenido de los registros binarios, especificando el archivo o los archivos que desee decodificar.

```
$ mysqlbinlog /var/lib/mysql/mysql-bin.000001
```

- ☐ La información del registro binario puede visualizarse también en el cliente de texto `mysql` con el comando `SHOW BINLOG EVENTS`.

Los tres formatos posibles para el registro binario (`STATEMENT`, `ROW` y `MIXED`) se

detallan en el capítulo Replicación, pero he aquí lo esencial:

- STATEMENT era el formato predeterminado hasta MySQL 5.6; por tanto, es todavía bastante actual. El texto de las peticiones se transcribe de forma directa en el registro binario, lo que hace que el registro sea fácil de interpretar por un humano.

He aquí un ejemplo de una entrada en el registro en formato STATEMENT:

```
$ mysqlbinlog mysql-bin.000003
# at 718
#160503 17:12:53 server id 13001  end_log_pos 718 CRC32 0xb82f4450
Intvar
SET INSERT_ID=1/*!*/;
#160503 17:12:53 server id 13001  end_log_pos 843 CRC32 0xa64c2939
Query
    thread_id=6          exec_time=0 error_code=0
SET TIMESTAMP=1462288373/*!*/;
insert into t (col) values ('Esto es una prueba')
/*!*/;
```

Uno de los inconvenientes de este formato que se pueden dar en la replicación es la aparición de discrepancias en el esclavo. Veremos un ejemplo en el capítulo Replicación.

- El formato ROW es el formato predeterminado a partir de MySQL 5.7. En general, más rápido y seguro para la replicación, es el formato preferido. Solo se conservan en el registro binario los cambios de cada fila. La petición original se pierde. El principal inconveniente de este formato es que hace que las peticiones sean difíciles de leer por un humano.

La misma inserción que hemos visto antes con el formato ROW:

```
$ mysqlbinlog mysql-bin.000003
# at 293
#160503 17:21:32 server id 13001  end_log_pos 342 CRC32 0x1477ef57
    Table_map: `my_app`.`t` mapped to number 108
# at 342
```

```
#160503 17:21:32 server id 13001 end_log_pos 399 CRC32 0x5d4c8e8c
  Write_rows: table id 108 flags: STMT_END_F
BINLOG '
/MEoVxPJMgAAMQAAAFYBAAAAAGwAAAAAAAAEABm15X2FwcAABdAACAw8CFAACV+93FA==
/MEoVx7JMgAAOQAAAI8BAAAAAGwAAAAAAAAEAAgAC//wCAAAAEENlY2kgZXN0IHVuIHRlc3SM

'/*!*/;
```

Con `-vv`, el contenido es un poco más claro:

```
$ mysqlbinlog -vv mysql-bin.000003
# at 342
#160503 17:21:32 server id 13001 end_log_pos 399 CRC32 0x5d4c8e8c
  Write_rows: table id 108 flags: STMT_END_F
BINLOG '
/MEoVxPJMgAAMQAAAFYBAAAAAGwAAAAAAAAEABm15X2FwcAABdAACAw8CFAACV+93FA==
/MEoVx7JMgAAOQAAAI8BAAAAAGwAAAAAAAAEAAgAC//wCAAAAEENlY2kgZXN0IHVuIHRlc3SM

'/*!*/;
### INSERT INTO `my_app`.`t`
### SET
###   @1=2 /* INT meta=0 nullable=0 is_null=0 */
###   @2='Esto es una prueba' /* VARSTRING(20) meta=20 nullable=1
is_null=0 */
```

- El formato `MIXED` pretende equilibrar las ventajas de los dos formatos anteriores: por defecto, las peticiones se almacenan en formato `STATEMENT`, salvo cuando se corre el riesgo de provocar incoherencias de datos en los esclavos de replicación, en cuyo caso se aplica automáticamente el formato `ROW`. En realidad, este formato sigue siendo peligroso porque es menos frecuente, padece de varios bugs que pueden ocasionar pérdidas de datos en los esclavos de replicación.

El registro binario es un registro incremental cuyo incremento administra el servidor. El

salto es de 1. El primer archivo binario creado tiene el formato `nombre_archivo.000001`; el siguiente, `nombre_archivo.000002`, y así sucesivamente. El paso al siguiente se realiza al cerrar el archivo en curso. Este paso de testigo puede realizarse de tres modos diferentes:

- En caso de reinicio del servidor.
- Si se alcanza el tamaño máximo del archivo especificado con la opción `max_binlog_size`.
- En caso de ejecutar el comando `FLUSH LOGS`.

Observe, sin embargo, que los antiguos archivos no se purgan por defecto. Tendremos varias posibilidades:

- Una eliminación automática por el servidor MySQL de los registros de más de N días con la opción `expire_logs_days = N`.
- Una eliminación manual:
 - Bien en comparación con un intervalo de tiempo: `PURGE BINARY LOGS BEFORE`
 - O bien en comparación con un número de registro: `PURGE BINARY LOGS TO`
 - O bien borrándolo todo y volviendo a empezar con el archivo `.000001`:
`RESET MASTER`.

La integridad de los datos replicados en caso de crash se mejoró con MySQL 5.6 mediante el cálculo de la suma de control (*checksum*) de los eventos del registro binario, asignando el valor CRC32 (valor por defecto) a la variable `binlog_checksum`. Este cálculo se puede desactivar con el valor `NONE`. El servidor solo comprueba entonces que el evento esté totalmente escrito en el registro.

```
mysql> SHOW VARIABLES LIKE 'binlog_checksum';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| binlog_checksum | CRC32 |
```

```
+-----+-----+
```

```
# at 201
#121020 13:18:55 server id 1 end_log_pos 340 CRC32 0x424b4f4c
Query   thread_id=2   exec_time=0   error_code=0
use world/*!*/;
SET TIMESTAMP=1350731935/*!*/;
INSERT INTO `CountryLanguage` VALUES ('BLZ','Spanish','F',31.6)
/*!*/;
```

He aquí un ejemplo de configuración, con en el archivo `my.cnf` (o `my.ini`) mostrando la activación del registro en el directorio `/usr/local/mysql/logs/` y que tiene como prefijo `mysql-logbin`. El índice de los registros binarios está en el mismo directorio y se llama `mysql-logbin.index`. Los registros de más de 10 días serán borrados por el servidor de forma automática (`expire_logs_days=10`).

Ejemplo de configuración del registro binario

```
[mysqld]
log-bin = /usr/local/mysql/logs/mysql-logbin
log-bin-index = /usr/local/mysql/logs/mysql-logbin.index
expire_logs_days = 10
```

El comando `SHOW MASTER STATUS` permite obtener información sobre el registro binario actual: su nombre (`File`) y su tamaño (que equivale a 120 bytes), es decir, la posición del último carácter en el archivo (`Position`). `Binlog_Do_DB` y `Binlog_Ignore_DB` proporcionan la lista de esquemas de forma respectiva, cuyas tablas son los únicos objetos replicados, y la lista de los esquemas que incluyen tablas no replicadas.

```
mysql> SHOW MASTER STATUS \G
***** 1. row *****
File: mysql-logbin.000001
```

```
Position: 120
Binlog_Do_DB:
Binlog_Ignore_DB:
Executed_Gtid_Set:
```

Las peticiones de escritura harán crecer el registro binario. Su tamaño aumentó de 120 a 424 bytes.

```
mysql> CREATE TABLE test(i int);
```

```
mysql> INSERT test VALUES (1);
```

```
mysql> SELECT * FROM test;
```

```
+-----+
| i      |
+-----+
| 1      |
+-----+
```

```
mysql> SHOW MASTER STATUS \G
```

```
***** 1. row *****
File: mysql-logbin.000001
Position: 424
Binlog_Do_DB:
Binlog_Ignore_DB:
Executed_Gtid_Set:
```

El comando `SHOW BINLOG EVENTS` permite ver que las dos peticiones de escritura están presentes en el registro, mientras que el `SELECT` no está presente. Cabe señalar que los 120 primeros bytes son información añadida por el servidor al crear el archivo.

```
mysql> SHOW BINLOG EVENTS IN 'mysql-logbin.000001' \G
```

```
***** 1. row *****
```

```
Log_name: mysql-logbin.000001
Pos: 4
Event_type: Format_desc
Server_id: 1
End_log_pos: 120
Info: Server ver: 5.7.12-log, Binlog ver: 4
***** 2. row *****
Log_name: mysql-logbin.000001
Pos: 120
Event_type: Query
Server_id: 1
End_log_pos: 218
Info: use `test`; CREATE TABLE test(i int)
***** 3. row *****
Log_name: mysql-logbin.000001
Pos: 218
Event_type: Query
Server_id: 1
End_log_pos: 297
Info: BEGIN
***** 4. row *****
Log_name: mysql-logbin.000001
Pos: 297
Event_type: Query
Server_id: 1
End_log_pos: 393
Info: use `test`; INSERT test VALUES (1)
***** 5. row *****
Log_name: mysql-logbin.000001
Pos: 393
Event_type: Xid
Server_id: 1
End_log_pos: 424
Info: COMMIT /* xid=16 */
```

El registro mysql-logbin.000001 se ha cerrado y mysql-logbin.000002 se ha creado al ejecutar el comando FLUSH LOGS.


```
mysql> FLUSH LOGS;
Query OK, 0 rows affected (0,16 sec)
mysql> SHOW BINARY LOGS ;
+-----+-----+
| Log_name          | File_size |
+-----+-----+
| mysql-logbin.000001 | 468       |
| mysql-logbin.000002 | 120       |
+-----+-----+
```

El registro binario activo es ahora `mysql-logbin.000002`.

```
mysql> SHOW MASTER STATUS \G
***** 1. row *****
File: mysql-logbin.000002
Position: 120
Binlog_Do_DB:
Binlog_Ignore_DB:
Executed_Gtid_Set:
```

Último punto por conocer acerca de los registros binarios con MySQL 5.7: el valor predeterminado del parámetro `sync_binlog` ha cambiado en comparación con versiones anteriores. El papel de esta opción es indicar con qué frecuencia las escrituras en el registro son sincronizadas en disco, garantizando que se registran de manera permanente. Hasta MySQL 5.6, el valor predeterminado era 0, lo que significa que la sincronización solo se efectuaba cuando el sistema operativo lo decidía (es decir, aproximadamente una vez por segundo). A partir de MySQL 5.7, este valor por defecto es 1, lo que significa que el registro se sincroniza en disco para cada transacción.

Las consecuencias son importantes: hasta MySQL 5.6 y con el valor por defecto, era posible que algunas transacciones validadas en InnoDB estuvieran ausentes en los registros binarios, lo que podía plantear problemas de pérdida de datos o problemas de

replicación. Esto ya no es posible con MySQL 5.7. Pero el aspecto negativo es la pérdida de rendimiento, que puede ser muy significativa con discos lentos: en efecto, la sincronización en el disco es una operación muy costosa, equivalente a una escritura aleatoria.

Dado que la mayoría de los usuarios mantienen `sync_binlog = 0` con MySQL 5.6, es tentador conservar este valor con MySQL 5.7. Como mínimo, hay que tenerlo en cuenta durante las pruebas de rendimiento para comparar las dos versiones. De lo contrario, MySQL 5.7 puede resultar artificialmente lento para las escrituras.

2. El registro de peticiones lentas

El registro de peticiones lentas (*slow query log*) tiene como principal función ayudar al administrador de la base de datos a identificar las peticiones cuyo tiempo de ejecución es un problema. Su activación se realiza en caliente o en frío con la opción `slow_query_log`, y todas las peticiones cuyo tiempo de ejecución supere el valor de la opción `long_query_time` (en segundos) serán registradas.

- ☐ Es posible almacenar la información sobre las peticiones lentas en una tabla (`mysql.slow_log`) en lugar de en un archivo con el parámetro `log_output = TABLE`. Sin embargo, este método de registro puede provocar pérdidas de rendimiento importantes y es mejor no utilizarlo nunca.

Ejemplo de configuración

```
[mysqld]
slow_query_log = ON
slow_query_log_file = /var/lib/mysql/mysql-slow.log
long-query-time = 1
```

Ejemplo de entrada en el registro

```
# Time: 2016-05-04T13:05:14.216282Z
# User@Host: root[root] @ localhost [] Id:      3
# Query_time: 0.000661  Lock_time: 0.000403 Rows_sent: 2
Rows_examined: 2
SET timestamp=1462367114;
select * from my_app.t;
```

El objetivo principal de un servidor de bases de datos es ejecutar lo antes posible las peticiones. El registro de peticiones lentas es una mina de oro para quien desea mejorar el rendimiento. Una primera forma de utilizar este registro es colocar la variable `long_query_time` con un valor alto (por ejemplo, 5 o 10) para identificar las peticiones extremadamente lentas. Otra forma de proceder es poner `long_query_time = 0` para grabar un histórico de todas las peticiones realizadas por el servidor. En este caso, se aconseja vigilar el espacio en disco, ya que un servidor cargado puede escribir 1 GB por minuto en el registro de peticiones lentas.

¿Cómo analizar las peticiones de un archivo que contiene todas las peticiones realizadas durante, por ejemplo, una hora? El contenido es demasiado voluminoso para ser examinado de forma manual. La mejor herramienta de análisis es `pt-query-digest`, uno de los scripts de Percona Toolkit. `pt-query-digest` agrupa todas las peticiones similares (por ejemplo, `SELECT * FROM t WHERE id=100` y `SELECT * FROM t WHERE id = 23`) y crea un informe que contiene toda la información necesaria para el análisis y la optimización de las peticiones: un resumen que contiene la lista de grupos de peticiones más lentas y, para cada grupo, muchas estadísticas sobre la ejecución, así como un ejemplo concreto de una petición.

- ☐ Es preciso comprender que, cuando se trata de mejorar los resultados globales de un servidor, es mucho más ventajoso optimizar las peticiones rápidas que se ejecutan a menudo que las peticiones lentas que no se ejecutan casi nunca. Si una petición toma 10 ms, pero se ejecuta 1000 veces por segundo, reducir su tiempo de respuesta a 8 ms supondrá una gran diferencia. Sin embargo, si una petición que toma 18 s solo se ejecuta 1 vez al día, nadie notará la diferencia si la mejoramos para que se ejecute en 2 s.

Ejemplo de informe con pt-query-digest (ligeramente formateado)

```
# Profile
# Rank Query ID          Response time    Calls R/Call V/M    Item
# ==== =====
#      1 0xDD9512F54A8FDC75 203.8130 47.4%   957 0.2130   0.01  SELECT
wp_redirection_items wp_redirection_groups wp_redirection_modules
#      2 0x92F3B1B361FB0E5B 148.5445  34.6%   959 0.1549   0.00  SELECT
wp_options
#      3 0xD6AA15532ACDD64  20.2834   4.7%   313 0.0648   0.00  SELECT
wp_users
#      4 0xE208B92518DF0A44  13.5583   3.2%    12 1.1299   0.01  SELECT
wp_posts
#      5 0x1425177AF56711BC   9.7609   2.3%    11 0.8874   0.32  SELECT
wp_posts
wp_term_relationships
#      6 0x94350EA2AB8AAC34   6.7514   1.6%   962 0.0070   0.04  UPDATE
wp_options
#      7 0x08D19B80DC07FF70   3.3142   0.8%   871 0.0038   0.00  UPDATE
wp_user_activity
#      8 0xB343B9825C05FD60   2.5817   0.6%    3 0.8606   0.00  UPDATE
dl_users
# MISC 0xMISC              21.0329   4.9% 39807 0.0005   0.0 <135
ITEMS>
```

Aquí puede observar que el 82 % del tiempo del servidor se empleó en ejecutar las peticiones #1 y #2: ahora sabemos que son las dos peticiones que hay que tratar de optimizar si deseamos mejorar el rendimiento del servidor. Es casi inútil mirar las otras peticiones, ya que tienen poco o ningún impacto en la carga del servidor.

Aquí tiene las estadísticas detalladas proporcionadas por la herramienta para el grupo de peticiones #1:

```
# Query 1: 0.48 QPS, 0.10x concurrency, ID 0xDD9512F54A8FDC75 at byte
```

9722374

Scores: V/M = 0.01

Time range: 2016-03-11 09:23:57 to 09:57:18

# Attribute	pct	total	min	max	avg	95%	stddev
-------------	-----	-------	-----	-----	-----	-----	--------

median							
--------	--	--	--	--	--	--	--

# Count	2	957					
---------	---	-----	--	--	--	--	--

# Exec time	47	204s	187ms	392ms	213ms	308ms	40ms
-------------	----	------	-------	-------	-------	-------	------

# Lock time	2	179ms	51us	116ms	187us	69us	4ms
-------------	---	-------	------	-------	-------	------	-----

# Rows sent	0	52	0	1	0.05	0.99	0.22
-------------	---	----	---	---	------	------	------

# Rows examine	0	3.74k	4	4	4	4	0
----------------	---	-------	---	---	---	---	---

# Rows affecte	0	0	0	0	0	0	0
----------------	---	---	---	---	---	---	---

# Rows read	0	3.74k	4	4	4	4	0
-------------	---	-------	---	---	---	---	---

# Bytes sent	0	1.47M	1.57k	1.81k	1.58k	1.69k	42.73
--------------	---	-------	-------	-------	-------	-------	-------

# Merge passes	0	0	0	0	0	0	0
----------------	---	---	---	---	---	---	---

# Tmp tables	0	0	0	0	0	0	0
--------------	---	---	---	---	---	---	---

# Tmp disk tbl	0	0	0	0	0	0	0
----------------	---	---	---	---	---	---	---

# Tmp tbl size	0	0	0	0	0	0	0
----------------	---	---	---	---	---	---	---

# Query size	2	510.78k	504	699	546.54	592.07	32.01
--------------	---	---------	-----	-----	--------	--------	-------

Boolean:

Full join 100% yes, 0% no

String:

Databases my_app

Hosts 54.179.163.217

Last errno 0

```

# Users          root
# Query_time distribution
# 1us
# 10us
# 100us
# 1ms
# 10ms
# 100ms
#####
# 1s
# 10s+
# Tables
# SHOW TABLE STATUS FROM `my_app` LIKE 'wp_redirection_items'\G
# SHOW CREATE TABLE `my_app`.`wp_redirection_items`\G
# SHOW TABLE STATUS FROM `my_app` LIKE 'wp_redirection_groups'\G
# SHOW CREATE TABLE `my_app`.`wp_redirection_groups`\G
# SHOW TABLE STATUS FROM `my_app` LIKE 'wp_redirection_modules'\G
# SHOW CREATE TABLE `my_app`.`wp_redirection_modules`\G
# EXPLAIN /*!50100 PARTITIONS*/
SELECT
wp_redirection_items.*,wp_redirection_groups.tracking,wp_redirection_
groups.position AS group_pos,wp_redirection_modules.id AS module_
id FROM wp_redirection_items INNER JOIN wp_redirection_groups ON
wp_redirection_groups.id=wp_redirection_items.group_id AND
wp_redirection_groups.status='enabled' INNER JOIN
wp_redirection_modules
ON wp_redirection_modules.id=wp_redirection_groups.module_id AND wp_
redirection_modules.type='wp' WHERE( wp_redirection_items.regex=1 OR
wp_
redirection_items.url='/bn/tips')\G

```

- ☐ Se proporcionan más detalles sobre pt-query-digest en el capítulo Herramientas de supervisión.

También podemos recuperar las peticiones que no utilizan índices con la opción

`log_queries_not_using_indexes`, sea cual sea su tiempo de ejecución. Sin embargo, esta opción no es siempre útil, ya que las peticiones sin índice no son necesariamente malas: si almacenamos en una tabla la lista de las comunidades autónomas con sus provincias, puede no ser necesario añadir un índice, ya que la tabla solo tendrá medio centenar de filas.

3. El registro de errores

El registro de errores (*error log*) es el único de los cuatro registros que se activa por defecto. Se reconoce a través de su extensión `.err`. El prefijo es por defecto el nombre del equipo. Su ubicación se obtiene con el comando `SHOW GLOBAL VARIABLES LIKE 'log_error'`. Puede cambiarse, así como el nombre del archivo, poniéndolo en la opción `log-error`. En caso de fallo o de comportamiento anormal, es la herramienta preferida del DBA en curso de investigación. Este registro contiene la información relativa a las advertencias y errores del servidor MySQL. Aquí encontraremos también los mensajes relacionados con los motores de almacenamiento o las funcionalidades avanzadas, como la replicación o el programador de eventos (*event scheduler*).

Uso del registro de errores

Elección de la ubicación del registro de errores en el archivo de configuración:

```
[mysqld]
log-error = /usr/local/mysql/logs/mysql-error.err
```

Visualización del contenido de un registro de errores:

```
$ tail mysql-error.err
121021 00:04:25 mysqld_safe Starting mysqld daemon with databases
from /home/daz/MySQL-5_6_7/data
121021 0:04:25 [Warning] TIMESTAMP with implicit DEFAULT value is
deprecated. Please use --explicit_defaults_for_timestamp server
```

```
option (see documentation for more details).
121021 0:04:25 [Note] Plugin 'FEDERATED' is disabled.
121021 0:04:25 InnoDB: The InnoDB memory heap is disabled
121021 0:04:25 InnoDB: Mutexes and rw_locks use GCC atomic builtins
121021 0:04:25 InnoDB: Compressed tables use zlib 1.2.3
121021 0:04:25 InnoDB: Using Linux native AIO
121021 0:04:25 InnoDB: CPU supports crc32 instructions
121021 0:04:25 InnoDB: Initializing buffer pool, size = 128.0M
121021 0:04:25 InnoDB: Completed initialization of buffer pool
121021 0:04:25 InnoDB: highest supported file format is Barracuda.
121021 0:04:25 InnoDB: 128 rollback segment(s) are active.
121021 0:04:25 InnoDB: Waiting for the background threads to start
121021 0:04:25 InnoDB: 1.2.7 started; log sequence number 10525020
121021 0:04:25 [Note] Server hostname (bind-address): '*'; port: 5605
121021 0:04:25 [Note] IPv6 is available.
121021 0:04:25 [Note] - '::' resolves to '::';
121021 0:04:25 [Note] Server socket created on IP: '::'.
121021 0:04:26 [Note] Event Scheduler: Loaded 0 events
121021 0:04:26 [Note] /home/daz/MySQL/bin/5.6.7/bin/mysqld: ready for
connections.
```

4. El registro general

El registro general (*general log*) registra los acontecimientos recibidos por `mysqld`, es decir, las peticiones válidas o no enviadas por los clientes y la información de conexión/desconexión de los clientes. Al igual que el registro de peticiones lentas, lo podemos activar y desactivar en caliente o en frío con la opción `general_log`.

También podemos almacenar la información, ya sea en un archivo o en una tabla, pero como para el registro de peticiones lentas, se debe siempre optar por usar un archivo en la medida de lo posible.

A diferencia del registro de peticiones lentas, que almacena las peticiones después de su ejecución, el registro general almacena las peticiones antes de su ejecución. Por lo tanto, el registro general no cuenta con ninguna información disponible en el registro de peticiones lentas. Solo sirve para mantener una constancia de las peticiones o conexiones con error.

Por lo tanto, es raro tener que activar el registro general, ya que puede llenar el disco con mucha rapidez y tener un impacto importante sobre el rendimiento.

Ejemplo de registro general

```
2016-05-04T13:35:23.707140Z    4 Connect  root@localhost on  using
Socket
2016-05-04T13:35:23.707594Z    4 Query    select @@version_comment
limit 1
2016-05-04T13:35:47.578209Z    4 Query    select * from my_app.t
2016-05-04T13:39:39.666551Z    4 Quit
2016-05-04T13:39:50.895276Z    5 Connect  javier@localhost
on using Socket
2016-05-04T13:39:50.895379Z    5 Connect  Access denied for user
'javier'@'localhost' (using password: NO)
```

5. Mejores prácticas

a. Configuración

Con un uso convencional de MySQL, la configuración estándar consiste en:

- Activar el registro binario. Es necesario para restaurar los datos modificados desde la última copia. Además, será obligatorio para la replicación.
- Activar el registro de peticiones lentas. Aquí encontraremos las peticiones que se deben optimizar.
- No activar el registro general, salvo necesidades específicas (auditoría, por ejemplo).

b. Supervisar el uso del disco

Implemente procedimientos para supervisar los espacios de almacenamiento. En función de la actividad del servidor (volumen y carga), algunos archivos pueden crecer de forma rápida. Este es concretamente el caso del registro general.

c. Impacto sobre el rendimiento

La activación de los registros tiene un impacto negativo sobre el rendimiento, que no es fácil de evaluar, ya que depende de nuestra aplicación y de los registros activados. Nos corresponde encontrar el equilibrio entre lo que aportan y las prestaciones que requerimos. Observe, sin embargo, el registro en las tablas del sistema (registro general y registro de peticiones lentas), ya que es mucho más pesado que el registro en archivos.



El modo SQL

El comportamiento predeterminado del servidor MySQL es bastante permisivo, en especial en lo que respecta a la coherencia de los datos. La directiva es en cierto modo: «Los datos recibidos son coherentes; por lo tanto, no es necesario comprobar nada», lo que en realidad no es siempre el caso. Si su aplicación comprueba todos los datos, se dispensa al servidor de realizar esta tarea y se evita así que disminuya el rendimiento. Sin embargo, si este no es el caso o si hay otra manera de acceder a los datos que no sea a partir de la aplicación, se debe utilizar el servidor para hacer las comprobaciones. Por ejemplo, si una cadena de caracteres es demasiado larga para insertarse en una columna, el servidor ejecutará la inserción truncando la cadena de caracteres y devolviendo una advertencia (*warning*) al cliente en la que dirá que los datos han sido truncados para que la inserción tenga éxito. Esto plantea dos problemas principales para la mayoría de las aplicaciones. En primer lugar, el dato presente en la tabla no es el enviado por el cliente, la coherencia de datos no está garantizada. En segundo lugar, nuestra experiencia nos ha demostrado que muy pocos desarrolladores tratan los avisos enviados por el servidor, lo cual quiere decir, por último, que se ha insertado un dato incorrecto en la base y que nadie está al corriente. La opción `sql_mode` que apareció en la versión 4.1 permite, entre otras cosas, resolver este tipo de problema. Esta opción puede cambiar el comportamiento del servidor:

- Reforzando la coherencia, impidiendo la inserción implícita de datos inválidos.
- Haciendo que el código SQL sea portátil.
- Y proporcionando un comportamiento del servidor similar al de otros SGBD.

1. Los modos habituales

En MySQL 5.6, el valor por defecto de la variable `sql_mode` es

`NO_ENGINE_SUBSTITUTION` (el comportamiento de este modo se explica un poco

más abajo).

```
mysql> SHOW SESSION VARIABLES LIKE 'sql_mode';
```

Variable_name	Value
sql_mode	NO_ENGINE_SUBSTITUTION

La tabla `char3tinyint` contiene dos columnas: `c` como `CHAR(3)`, cadena que contiene tres caracteres como máximo, e `i` en `TINYINT`, que solo acepta los números enteros de -128 a 127.

```
mysql> SHOW CREATE TABLE char3tinyint \G
```

```
***** 1. row *****
```

```
Table: char3tinyint
```

```
Create Table: CREATE TABLE `char3tinyint` (
```

```
  `c` char(3) DEFAULT NULL,
```

```
  `i` tinyint(4) DEFAULT NULL
```

```
);
```

```
mysql> INSERT INTO char3tinyint VALUES('abc',127);
```

```
Query OK, 1 row affected (0,00 sec)
```

```
mysql> SELECT * FROM char3tinyint;
```

c	i
abc	127

La cadena de caracteres LeMUG es demasiado larga para ser almacenada en su totalidad en

una columna de tipo CHAR (3) ; el modo SQL por defecto es permisivo, el servidor almacenará lo que pueda, es decir, los tres primeros caracteres, e ignorará los dos siguientes. Por lo tanto, trunca la cadena de caracteres. En la segunda columna, el entero 2008 supera el límite máximo del tipo TINYINT, que vale 127. Al encontrarse por encima de este límite, se almacenará el valor máximo, es decir, 127 (si el número fuera inferior a -128, ese es el último número que habría sido almacenado). Para cada una de estas columnas, el servidor devuelve un aviso que describe la operación que ha realizado de manera implícita.

```
mysql> INSERT INTO char3tinyint VALUES('LeMUG',2008);
Query OK, 1 row affected, 2 warnings (0,00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level   | Code | Message                                     |
+-----+-----+-----+
| Warning | 1265 | Data truncated for column 'c' at row 1    |
| Warning | 1264 | Out of range value for column 'i' at row 1 |
+-----+-----+-----+

mysql> SELECT * FROM char3tinyint;
+-----+-----+
| c     | i     |
+-----+-----+
| abc   | 127   |
| LeM   | 127   |
+-----+-----+
```

El modo SQL pasa a ser estricto para la sesión mediante al valor STRICT_ALL_TABLES. Los datos inválidos son rechazados y se devuelve un error, ya que la inserción no se realiza.

```
mysql> SET SESSION
sql_mode=CONCAT(@@sql_mode,',','STRICT_ALL_TABLES');
```

Query OK, 0 rows affected (0,00 sec)

```
mysql [localhost] {msandbox} (test) > SHOW SESSION VARIABLES LIKE  
'sql_mode';
```

```
+-----+-----+  
| Variable_name | Value |  
+-----+-----+  
| sql_mode      | STRICT_ALL_TABLES,NO_ENGINE_SUBSTITUTION |  
+-----+-----+  
1 row in set (0,00 sec)
```

```
mysql> INSERT char3tinyint VALUES('LeMUG',2008);  
ERROR 1406 (22001): Data too long for column 'c' at row 1
```

```
mysql> SELECT * FROM char3tinyint;
```

```
+-----+-----+  
| c      | i      |  
+-----+-----+  
| abc    | 127    |  
| LeM    | 127    |  
+-----+-----+
```

Entre las variables que permiten mejorar la coherencia de los datos, se encuentra también:

- **NO_ENGINE_SUBSTITUTION**: Durante un comando `CREATE TABLE` o `ALTER TABLE`, si no existe el motor de almacenamiento solicitado en la creación o modificación de una tabla, el servidor devuelve un error y no ejecuta, por tanto, el comando. En caso contrario, se devuelve una advertencia (*warning*) y la tabla se crea o modifica con el motor de almacenamiento por defecto.

En el siguiente ejemplo, el motor `ARIA` es desconocido en este servidor. Si se suprime el modo `SQL NO_ENGINE_SUBSTITUTION`, el servidor crea la tabla con el motor por defecto, `InnoDB`.

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

```
mysql> SHOW SESSION VARIABLES LIKE 'sql_mode';
```

Variable_name	Value
sql_mode	

```
mysql> CREATE TABLE t_ARIA(i int)ENGINE = ARIA;
```

```
mysql> SHOW WARNINGS;
```

Level	Code	Message
Warning	1286	Unknown table engine 'ARIA'
Warning	1266	Using storage engine InnoDB for table 't_ARIA'

```
mysql> SHOW CREATE TABLE t_ARIA \G
```

```
***** 1. row *****
```

```
Table: t_ARIA
```

```
Create Table: CREATE TABLE `t_ARIA` (  
  `i` int(11) DEFAULT NULL  
  ) ENGINE=InnoDB;
```

```
mysql> SET SESSION sql_mode='NO_ENGINE_SUBSTITUTION';
```

```
mysql> CREATE TABLE t_ARIA2(i int)ENGINE = ARIA;
```

- **STRICT_ALL_TABLES** y **STRICT_TRANS_TABLES**: si el modo SQL contiene uno de estos dos parámetros, se habla de modo estricto. Los datos inválidos son rechazados y se devuelve un error porque la inserción no se realiza. Con una u otra de las opciones, el comportamiento del servidor es idéntico en las tablas que utilizan un motor de almacenamiento transaccional, como es el caso con InnoDB. En las tablas que tienen un motor no transaccional (MyISAM, por ejemplo), su comportamiento difiere en inserciones múltiples, es decir, en caso de inserción de varios registros en una única petición INSERT.

Con el modo `STRICT_ALL_TABLES`, el servidor devuelve un error en el primer dato inválido. El motor no es transaccional, los datos ya insertados se mantienen, lo que puede causar problemas de coherencia. Para evitar esto, se puede escribir las peticiones `INSERT` con un solo registro a la vez. Mientras que, en el mismo contexto, con el modo `STRICT_TRANS_TABLES`, los datos inválidos se insertan y se convierten en el valor válido más cercano. El servidor genera, sin embargo, un aviso.

Cabe señalar que, si los dos modos están activados, el comportamiento de modo `STRICT_ALL_TABLES` es el que tiene la preferencia.

```
mysql> SET SESSION sql_mode='STRICT_ALL_TABLES';
```

```
mysql> INSERT INTO t_myisam VALUES ('a',100),('ab',200);
ERROR 1264 (22003): Out of range value for column 'i' at row 2
```

```
mysql> SELECT * FROM t_myisam;
```

+	-----+	-----+
	c	i
+	-----+	-----+
	a	100
+	-----+	-----+

```
mysql> SET SESSION sql_mode='STRICT_TRANS_TABLES';
```

```
mysql> INSERT INTO t_myisam VALUES ('b',111),('ab',200);
Query OK, 2 rows affected, 1 warning (0,00 sec)
Records: 2 Duplicates: 0 Warnings: 1
```

```
mysql> SHOW WARNINGS;
```

+	-----+	-----+	-----+
	Level	Code	Message
+	-----+	-----+	-----+
	Warning	1264	Out of range value for column 'i' at row 2
+	-----+	-----+	-----+

```
mysql> SELECT * FROM t_myisam;
```



```

+-----+-----+
| c | i |
+-----+-----+
| a | 100 |
| b | 111 |
| ab | 127 |
+-----+-----+

```

```
mysql> SET SESSION sql_mode='STRICT_TRANS_TABLES,
STRICT_ALL_TABLES';
```

```
mysql> INSERT INTO t_myisam VALUES ('abc',1),('abcdef',50);
ERROR 1406 (22001): Data too long for column 'c' at row 2
```

```
mysql> SELECT * FROM t_myisam;
```

```

+-----+-----+
| c | i |
+-----+-----+
| a | 100 |
| b | 111 |
| ab | 127 |
| abc | 1 |
+-----+-----+

```

```
4 row in set (0,00 sec)
```

- `ERROR_FOR_DIVISION_BY_ZERO`: devuelve un aviso cuando la actualización de un campo resulta en una división por 0 o módulo por 0.

```
mysql> SET SESSION sql_mode='ERROR_FOR_DIVISION_BY_ZERO';
```

```
mysql> INSERT mi_tabla VALUES (1/0);
```

```
Query OK, 1 row affected, 1 warning (0,00 sec)
```

```
mysql> SHOW WARNINGS;
```

```

+-----+-----+-----+
| Level | Code | Message |

```

```
+-----+-----+-----+
| Error | 1365 | Division by 0 |
+-----+-----+-----+
```

- NO_ZERO_DATE: devuelve un aviso si la fecha escrita en la tabla es cero ('0000-00-00', '0000/00/00'...). Combinado con los modos STRICT_TRANS_TABLES o STRICT_ALL_TABLES, el dato modificado (INSERT o UPDATE) no se actualiza y se devuelve un error.

```
mysql> SET SESSION sql_mode='NO_ZERO_DATE,STRICT_TRANS_TABLES';
```

```
mysql> INSERT INTO mi_tabla values('0000-00-00 00:00:00');
ERROR 1292 (22007): Incorrect datetime value: '0000-00-00
00:00:00' for column 'd' at row 1
```

- NO_ZERO_IN_DATE: devuelve un aviso si el año, mes o el día son cero. Combinado con los modos STRICT_TRANS_TABLES o STRICT_ALL_TABLES, el dato modificado (INSERT o UPDATE) no se actualiza y se devuelve un error. Cabe señalar que el valor cero ('0000-00-00', '0000/00/00'...) es aceptado.

```
mysql> SET SESSION sql_mode ='STRICT_ALL_TABLES,NO_ZERO_IN_DATE';
```

```
mysql> INSERT INTO t1 VALUES ('0000-00-00');
Query OK, 1 row affected (0,00 sec)
```

```
mysql> INSERT INTO t1 VALUES ('0000-00-01');
ERROR 1292 (22007): Incorrect datetime value: '0000-00-01' for
column 'd' at row 1
```

```
mysql> INSERT INTO t1 VALUES ('0000-01-00');
ERROR 1292 (22007): Incorrect datetime value: '0000-01-00' for
column 'd' at row 1
```

```
mysql> INSERT INTO t1 VALUES ('0001-00-00');  
ERROR 1292 (22007): Incorrect datetime value: '0001-00-00' for  
column 'd' at row 1
```

```
mysql> INSERT INTO t1 VALUES ('1974-11-12');  
Query OK, 1 row affected (0,00 sec)
```

- NO_AUTO_CREATE_USER: se prohíbe la creación de cuentas de usuarios sin contraseñas con el comando GRANT. Sin embargo, podemos crear un usuario sin contraseña, con el comando CREATE USER.

```
mysql> GRANT USAGE ON *.* TO 'IT';  
ERROR 1133 (42000): Can't find any matching row in the user table
```

```
mysql> GRANT USAGE ON *.* TO 'IT' IDENTIFIED BY 'pal2cont3';  
Query OK, 0 rows affected (0,00 sec)
```

```
mysql> CREATE USER 'viadeo';  
Query OK, 0 rows affected (0,00 sec)
```

- ☐ INSERT IGNORE / UPDATE IGNORE: las peticiones INSERT IGNORE y UPDATE IGNORE permiten evitar los modos SQL que refuerzan la coherencia de los datos, en modo estricto. La modificación de datos será posible, pero el servidor devuelve un aviso. En otras palabras, con la cláusula IGNORE el funcionamiento del servidor es similar al `mode_sql=''`.

2. Las combinaciones de modos

Podemos combinar varios modos SQL. Basta con ponerlos uno a continuación del otro,

separados por comas y sin espacios, todo en un único comando. Esta operación puede hacerse en caliente, para la sesión o de manera global. Sin embargo, como pasa con otras opciones, es preferible definirlos en el archivo de configuración.

```
[mysqld]
sql_mode=STRICT_TRANS_TABLES,STRICT_ALL_TABLES,NO_ZERO_IN_DATE,
NO_ZERO_DATE, NO_ENGINE_SUBSTITUTION
```

La opción `sql_mode` también puede tomar como valor combinaciones de modos predefinidos. Para obtener un comportamiento estricto del servidor, utilice el modo SQL `TRADITIONAL`. Es equivalente a los modos `'STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION'`. Para tener un código lo más cercano posible al estándar SQL ANSI, utilice el valor ANSI equivalente a `'REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE'`.

3. Modo SQL por defecto

El valor por defecto del modo SQL cambia según las versiones. Debemos estar atentos durante una actualización de versión mayor, ya que las peticiones válidas con una versión antigua pueden no serlo con una nueva versión. Así, con MySQL 5.5, el modo SQL por defecto es:

```
mysql> show global variables like 'sql_mode'\G
***** 1. row *****
Variable_name: sql_mode
Value:
```

Con MySQL 5.6:

```
mysql> show global variables like 'sql_mode'\G
***** 1. row *****
Variable_name: sql_mode
Value: NO_ENGINE_SUBSTITUTION
```

Y con MySQL 5.7:

```
sb1> show global variables like 'sql_mode'\G
***** 1. row *****
Variable_name: sql_mode
Value: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_
DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_
SUBSTITUTION
```

Otros modos SQL permiten obtener un comportamiento del servidor similar al de las antiguas versiones de MySQL (MYSQL323 y MYSQL40) e incluso otros SGBD (MSSQL / DB2 / ORACLE / MAXDB / POSTGRESQL). Cabe señalar que la compatibilidad no es segura al 100 %.

Encontraremos la lista completa de los valores de la opción `sql_mode` en la documentación de MySQL:<http://dev.mysql.com/doc/refman/5.7/en/sql-mode.html>

```
mysql> SET SESSION sql_mode='TRADITIONAL';
Query OK, 0 rows affected (0,00 sec)

mysqlt> SHOW VARIABLES LIKE 'sql_mode'\G
***** 1. row *****
Variable_name: sql_mode
Value:
STRICT_TRANS_TABLES,STRICT_ALL_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,
ERROR_FOR_DIVISION_BY_ZERO,TRADITIONAL,NO_AUTO_CREATE_USER,
NO_ENGINE_SUBSTITUTION
```

1 row in set (0,00 sec)

- ☐ Modo SQL y particiones: una vez creadas las particiones, no se recomienda cambiar el valor de la opción `sql_mode` porque corremos el riesgo de corromper o perder los datos de las tablas particionadas. Si además replicamos estas tablas, el servidor maestro y el servidor esclavo deben tener el mismo modo SQL.

Otros parámetros

1. Parámetros MyISAM

El único verdadero parámetro importante para las tablas MyISAM es el tamaño de la caché de índice, comandado por la variable `key_buffer_size`.

Para encontrar un buen tamaño, deberá fijarse en los valores de tres de las variables de estado del resultado del comando:

```
mysql> SHOW GLOBAL STATUS LIKE 'Key_%';
```

Las dos primeras variables que se deben considerar son `key_reads` y `Key_read_requests`. `Key_reads` indica el número de peticiones de lecturas de índice que no han podido ser satisfechas por la caché, y `Key_read_requests` indica el número de lecturas en el índice. Si calculamos el porcentaje de utilización de la caché usando la fórmula $(1 - \text{Key_reads} / \text{Key_read_requests}) \times 100$, su caché debería estar bien configurada si su nivel es cercano al 100 %. Si la tasa es baja, la caché es sin duda demasiado pequeña y podemos aumentar el valor de `key_buffer_size`.

La tercera variable que debe comprobarse es `key_blocks_unused`, que nos da el número de bloques disponibles en la caché. Si la tasa de utilización es baja y el número de bloques disponibles es también bajo, su caché es en realidad demasiado pequeña.

☐ Debe saber que el hecho de prever una caché sobredimensionada no penaliza,

ya que la memoria solo se adjudicará en caso de necesidad, a diferencia de la caché de peticiones.

No existe ninguna opción para almacenar en caché los datos de tablas MySQL. Solo el sistema operativo pone en caché los datos, lo que es mucho menos eficaz que una caché especializada, como la que existe para los índices.

2. Caché de peticiones

a. Función de la caché

MySQL mantiene un conjunto de resultados de peticiones de tipo `SELECT` en una estructura en la memoria correspondiente llamada caché de peticiones. Cuando un `SELECT` se ejecuta, el servidor almacena el resultado en la caché, de modo que, si un cliente solicita la misma petición en un tiempo lo bastante corto para que las tablas no hayan cambiado, el servidor pueda remitir directamente el resultado que ha conservado en la memoria, en lugar de pasar por todas las fases de descomposición, optimización y ejecución de la petición.

La caché de peticiones puede, en teoría, reducir de forma considerable la carga de un servidor MySQL evitando la ejecución de muchas peticiones.

Tenga en cuenta que la caché de peticiones tiene un coste en recursos, ya que, por un lado bloquea una parte de la memoria RAM del sistema y, por otro, el mecanismo de caché de las peticiones provoca operaciones adicionales para todas las peticiones, incluso aquellas que no pueden beneficiarse de la caché. Pero, sobre todo, el principal problema de la caché de peticiones es un bloqueo global que hace que todos los accesos sean secuenciales. Si el servidor ejecuta pocas peticiones en paralelo, esta serialización no suele ser un problema. Pero si son muchos los accesos concurrentes, la serialización de la caché de peticiones es tan importante que es mejor desactivar la caché.

b. Activación de la caché

Por defecto, la caché de peticiones está desactivada desde MySQL 5.6. Para comprobarlo, basta con mirar el valor de la variable `query_cache_type`:

|


```
mysql> SHOW VARIABLES LIKE 'query_cache_type';
```

Variable_name	Value
query_cache_type	OFF

Esta variable puede tener tres valores diferentes:

- ON: el servidor intenta poner en caché todas las peticiones SELECT, salvo las que incluyen las palabras-clave SQL_NO_CACHE (SELECT SQL_NO_CACHE . . .).
- OFF: no se pone ninguna petición en la caché.
- DEMAND: el servidor solo intenta poner en caché las peticiones SELECT con la palabra clave SQL_CACHE (SELECT SQL_CACHE . . .).

c. Peticiones excluidas de la caché

No todas las peticiones SELECT pueden ponerse en caché. Al ejecutar una petición, si el servidor observa que la petición tiene una propiedad que no la hace apta para ponerse en caché, señala que esta no puede ser puesta en caché. ¿Cuáles son los criterios que impiden poner en caché una petición?

En primer lugar, solo las peticiones SELECT deterministas, es decir, capaces de dar el mismo resultado si se ejecutan varias veces con el mismo juego de datos, se almacenarán en la caché. Así, por ejemplo, la petición SELECT * FROM t dará siempre el mismo resultado mientras la tabla t no sea modificada, mientras que la petición SELECT CURRENT_TIMESTAMP() dará un resultado que variará a lo largo del tiempo. MySQL podrá poner en caché la primera, pero no la segunda. Cualquier llamada a una función no determinista como NOW(), RAND(), CURRENT_DATE(), UUID(), CURRENT_USER() y algunas otras impide la puesta en caché.

Por otra parte, las peticiones SELECT con el uso de tablas temporales, variables de usuario o que se encuentran en procedimientos almacenados, vistas, triggers o eventos no se ponen en caché.

Por último, todas las peticiones que llevan la palabra clave `SQL_NO_CACHE` quedan excluidas de la caché de peticiones.

- ☐ Cuando desee medir el tiempo de ejecución de una petición para probar optimizaciones, piense en añadir la palabra clave `SQL_NO_CACHE`. De lo contrario, sus peticiones pueden ser servidas por la caché: las respuestas siempre serán instantáneas y no será capaz de determinar si un cambio es beneficioso o no.

d. Llamada a un elemento de caché

Cuando la caché de peticiones está activada, no se requiere ninguna acción por parte del usuario o administrador para aprovecharla. Una vez que el servidor recibe una petición `SELECT`, determina si la petición y su resultado se encuentran ya en la caché. Esta búsqueda se efectúa pasando por una función hash el texto de la petición, así como algunos parámetros adicionales, y comparando el valor obtenido con los valores existentes en la caché. La función hash tiene en cuenta las mayúsculas y minúsculas, lo que significa que dos peticiones equivalentes de forma semántica se consideran distintas por una simple diferencia de minúscula y mayúscula (los espacios se tienen también en cuenta). Así, la caché de peticiones considera que todas las peticiones siguientes son diferentes:

- `SELECT id, nombre FROM t`
- `select id, nombre FROM t`
- `SELECT nombre, id FROM t`

Simple convenciones de escritura entre los desarrolladores pueden ahorrar en rendimiento, evitando al servidor considerar como diferentes las peticiones que son idénticas desde el punto de vista funcional.

Si el servidor encuentra la petición en la caché, el resultado se devuelve directamente sin pasar por las distintas fases de ejecución.

Si el servidor no encuentra la petición en la caché, la petición se realiza de forma normal. Una vez ejecutada, la petición será escrita en la caché con su resultado, salvo que durante la ejecución MySQL marque la petición como no apta para su puesta en caché.

e. Actualización de la caché

El servidor MySQL actualiza la caché de forma automática, sin ninguna intervención externa. Además del almacenamiento de un nuevo resultado de una petición, pueden ocurrir otros dos acontecimientos: anulación o borrado.

Una anulación se produce cuando una modificación se realiza en una tabla: actualización, adición o eliminación de uno o varios datos, y también la modificación de la estructura de la tabla. En este caso, todas las peticiones que implican las tablas modificadas serán borradas de la caché. Este método tiene la ventaja de ser rápido y por defecto ser claro: en efecto, es perfectamente posible que una actualización de datos en una tabla no cambie los resultados de todas las peticiones que intervienen en ella. Aquí, la eficacia no tiene en cuenta la exactitud.

Un borrado tiene lugar cuando el servidor intenta almacenar una nueva entrada en la caché, pero la memoria disponible es insuficiente. Los resultados donde la fecha de la última petición son anteriores se eliminan para dar cabida a nuevos resultados.

f. Fragmentación

Una descripción simplificada de los mecanismos de asignación de la memoria en la puesta en caché permitirá comprender por qué la caché se puede fragmentar. Esta fragmentación se vuelve molesta cuando el servidor está obligado a borrar las entradas, ya que no logra encontrar un número suficiente de memoria contigua para registrar nuevos valores.

El servidor almacena los resultados de la caché a medida que se recuperan, lo que es más eficaz que colocar estos resultados en una caché de memoria temporal para transferirla a la caché de peticiones.

Este método tiene, sin embargo, un inconveniente: el servidor no conoce de antemano el número de resultados que se necesita guardar en la caché, y no puede determinar la cantidad de memoria necesaria. Por eso, la memoria de caché se asigna en pequeños bloques contiguos de un tamaño predefinido.

Cuando el resultado está grabado por completo, la memoria restante del último bloque se recupera y vuelve a estar disponible para almacenar otros resultados. Pero cuando varios resultados se registran al mismo tiempo, entre todos los bloques que se utilizan de forma parcial, solo aquellos cuya posición es contigua a los bloques libres pueden recuperarse.

Entre los bloques de datos, pueden existir bloques de espacio libre, aunque su tamaño es demasiado pequeño para poder reutilizarlos. Estos bloques libres se pierden de forma irremediable. La peor situación se alcanza cuando entre cada bloque de datos se encuentra un bloque libre pero inutilizable. En este caso, la caché está muy fragmentada.

A continuación, se presentará un método para detectar una gran fragmentación de una caché.

g. Herramienta de caché

Como se indicó antes, la caché de peticiones puede ser muy ventajosa en términos de rendimiento, pero existen muchos casos en que la caché puede degradar el rendimiento. Dejando a un lado la serialización de acceso a la que ya hemos hecho referencia, debemos también considerar los otros costes ocultos de la caché de peticiones.

En primer lugar, para todas las peticiones `SELECT`, el servidor comprueba si la petición se encuentra o no en la caché. Esta verificación se efectúa para todas las peticiones, incluso para aquellas que tienen un elemento que las hace no adecuadas para la puesta en caché. El coste de esta verificación es muy bajo, dado que se trata solo de buscar si un elemento está presente en una tabla de control en la memoria.

En segundo lugar, cada petición `SELECT` candidata a ponerse en caché requiere operaciones específicas. Si hay suficiente espacio en la caché para almacenar los resultados de la petición, la operación es rápida. Pero si hay que borrar datos de la caché para recuperar espacio, la operación puede ser mucho más lenta.

Por último, todas las peticiones de modificación se ven ralentizadas, ya que hay que eliminar todas las entradas de caché correspondientes en las tablas modificadas.

Como se ve, si la aplicación sufre muchas lecturas idénticas, la caché resultará muy útil. Sin embargo, la caché de peticiones puede perder su interés e incluso ser perjudicial si se encuentra en uno de los casos siguientes:

- Existen muchas escrituras, provocando un gran número de invalidaciones.
- Las lecturas no se repiten varias veces.
- La mayoría de las lecturas no pueden ponerse en caché.

Debemos, pues, estudiar la información que pone a disposición MySQL sobre el uso de la caché para estimar si es preferible o no activarla.

h. Parámetros asociados a la caché

La caché se configura mediante cinco variables principales:

- `query_cache_size`: se trata del tamaño de la caché. MySQL asigna siempre una dimensión múltiplo de 1024, aunque el tamaño real de la caché no es necesariamente la misma que hemos solicitado. Tenga en cuenta que la totalidad de la memoria definida por este parámetro se asigna de forma inmediata.
- `query_cache_type`: indica si la caché está activa o no. Esta opción se detalló antes.
- `query_cache_limit`: si un resultado de una petición tiene un tamaño superior a `query_cache_limit`, este no se almacenará en la caché.
- `query_cache_min_res_unit`: la memoria de caché se asigna por bloques de un tamaño mínimo determinado por esta variable. Si los resultados por almacenar son muy compactos, un valor demasiado elevado puede conducir a perder mucho espacio en memoria.
- `query_cache_wlock_invalidate`: cuando un cliente ha bloqueado una tabla para escritura, por norma, otro cliente no podrá hacer lecturas sobre la tabla. Cuando el valor de esta opción está en `OFF` (valor por defecto), el servidor puede devolver un resultado de una petición procedente de la caché aunque la tabla esté bloqueada. Por lo general, no es necesario modificar esta variable.

Todas estas variables pueden modificarse de manera interactiva, por ejemplo:

```
mysql> SET GLOBAL query_cache_size=10*1024*1024;
```

O modificarse en el archivo de configuración `my.cnf` o `my.ini`, por ejemplo:

```
[mysqld]  
query_cache_size = 10M
```

En el caso de una modificación interactiva, los cambios se tienen en cuenta de forma inmediata, pero se pierden al reiniciar el servidor, mientras que para las modificaciones hechas en el archivo de configuración se requiere un reinicio del servidor.

i. Configuración del tamaño de la caché

Para determinar si el tamaño de la caché se adapta a la aplicación, es preciso examinar diversas variables de estado para obtener la máxima información posible sobre la manera en que la caché se utiliza. Con todas estas indicaciones, podremos ajustar el tamaño de la caché.

Se puede acceder a todas las variables de estado referentes a la caché de peticiones de la siguiente manera:

```
mysql> SHOW GLOBAL STATUS LIKE 'Qcache%';
+-----+-----+
| Variable_name      | Value      |
+-----+-----+
| Qcache_free_blocks | 3872       |
| Qcache_free_memory | 258067840  |
| Qcache_hits        | 15245675   |
| Qcache_inserts     | 11487035   |
| Qcache_lowmem_prunes | 0          |
| Qcache_not_cached  | 2440772    |
| Qcache_queries_in_cache | 8189      |
| Qcache_total_blocks | 20420      |
+-----+-----+
```

La primera variable por examinar es `Qcache_free_memory`, que indica el espacio libre de la caché. Un valor pequeño en comparación con el tamaño de la caché puede ser indicio de una caché demasiado pequeña; un valor cercano al tamaño de la caché indica una caché sobredimensionada.

La segunda es `Qcache_lowmem_prunes`, que devuelve el número de entradas suprimidas en la caché por falta de espacio. Un valor elevado es siempre sospechoso, y puede deberse bien a una caché demasiado pequeña, bien a una caché fragmentada.

La tercera es `Qcache_hits`, correspondiente al número de veces que el resultado de una petición pudo ser recuperado de la caché. Si el tamaño de la caché es pequeño, las peticiones insertadas en la caché serán sin duda muy pronto suprimidas por falta de espacio, y pocos resultados podrán recuperarse de la caché.

Estos tres parámetros deben permitirnos encontrar un tamaño de caché adecuado a la carga. Tenga en cuenta que 256 MB es un tamaño que debemos intentar no sobrepasar. Aunque nada impide en teoría utilizar un tamaño mayor, en la práctica, cuando la caché excede los 256 MB, las anulaciones o eliminaciones se vuelven largas y causan problemas de rendimiento.

j. Determinación de la eficiencia de la caché

A partir de las variables de estado, podemos calcular dos tipos de tasas para determinar la eficacia de la caché. Para estos cálculos, necesitaremos de una variable de condición adicional denominada `Com_select`, que devuelve el número de peticiones `SELECT` que fueron ejecutadas por el servidor y no incluidas por la caché de peticiones:

```
mysql> SHOW GLOBAL STATUS LIKE 'Com_select';
```

```
+-----+-----+
| Variable_name | Value      |
+-----+-----+
| Com_select    | 13882865   |
+-----+-----+
```

La tasa de aciertos, que se obtiene mediante la fórmula $100 * \frac{Qcache_hits}{Qcache_hits + Com_select}$, nos da la tasa de peticiones servidas por la caché en relación con el número total de `SELECT` solicitados al servidor. Cuanto mayor sea la tasa, más interesante será el uso de la caché. Sin embargo, no es posible definir de manera absoluta lo que es una buena o mala tasa de hits.

En algunos casos, el 70 % es perfectamente aceptable, mientras que en otros, hay que apuntar al 99,9 %. Por lo tanto, evite permanecer centrado por completo en la mejora de este tipo y no confíe en las herramientas que pueda encontrar en Internet y que indican de manera arbitraria si la tasa de hits es buena o no.

La tasa de inserción en la caché ($100 * Qcache_inserts / Com_select$) indica la proporción de peticiones SELECT que se encuentran en la caché cuando el servidor las ejecuta.

Si la tasa de inserción y la tasa de hits son elevadas, la caché es sin duda eficaz y está bien configurada.

Si la tasa de inserción y la tasa de hits es baja, la caché es sin duda poco eficaz. Podemos entonces intentar desactivarla para ver si los resultados son mejores sin caché.

Si las dos tasas tienen valores promedio con uno alto y otro bajo, estos indicadores no son concluyentes.

En cualquier caso, es interesante monitorizar la información absoluta dada por `Qcache_hits`, y en menor medida por `Qcache_inserts`.

☐ Recuerde que, al calcular un porcentaje (hits o inserciones), perdemos información esencial: la del número absoluto de operaciones realizadas por la caché. ¿Qué es más útil? ¿Una caché con una tasa de hits del 60 % pero que sirva 1000 peticiones o una caché con una tasa de hits del 99 % pero que solo sirva una petición por segundo? Configurar la caché de peticiones con la única información de tipo hits y la tasa de inserción no siempre permite concluir en forma segura.

☐ A menudo, la mejor manera de saber si la caché de peticiones es útil o no consiste en examinar el tiempo de respuesta de las peticiones con y sin la caché de peticiones.

k. Reducción de la fragmentación

Dos variables nos permitirán estimar la fragmentación de la caché: se trata de `Qcache_free_blocks`, que indica el número de bloques libres, y `Qcache_total_blocks`, que da el número total de bloques de la caché.

Si el resultado de `Qcache_free_blocks/Qcache_total_blocks` se aproxima a 1/2, es decir, que entre cada bloque de datos hay un bloque libre, la caché está

muy fragmentada.

Si nuestra caché está fragmentada y la variable `Qcache_lowmem_prunes` tiene un valor alto, entonces el servidor está obligado a eliminar entradas de la caché a causa de la fragmentación, lo que no es recomendable. En ese caso, podemos defragmentar la caché con el comando `FLUSH QUERY CACHE` o cambiar el valor de `query_cache_min_res_unit`.

La defragmentación de la caché puede llevar tiempo, lo que también explica por qué es preferible dimensionar la caché en 256 MB como máximo.

A pesar de su nombre, el comando `FLUSH QUERY CACHE` no vacía el contenido de la caché. El comando `RESET QUERY CACHE` se encarga de dicha operación.

Para encontrar un buen tamaño para `query_cache_min_res_unit`, podemos calcular el tamaño promedio de una entrada en la caché con la fórmula $(\text{query_cache_size} - \text{Qcache_free_memory}) / \text{Qcache_queries_in_cache}$. Si tenemos una mezcla de resultados de pequeño tamaño y resultados de gran tamaño, este valor promedio no será sin duda de gran ayuda. Más vale entonces, por ejemplo, no poner en caché los resultados de gran tamaño, seleccionando un valor bajo de `query_cache_limit` o poner la variable `query_cache_type` en `DEMAND` para seleccionar las peticiones que se pondrán en caché.

3. Otras variables

a. Número de conexiones simultáneas

MySQL utiliza la variable `max_connections` para limitar el número de conexiones simultáneas al servidor.

Dos indicadores pueden avisarnos cuando este valor sea demasiado bajo: la variable de estado `max_used_connections` vale o está cerca de `max_connections` o el servidor rechaza las conexiones con el error `Too Many connections`.

Sin embargo, es preferible reflexionar sobre las posibles consecuencias de un aumento del valor de este parámetro, ya que su objetivo es impedir que el servidor se derrumbe bajo una carga demasiado importante. Si el valor de `max_connections` necesario para evitar el

error `Too many connections` resulta tan alto que el rendimiento de todo el servidor se degrada, debemos sin duda pensar en otras modificaciones (replicación, cambio de equipo, cambio de software...).

- MySQL mantiene una conexión disponible de forma local para el usuario `root` en caso de problema. Por eso `max_connections` tiene por defecto el valor 151 y no 150, autorizando de forma efectiva 150 conexiones a aplicaciones.

b. Cachés de tablas

Estas cachés almacenan información sobre las tablas, lo que permite, cuando el servidor necesita utilizar una tabla, acelerar el acceso.

A partir de MySQL 5.1, la caché se divide en dos partes: una caché de tablas abiertas (`table_open_cache`) y una caché de definición de tabla (`table_definition_cache`). El ajuste de estos dos parámetros se realiza de forma similar a como se hace al utilizar las variables de estado siguientes:

- `Opened_tables` para `table_open_cache`. Si el valor de esta variable aumenta constantemente, es recomendable aumentar el valor de `table_open_cache`.
- `Opened_table_definitions` para `table_definition_cache`, bajo el mismo principio. En general, lo más sencillo es configurar `table_definition_cache` con un valor ligeramente más alto que el número de tablas existentes en el servidor.

Si tenemos varias decenas de miles de tablas en el sistema, no es recomendable utilizar valores más altos que 10000 para estas dos cachés, ya que pueden aparecer problemas de rendimiento. Observe que, a partir de MySQL 5.6, existe la variable `table_open_cache_instances`, que permite usar varias instancias de `table_open_cache`. Los valores superiores a 10000 para `table_open_cache` pueden utilizarse gracias a esta nueva opción.

c. Caché de threads

Cuando una conexión al servidor MySQL termina, el thread asociado a esta conexión se coloca en una caché para poder reutilizarse, evitando una operación de creación de un nuevo thread.

La variable correspondiente es `thread_cache_size` y, para ajustarla, basta con mirar la evolución de la variable de condición `Threads_created` en el tiempo: aumentando el tamaño de la caché, podemos llegar a garantizar que el valor de `Threads_created` aumente al menos a 5 por segundo.

- ☐ La creación de un thread es rápida con MySQL (aún más con MySQL 5.7); no debemos, por lo tanto, esperar ganancias importantes al aumentar el tamaño de la caché de threads.

d. Parámetros que no deben modificarse

Esta sección puede parecer extraña, pero algunos parámetros tienen un valor predefinido muy bajo, y muchas personas tratan de configurar un valor más alto con la esperanza de mejorar el rendimiento. Por desgracia, en el mejor de los casos, el impacto será invisible, y en el peor, podremos observar degradaciones de rendimiento.

No cambie, pues, nunca (excepto si comprende perfectamente el impacto de los cambios) `read_buffer_size`, `read_rnd_buffer_size`, `join_buffer_size` y `sort_buffer_size`.

Introducción

La seguridad de los datos del sistema informático de la empresa no es solo responsabilidad del RSSI (Responsable de la Seguridad de los Sistemas de Información). Los técnicos y las buenas prácticas necesarias para protegerse deben ser conocidas y aplicadas por cualquier persona. Los elementos de la cadena de aplicaciones son interdependientes; la fortaleza de esta cadena va a depender de la solidez del eslabón más débil. Una prueba no efectuada por la aplicación, una cuenta de usuario que tenga demasiados derechos sobre la base de datos, una actualización de seguridad no efectuada o simplemente una torpeza en el servidor de producción pueden ser la causa del robo o deterioro de los números de tarjetas bancarias de sus clientes, por ejemplo. Las consecuencias pueden ser desastrosas tanto financieramente como en términos de imagen para la empresa. Si bien la seguridad absoluta no existe, vamos a tratar de mostrar todas las prácticas necesarias para minimizar los riesgos en este capítulo.

Securización del servidor MySQL

Las cuestiones relacionadas con la seguridad deben plantearse desde la instalación del servidor de base de datos. En la filosofía de MySQL, el usuario debe poder descargar, instalar y utilizar el SGBDR (Sistema de Gestión de Bases de Datos Relacionales) sin ninguna dificultad. Esta simplicidad es uno de los puntos fuertes de MySQL, ya que, por un lado, esto permitió democratizar el uso de una base de datos haciendo accesible este universo (muchos desarrolladores han experimentado las bases de datos mediante MySQL) y, por otro, esto da la posibilidad de probar fácilmente una aplicación. El inconveniente principal es que la instalación predeterminada es muy permisiva, sobre todo en materia de seguridad...



1. Securización de la instalación

El objetivo de esta sección no es volver a la instalación del servidor que se detalla en el capítulo Instalación del servidor, sino recordar algunos puntos cruciales en materia de seguridad, que deberá adaptar en función del tipo de instalación y de su sistema operativo.

a. Controlar los permisos

Debemos crear un grupo y un usuario dedicado para iniciar la instancia `mysqld`:

```
$ groupadd mysql
$ useradd -g mysql mysql
```

El directorio de datos contiene información sensible; por tanto, debe preservarse de actos dolosos o de la visualización por personas no autorizadas

```
$ cd /usr/local/mysql/  
$ chown -R root .  
$ chown -R mysql data  
$ chgrp -R mysql .
```

mysqld no debe, en ningún caso, ejecutarse con el administrador del sistema `root` en UNIX (o administrador con MS Windows). Un usuario con, por ejemplo, el derecho `FILE` puede crear archivos como `root`.

b. Poner contraseña a la cuenta root

Es el superusuario de MySQL (no confundir con el administrador del sistema `root` bajo UNIX). Cuenta, por lo tanto, con todos los derechos sobre el servidor. Debe protegerse mediante contraseña. Esto es válido para cualquier sistema operativo.

```
$ mysql -u root # conexión al servidor con el usuario root sin  
contraseña  
mysql> SET PASSWORD FOR root@localhost = password('m0T2p4ss3'); /*  
el comando set password permite implementar una contraseña para una  
cuenta de usuario */
```

Como veremos un poco más adelante, un usuario MySQL está compuesto por un nombre, «user», y el nombre del equipo desde el cual el usuario se conecta, «host». Podemos tener una cuenta `'root'@'localhost'`, que permite conectarse al servidor con el usuario `root`, siempre que el cliente esté en el mismo equipo que el servidor MySQL (local), y, por ejemplo, una cuenta `'root'@'123.45.67.89'`, que permite conectarse como `root` desde el equipo que tiene como dirección IP 123.45.67.89. Son dos cuentas diferentes, que pueden tener derechos y contraseñas diferentes. Esta posibilidad que ofrece el servidor permite una gestión de derechos muy fina. Sin embargo, por experiencia, recomendamos solo tener una ocurrencia por usuario, en aras de simplificar la gestión de los derechos y, por lo tanto, disminuir el riesgo de errores. En otras palabras, guardemos

solo el usuario `root'@'localhost'`. Si necesitamos administrar el servidor en remoto, en lugar de tener `'root'@'%'` (que permite conectarse con el usuario root desde cualquier equipo), usaremos el protocolo de comunicación segura SSH o equivalente.

```
mysql> SELECT user, host, password FROM mysql.user WHERE user = 'root'
\G
***** 1. row *****
user: root
host: localhost
password: *228F5395471FEFD9B0BB291954D2189384329691
***** 2. row *****
user: root
host: 123.456.78.9
password: *63D85DCA15EAF5C58C908FD2FAE50CCBC60C4EA2

mysql> DROP USER 'root'@'123.456.78.9' ;

mysql> SELECT user, host, password FROM mysql.user WHERE user = 'root'
\G
***** 1. row *****
user: root
host: localhost
password: *228F5395471FEFD9B0BB291954D2189384329691
```

- ☐ Podemos renombrar la cuenta de administrador para que sea más difícil de encontrar por una persona malintencionada: `RENAME USER 'root'@'localhost' TO 'leader'@'localhost';`

c. Eliminar las cuentas anónimas

Para facilitar el manejo del software, podemos encontrar durante la instalación del servidor una cuenta anónima, es decir, una cuenta que tiene el campo `user` vacío. Esta permite conectarse al servidor con cualquier usuario, en local y sin contraseña. No hace falta argumentar más para comprender que, en un servidor de producción, ese usuario no debe

existir. Hay que suprimirlo. Este principio debe extenderse a todas las cuentas no utilizadas; así eliminamos muchos problemas potenciales.

```
mysql> SELECT user, host, password FROM mysql.user WHERE user = '' \G
***** 1. row *****
user:
host: localhost
password:

mysql> DROP USER ''@localhost;

mysql> SELECT user, host, password FROM mysql.user WHERE user = '' \G
```

- ☐ A partir de MySQL 5.7, no se crean cuentas anónimas durante la instalación del servidor, lo que sí sucedía en todas las versiones anteriores.

d. Eliminar el esquema test

El esquema `test` es creado por MySQL durante la instalación para todas las versiones salvo 5.7, y todos los usuarios tienen derecho a acceder en lectura y escritura sin que sea necesario especificar de forma explícita los derechos adecuados. Un usuario malintencionado podría llenar de datos el disco duro y así impedir el buen funcionamiento de nuestra aplicación.

```
mysql> SHOW SCHEMAS;
+-----+
| Database          |
+-----+
| information_schema|
| mysql             |
| test              |
+-----+
```



```
mysql> DROP SCHEMA test;
```

```
mysql> SHOW SCHEMAS;
```

```
+-----+
| Database          |
+-----+
| information_schema |
| mysql              |
+-----+
```

Cabe señalar que esta recomendación se aplica también a todo esquema `test` que se cree a continuación, y también a cualquier esquema que comience por la palabra `test_`.

e. Securizar la instalación con la herramienta `mysql_secure_installation`

La aplicación de estas recomendaciones puede efectuarse con la herramienta `mysql_secure_installation`. Esta herramienta no está disponible en MS Windows; sin embargo, el instalador gráfico también ofrece la posibilidad de securizar la instalación.

```
$ mysql_secure_installation
```

```
NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL
MySQL SERVERS IN PRODUCTION USE! PLEASE READ EACH STEP CAREFULLY!
```

In order to log into MySQL to secure it, we'll need the current password for the root user. If you've just installed MySQL, and you haven't set the root password yet, the password will be blank, so you should just press enter here.

Enter current password for root (enter for none):

2. Utilización de SSL

Los datos viajan en texto sin cifrar entre el cliente y el servidor MySQL. A partir de MySQL 4, podemos encriptarlos usando el protocolo SSL (*Secure Sockets Layer*), que permite garantizar la seguridad, integridad y confidencialidad de los datos intercambiados. Sin embargo, las operaciones adicionales tienen un impacto significativo sobre el rendimiento, reduciéndolo en cerca del 30 %.

a. Las opciones

SSL ofrece una lista de opciones. Existe más información disponible en el sitio de MySQL: <https://dev.mysql.com/doc/refman/5.6/en/secure-connection-options.html>. He aquí una breve descripción:

- `ssl` permite al servidor autorizar conexiones SSL y al cliente conectarse utilizando este protocolo.
- `ssl-ca` permite especificar el archivo que contiene el certificado de autoridad (CA).
- `ssl-capath` es el directorio de archivos que contienen los certificados de autoridad SSL.
- `ssl-cert` es el nombre del certificado SSL que debe utilizarse para establecer una conexión segura.
- `ssl-key` es el nombre del archivo de clave SSL que debe utilizarse para establecer una conexión segura.
- `ssl-cipher` es la lista de cifrados (*cipher*) autorizados que deben utilizarse con SSL.

b. Las principales etapas

Para poner en funcionamiento SSL, el servidor debe haber sido compilado con soporte para SSL, utilizando la opción `--with-openssl` o `--with-yassl`. Se requieren algunas etapas preparatorias antes de poder cifrar las comunicaciones. Las siguientes son las principales. Encontraremos más información en la documentación oficial de MySQL en la dirección: <https://dev.mysql.com/doc/refman/5.6/en/secure-connection-options.html>

Creación del certificado de autoridad y una clave privada, en el formato estándar x509:

|

```
openssl req -new -x509 -keyout ca-key.pem -out ca-cert.pem -days 365
```

Creación del certificado y la clave privada del servidor:

```
openssl req -new -keyout server-key.pem -out server-req.pem -days 365
openssl x509 -req -in server-req.pem -days 365 -CA ca-cert.pem -CAkey
ca-key.pem -set_serial 01 -out server-cert.pem
```

Creación del certificado y la clave privada del cliente:

```
openssl req -new -keyout client-key.pem -out client-req.pem -days 365
openssl x509 -req -in client-req.pem -days 365 -CA ca-cert.pem -CAkey
ca-key.pem -set_serial 01 -out client-cert.pem
```

Retirada de la contraseña de las claves:

```
openssl rsa -in server-key.pem -out server-key.pem
openssl rsa -in client-key.pem -out client-key.pem
```

Configuración del servidor y los clientes que deberán conectarse a SSL:

```
[mysqld]
ssl-ca=/etc/ssl/mysql/ca-cert.pem
ssl-cert=/etc/ssl/mysql/server-cert.pem
ssl-key=/etc/ssl/mysql/server-key.pem
```

```
[client]
ssl-ca=/etc/ssl/mysql/ca-cert.pem
ssl-cert=/etc/ssl/mysql/client-cert.pem
ssl-key=/etc/ssl/mysql/client-key.pem
```

Después de reiniciar el servidor para que estas opciones se tengan en cuenta, conéctese con el cliente texto.

El comando `SHOW VARIABLES LIKE '%ssl%'` nos indica que la configuración está funcionando:

```
mysql> SHOW VARIABLES LIKE '%ssl%';
```

Variable_name	Value
have_openssl	YES
have_ssl	YES
ssl_ca	/etc/ssl/mysql/ca-cert.pem
ssl_capath	
ssl_cert	/etc/ssl/mysql/server-cert.pem
ssl_cipher	
ssl_key	/etc/ssl/mysql/server-key.pem

Cifrado de datos

Para hacer frente a las necesidades de cifrado de información sensible (financiera, médica...) o para tratar de protegerse contra el robo de datos, contra administradores poco escrupulosos, puede ser necesario encriptar los datos. Antes de la versión 5.7, MySQL carecía de módulos criptográficos capaces, por ejemplo, de encriptar el contenido de una columna o de una tabla. Sin embargo, podemos cifrar/descifrar los datos con las funciones MySQL `AES_ENCRYPT()` / `AES_DECRYPT()`, `DES_ENCRYPT()` / `DES_DECRYPT()` y `ENCODE()` / `DECODE()` (para más detalles, consulte la página siguiente: <https://dev.mysql.com/doc/refman/5.7/en/encryption-functions.html>).



Los datos son entonces cifrados en el disco. Sin embargo, están en texto claro, incluida la clave de cifrado en el código SQL, así como en la red.

```
mysql> INSERT INTO t_crypt VALUES (1, AES_ENCRYPT('Frase cifrada',  
'crypt_key'));
```

```
Query OK, 1 row affected (0,00 sec)
```

```
mysql> SELECT i, b FROM t_crypt;
```

```
+-----+-----+  
| i      | data_crypt      |  
+-----+-----+  
| 1      | ?z#?uگb?7?G[g,< |  
+-----+-----+
```

```
1 row in set (0,00 sec)
```

```
mysql> SELECT i, AES_DECRYPT(b, 'crypt_key') AS data_crypt FROM  
t_crypt;
```

```

+-----+-----+
| i      | data_crypt      |
+-----+-----+
| 1      | Frase cifrada   |
+-----+-----+
1 row in set (0,00 sec)

```

Otra posibilidad es cifrar los datos a nivel de la aplicación antes de almacenarlos en la base de datos. No aparecerán en claro en la base de datos, ni durante la transferencia entre el cliente y el servidor.

El cifrado también se puede hacer a nivel del montaje del sistema de archivos, lo que permite a toda una partición del sistema de archivos ser cifrada y descifrada por el sistema operativo.

A partir de MySQL 5.7, el cifrado está disponible de forma nativa para las tablas InnoDB usando la opción `innodb_file_per_table`.

Debemos configurar primero un plugin específico para que un anillo de claves (*keyring*) esté disponible. La instalación se detalla en la página

<https://dev.mysql.com/doc/refman/5.7/en/keyring.html> y podemos comprobar la correcta instalación con el siguiente comando:

```

mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
        FROM INFORMATION_SCHEMA.PLUGINS
        WHERE PLUGIN_NAME LIKE 'keyring%';

```

```

+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| keyring_file | ACTIVE        |
+-----+-----+

```

Después, podemos crear una tabla cifrada con la siguiente sintaxis:

```
mysql> CREATE TABLE t (...) ENCRYPTION='Y' ;
```

También podemos convertir una tabla no cifrada en una tabla cifrada usando el comando ALTER TABLE:

```
mysql> ALTER TABLE t ENCRYPTION='Y' ;
```

Observe que solo los archivos de datos son codificados, los distintos registros (registros binarios, redo logs, por ejemplo) no lo son.

Para más información, consulte la documentación en línea:

<https://dev.mysql.com/doc/refman/5.7/en/innodb-tablespace-encryption.html>

- ☐ MariaDB 10 también dispone de un sistema de cifrado, con una implementación distinta. Este sistema se describe en la página siguiente:

<https://mariadb.com/kb/en/mariadb/data-at-rest-encryption/>

Las opciones para reforzar la seguridad

1. skip-networking

En el caso de arquitecturas en las que el cliente se encuentra en el mismo equipo que el servidor, puede ser interesante prohibir las conexiones remotas. No solo prevenimos la latencia y posibles problemas relacionados con la red, sino que es a su vez un medio para limitar los riesgos de ataques remotos. La opción `skip-networking` permite al servidor MySQL no escuchar las conexiones TCP/IP. Las conexiones al servidor MySQL solo podrán efectuarse en local, con los sockets UNIX. En un entorno MS Windows, el principio es el mismo, pero con los protocolos named `pipes` o `shared memory`.

En el archivo de configuración `my.cnf` (o `my.INI`), activamos `skip-networking`:

```
[mysqld]  
skip-networking
```

Ahora es imposible conectar al servidor MySQL utilizando el protocolo TCP/IP.

```
$ mysql --host=localhost --protocol=tcp  
ERROR 2003 (HY000): Can't connect to MySQL server on 'localhost' (111)
```


Solo queda el protocolo UNIX socket (named pipe o shared memory en Windows):

```
$ mysql --host=localhost --protocol=socket
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.1.35-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

Mysql>
```

2. bind-address

La opción `bind-address` permite asociar (*bind*) una dirección IP al servidor MySQL; en concreto, a sus interfaces de red. En otras palabras, las conexiones de cliente solo podrán pasar por esta dirección IP para alcanzar el servidor MySQL.

Por ejemplo, para impedir el acceso al servidor a los clientes remotos, hay que tener la configuración siguiente: `bind-address = 127.0.0.1` o `bind-address = localhost`. Por defecto, `bind-address = '0.0.0.0'`, es decir, que las conexiones se pueden hacer a partir de las diferentes interfaces de red del servidor.

Si especificamos una dirección IP, solo podremos conectar a una sola, y esta última solo puede venir de una interfaz de red perteneciente al equipo. No podemos, pues, atribuir la dirección IP de otro servidor.

Por ejemplo, tomemos una configuración con las tres interfaces de red siguientes:

- `Eth0: 10.60.1.177` (cable).
- `lo: 127.0.0.1` (localhost).
- `wlan0: 10.60.1.178` (la Wi-Fi).

Y una configuración de `bind-address` predeterminada (`0.0.0.0`, la más permisiva). Si

el protocolo TCP/IP está activado en el servidor MySQL, entonces podemos conectarnos de las siguientes formas:

```
mysql --host=10.60.1.177
mysql --host=127.0.0.1
mysql --host=10.60.1.178
```

Sin embargo, con `bind-address = '10.60.1.177'`, de los tres intentos, solo `mysql--host = 10.60.1.177` aceptará la conexión al servidor. Los otros dos intentos terminarán con un fallo y devolverán el error: `ERROR 2003 (HY000) : Can't connect to MySQL server on '127.0.0.1' (111)` (Traducción: Imposible conectarse al servidor MySQL en '127.0.0.1' (111)).

3. skip-name-resolve

La opción `skip-name-resolve` desactiva la búsqueda de los nombres de HOSTS por DNS durante la conexión de los clientes al servidor MySQL. Esto obliga a las aplicaciones a utilizar solo las cuentas usando como nombre de host una dirección IP o `localhost`.

Las ganancias al activar esta opción son dos: rendimiento y seguridad. Cuando el tiempo de ejecución de la petición es mucho más largo con un cliente remoto que con un cliente local, esto puede deberse a una resolución lenta del nombre de host, sobre todo si tenemos muchos. En este caso, `skip-name-resolve` puede mejorar la duración de la petición y disminuir el tiempo de conexión del cliente al servidor.

La otra ventaja es reducir los riesgos de ataques por denegación de servicio DNS, ya que el mecanismo de resolución del equipo MySQL es muy costoso.

El principal inconveniente es aumentar el riesgo de que la conexión de clientes al servidor se vea bloqueada, sobre todo si la red tiene una fiabilidad dudosa. Para tomar precauciones, establezca la variable `max_connect_errors` con un valor bastante grande (por ejemplo, 100000000). En el caso de que esto no sea suficiente y de que el cliente se encuentre en una lista negra, deberemos reiniciar la caché de las máquinas ejecutando el

comando `FLUSH hosts` o utilizando el cliente `mysqladmin flush-hosts`.

- Para comprobar si tenemos cuentas de usuario con nombres de host, usamos el comando: `SELECT user, host FROM mysql.user WHERE host <> 'localhost' AND host RLIKE '[a-z]';`

4. skip-show-database

Por defecto, cualquier usuario puede ejecutar el comando `SHOW SCHEMAS` (o `show DATABASES`), que permite ver la lista de los esquemas del servidor. Ver los esquemas no significa forzosamente poder visualizar su contenido o manipular los objetos que contienen. Sin embargo, es información que no es necesario compartir con todas las aplicaciones.

Si activamos la opción `skip-show-database`, solo las cuentas que tengan el derecho `SHOW SCHEMAS` (o `show DATABASES`) podrán ver la lista de todos los esquemas del servidor. Sin embargo, siempre se podrá ver la lista de esquemas para los cuales el usuario tenga derechos utilizando la tabla `information_schema.SCHEMATA`.

```
mysql> SHOW SCHEMAS;
ERROR 1227 (42000): Access denied; you need the SHOW DATABASES
privilege for this operation
```

```
mysql> SELECT schema_name FROM information_schema.SCHEMATA;
+-----+
| schema_name          |
+-----+
| information_schema    |
| world                 |
+-----+
```

```
mysql> SHOW GRANTS;
+-----+
| Grants for bi@localhost |
```

```
+-----+
| GRANT USAGE ON *.* TO 'maria'@'localhost' |
| GRANT SELECT ON `world`.* TO 'maria'@'localhost' |
+-----+
```

5. secure-file-priv

MySQL permite dedicar un directorio para los archivos cargados con el comando `LOAD DATA INFILE` o la función `LOAD_FILE()`, y para los archivos creados, con la petición `SELECT ... INTO OUTFILE`.

```
[mysqld]
secure-file-priv=/sql/file/

mysql> LOAD DATA INFILE '/home/daz/sql/city.txt' INTO TABLE City;
ERROR 1290 (HY000): The MySQL server is running with the --secure-
file-priv option so it cannot execute this statement
```

El error 1290 mostrado en este ejemplo indica que el archivo que se desea cargar no está en el directorio definido por la opción `secure_file_priv`.

6. chroot

`chroot` es un comando UNIX que se utiliza con frecuencia en los hosts web. Permite, por ejemplo, ejecutar de forma independiente varias instancias de un mismo programa, cada una con su propio árbol de archivos, en un equipo host compartido, o incluso ejecutar aplicaciones de 32 bits en un sistema de 64 bits. MySQL se puede ejecutar en un entorno «chroot», es decir, que se encuentra aislado en una parte del árbol y cuyo directorio raíz es la ruta pasada por la opción `chroot`. En caso de ataque, el atacante no podrá acceder a los archivos que no deriven de la raíz especificada, ni por lo tanto del directorio raíz de la máquina host. Solo se verá impactado el árbol MySQL, lo que es un mal menor. Cabe

señalar que la utilización de algunas peticiones, como `SELECT . . . INTO OUTFILE` y `LOAD DATA INFILE`, puede verse limitada.

Gestión de usuarios y contraseñas

La gestión de los usuarios de MySQL se realiza por completo a través de tablas específicas que pertenecen al esquema de sistema `mysql: user, db, tables_priv, columns_priv, procs_priv`.

La tabla `user` contiene la lista de cuentas de usuarios. Se describen con, entre otras, las columnas `host` (el nombre de host) y `user` (el nombre). Estas dos columnas representan la clave primaria de la tabla. También encontraremos la columna `password` (la contraseña de la cuenta), que se almacena recortada (gracias a la función hash MySQL `password()`), así como los derechos globales del usuario, como el derecho de hacer inserciones (`INSERT`), lecturas (`SELECT`)...

La tabla `db` contiene los derechos específicos a la base de datos (o esquemas) de cada usuario.

La tabla `tables_priv` contiene los derechos específicos a una tabla o a una vista.

La tabla `columns_priv` contiene los derechos específicos a una columna.

La tabla `procs_priv` contiene los derechos para las rutinas almacenadas, es decir, los procedimientos y funciones almacenados.

1. Conexión a cuentas de usuario

Al arrancar el servidor, las tablas de sistema se cargan en memoria. La tabla `user` ve entonces sus datos clasificados en función de la columna `host`, del más específico (`localhost`, dirección IP) al menos específico (%). Para dos valores de `host` idénticos, se efectuará una selección adicional sobre la columna `user`, siempre del más específico (un

nombre de cuenta de usuario) al menos específico, es decir, la cuenta del usuario anónimo (''). Cuando el cliente se conecta, la cuenta de usuario elegida es la primera que corresponde al host (host) de la tabla user ordenada y luego al usuario (user) con la contraseña correcta (password).

El servidor tiene dos usuarios, 'daz'@'%' y 'daz'@'localhost'.

```
mysql_root> SELECT user, host FROM mysql.user WHERE user = 'daz';
+-----+-----+
| user | host      |
+-----+-----+
| daz  | %         |
| daz  | localhost |
+-----+-----+
2 rows in set (0,00 sec)
```

Conectándose con el usuario daz desde el cliente local (localhost), la conexión se realiza con la cuenta daz@localhost, ya que es la más específica de las dos.

```
mysql_daz_local> SELECT USER(), CURRENT_USER();
+-----+-----+
| USER()          | CURRENT_USER() |
+-----+-----+
| daz@localhost   | daz@localhost  |
+-----+-----+
1 row in set (0,00 sec)
```

Sin embargo, conectándose desde un cliente remoto (dasini.net) con el usuario daz, solo la cuenta 'daz'@'%' corresponde:

```
mysql_daz_distant> SELECT USER(), CURRENT_USER();
+-----+-----+
```

```

| USER() | CURRENT_USER() |
+-----+-----+
| daz@dasini.net | daz@% |
+-----+-----+
1 row in set (0,00 sec)

```

En las antiguas versiones de MySQL (3.23, 4.0), era necesario gestionar los derechos modificando de forma directa estas tablas. Esta forma de proceder es obsoleta y ya no se recomienda.

Los comandos necesarios para la gestión de los derechos son los siguientes:

- `CREATE USER`: permite crear una cuenta de usuario y asignarle una contraseña.
- `GRANT`: permite otorgar derechos, proporcionar una contraseña, especificar los límites de utilización de recursos (como limitar el número de conexiones por hora) y configurar las opciones SSL (punto que hemos visto antes). También puede desempeñar el papel del comando `CREATE USER`.
- `REVOKE`: este comando es el opuesto a `GRANT`. Permite eliminar los derechos de una cuenta de usuario pero no borrar la cuenta.
- `DROP USER`: permite eliminar una cuenta de usuario (y en consecuencia, todos sus derechos).
- `SHOW GRANTS`: permite ver los derechos.

Pérdida de contraseña de administrador

Perder la contraseña del administrador es una situación muy embarazosa, que impide además administrar el servidor. Como se almacena recortada, no es posible recuperarla; sin embargo, se proporciona una solución.

En primer lugar, debemos detener el servidor.

Luego, reiniciar con la opción `skip-grant-tables`. Esta opción le impide al servidor utilizar las tablas que gestionan los derechos de los usuarios. Podremos conectarnos al servidor, con cualquier usuario existente o no, sin introducir la contraseña.

Sin embargo, debemos ser conscientes de que en ese momento, el servidor es vulnerable porque cualquiera puede conectarse y tomar el control. Asegúrese de que ningún otro

cliente (aplicación de batch, por ejemplo) puede acceder al servidor. Para ello, inicie el servidor con un socket (o un puerto) no estándar. La opción `skip-networking`, que hemos visto en la sección de las opciones para reforzar la seguridad, será útil.

La opción `console` también es opcional. Esta permite mostrar los mensajes de error en la pantalla.

```
$ mysqld --skip-grant-tables --skip-networking --console  
--socket=/tmp/maintenance.sock &
```

Conéctese al servidor con el cliente texto `mysql` y reemplace el bloqueo ejecutando el comando `FLUSH PRIVILEGES`. Esto obligará al servidor a comprobar de nuevo los nombres de los usuarios (`user`), los nombres de host (`host`) y las contraseñas (`password`) de los posibles clientes que intentan conectarse.

```
$ mysql --socket=/tmp/maintenance.sock  
mysql> FLUSH PRIVILEGES;
```

Genere una nueva contraseña. La antigua no puede recuperarse porque se encuentra recortada (hash) de forma irreversible.

```
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('N0uye#u_pwd');
```

Detenga el servidor MySQL y reinícielo de forma normal. ¡Hemos reparado el error!

- ☐ Encontrará una alternativa en la documentación de MySQL:
<http://dev.mysql.com/doc/refman/5.6/en/resetting-permissions.html>

2. Gestión de cuentas de usuario

La sintaxis para crear una cuenta de usuario es la siguiente: `CREATE USER 'usuario'@ 'host' IDENTIFIED BY 'contraseña'`

La cuenta de usuario (`'usuario'@'máquina'`) está compuesta por un nombre de usuario y un nombre de host (la dirección IP o el nombre de la máquina) entre comillas (") o apóstrofos (') y separados por una arroba (@). Si no se especifica un nombre de host, la de usuario se creará con el host %, que significa *cliente que se conecta desde cualquier host*.

Un error común es abrir los apóstrofos antes del nombre de usuario y cerrarlos tras el nombre de host. Por ejemplo, `'usuario@host'` creará una cuenta de usuario `'usuario@host'@' %'`, que no es la misma cuenta de usuario que `'usuario'@' %'`.

La contraseña, aunque opcional, es muy aconsejable, en especial en un servidor en producción. Es una cadena de caracteres que debemos poner también entre comillas o apóstrofos.

```
mysql> CREATE USER 'olivier'@'localhost' IDENTIFIED BY 'Pa55w0rD';  
mysql> CREATE USER 'olivier' IDENTIFIED BY 'otro_Pa55w0rD';
```

- ☐ Observe: hasta MySQL 5.6.2 inclusive, la contraseña estaba en texto visible en el comando `CREATE USER`, y podía, por lo tanto, ser visible en los registros MySQL (como el registro general o general log) o en los registros del sistema (como `~/.mysql_history`). Estos archivos deben ser protegidos del acceso por personas no autorizadas.

A partir de MySQL 5.6.3, este no es el caso. Las contraseñas en texto visible, pasadas en los comandos `CREATE USER`, `GRANT` y `SET PASSWORD`, ya no son visibles en los archivos de registro general de peticiones lentas y binarios. Sin embargo, podemos obtener el antiguo comportamiento al arrancar el servidor con la opción `Log-raw`. Por supuesto, esto no es aconsejable.

```
$ tail -f mysql-general.log
/mysql, Version: 5.6.7-rc (MySQL Community Server (GPL)). started
with:
Tcp port: 5605  Unix socket: /tmp/mysql_5605.sock
Time           Id Command      Argument
121030 20:32:28      9 Query      SET PASSWORD FOR `olivier`=
<secret>
121030 20:33:55      2 Query      CREATE USER 'daz'@'localhost'
IDENTIFIED BY PASSWORD '*F02F5BFE01D376A3C4D9AD9E930D25C9D0CED644'
121030 20:35:12      2 Query      GRANT SELECT ON `world`. * TO
'olivier'@'localhost' IDENTIFIED BY PASSWORD
'*7D85CAE80F5E385051CCEB3035293498ED08715A'
```

Estos dos comandos permiten la creación de dos cuentas de usuario distintas. Lo podemos comprobar mirando la tabla `user` del esquema `mysql`:

```
mysql> SELECT user, host, password FROM mysql.user WHERE user =
'olivier';
```

user	host	password
olivier	localhost	*DEDC92F45B664030BBF191A8A7732A52044B5EBF
olivier	%	*C0F5BEAC30750E7A522360CAD46A7AFD450CDA12

El usuario `olivier@localhost` solo se utilizará en caso de conexión desde un cliente local, es decir, un cliente que está en el mismo equipo que el servidor MySQL y pasando por el bucle local (`localhost` o `127.0.0.1`). En todos los demás casos, se utilizará el usuario `Olivier@' %'`.

Cabe señalar también que la contraseña se almacena recortada en la tabla `mysql.user` mediante la función MySQL `password()`.

- Tener varias cuentas con la misma contraseña puede ser peligroso, ya que en caso de que esta se vea comprometida, podrían verse afectadas varias cuentas . Una forma sencilla de saber si tenemos varias ocurrencias de la misma contraseña es ejecutar la siguiente petición:

```
SELECT count(*), password FROM mysql.user GROUP BY password HAVING count(user)>1;
```

El carácter % puede utilizarse también como comodín en el nombre de host. Por ejemplo, `CREATE USER 'fresh'@'%.dasini.net'` permite crear una cuenta de usuario para conectarse a partir de los equipos cuyo nombre termina por `.dasini.net`. Otro ejemplo, `CREATE USER 'emma'@'.168%'` permite crear una cuenta de usuario para conectarse a partir de los equipos cuya dirección IP comienza por `192.168.`

`CREATE USER 'kams'@'.168.10.1_'` permite crear una cuenta de usuario para conectarse a partir de los equipos cuya dirección IP está comprendida entre `192.168.10.10` y `192.168.10.19`.

Dicho esto, no debemos abusar de los comodines en el nombre de host si tenemos la posibilidad de bloquear la cuenta de usuario a una dirección IP (véase un nombre de host). Para recuperar la lista de cuentas con %, ejecute la siguiente petición:

```
SELECT user, host FROM mysql.user WHERE host LIKE '%\%';
```

- Seguridad: una buena práctica consiste en limitar las interferencias entre diferentes aplicaciones que utilizan la misma instancia de MySQL, creando una cuenta de usuario para cada una de ellas. Por ejemplo:
`'webmail'@'222.33%'` para la aplicación webmail con un rango de direcciones que comienza por `222.33`.

'foro'@'222.33.4.55' para el foro que está en la dirección 222.33.4.55.

'conta'@'localhost' para la aplicación de contabilidad que está en local.

Como hemos visto antes, la contraseña se recorta antes de almacenarse. Una función hash es una función que toma una cadena de caracteres de entrada y devuelve otra cadena de caracteres de salida. La cadena de caracteres resultante tiene siempre la misma longitud y es exactamente igual para una misma entrada. A diferencia del cifrado, el hash no permite recuperar la cadena de caracteres fuente, ya que es irreversible. Entre los algoritmos más conocidos, cabe citar el MD5 (*Message Digest 5*) o el SHA1 (*Secure Hash Algorithm*). La función MySQL `password()`, que recorta las contraseñas, es un buen compromiso entre rendimiento (duración del recorte de la cadena de caracteres) y seguridad (tamaño de la cadena resultante). No obstante, no se aconseja su uso en el código porque su algoritmo puede variar en función de la versión de MySQL. Ese fue el caso, para mejorar la seguridad del recorte, durante la transición a la versión MySQL 4.1.1. Este antiguo algoritmo está disponible desde entonces con la función `old_password()`. Encontraremos un artículo (en francés) sobre las diferentes funciones de recorte (hash) de MySQL y sus resultados en dasini.net (<http://dasini.net/blog/2008/11/11/hash-securite-mysql/>).

```
mysql> SELECT password('Nat&Mat');
+-----+
| password('Nat&Mat') |
+-----+
| *6CB1A82BF5B9D12B253F7E0BF3EDFAE4314F79C7 |
+-----+
```

```
mysql> SELECT old_password('Nat&Mat');
+-----+
| old_password('Nat&Mat') |
+-----+
| 05ebcf1969421b4f |
+-----+
```

La opción servidor `secure-auth` permite prohibir la identificación de las cuentas de la contraseña recortadas con el antiguo algoritmo.

- En MySQL 5.6, el cliente `mysql_upgrade` emite un aviso si se encuentra una cuenta de usuario con una contraseña recortada con el antiguo algoritmo de hash de MySQL (versiones iguales o anteriores a MySQL 4.0). Es muy recomendable actualizar la contraseña al algoritmo actual, mucho más seguro.

Con MySQL 5.6, se permite un tercer valor para la variable `old_password`. Además de los valores 0 (OFF): desactivar el antiguo método de recorte, 1 (ON): activar el antiguo método de control, la función puede tomar el valor 2, que permite usar el algoritmo de hash SHA-256. La función `password()` se ve impactada por el valor de la variable `old_password`, ya que utiliza el algoritmo definido por esta última.

Con MySQL 5.7, no se puede tener `old_password = 1`, y `secure_auth` está activado obligatoriamente.

Valor hash por defecto:

```
mysql> SET SESSION old_passwords=0;
```

```
mysql> SHOW SESSION VARIABLES LIKE 'old_passwords';
```

Variable_name	Value
old_passwords	0

```
mysql> SELECT PASSWORD('freshdaz');
```

PASSWORD('freshdaz')
*1E18977CC59048E063BB2355DE4A002BCCE0A220

```
mysql> SELECT OLD_PASSWORD('freshdaz');
+-----+
| OLD_PASSWORD('freshdaz') |
+-----+
| 139638993a07a3ee        |
+-----+
```

Valor hash anterior a MySQL 4.1.1:

```
mysql> SET SESSION old_passwords=1;

mysql> SHOW SESSION VARIABLES LIKE 'old_passwords';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| old_passwords | 1     |
+-----+-----+

mysql> SELECT PASSWORD('freshdaz');
+-----+
| PASSWORD('freshdaz') |
+-----+
| 139638993a07a3ee    |
+-----+

mysql> SELECT OLD_PASSWORD('freshdaz');
+-----+
| OLD_PASSWORD('freshdaz') |
+-----+
| 139638993a07a3ee        |
+-----+
```

Nuevo hash (SHA-256) disponible a partir de MySQL 5.6:

```
mysql> SET SESSION old_passwords=2;
```

```
mysql> SHOW SESSION VARIABLES LIKE 'old_passwords';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| old_passwords | 2     |
+-----+-----+
```

```
mysql> SELECT PASSWORD('freshdaz');
```

```
+-----+
| PASSWORD('freshdaz') |
+-----+
| $5$K:~# ^#~!m##g hy$ClGpSNUzq/oNi38cZ0pGvXcD/avF1tiFr0zUWWwIKZ5 |
+-----+
```

```
mysql> SELECT OLD_PASSWORD('freshdaz');
```

```
+-----+
| OLD_PASSWORD('freshdaz') |
+-----+
| 139638993a07a3ee        |
+-----+
```

- ☐ La contraseña podrá presentarse directamente en su forma recortada durante la creación del usuario. Se debe especificar a MySQL la función hash usada.

```
mysql> SELECT password('m3r3gran')
```

```
+-----+
| password('m3r3gran') |
+-----+
| *862EB6CE84D1AADF7F10BE749DD8C44A2053842B |
+-----+
```



```
mysql> CREATE USER 'valerie'@'localhost' IDENTIFIED BY PASSWORD
'*862EB6CE84D1AADF7F10BE749DD8C44A2053842B';
```

```
mysql> SELECT user, host, password FROM mysql.user WHERE user =
'valerie';
```

user	host	password
valerie	localhost	*862EB6CE84D1AADF7F10BE749DD8C44A2053842B

Una cuenta de usuario puede ser renombrada con `RENAME USER` y eliminada con `DROP USER`. Estos dos comandos aparecen seguidos por el nombre del usuario.

```
mysql> SELECT user, host FROM mysql.user WHERE user='olivier';
```

user	host
olivier	%
olivier	localhost

```
mysql> RENAME USER 'olivier'@'%' TO 'olivier'@'dasini.net';
```

```
mysql> SELECT user, host FROM mysql.user WHERE user='olivier';
```

user	host
olivier	dasini.net
olivier	localhost

```
mysql> DROP USER 'olivier'@'dasini.net';
```

```
mysql> SELECT user, host FROM mysql.user WHERE user='olivier';
```

user	host
------	------

user	host
olivier	localhost

En la creación de una cuenta de usuario, los derechos de los usuarios que utilizan esta cuenta se ven especialmente reducidos. Solo disponen del derecho USAGE, que permite conectarse al servidor MySQL únicamente.

Lo podemos comprobar con el comando `SHOW GRANTS`, que permite ver los derechos de una cuenta:

```
mysql> SHOW GRANTS FOR 'olivier'@'localhost';
+-----+
| Grants for olivier@localhost |
+-----+
| GRANT USAGE ON *.* TO 'olivier'@'localhost' IDENTIFIED BY
PASSWORD '*DEDC92F45B664030BBF191A8A7732A52044B5EBF' |
+-----+
```

El derecho USAGE solo permite conectarse al servidor y acceder al esquema virtual `information_schema`, que está compuesto por las vistas de sistema generadas por MySQL y que contiene los metadatos (ver el capítulo Herramientas de supervisión para más detalles). Para que el usuario pueda manipular los datos, hay que darle otros derechos.

3. Roles

El concepto de rol nunca formó parte de las funcionalidades disponibles en MySQL, y la versión 5.7 no es una excepción. Sin embargo, con MySQL 5.7, es posible emular los roles utilizando los usuarios proxy.

La solución no es perfecta; por una parte, porque la operación es bastante compleja, y por otra, porque la documentación en línea no es explícita acerca de la posibilidad de utilizar los usuarios proxy. Sin embargo, encontraremos una excelente descripción del proceso que es preciso seguir en el artículo siguiente:

<http://mysqlblog.fivefarmers.com/2015/04/08/emulating-roles-with-expanded-proxy-user-support-in-5-7-7/>

- ☐ Los roles están disponibles en MariaDB 10. Vea la página siguiente:
<https://mariadb.com/kb/en/mariadb/roles-overview/>

4. Plug-ins de autenticación

A partir de MySQL 5.6, la autenticación se implementa mediante plug-ins. El método de autenticación por defecto, visto en la sección anterior, utiliza el plug-in `mysql_native_password`, que viene con el servidor MySQL. Además se incluyen otros dos plug-ins en el servidor MySQL: `mysql_old_password` (método de control de versiones anteriores a MySQL 4.1.1) y `sha256_password`, que utiliza el algoritmo de hash SHA-256.

```
mysql> pager grep AUTHENTICATION
PAGER set to 'grep AUTHENTICATION'
```

```
mysql> SHOW PLUGINS;
```

<code>mysql_native_password</code>	ACTIVE	AUTHENTICATION	NULL	GPL	
<code>mysql_old_password</code>	ACTIVE	AUTHENTICATION	NULL	GPL	
<code>sha256_password</code>	ACTIVE	AUTHENTICATION	NULL	GPL	

Para utilizar el método de hash SHA-256, existen dos posibilidades. La primera consiste en añadir la línea `default-authentication-plugin=sha256_password` en la sección `[mysqld]` del archivo de configuración del servidor MySQL. La segunda, en especificar de forma explícita el plug-in usado (`sha256_password`) y el método de control (`old_passwords=2`) al crear el usuario:

```

mysql> CREATE USER daz_strong@localhost IDENTIFIED WITH
sha256_password;

mysql> SET SESSION old_passwords=2;

mysql> SET PASSWORD FOR daz_strong@localhost =
PASSWORD('m0t_2_p45S3');

mysql> SHOW GRANTS FOR daz_strong@localhost;
+-----+
| Grants for daz_strong@localhost |
+-----+
| GRANT USAGE ON *.* TO 'daz_strong'@'localhost' |
+-----+

mysql> SELECT user, host, password, plugin, authentication_string
FROM mysql.user WHERE user = 'daz_strong'\G
***** 1. row *****
          user: daz_strong
          host: localhost
      password:
      plugin: sha256_password
authentication_string: $5$^+y
%gk5YgR;5#(w4SSkZ$UE0.PD0AhUrI.gFI9gQuGFAMmLeGrC1dfiIUzh8hjV.

```

Esta autenticación a través de plug-ins permite conectarse al servidor MySQL de muchas otras maneras (LDAP, PAM...). Para ir más allá, no dude en consultar la documentación oficial de MySQL: <http://dev.mysql.com/doc/refman/5.6/en/pluggable-authentication.html>

5. Plug-in de validación de las contraseñas

A partir de MySQL 5.6, el plug-in `validate_password` permite mejorar la seguridad de las cuentas de usuario, probando la robustez de una contraseña mediante la función `VALIDATE_PASSWORD_STRENGTH()` y, llegado el caso, impidiendo al

usuario crear una contraseña que no es lo bastante segura, durante la ejecución de los comandos CREATE USER, GRANT y SET PASSWORD. Cuando este plug-in no está activado, es muy fácil crear una contraseña poco segura:

```
mysql> SET PASSWORD = PASSWORD('qwerty');
Query OK, 0 rows affected (0,00 sec)
```

El comando SHOW PLUGINS permite saber si el plug-in validate_password está activado.

```
mysql> SHOW PLUGINS;
+-----+-----+-----+-----+-----+
| Name                | Status | Type                | Library | License |
+-----+-----+-----+-----+-----+
| mysql_native_password| ACTIVE | AUTHENTICATION      | NULL    | GPL      |
| mysql_old_password   | ACTIVE | AUTHENTICATION      | NULL    | GPL      |
| sha256_password      | ACTIVE | AUTHENTICATION      | NULL    | GPL      |
| ...
+-----+-----+-----+-----+-----+
43 rows in set (0,00 sec)
```

Si no fuera el caso, la instalación se hace con facilidad con el comando INSTALL PLUGIN:

```
mysql> INSTALL PLUGIN validate_password SONAME
'validate_password.so';

mysql> SHOW PLUGINS;
+-----+-----+-----+-----+-----+
--+
| Name                | Status | Type                | Library | License |
```

```

+-----+-----+-----+-----+
--*
| mysql_native_password | ACTIVE | AUTHENTICATION | NULL | GPL
|
| mysql_old_password   | ACTIVE | AUTHENTICATION | NULL | GPL
|
| sha256_password      | ACTIVE | AUTHENTICATION | NULL | GPL
|
...
| validate_password    | ACTIVE | VALIDATE PASSWORD | validate_ | GPL
|
                                                                    password.so
+-----+-----+-----+-----+
--+
```

Una vez activado, si la contraseña elegida no es lo bastante segura, no se puede crear:

```

mysql> SET PASSWORD = PASSWORD('qwerty');
ERROR 1819 (HY000): Your password does not satisfy the current policy
requirements
```

```

mysql> SET PASSWORD = PASSWORD('P4s5W0rD');
Query OK, 0 rows affected (0,00 sec)
```

Observe: ¡sigue siendo posible tener una cuenta de usuario sin contraseña!

```

mysql_simple_user> SET PASSWORD = PASSWORD('');
Query OK, 0 rows affected (0,00 sec)
```

El plug-in `validate_password` incluye seis variables:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'validate%';
```

Variable_name	Value
validate_password_dictionary_file	
validate_password_length	8
validate_password_mixed_case_count	1
validate_password_number_count	1
validate_password_policy_number	MEDIUM
validate_password_special_char_count	1

La permisividad de MySQL a nivel de la robustez de la contraseña será administrada por la variable `validate_password_policy_number`.

Si esta tiene un valor:

- LOW o 0, solo se verifica la longitud de la contraseña. El tamaño debe ser igual o superior al valor de la variable `validate_password_length`.
- MEDIUM o 1, se comprueba el tamaño (`validate_password_length`) y la presencia de al menos un número (`validate_password_number_count`), una letra minúscula y una letra mayúscula (`validate_password_mixed_case_count`), un carácter especial (`validate_password_special_char_count`).
- STRONG o 2, se comprueba el tamaño (`validate_password_length`) la presencia de al menos un número (`validate_password_number_count`), una letra minúscula y una letra mayúscula (`validate_password_mixed_case_count`), un carácter especial (`validate_password_special_char_count`) y que no esté en una lista de palabras prohibidas introducidas en la variable `validate_password_dictionary_file`.

El otro aporte del plug-in `validate_password` es la función

`VALIDATE_PASSWORD_STRENGTH()`, que devuelve un número entero que indica

la robustez de la contraseña pasada como parámetro. Los valores devueltos por la función son:

- 0 si el tamaño de la contraseña es inferior a 4.
- 25 si el tamaño de la contraseña es inferior o igual a 4 e inferior al valor de la variable `validate_password_length`.
- 50 si se satisfacen las limitaciones impuestas en el modo `validate_password_policy_Number = 'LOW'`.
- 75 si se satisfacen las limitaciones impuestas en el modo `validate_password_policy_number = 'MEDIUM'`.
- 100 si se satisfacen las limitaciones impuestas en el modo `validate_password_policy_number = 'STRONG'`.

```
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('qwerty') ;
+-----+
| VALIDATE_PASSWORD_STRENGTH('qwerty') |
+-----+
|                                25 |
+-----+
```

```
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('P4s5W0rD') ;
+-----+
| VALIDATE_PASSWORD_STRENGTH('P4s5W0rD') |
+-----+
|                                50 |
+-----+
```

```
mysql [localhost] {t} ((none)) > SELECT
VALIDATE_PASSWORD_STRENGTH('_P4s5W0rD') ;
+-----+
| VALIDATE_PASSWORD_STRENGTH('_P4s5W0rD') |
+-----+
|                                100 |
+-----+
```


6. Expiración de contraseña

A partir de MySQL 5.6, podemos obligar al usuario a cambiar su contraseña con el comando `ALTER USER`. Al expirar, el usuario no podrá realizar ninguna petición hasta que no indique una contraseña. Observe, sin embargo, que nada impide al usuario introducir de nuevo la misma contraseña:

```
-- usuario DBA
mysql_DBA> CREATE USER olivier@localhost IDENTIFIED BY 'P4s5W0rD'\G
Query OK, 0 rows affected (0,00 sec)

mysql_DBA> SELECT user, host, password, password_expired FROM
mysql.user
WHERE user='olivier' AND host='localhost'\G
***** 1. row *****
      user: olivier
      host: localhost
      password: *F02F5BFE01D376A3C4D9AD9E930D25C9D0CED644
password_expired: N

mysql_DBA> GRANT SELECT ON world.* TO olivier@localhost;

-- usuario olivier
mysql_olivier> SELECT count(*) FROM world.City;
+-----+
| count(*) |
+-----+
|      4079 |
+-----+

-- usuario DBA
mysql_DBA> ALTER USER olivier@localhost PASSWORD EXPIRE;

mysql_DBA> SELECT user, host, password, password_expired FROM
```

```

mysql.user
WHERE user='olivier' AND host='localhost'\G
***** 1. row *****

      user: olivier
      host: localhost
      password: *F02F5BFE01D376A3C4D9AD9E930D25C9D0CED644
password_expired: Y

-- usuario olivier
mysql_olivier> SELECT count(*) FROM world.City;
ERROR 1820 (HY000): You must SET PASSWORD before executing this
statement

mysql_olivier> SET PASSWORD = PASSWORD('nEw_P445s');
Query OK, 0 rows affected (0,02 sec)

mysql_olivier> SELECT count(*) FROM world.City;
+-----+
| count(*) |
+-----+
|      4079 |
+-----+

-- usuario DBA
mysql_DBA> SELECT user, host, password, password_expired FROM
mysql.user
WHERE user='olivier' AND host='localhost'\G
***** 1. row *****

      user: olivier
      host: localhost
      password: *5166054D66DB139C1EE7CC840E76FFB465B9A4C0
password_expired: N

```

7. Utilidad de configuración de contraseñas

La herramienta `mysql_config_editor` permite almacenar la información de autenticación en el archivo encriptado `.mylogin.cnf`. Este archivo se crea en el «home directory» del usuario para los sistemas operativos derivados de UNIX y en `%APPDATA%\MySQL` en Microsoft Windows. Los datos almacenados, el nombre de la cuenta de usuario (`user`), la contraseña (`password`) y el equipo a partir del cual el cliente puede conectarse (`host`) permitirán al cliente conectarse al servidor sin tener que proporcionar la contraseña en texto visible.

La estructura del archivo cifrado `.mylogin.cnf` es similar a la del archivo de configuración `my.cnf` (o `my.ini` en MS Windows). Está compuesto por secciones (o grupos), los *login paths*. Cada sección tiene un nombre que debe ser único y que corresponde al perfil del cliente que va a buscar su información de conexión y los tres datos antes citados, es decir, el nombre de la cuenta de usuario (`user`), la contraseña (`password`) y el equipo desde el que se conecta el cliente (`host`).

El proceso de almacenamiento de los datos de autenticación se realiza con la opción `SET`:

```
$ mysql_config_editor set --login-path=localtest --host=localhost
--user=olivier --password
Enter password:
```

Para ver el contenido de un login path, usamos la opción `PRINT`:

```
$ mysql_config_editor print --login-path=localtest
[localtest]
user = olivier
password = *****
host = localhost
```

El cliente usa la opción `login-path` para recuperar la información de conexión del archivo `.mylogin.cnf`:

```
mysql --login-path=localtest
Welcome to the MySQL monitor.  Commands end with ; or \g.
...
mysql>
```

Los comandos principales de la herramienta `mysql_config_editor` son:

- `set`: devuelve los datos de autenticación de un login path.
- `print`: muestra el contenido del archivo.
- `remove`: elimina un login path.
- `reset`: elimina el archivo (si existe) y lo vuelve a crear.

8. Asignación de los derechos

El comando `GRANT` permite asignar derechos y también crear una cuenta si esta no existe todavía. La sintaxis es la siguiente: `GRANT tipo_de_derechos ON alcance_de_derechos TO cuenta_usuario` con:

- `tipo_de_derechos`: derechos asignados (`SELECT`, `PROCESS`, `EXECUTE`...).
- `alcance_de_derechos`: todo el servidor (`* . *`), todos los objetos de un esquema: (`world . *`), una tabla concreta (`world.City`)...
- `Cuenta_usuario`: la cuenta o las cuentas a las que se asignan los derechos (`'olivier'@'localhost'`, `'pedro'@'localhost'`...).

Los derechos pueden clasificarse en cinco categorías, en función de su alcance y su función.

a. Los derechos de administración

Permiten administrar el servidor MySQL. Deben, en principio, estar reservados a las cuentas de administración. Se definen a nivel global, es decir, para todos los objetos del servidor. El cuadro siguiente muestra la lista de los diferentes derechos de administración.

Derecho	Descripción
CREATE TEMPORARY TABLES	Crear tablas temporales (CREATE TEMPORARY TABLE).
CREATE USER	Crear, editar, eliminar las cuentas de usuario (CREATE, DROP, RENAME).
FILE	Leer o escribir en los archivos almacenados en el equipo host del servidor MySQL (SELECT ... INTO OUTFILE, LOAD DATA).
GRANT OPTION	Permite transmitir sus derechos a otras cuentas.
LOCK TABLES	Bloquear las tablas (LOCK TABLES).
PROCESS	Ver los threads (SHOW PROCESSLIST).
PROXY	Activar el modo usuario proxy.
RELOAD	Reiniciar los registros, las tablas, estadísticas... (FLUSH o RESET).
REPLICATION CLIENT	Supervisar y administrar la replicación (SHOW MASTER STATUS, SHOW SLAVE STATUS).
REPLICATION SLAVE	Utilizado por la cuenta de replicación para recuperar los eventos del registro binario del maestro.
SHOW SCHEMAS / DATABASES	Ver la lista de esquemas (base de datos).
SHUTDOWN	Detener el servidor (mysqladminshutdown).
SUPER	Ejecutar varios comandos de administración (CHANGE

```
MASTER TO, KILL (otro thread que no sea el suyo),  
SET GLOBAL...).
```

Otorgar derechos de administración

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'administrador'@'localhost'; /*  
Otorga todos los derechos al conjunto del servidor*/
```

```
mysql> GRANT REPLICATION SLAVE ON *.* TO 'replicador'@'%'; /* Otorga  
los  
derechos requeridos para la replicación */
```

```
mysql> GRANT SHUTDOWN ON *.* TO 'detener_servidor'@'dasini.net'; /*  
tiene  
el derecho de detener el servidor MySQL */
```

- ☐ La opción `secure_file_priv` permite leer (con la petición `LOAD DATA` o la función `LOAD_FILE()`) o escribir (con la petición `SELECT ... INTO OUTFILE`) en los archivos contenidos en el directorio especificado:

```
[mysqld]
```

```
secure-file-priv=/mi/directorio/
```

```
mysql> SELECT * FROM City INTO OUTFILE '/tmp/city.txt';  
ERROR 1290 (HY000): The MySQL server is running with the --secure-  
file-priv option so it cannot execute this statement
```

```
mysql> SELECT * FROM City INTO OUTFILE '/mi/directorio/city.txt';  
Query OK, 4079 rows affected (0,02 sec)
```

b. Los derechos a nivel de los esquemas

A veces es necesario definir derechos para todo un esquema (o varios). Los derechos se refieren entonces al conjunto de objetos pertenecientes a este esquema. El cuadro siguiente muestra la lista de los derechos de los esquemas.

Derecho	Descripción
ALTER	Modificar los esquemas y las tablas (ALTER SCHEMA/DATABASE/TABLE).
CREATE	Crear esquemas y tablas (CREATE SCHEMA/DATABASE/TABLE).
CREATE TEMPORARY TABLE	Crear tablas temporales (CREATE TEMPORARY TABLES).
CREATE VIEW	Crear vistas (CREATE VIEW).
DELETE	Borrar los registros de una tabla (DELETE).
DROP	Eliminar esquemas y tablas (DROP SCHEMA/DATABASE/TABLE).
EVENT	Programar los eventos del administrador de eventos (<i>event scheduler</i>) (CREATE EVENT).
GRANT OPTION	Permite transmitir sus derechos a otras cuentas.
INDEX	Crear y eliminar índices (CREATE/DROP INDEX).
INSERT	Insertar registros en una tabla (INSERT).
LOCK TABLES	Bloquear las tablas (LOCK TABLES).

SELECT	Mostrar los registros de una tabla, ver la estructura de las tablas (SELECT, DESCRIBE, SHOW CREATE TABLE...).
SHOW VIEW	Ver el código SQL de una vista (SHOW CREATE VIEW).
UPDATE	Modificar los registros (UPDATE).

Otorgar derechos sobre los esquemas

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON base_sms.* TO
'stat'@'10.175.201.%', 'poll'@'10.175.201.%'; /* Proporciona los
derechos
de leer y escribir en todas las tablas del esquema base_sms para los
usuarios stat y poll que se conectan a partir de clientes
con una dirección IP que comienza por 10.175.201. */

mysql> GRANT SELECT ON world.* TO 'lectura_world'@'localhost'; /*
Otorga solo el derecho de lectura a la cuenta que sirve para leer las
tablas del esquema world */

mysql> SELECT GRANT, LOCK TABLES, FILE ON *.* TO
'userdump'@'localhost'; /* Otorga los derechos suficientes para
respaldar todos los objetos, con el cliente mysqldump */
```

c. Los derechos a nivel de las tablas

Puede ocurrir que su aplicación solo inserte datos en una tabla, por ejemplo, en el marco del registro de datos. En tal caso, se pueden especificar derechos para permitir modificar únicamente la tabla y sus datos.

Lista de los derechos para las tablas

Derecho	Descripción
---------	-------------

ALTER	Modificar las tablas (ALTER TABLE).
CREATE	Crear tablas (CREATE TABLE).
DELETE	Borrar los registros de una tabla (DELETE).
DROP	Eliminar tablas (DROP TABLE).
GRANT OPTION	Permite transmitir sus derechos a otras cuentas.
INDEX	Crear y eliminar índices (CREATE/DROP INDEX).
INSERT	Insertar registros en una tabla (INSERT).
SELECT	Mostrar los registros de una tabla, ver la estructura de las tablas (SELECT, DESCRIBE, SHOW CREATE TABLE...).
TRIGGER	Crear y eliminar disparadores o triggers (CREATE/DROP TRIGGER).
UPDATE	Modificar los registros (UPDATE).

Otorgar derechos sobre una tabla

```
mysql> GRANT INSERT ON appli.journal TO 'registro'@'localhost'; /*
Otorga el derecho de insertar datos en la tabla journal del esquema
appli
*/
```

```
mysql> GRANT SELECT ON world.City TO 'lectura_city1'@'localhost',
'lectura_city2'@'localhost'; /* Otorga el derecho de lectura de los
datos
de la tabla City del esquema world a las cuentas lectura_city1 y
```

d. Los derechos a nivel de columnas

Un nivel más bajo nos ofrece la posibilidad de añadir, visualizar y modificar los datos de algunas columnas de una tabla.

Derecho	Descripción
INSERT	Insertar registros en una tabla (INSERT).
SELECT	Mostrar los registros de una tabla, ver la estructura de las tablas (SELECT, DESCRIBE, SHOW CREATE TABLE).
UPDATE	Modificar los registros (UPDATE).

- ☐ Los derechos sobre las columnas impiden a las peticiones ponerse en la caché de peticiones.
- ☐ No es posible tener el derecho DELETE para una columna porque el nivel más bajo de borrado es el registro.

Otorgar derechos sobre las columnas de una tabla

```
mysql> GRANT SELECT(name, CountryCode), UPDATE(CountryCode) ON  
world.City TO 'ramon'@'%'; /* otorga los derechos SELECT en las  
columnas name, CountryCode y UPDATE en la columna CountryCode de  
la tabla City del esquema world */
```

e. Los derechos para las rutinas almacenadas

El sistema de derechos se aplicará a su vez para controlar si un usuario puede crear, modificar o suprimir las rutinas almacenadas.

Derecho	Descripción
CREATE ROUTINE	Establecer procedimientos y funciones almacenadas.
ALTER ROUTINE	Modificar o eliminar los procedimientos y funciones almacenadas.
EXECUTE	Ejecutar los procedimientos y funciones almacenadas.
GRANT OPTION	Permite transmitir sus derechos a otras cuentas.

Otorgar derechos a un procedimiento almacenado

```
GRANT CREATE ROUTINE, EXECUTE ON ps.* TO 'creador_ps'@'localhost'; /*
Otorga el derecho de crear y ejecutar los procedimientos almacenados
en el esquema ps */
```

Para dar todos los derechos a una cuenta de usuario, es necesario utilizar la cláusula ALL PRIVILEGES (o ALL). Cada cliente que utiliza la cuenta tiene el conjunto de los privilegios en todo el servidor con *.* , o sobre todos los objetos de un esquema con nombre_esquema.* , con una excepción: el derecho GRANT OPTION . Este último solo podrá concederse con la cláusula WITH GRANT OPTION al final del comando GRANT . Sin embargo, aunque sea muy cómodo, tener todos los derechos puede ser fuente de problemas en caso de errores de manipulación o de compromiso. Además, dos usuarios que dispongan cada uno de la mitad de los derechos y de la cláusula GRANT OPTION podrán intercambiárselos y tenerlos así todos.

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'katherine'@'localhost';
Query OK, 0 rows affected (0,00 sec)
```

```
mysql> SHOW GRANTS FOR 'katherine'@'localhost';
```

```
+-----+
| Grants for katherine@localhost |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'katherine'@'localhost' |
+-----+
1 row in set (0,00 sec)
```

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'katherine'@'localhost' WITH
GRANT OPTION;
```

```
mysql> SHOW GRANTS FOR 'katherine'@'localhost';
```

```
+-----+
+
| Grants for katherine@localhost |
+-----+
+
| GRANT ALL PRIVILEGES ON *.* TO 'katherine'@'localhost' WITH
GRANT OPTION |
+-----+
+
```

9. Limitación del uso de recursos

Además de la gestión de acceso a los objetos de la base de datos, MySQL permite gestionar los recursos, ofreciéndonos la posibilidad de limitar el número total de peticiones

(MAX_QUERIES_PER_HOUR), peticiones de escritura

(MAX_UPDATES_PER_HOUR) y de conexiones

(MAX_CONNECTIONS_PER_HOUR) por hora.

La opción MAX_USER_CONNECTIONS permite limitar el número de conexiones simultáneas con la misma cuenta de usuario.

Limitación de recursos a dos peticiones por hora

```
mysql> GRANT SELECT on world.City TO 'gen_html'@'localhost' IDENTIFIED
by 'M0T2p4ss3' WITH GRANT OPTION MAX_QUERIES_PER_HOUR 2;
```

```
$ mysql -u gen_html -Pm0T2p4ss3 -H -e"SELECT * FROM world.City" >
Ciudad.html /* genera el archivo de la ciudad.html */
```

```
$ mysql -u gen_html -Pm0T2p4ss3 -H -e"SELECT * FROM world.City" >
Ciudad.html /* devuelve un error, ya que se ha alcanzado la cuota */
ERROR 1226 (42000) at line 1: User 't' has exceeded the
'max_questions'
resource (current value: 2)
```

Cabe señalar que la conexión al servidor consume una petición de la cuota.

El reinicio de los contadores se realiza con el comando `FLUSH USER_RESOURCES`. Otras alternativas consisten en recargar las tablas de derechos con el comando `FLUSH PRIVILEGES` o con el cliente `mysqladmin reload`, ejecutando de nuevo el comando `GRANT` y, por supuesto, reiniciando el servidor.

Por último, el comando `GRANT` también ofrece la posibilidad de configurar las opciones SSL para comprobar los certificados. Este tema se aborda en la sección Usar SSL.

10. Visualización de los derechos

El comando `SHOW GRANTS` nos permite ver nuestros propios derechos o, si tenemos el derecho `SELECT` en el esquema `mysql`, los derechos de las demás cuentas de usuario.

```
mysql> SHOW GRANTS;
+-----+
| Grants for daz@localhost |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'daz'@'localhost' IDENTIFIED BY
```

```
PASSWORD '*35DC09EF57CBBE40E97DC3B8658F6A7CD1FB9670' WITH GRANT  
OPTION |
```

```
+-----+  
-----+
```

```
mysql> SHOW GRANTS FOR 'clara'@'localhost';
```

```
+-----+  
| Grants for clara@localhost |  
+-----+  
| GRANT USAGE ON *.* TO 'clara'@'localhost' |  
| GRANT ALL PRIVILEGES ON `lolcat`.* TO 'clara'@'localhost' |  
+-----+
```

11. Entrada en vigor de los derechos

Los derechos relacionados con las tablas y columnas tendrán efecto de forma inmediata.

```
mysql_root> GRANT SELECT ON world.City to fresh;
```

```
mysql_fresh> SHOW TABLES IN world;
```

```
+-----+  
| Tables_in_world |  
+-----+  
| City            |  
+-----+
```

El cambio de derechos para el esquema actual no surtirá efecto hasta el momento de la reconexión a este:

```
mysql_fresh> SHOW GRANTS;
```

```
+-----+  
| Grants for fresh@% |  
+-----+
```

```
| GRANT USAGE ON *.* TO 'fresh'@'%' |
| GRANT SELECT, INSERT ON `world`.* TO 'fresh'@'%' |
+-----+

mysql_fresh> SELECT * FROM world.City LIMIT 1;
+-----+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Population |
+-----+-----+-----+-----+-----+
| 1 | Kabul | AFG | Kabul | 1780000 |
+-----+-----+-----+-----+-----+

mysql_root> REVOKE SELECT ON world.* FROM fresh;

mysql_fresh> SHOW GRANTS;
+-----+
| Grants for fresh@% |
+-----+
| GRANT USAGE ON *.* TO 'fresh'@'%' |
| GRANT INSERT ON `world`.* TO 'fresh'@'%' |
+-----+

mysql_fresh> SELECT * FROM world.City LIMIT 1;
+-----+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Population |
+-----+-----+-----+-----+-----+
| 1 | Kabul | AFG | Kabul | 1780000 |
+-----+-----+-----+-----+-----+

mysql_fresh> USE world
Database changed

mysql_fresh> SELECT * FROM world.City LIMIT 1;
ERROR 1142 (42000): SELECT command denied to user 'fresh'@'localhost'
for table 'City'
```

En cuanto a las contraseñas y a los derechos globales, solo se aplican al volver a conectar al servidor MySQL.

```
mysql_root> GRANT ALL ON *.* TO fresh;
```

```
mysql_fresh> SHOW GRANTS;
```

```
+-----+
| Grants for fresh@%                |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'fresh'@'%' |
+-----+
```

```
mysql_fresh>> USE world
```

```
ERROR 1044 (42000): Access denied for user 'fresh'@'%' to database
'world'
```

12. Eliminación de los derechos

Los derechos de una cuenta pueden eliminarse con el comando REVOKE. Su sintaxis es similar a la del comando GRANT. La diferencia es sutil y lingüística: la palabra clave TO es sustituida por FROM:

```
REVOKE tipo_de_derechos ON alcance_de_derechos
FROM cuenta_usuario
```

Con:

- *tipo_de_derechos*: derechos asignados (SELECT, PROCESS, EXECUTE...).
- *alcance_de_derechos*: todo el servidor (* . *), todos los objetos de un esquema (world . *), una tabla concreta (world.City)...
- *cuenta_usuario*: la cuenta o las cuentas a las que se otorguen los derechos ('olivier'@'localhost').

REVOKE borra los derechos, pero en ningún caso elimina el usuario. Esta tarea le corresponde al comando DROP USER.

Ejemplo

```
mysql> SHOW GRANTS FOR 'administrador'@'localhost';
+-----+
| Grants for administrador@localhost |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'administrador'@'localhost' |
+-----+

mysql> REVOKE ALL PRIVILEGES ON *.* FROM 'administrador'@'localhost';

mysql> SHOW GRANTS FOR 'administrador'@'localhost';
+-----+
| Grants for administrador@localhost |
+-----+
| GRANT USAGE ON *.* TO 'administrador'@'localhost' |
+-----+

mysql> DROP USER 'administrador'@'localhost';

mysql> SHOW GRANTS FOR 'administrador'@'localhost';
error 1141 (42000): There is no such grant defined for user
'administrador' on host 'localhost'
```

Para eliminar varios derechos a la vez, debemos listarlos con una sintaxis similar al comando GRANT, mientras que, para borrar todos los derechos de una cuenta, REVOKE permite una sintaxis alternativa: REVOKE ALL PRIVILEGES, GRANT OPTION FROM cuenta_usuario

Siempre nos podremos conectar al servidor con esta cuenta, pero no nos permitirá manipular algunos datos del esquema INFORMATION_SCHEMA o ejecutar determinados comandos (SHOW SCHEMAS, SHOW TABLES...).

```
mysql> SHOW GRANTS FOR 'userdump'@'localhost';
```

```

+-----+
| Grants for userdump@localhost |
+-----+
| GRANT SELECT, FILE, LOCK TABLES ON *.* TO 'userdump'@'localhost' |
+-----+

mysql> REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'userdump'@'localhost'
OK, 0 rows affected (0,00 sec)

mysql> SHOW GRANTS FOR 'userdump'@'localhost';
+-----+
| Grants for userdump@localhost |
+-----+
| GRANT USAGE ON *.* TO 'userdump'@'localhost' |
+-----+

```

13. Mejores prácticas de gestión de los derechos

Como hemos visto en las secciones anteriores, la gestión de derechos tiene un impacto directo en las tablas del esquema de sistema `mysql`. Este último debe protegerse para evitar cualquier manipulación desafortunada o maliciosa. En otras palabras, una cuenta de usuario estándar no debe en ningún caso tener derechos sobre estas; si no, le estamos dando acceso total al servidor MySQL...

En términos generales, el principio que debe prevalecer es el del menor derecho: un usuario solo debe tener el derecho de hacer lo que tiene que hacer ¡y nada más!

Algunos derechos son más peligrosos que otros, por lo que solo deben concederse si es imprescindible:

- **SUPER**: ofrece la posibilidad al usuario torpe o malintencionado de colgar el servidor (cambiar el tamaño de los buffers, cambiar el número máximo de conexiones simultáneas permitidas...).
- **PROCESS**: permite ver las peticiones en curso de ejecución: concretamente, la contraseña en texto visible de los comandos `CREATE USER` y `GRANT`.

- `FILE`: puede utilizarse para leer un archivo sensible y cargar su contenido en una tabla.
- `WITH GRANT OPTION`: permite dar sus derechos a otra cuenta.

Por último, opte por el cliente `mysqladmin password` para cambiar las contraseñas, en lugar del cliente de texto `mysql`, porque los comandos de este último se registran en el archivo `~/.mysql_history`.

Eliminar el registro de comandos del cliente de texto `mysql`

Las órdenes con el cliente de texto (`mysql`) se registran en el archivo `~/.mysql_history`. Debemos garantizar la protección al acceso a este archivo. Para desactivar esta funcionalidad, existen varias posibilidades:

- Cierre el cliente de texto `mysql`, elimine el archivo `~/.mysql_history` y vuelva a crearlo como enlace simbólico a `/dev/null:ln -s /dev/null $HOME/.mysql_history`
- Otro método: edite el archivo de arranque shell dando valor `' /dev/null '` a la variable de entorno `MYSQL_HISTFILE`.
- O simplemente ejecute después del cierre del cliente de texto: `cat /dev/null > ~/.mysql_history`

Una alternativa consiste en ejecutar el cliente `mysql` con la opción `--batch` (o `-B`).

Securización de las vistas y rutinas almacenadas

Como hemos visto en la sección sobre derechos, el acceso a los objetos de la base de datos está protegido a nivel de cuentas de usuario por un sistema que nos puede autorizar o no. Sin embargo, aunque tengamos derechos para acceder al objeto (vistas, procedimientos y funciones almacenadas), no es seguro que tengamos el derecho para acceder a los datos a los que se refiere. Para ejecutar una vista (o una rutina), es necesario que el usuario tenga derecho a manipular todos los objetos a los que accede la vista (o la rutina), siempre que la vista (o la rutina) se ejecute a través de un usuario que tenga los derechos adecuados (casi como si usáramos el comando de Linux `su`).

Para una vista, es la cláusula SQL `SECURITY` la que permite configurar este comportamiento con los valores:

- **INVOKER**: la vista se ejecuta con los derechos del usuario.
- **DEFINER**: la vista se ejecuta con los derechos de su creador.

```
mysql_root> CREATE SQL SECURITY INVOKER VIEW City_ESP1 AS SELECT name  
FROM world.City WHERE countrycode='ESP';
```

```
mysql_root> CREATE SQL SECURITY DEFINER VIEW City_ESP2 AS SELECT name  
FROM world.City WHERE countrycode='ESP';
```

```
mysql_user_test> SELECT * FROM City_ESP1 LIMIT 1;  
ERROR 1356 (HY000): View 'test.City_ESP1' references invalid table(s)  
or column(s) or function(s) or definer/invoker of view lack rights to
```

use them

```
mysql_user_test> SELECT * FROM City_ESP2 LIMIT 1;
```

```
+-----+
```

```
| name  |
```

```
+-----+
```

```
| Madrid |
```

```
+-----+
```

```
1 row in set (0,00 sec)
```

La cláusula SQL SECURITY existe también para las rutinas. Su comportamiento es, por supuesto, el mismo. Los procedimientos y funciones almacenados se abordan en el capítulo Otras funcionalidades - Rutinas almacenadas/Disparadores (triggers)/Eventos.

Líneas generales

1. Introducción

¿Cuántas veces hemos perdido datos y nos hemos dado cuenta de que no existía ningún medio de recuperarlos? Es el riesgo que cualquier administrador corre cuando no se toma el tiempo de implementar una política de protección adecuada. En efecto, algunos creen que al establecer una replicación entre dos servidores MySQL quedan protegidos de una pérdida o corrupción de datos, pero no es así. Un fallo en las operaciones en uno de los servidores, y lo perderán todo. ¡No haga como ellos!

Del mismo modo, no se puede estar seguro de recuperar los datos mientras no se ha probado el procedimiento de restauración. Haber guardado sus datos es una buena idea, pero si el procedimiento de restauración no ha sido debidamente probado y validado, esto no servirá de gran cosa. Este capítulo tiene por objeto familiarizarnos con los conceptos inherentes al respaldo y presenta los distintos medios disponibles que nos permitirán llegar a la solución más conveniente.

Recuerde guardar los archivos de configuración. El comportamiento de la base de datos puede cambiar por completo (rendimiento, generación de los archivos binarios...) si no utilizamos después de la restauración los mismos parámetros que hemos usado para el respaldo. Esto también puede ser el caso de la configuración de los sistemas de archivos. Por ejemplo, en Linux tenemos la posibilidad de desactivar la actualización de las fechas de acceso (parámetro `mtime` para `ext2/ext3`) en los archivos de la base de datos que resultan inútiles para su correcto funcionamiento. Se trata, en la medida de lo posible, de guardar las partes de su entorno que influyen en el funcionamiento de la base de datos. De esta forma, durante la restauración evitaremos sorpresas.

La frecuencia de los respaldos, la duración de retención de los registros binarios (binlogs) y

la frecuencia de sincronización de archivos de respaldo en servidores remotos deben estar en consonancia con la pérdida de datos máxima admisible (RPO, *Recovery Point Objective*). En efecto, si el RPO es de dos horas, no tenemos activados los registros binarios y estamos haciendo una copia de seguridad diaria, en caso de un crash que ocurra cuatro horas después del respaldo, no podremos cumplir el RPO. Una solución consistiría en seguir estas recomendaciones:

- Activar los registros binarios (`log-bin`).
- Forzar el cambio del registro cada dos horas añadiendo en el gestor de tareas (`crontab` en UNIX) el siguiente comando:

```
mysql --defaults-file=/etc/mysql/debian.cnf -e "FLUSH LOGS"
```

- Sincronizar los registros binarios en un servidor remoto cada dos horas.
- Realizar dos respaldos diarios (a replicar también en otro servidor en caso de pérdida del servidor fuente) o más, según la cantidad de modificaciones efectuadas en la base de datos.

Por último, si hemos establecido la replicación, podemos llevar a cabo los respaldos en un esclavo para no sobrecargar el maestro o no interrumpir el servicio si realiza una copia de seguridad física en frío (servidor detenido).

¡No olvide que la replicación no constituye una copia de seguridad! Tener una copia de sus datos en un esclavo no exime en absoluto de hacer respaldos regulares. ¿Por qué? Si alguien hace un `DROP TABLE` en una tabla grande de su aplicación, este comando se replicará de forma inmediata y los datos se borrarán del servidor esclavo. Confundir la replicación y la copia de seguridad es por desgracia un error muy común. ¡No caiga en esta trampa! Vea la replicación más bien como una ayuda al respaldo (de todas las cosas que puede aportar la replicación). Por otra parte, algunos crean el esclavo de la replicación con un retraso voluntario, bien utilizando `pt-slave-delay`, o a partir de MySQL 5.6 con el comando `CHANGE MASTER TO MASTER_DELAY = N`. Esto puede permitir recuperar de forma sencilla los datos antes de que sean borrados por la replicación, sin recurrir a técnicas de restauración a menudo engorrosas.

Otro parámetro importante cuando se trata de restauración es el RTO (*Recovery Time*

Objective): se trata del tiempo que asignamos para reconstruir los datos en caso de problema. Por ejemplo, es inútil pasar tiempo estudiando una solución para que la restauración se ejecute en 12 horas si necesita restablecer la totalidad de los datos en menos de dos horas.

Por último, recuerde que el objetivo final de los respaldos es poder restaurar los datos: si no probamos de vez en cuando una restauración, corremos el riesgo de no darnos cuenta de que los respaldos no son correctos hasta el día en que realmente necesitemos restaurar sus datos.

2. Respaldo lógico

El respaldo lógico consiste en extraer de la base de datos las instrucciones SQL para regenerar la estructura, los datos, los procedimientos, las funciones almacenadas, etc.

La exportación de datos requiere que el servidor MySQL esté arrancado porque es él quien accede al contenido de los archivos de datos. La herramienta que se utiliza más frecuentemente es `mysqldump`. Detallaremos sus principales funcionalidades al final de este capítulo. Además, esta permite realizar simples `SELECT *` en las tablas que se deben respaldar.

Los respaldos lógicos tienen las siguientes ventajas:

- Son archivos de texto que pueden, por tanto, visualizarse de forma simple y editarse mediante las herramientas convencionales (`cat`, `sed`, `grep`, `awk`; por ejemplo, en Linux).
- El respaldo es flexible en principio porque `mysqldump` dispone de un gran número de opciones. De esta forma, podemos guardar una porción de una tabla, una o varias tablas, una o varias bases de datos o la base de datos completa.
- La restauración es sencilla.
- La restauración es flexible, ya que podemos inyectar el respaldo en un servidor con una configuración diferente e incluso un motor de almacenamiento distinto.

Sin embargo, este tipo de respaldo también tiene inconvenientes. El más importante es el tiempo necesario para la restauración, lo que, en la práctica, limita los respaldos lógicos a las bases de datos de pequeño tamaño (hasta 20 GB para dar una estimación de magnitud).

3. Respaldo físico

El respaldo físico consiste en guardar de forma directa los archivos con las herramientas de sistemas estándar (`cp`, `tar`, `gzip`, por ejemplo). El tamaño del respaldo físico suele ser mayor que el respaldo lógico, ya que se copian todos los segmentos (tablas, índices, rollback). Este puede hacerse con la base de datos abierta (en línea) o cerrada. En el caso de un respaldo con la base cerrada, basta con detener el servidor y respaldar todo el árbol del directorio indicado en la variable `data_directory` (identificable en el archivo de configuración). Sin embargo, en el caso de un respaldo con la base abierta, hay que tomar ciertas precauciones porque hay una interacción entre el respaldo y el servidor MySQL activo.

Para efectuar una analogía con el respaldo lógica, veamos las ventajas de los respaldos físicos:

- La restauración es sencilla, ya que basta con copiar los archivos guardados en el lugar correcto, verificar los permisos y reiniciar el servidor.
- Los binarios guardados pueden ser portados a otros sistemas operativos mientras los motores de almacenamiento utilizados sean MyISAM o InnoDB.

Por supuesto, los respaldos físicos también tienen desventajas:

- Su tamaño es mucho mayor que el de una copia de seguridad lógica (que, además, se comprime con mucha facilidad).
- Un solo archivo corrupto puede inutilizar el conjunto del respaldo. Por ello, es muy importante probar los respaldos.

4. Respaldo completo/incremental

Si cada vez que almacenamos la base de datos completa debemos observar una cierta retención de los datos guardados (por ejemplo, seis semanas), hemos de saber que el espacio de almacenamiento necesario puede aumentar con el tiempo hasta alcanzar un punto de ruptura. Una duración de los procesos de respaldo que supere la ventana disponible, el espacio necesario para que supere el disponible y la duración estimada de la

restauración pueden ser puntos de ruptura que debemos anticipar. Debemos tener en cuenta todos estos elementos disponibles a la hora de decidir efectuar una copia de seguridad lógica o física, completa o incremental.

Si deseamos utilizar respaldos incrementales, tendremos siempre que comenzar con una copia de seguridad completa. Por ejemplo, podemos realizar cada domingo una copia de seguridad completa, mientras que el resto de los días de la semana será más práctico una copia incremental. Esto puede ser útil cuando tenemos pocos cambios diarios, y nos puede permitir mantener un mayor número de respaldos en el disco del servidor en cuestión antes de tener que utilizar una librería de respaldos.

Tomemos como ejemplo un servidor en el que un respaldo completo ocupa 200 GB y en el que se modifican 100 MB de datos a diario. Si además debemos observar una retención de seis semanas, una copia de respaldo completa diaria necesitaría 8,2 TB ($6 \times 7 \times 200$ GB). Si ahora elegimos realizar una copia de respaldo completa cada domingo y una copia incremental el resto de la semana, será necesario almacenar 1,2 TB ($((6 \times 0,1 + 200) \times 6)$).

De hecho, sea cual sea el número de semanas que se desee conservar, será necesario almacenar dos veces más datos en comparación con respaldos completos diarios. ¡Esto es importante!

En cuanto a la restauración, si perdemos la BBDD un jueves y debemos restaurar los datos de la semana anterior (en caso de corrupción, por ejemplo), deberemos restablecer el respaldo completo del domingo anterior al jueves y restaurar los respaldos de incrementales lunes, martes, miércoles y jueves para aplicarlos hasta la hora exacta decidida.

¿Están adaptados los respaldos incrementales a nuestro sistema? Si tenemos un volumen bajo (< 100 GB), sin duda. El principal interés de los respaldos incrementales es reducir el espacio necesario para los respaldos. El inconveniente principal es que la restauración es mucho más compleja porque, en lugar de solo restaurar una copia de seguridad, tendremos que recuperar varias, una tras otra, lo que aumenta de forma considerable el riesgo de error. Si la base de datos es pequeña, la ganancia de espacio en disco no justifica el riesgo asumido en el respaldo. Por el contrario, si tenemos más de 1 TB de datos, contamos con buenas posibilidades de poder ahorrar mucho espacio en disco con respaldos incrementales, lo que puede contrarrestar el riesgo asumido en la restauración.

En la inmensa mayoría de los casos, si elegimos respaldos incrementales con la intención de hacer al menos un respaldo completo por semana, debemos probar el proceso de

restauración. De lo contrario, el riesgo de no poder restaurar los datos es muy grande.

5. Respaldo y replicación

El objetivo principal de los respaldos es, por supuesto, poder recuperar los datos en caso de problema, pero también existe otro caso en el que los respaldos son necesarios: la replicación. Todos los detalles sobre la replicación se tratan en el capítulo Replicación. Si no está familiarizado con la replicación, es conveniente que lea primero este capítulo antes de abordar esta sección.

Cuando queramos preparar un esclavo, después de haber procedido a la configuración de MySQL, debemos:

- Restaurar una copia de seguridad.
- Proporcionar la ubicación en los registros binarios del maestro a partir del cual debe iniciar la replicación.

No podemos adivinar esta ubicación a partir del estado de los datos. Dicho de otra forma, si el respaldo se hizo sin precaución, no podemos deducir la posición donde debe comenzar la replicación. Por lo tanto, no todos los respaldos permiten iniciar la replicación.

Algunas herramientas incorporan de forma nativa la posición de replicación en la copia de seguridad (es el caso de Percona XtraBackup, por ejemplo), pero para otras necesitaremos indicar de forma explícita una opción adicional para obtener esta posición (es el caso de `mysqldump`).

- ☐ Preste mucha atención al servidor que desee respaldar: la posición no se obtiene de la misma manera si el servidor es un maestro que si el servidor es un esclavo. Y las herramientas que añaden la posición de replicación de forma automática no comprueban, por lo general, cuál es el rol del servidor, lo que también puede conducir a respaldos que luego no pueden utilizarse. Necesitaremos, por lo tanto, prestar atención al escribir los scripts de respaldo. Veremos en la continuación de este capítulo cuáles son las opciones que se deben utilizar para `mysqldump` y Percona XtraBackup.

Incluso si no utilizamos la replicación, debemos incluir siempre la posición en los registros binarios en los respaldos. En efecto, es la única manera de poder realizar una restauración hasta un punto determinado (PITR o *Point In Time Recovery*).

Imaginemos la situación siguiente: hacemos una copia de todos los días a medianoche, y un día, a las 20 h, los discos dejan de funcionar. Entonces debemos restablecer el respaldo, pero, si nos contentamos con restaurar la copia de seguridad de la noche anterior, perderemos 20 horas de datos, lo que no suele ser aceptable. ¿Qué hacer? El procedimiento de recuperación se llevará a cabo en dos etapas:

- Restablecemos el último respaldo. Tendremos el estado de la base de datos a medianoche de la noche anterior.
- Aplicamos los registros binarios de medianoche hasta el momento en el que el servidor se detuvo. Es aquí cuando la posición en los registros binarios en el momento del respaldo tiene su importancia: si no conocemos esta posición, es imposible saber qué registros binarios deben aplicarse para recuperar el último estado conocido de la base de datos. Supongamos que la posición grabada es (`mysql-bin.000034`, 123456) y que el último registro binario disponible es `mysql-bin.000037`. Entonces debemos ejecutar el comando siguiente:

```
$ mysqlbinlog -D --start-position=123456 mysql-bin.000034  
mysql-bin.000035 mysql-bin.000036 mysql-bin.000037 | mysql
```

- Encontrará más detalles en la sección de restauración de este capítulo.

Los lectores astutos se harán sin duda la siguiente pregunta: ¿cómo se pueden utilizar los registros binarios del maestro para la recuperación de datos si el servidor se ha vuelto inservible (discos que ya no funcionan, por ejemplo)? ¡La respuesta es que simplemente debemos proteger también los registros binarios!

Para esto, existen varias maneras:

- Copie los registros binarios de forma periódica en otro servidor. Ya que la información en los registros binarios nunca se modifica una vez escrita, podemos

tener constancia del último archivo escrito y copiar solo los archivos nuevos.

- Use `mysqlbinlog` 5.6 o 5.7, que permite transmitir de forma continua los registros binarios a un servidor remoto. Esta solución es mejor que la anterior porque garantiza que siempre tenemos a disposición una copia de las últimas entradas de los registros binarios. `mysqlbinlog` es independiente del servidor MySQL, podemos usar `mysqlbinlog` 5.6 si el servidor se ejecuta con la versión 5.5 o una versión anterior. Un buen ejemplo de script con `mysqlbinlog` se encuentra en la siguiente dirección: <https://www.percona.com/blog/2012/01/18/backing-up-binary-log-files-with-mysqlbinlog/>

6. Respaldo y motores de almacenamiento

Todos los motores de almacenamiento son diferentes cuando se trata de respaldos. Por consiguiente, es importante conocer los motores de almacenamiento que pensamos utilizar para poder escribir los scripts de respaldo de forma correcta.

Los motores de almacenamiento más corrientes son sin duda InnoDB y MyISAM.

Veamos cómo realizar copias de seguridad con estos motores. Si usamos otro motor, lo mejor es consultar la documentación para conocer las posibles restricciones sobre los respaldos.

a. MyISAM

En cuanto a MyISAM, lo primero que debe saber es que es necesario suprimir las escrituras para obtener una copia de seguridad coherente. Si no detenemos las escrituras y el respaldo comienza a las 9:00 y finaliza a las 10:00, tendremos para algunas tablas los datos de las 9:00, para otras tablas los datos de las 9:30, etc. Es decir, tendremos una copia de seguridad, pero será imposible de utilizar.

- ☐ Tenga en cuenta que olvidar interrumpir las escrituras para las tablas MyISAM es un error muy común por desgracia. ¡No caiga en esta trampa!

Interrumpir las escrituras no es un problema cuando la base de datos es pequeña o cuando hay ventanas sin tráfico, pero para los sistemas que deben funcionar de forma continua y

que tienen una base voluminosa, MyISAM hace que los respaldos sean difíciles. La mejor solución es utilizar la replicación y realizar los respaldos en un esclavo dedicado: ahora podemos cortar la replicación en el esclavo, deteniendo así las escrituras sobre el esclavo sin tener ningún impacto en el maestro, efectuar el respaldo y reiniciar la replicación.

b. InnoDB

Ya hemos mencionado que InnoDB es, en general, mucho mejor que MyISAM para las cuestiones de rendimiento o persistencia de datos. Pero InnoDB también hace los respaldos mucho más sencillos. Podemos utilizar la funcionalidad de multiversión de InnoDB para respaldar los datos de manera coherente, mientras el sistema recibe las escrituras. Lo único que debe conocer es la posible opción para activar esta funcionalidad (`--single-transaction` con `mysqldump`, por ejemplo).

c. MyISAM e InnoDB

En algunos casos, el sistema será heterogéneo a nivel de los motores de almacenamiento: algunas tablas utilizarán MyISAM y otras, InnoDB. Esta opción no es ideal desde un punto de vista de configuración, ya que es necesario, por ejemplo, compartir la memoria del servidor entre los dos motores de almacenamiento, pero en algunos casos no podremos simplemente convertir todas las tablas a InnoDB. En este caso, en general, estaremos obligados a cortar las escrituras en todas las tablas antes de guardar los datos, como si la base de datos solo tuviera tablas MyISAM.

Percona XtraBackup aporta una solución interesante: la herramienta es capaz de respaldar primero las tablas InnoDB sin interrumpir las escrituras y luego, las tablas MyISAM cortando las escrituras. Así, obtenemos una copia de seguridad coherente cortando las escrituras solo durante el respaldo de las tablas MyISAM y no durante toda la operación.

- ☐ En realidad, aunque todas nuestras tablas usen InnoDB, el sistema es una mezcla de MyISAM e InnoDB, ya que las tablas de la base de sistema `mysql` utilizan MyISAM. Esto no suele ser un problema, ya que estas tablas no reciben ningún tráfico de escritura.

7. Restauración

Cuando llevamos a cabo la restauración de los datos guardados, podemos volver a ejecutar los registros binarios, o en cualquier caso una parte de ellos, para obtener la base de datos en el estado en el que estaba antes del crash o la pérdida de datos.

Este proceso se llama PITR (*Point-In-Time Recovery*) y debe formar parte de los procedimientos de restauración que lleva a cabo el administrador.

Sin embargo, debemos tener en cuenta:

- No aplicar los registros binarios de forma independiente unos de otros. Una transacción MySQL puede ser almacenada en dos registros binarios, lo que hace que si aplicamos los registros, uno por uno, la transacción será invalidada por no estar COMMITED. Debemos siempre aplicar el conjunto de los registros binarios a la vez.

```
shell> mysqlbinlog mysql-bin.000004 mysql-bin.000005  
mysql-bin.000006 | mysql db1
```

- Aplicar los registros binarios sin volver a ejecutar las órdenes defectuosas. Si realizamos una recuperación, esto puede deberse a una orden que se realizó por error en producción (y sí, ¡esto le pasa a todo el mundo!). Imaginemos que una persona ha eliminado una tabla muy importante, pero que es la única orden que se necesita rechazar. Primero debemos buscar esta orden en los registros binarios disponibles.

```
shell>for binary_log in mysql-bin.0*  
do  
    mysqlbinlog $binary_log | grep -qil DROP && echo $binary_log &&  
break  
done  
mysql-bin.000004
```

Luego, buscamos las posiciones de inicio y final de este evento en el registro correspondiente:

```
shell> mysqlbinlog mysql-bin.000004|grep -B 3 -i DROP
# at 463303
#080923 10:49:52 server id 1 end_log_pos 463381 Query thread_id=23
exec_time=0 error_code=0
SET TIMESTAMP=1222159792/*!*/;
drop table tabla_mega_importante
```

Por último, solo queda aplicar todos los registros, excepto la orden de eliminación de la tabla `_mega_importante` (aquí tenemos menos de 10 registros, por lo que se puede utilizar el carácter `?` del shell en lugar de enumerar todos los archivos):

```
mysqlbinlog -D --stop-position=463303 mysql-bin.00000?|mysql db1
mysqlbinlog -D --start-position=463381 mysql-bin.00000?|mysql db1
```

La opción `-D` añade `SET SQL_LOG_BIN=0` en la salida de `mysqlbinlog` para no ejecutarlo sobre los esclavos que, por ejemplo, pueden no estar corruptos, o simplemente ser restaurados en paralelo al maestro.

En la Práctica

1. Importación/exportación manual

MySQL permite exportar el resultado de una petición en un archivo, que se almacena en el servidor, cuando el usuario tiene el privilegio `FILE`. Este método es más rápido para importar/exportar el contenido de una tabla que una copia de seguridad SQL, ya que no es necesario interpretar o generar las órdenes SQL.

Para ello, tenemos el comando `SELECT . . . INTO OUTFILE` que respeta el formato siguiente:

```
SELECT col1,col2,... INTO OUTFILE 'nombre_archivo'
FIELDS TERMINATED BY cadena1OPTIONNALLY ENCLOSE BY cadena2
LINES TERMINATED BY cadena3FROM nombre_de_tabla.
```

con:

- `col1,col2 . . .`: el nombre de las columnas que deben escribirse en la salida.
- `nombre_archivo`: el nombre del archivo que contendrá los datos exportados.
- `cadena1`: el separador de datos.
- `cadena2`: el delimitador de datos.
- `cadena3`: el separador de los registros del archivo de salida.
- `nombre_de_tabla`: el nombre de la tabla de la que se recuperan los registros.

Este nombre puede ser sustituido por una subconsulta.

Así, la siguiente petición exporta todos los datos de la tabla `actor`:

```
mysql> SELECT * FROM actor INTO OUTFILE '/var/tmp/actor.txt';  
Query OK, 200 rows affected (0.02 sec)
```

Cuando el comando termina, se encuentra el contenido de la tabla en el archivo `/var/tmp/actor.txt`, los valores se separan por tabuladores y los valores NULL se sustituyen por `\N`. El tamaño y el formato del archivo `actor.txt` permiten una operación más simple que la salida SQL que se habría obtenido con `mysqldump`.

En efecto, en UNIX/Linux, podemos utilizar los comandos de sistema basados en expresiones regulares para contar un número de apariciones.

```
shell> awk -F"\t" '/PENELOPE/ {print $2}' /var/tmp/actor.txt |uniq  
-c  
4 PENELOPE
```

O modificar un apellido:

```
shell> sed -i 's/PENELOPE/JENNY/' /var/tmp/actor.txt  
shell> grep -c JENNY /var/tmp/actor.txt  
4
```

Con la salida de `mysqldump`, habríamos necesitado abrir el archivo de export y editarlo de forma manual, o desactivar la opción `extended-insert` para solo obtener inserciones unitarias y facilitar la modificación utilizando los comandos `shell`.

No obstante, ya no debe existir el archivo de salida, en cuyo caso el comando fracasa y el archivo es creado en el equipo del servidor MySQL. Para ello, podemos usar, por ejemplo, el comando `system` o el atajo `\!` en el intérprete `mysql` para borrar el archivo previamente:

```
mysql> SYSTEM rm /var/tmp/actor.txt
```

El contenido podrá recargarse usando el comando `LOAD DATA INFILE`, que es el comando reciproco `SELECT ... INTO OUTFILE`. Estos dos comandos disponen de una amplia gama de opciones que permiten, entre otras cosas, especificar el formato utilizado, así como añadir una operación SQL. Podemos, por ejemplo, generar de forma rápida un archivo CSV a partir de los datos de la base de datos.

```
mysql> SELECT 'id','name','address' UNION ALL  
SELECT id,name,address FROM customer_list LIMIT 10  
INTO OUTFILE '/var/tmp/customer_list.csv' FIELDS TERMINATED BY ','  
OPTIONALLY ENCLOSED BY '"';  
Query OK, 10 rows affected (0,03 sec)
```

Luego, podremos cargar el archivo `/var/tmp/customer_list.csv` usando la primera línea como marcador de las diferentes columnas:

```
shell> cat /var/tmp/customer_list.csv  
"id","name","address"  
"218","VERA MCCOY","1168 Najafabad Parkway"  
"441","MARIO CHEATHAM","1924 Shimonoseki Drive"  
"69","JUDY GRAY","1031 Daugavpils Parkway"  
"176","JUNE CARROLL","757 Rustenburg Avenue"  
"320","ANTHONY SCHWAB","1892 Nabereznyje Telny Lane"  
"528","CLAUDE HERZOG","486 Ondo Parkway"  
"383","MARTIN BALES","368 Hunuco Boulevard"  
"381","BOBBY BOUDREAU","1368 Maracabo Boulevard"  
"359","WILLIE MARKHAM","1623 Kingstown Drive"
```

Por supuesto, también podemos realizar operaciones más complejas, cargando, por ejemplo, un archivo que contenga datos separados por el carácter `@`, al tiempo que se efectúan operaciones en los valores encontrados, para almacenar el resultado de una prueba

o de una operación y recuperando solo algunas columnas. Imaginemos que disponemos del siguiente contenido en el archivo `contenido.dat`:

```
Talla@Peso@Nombre
180@80@Bill
140@80@LARRY
175@90@NICOLAS
168@110@TOM
```

Vamos a tratar de cargar la tabla `imc_test`, que permitirá saber si las personas tienen un índice de masa corporal dentro de la norma. La fórmula para calcular este índice es la siguiente:

$$\text{IMC} = (\text{Peso en kg}) / (\text{Estatura en metros})^2$$

En primer lugar, creamos la tabla `imc_test`:

```
mysql> CREATE TABLE imc_test (nombre VARCHAR(20), buen_imc BOOL);
```

Luego cargamos los datos del archivo `contenido.dat` saltándonos la primera línea, que corresponde a la definición de las columnas, indicando que los valores están separados por el carácter `@` y utilizando la función `IF` para determinar si el IMC es correcto:

```
mysql> LOAD DATA INFILE '/tmp/contenido.dat'
INTO TABLE imc_test FIELDS TERMINATED BY '@'
IGNORE 1 LINES (@talla,@peso,nombre)
SET buen_imc=IF(@peso/pow(@talla/100,2) BETWEEN 18.5
AND 24.9,TRUE,FALSE);
Query OK, 4 rows affected (0,06 sec) Records: 4 Deleted:
0 Skipped: 0 Warnings: 0
```

Ahora podemos comprobar el resultado:

```
mysql> SELECT * FROM imc_test;
```

```
+-----+-----+
| nombre | buen_imc |
+-----+-----+
| BILL   |         1 |
| LARRY  |         0 |
| NICOLAS|         0 |
| TOM    |         0 |
+-----+-----+
4 rows in set (0,00 sec)
```

Solo queda ponerse en contacto con Larry, Nicolas y Tom para anunciarles la mala noticia.

2. mysqldump

`mysqldump` es la herramienta de respaldo para MySQL más conocida y más utilizada. Su éxito se debe a varios factores: se suministra de forma estándar con los servidores; existe desde hace mucho tiempo y dispone de un gran número de opciones que la hacen muy flexible. Este cliente funciona de forma independiente del motor utilizado y pondrá a disposición del administrador algunas opciones para acelerar tanto la exportación como la futura importación. Este método permite extraer toda la base de datos sin tener que respaldar los índices, ya que solo se reunirán las peticiones SQL para regenerar el archivo de salida. Esto permite ganar un espacio en disco, que no es, sin embargo, comparable al del comando `SELECT . . . INTO OUTFILE`. Podemos cambiar con un editor de texto las peticiones de creación de estructuras, lo que puede permitir, por ejemplo, cambiar el tipo de motor utilizado para las tablas, renombrarlas, etc. Este método requiere que el servidor MySQL se inicie para que el cliente pueda conectarse y generar todas las peticiones SQL enunciadas más arriba, lo que puede retrasar las peticiones SQL que se estén ejecutando en el servidor.

`mysqldump` permite hacer respaldos lógicos, por lo que no hay que olvidar que la restauración puede ser muy larga si la base de datos es voluminosa.

Estas son las principales opciones de `mysqldump`:

- `-A, --all-databases`: extrae todas las bases.
- `-B, --databases`: exporta todas las bases pasadas como parámetro separadas por un espacio.
- `--single-transaction`: realiza la copia de seguridad dentro de una transacción para conservar la consistencia de todas las tablas InnoDB accedidas.
- `--tables`: exporta las tablas pasadas como parámetro separadas por un espacio.
- `-u, --user`: especifica el login necesario para conectarse al servidor MySQL.
- `-p--password`: especifica la contraseña necesaria durante la conexión al servidor MySQL.
- `-p, --port`: especifica el puerto que debe utilizarse para conectarse al servidor MySQL.
- `-R, --routines`: extrae las funciones y procedimientos. **¡Esta opción no está activada por defecto!**
- `--skip-extended-insert`: desactiva las inserciones múltiples, lo que puede ser útil cuando se quiere hacer cambios en el archivo de salida, como suprimir algunas líneas.
- `--opt`: alias para `--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset`. **Esta opción está activada por defecto.**
- `-d--no-data`: no exporta el contenido de las tablas. Puede resultar útil si queremos recuperar la estructura de las bases de datos.
- `-w, --where`: pone un filtro en los registros de exportación.
- `--skip-add-drop-database, --skip-add-drop-table, --no-create-db, --no-create-info`: desactiva respectivamente las órdenes de supresión de bases, tablas y las órdenes de creación de bases y tablas.
- `--master-data`: añade el comando `CHANGE MASTER TO` en la salida, lo que es útil para crear de forma rápida un esclavo a partir de la exportación de un maestro.

- `--dump-slave`: añade el comando `CHANGE MASTER TO` en la salida. Si tomamos los respaldos de un esclavo, debemos usar esta opción en lugar de `--master-data`.
- `--tab`: genera para cada tabla los archivos `nombre_tabla.sql`, que contienen la estructura de la tabla, y `nombre_tabla.txt`, que almacena el contenido de la tabla formateado con tabuladores (mismo formato que para el comando `SELECT ... INTO OUTFILE`).
- `--flush-logs`: fuerza la creación de un nuevo archivo de registro binario antes de comenzar a exportar la base, o cada vez que se exporta una base diferente cuando la opción `-A` se utiliza, salvo si las opciones `--lock-all-tables` o `--master-data` están activadas.

He aquí, por ejemplo, cómo guardar varias bases de datos de una misma instancia, incluyendo los procedimientos almacenados asociados.

```
shell> mysqldump -u mi_usuario -p -R -B sakila test > mi_dump1.sql
Enter password:
```

Una vez se introduce y se confirma la contraseña, las bases de datos `sakila` y `test` se exportan integrando las definiciones de los procedimientos y funciones almacenadas en el archivo `mi_dump1.sql`.

Del mismo modo, podemos guardar una tabla recuperando solo las primeras 10 inserciones:

```
shell> mysqldump -u mi_usuario -p --no-create-info --skip-extended-
insert sakila actor| awk 'BEGIN{n=0} /SET|INSERT/ {print
$0;if($1=="INSERT"){n++;if(n>10){exit;}}}' > mi_dump2.sql
Enter password:
```

En este último comando, utilice el comando de sistema `awk` para conservar los comandos `SET` que preceden a las 10 inserciones que se desea recuperar.

Último ejemplo: si solo necesita recuperar los datos de la base `sakila` que conciernen a los actores cuyo apellido empieza por la letra N, puede utilizar el siguiente comando:

```
shell> mysqldump -u mi_usuario -p --where 'first_name like "N%"'
sakila actor | grep INSERT
Enter password:

INSERT INTO `actor` VALUES (2,'NICK','WAHLBERG','2006-02-15
03:34:33');
INSERT INTO `actor` VALUES (44,'NICK','STALLONE','2006-02-15
03:34:33');
INSERT INTO `actor` VALUES (50,'NATALIE','HOPKINS','2006-02-15
03:34:33');
INSERT INTO `actor` VALUES (166,'NICK','DEGENERES','2006-02-15
03:34:33');
```

En todos los ejemplos anteriores, hemos visto cómo obtener una copia de seguridad SQL. Para restaurar la copia de seguridad en un servidor MySQL, podemos proceder directamente si el servidor de destino es diferente del servidor fuente:

```
shell> mysqldump -p -B test2 |
mysql -umsandbox -p
Enter password: Enter password:
```

Si no, podemos transferir el respaldo al segundo servidor (si restablece el respaldo en otro servidor), y enviar el archivo a través de una tubería con el comando `mysql`:

```
shell> mysqldump -p -B test2 >
/var/tmp/dump.sql
Enter password:

shell> cat /var/tmp/dump.sql|mysql -p
```

Enter password:

```
# no olvide eliminar el volcado de prueba para no congestionar  
el directorio  
shell> rm -f /var/tmp/dump.sql
```

Una última posibilidad es cargar el archivo dump directamente en la consola `mysql`:

```
mysql> source /var/tmp/dump.sql  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 1 row affected (0.00 sec)
```

Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

```
Database changed  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.01 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)  
Query OK, 128 rows affected (0.00 sec)  
Records: 128 Duplicates: 0 Warnings: 0
```

Cuando exportamos los datos de una base, hay que plantearse la cuestión de la coherencia entre los datos de las distintas tablas que se exportan, es decir, preocuparse por el hecho de que algunas tablas pueden ser modificadas durante la exportación y, por tanto, contener las grabaciones más recientes o una revisión diferente. Esto puede tener un impacto importante si la aplicación que utiliza la base gestiona por sí misma la clave externa, y la exportación no ha bloqueado las tablas dependientes utilizando el método apropiado para el motor de almacenamiento.

Por ejemplo, si nuestra base de datos contiene solo tablas MyISAM y deseamos que todas las tablas sean coherentes al exportarlas, usaremos el comando:

```
shell> mysqldump -u mi_usuario -p sakila > dump_myisam.sql
Enter password:
```

Debemos saber que la opción `--opt`, que implica la opción `--lock-tables`, está activada por defecto. Todas las tablas de la base `sakila` se cerrarán con la opción `READ LOCAL` para autorizar las inserciones concurrentes, permitiendo exportar todos los datos que forman parte de la misma revisión, es decir, el conjunto de los datos que datan de principios de la exportación. Sin embargo, esto no es válido para las tablas de una misma base. Cuando se quieran exportar varias bases de datos, se deberá mantener la coherencia entre las tablas de las diferentes bases utilizando la opción `--lock-all-tables`, que plantea un bloqueo global:

```
shell> mysqldump -u mi_usuario -p --lock-all-tables -B sakila blog >
dump_2db_myisam.sql
Enter password:
```

En el caso de que nuestra base de datos utilice solo el motor InnoDB, habría que usar la opción `--single-transaction` para no causar bloqueo a nivel de tabla, ya que el motor de almacenamiento InnoDB presenta por defecto una visión coherente del conjunto de las tablas para cada nueva transacción a través de su sistema de multiversión MVCC:

```
shell> mysqldump -u mi_usuario -p --lock-all-tables -B sakila blog >
dump_innodb.sql
Enter password:
```

Una última cuestión puede plantearse cuando se utilizan diferentes motores de almacenamiento. En efecto, hay que saber que las opciones `--lock-tables` y `--single-transaction` son mutuamente excluyentes. En este caso, si realmente queremos que todos los datos sigan siendo coherentes, estaremos obligados a plantear bloqueos a nivel de las tablas, ya que `--single-transaction` no tendrá ningún efecto sobre las tablas MyISAM.

```
shell> mysqldump -u mi_usuario -p blog t1_myisam t2_innodb >
dump_mixed.sql
Enter password:
```

Cabe destacar que, si la coherencia de la base de datos es primordial, habrá que utilizar bloqueos a nivel de tabla (`--lock-tables` o `--lock-all-tables`), si algunas tablas no soportan el modo transaccional, y exportar todos los datos en una única transacción (`--single-transaction`) si todas las tablas utilizan un motor transaccional como InnoDB. Tenga en cuenta que si usamos `--lock-tables` o `--lock-all-tables` con las tablas InnoDB, las tablas estarán bloqueadas en lectura y no podrán editarse, mientras que con `--single-transaction`, se abre una transacción para efectuar la exportación, por lo que evita tener que plantear el bloqueo.

- ☐ Si desea usar copias de seguridad lógicas y `mysqldump` no le parece suficientemente eficaz, puede optar por `mydumper` (<https://github.com/maxbube/mydumper>). Se trata en realidad de dos herramientas complementarias: `mydumper` para respaldos multithread y `myloader` para restauraciones multithread.

3. Percona XtraBackup

XtraBackup es una herramienta open source disponible en el sitio de Percona (<http://www.percona.com/software/mysql-database/percona-xtrabackup>) y que permite hacer respaldos físicos no bloqueantes para las tablas InnoDB, es decir, en un servidor en funcionamiento sin dejar de servir las peticiones. Es la única herramienta libre disponible hoy para este tipo de operaciones y compite con el MySQL Enterprise Backup de Oracle.

XtraBackup está en realidad compuesta por dos herramientas: `xtrabackup`, que realiza el respaldo, e `innobackupex`, que es una simple capa escrita en Perl. ¿Por qué esta capa? Sencillamente porque la utilización directa de `xtrabackup` no es fácil, sobre todo debido a que la versión que debe utilizar depende de la versión de su servidor MySQL. `innobackupex` permite liberarse de este tipo de detalles y unifica la manera de invocar una copia de seguridad.

- `innobackupex` desaparecerá de manera progresiva. Hasta la versión 2.3, este script era la forma recomendada de invocar Percona XtraBackup. Con la versión 2.4, `innobackupex` se convirtió en un simple enlace simbólico a `xtrabackup`, y está previsto que `innobackupex` desaparezca por completo en una próxima versión. Para disponer de comandos utilizables en todas las versiones, `innobackupex` se usará en la continuación de esta sección.

Debe saber que XtraBackup tiene muchas funcionalidades: utilización de varios threads en paralelo, respaldos incrementales, respaldos comprimidos, cifrado y mucho más.

Empiece por crear un usuario dedicado para los respaldos:

```
mysql> CREATE USER 'xbackup'@'localhost' IDENTIFIED BY 'mi_cont';
mysql> GRANT RELOAD, LOCK TABLES, REPLICATION CLIENT ON *.*
TO 'xbackup'@'localhost';
```

Las copias de seguridad se desarrollan en dos etapas. La primera consiste en copiar los archivos MySQL al directorio objetivo, observando los cambios que tuvieron lugar en la base de datos durante la copia.

```
shell> innobackupex -user=xbackup -password=mi_cont /data/backups
```

Cuando termina el respaldo, aparece un mensaje del tipo:

```
121212 11:55:07 innobackupex: completed OK!
```

En tal caso, podemos pasar a la segunda etapa, que consiste en aplicar los cambios que se registraron durante la primera fase:

```
shell> innobackupex --apply-logs /data/backups/2012-12-12_12-00-00/
```

También aparecerá un mensaje indicando el éxito de la operación.

```
121212 12:05:29 innobackupex: completed OK!
```

Los archivos obtenidos pueden copiarse directamente en el directorio de datos si se requiere una restauración. Después de haber comprobado los permisos de archivos y directorios, podemos reiniciar MySQL.

4. Otras soluciones

a. mylvmbbackup

Esta herramienta escrita en Perl funciona en un sistema Linux en el que la opción LVM está activada. Permite automatizar las ocho etapas necesarias para la creación de una copia de

seguridad física usando LVM (*Linux Volume Manager*). Estas son:

- Bloquear el comando de escritura para todas las tablas con el comando `FLUSH TABLES WITH READ LOCK`.
- Crear una instantánea.
- Desbloquear las tablas.
- Montar el volumen lógico correspondiente a la instantánea.
- Efectuar la recuperación de archivos si es posible (soporte de LVM2 con las imágenes accesibles en escritura).
- Guardar todos los archivos de la base de datos.
- Desmontar el volumen lógico.
- Eliminar la instantánea.

Se requiere que la herramienta se ejecute en el servidor en el que se alojan los datos, pero podemos usar `rsync` para almacenar las copias de seguridad en servidores remotos.

`mylvmbackup` se basa en la utilización de instantáneas para reducir el tiempo de bloqueo del servidor MySQL (parada de lecturas/escrituras) a unos pocos segundos.

He aquí un ejemplo de uso con MySQL Sandbox (para la instalación, ver el capítulo Instalación del servidor) y MySQL 5.5.25:

```
Shell> echo "Creación de los directorios necesarios para mylvmbackup"
shell> mkdir -p /var/tmp/mylvmbackup/backup /var/tmp/mylvmbackup/mnt
shell> echo "se hace en el directorio raíz de MySQL 5.5 para
no tener errores en la ejecución del respaldo"
shell> cd /usr/local/mysql/5.5.25
shell> echo "Se inicia el respaldo"
shell> mylvmbackup --vgname=vadmin --lvname=lv_mysql_sandbox
--lvsize=1G --backuptype=rsync --user=msandbox
--password=msandbox --socket=/tmp/mysql_sandbox5525.sock
--mycnf=/opt/sandboxes/msb_5_5_25/my.sandbox.cnf
--innodb_recover --skip_flush_tables --skip_hooks
--mysqld_SAFE=/usr/local/mysql/5.5.25/bin/mysqld_safe
```

En la salida del script de respaldo, podemos recuperar las líneas `Info` para analizar lo que hace el script:

```
Info: Connecting to database...
Info: Taking position record into /tmp/mylvmbbackup-backup-
20100103_061219_mysql-EiANsL.pos...
Info: Running: lvcreate -s --size=1G
--name=lv_mysql_sandbox_snapshot
/dev/vadmin/lv_mysql_sandbox
Info: DONE: taking LVM snapshot
Info: Unlocking tables...
Info: Disconnecting from database...
Info: Mounting snapshot...
Info: Running: mount -o rw
/dev/vadmin/lv_mysql_sandbox_snapshot
/var/tmp/mylvmbbackup/mnt/backup
Info: DONE: mount snapshot
Info: Copying /tmp/mylvmbbackup-backup-
20100103_061219_mysql-EiANsL.pos to
/var/tmp/mylvmbbackup/mnt/backup-pos/backup-
20100103_061219_mysql.pos...
Info: Recovering InnoDB...
Info: Running: echo 'select 1;' |
/usr/local/mysql/5.5.25/bin/mysqld_safe
--socket=/tmp/mylvmbbackup.sock --pid-
file=/var/tmp/mylvmbbackup_recoverserver.pid --log-
error=/tmp/mylvmbbackup_recoverserver.err
--datadir=/var/tmp/mylvmbbackup/mnt/backup --skip-networking
--skip-grant --bootstrap --skip-ndbcluster --skip-slave-
start
Info: DONE: InnoDB recovery on snapshot
Info: Copying /opt/sandboxes/msb_5_5_25/my.sandbox.cnf to
/var/tmp/mylvmbbackup/mnt/backup-pos/backup-
20100103_061219_mysql_my.sandbox.cnf...
Info: Taking actual backup...
Info: Archiving with rsync to
/var/tmp/mylvmbbackup/backup/backup-20100103_061219_mysql
```



```

Info: Running: rsync -avWP /var/tmp/mylvmbbackup/mnt/backup/
/var/tmp/mylvmbbackup/mnt/backup-pos/backup-
20100103_061219_mysql.pos /var/tmp/mylvmbbackup/mnt/backup-
pos/backup-20100103_061219_mysql_my.sandbox.cnf
/var/tmp/mylvmbbackup/backup/backup-
20100103_061219_mysql.INCOMPLETE-0043648/
Info: DONE: create rsync archive
Info: Cleaning up...
Info: Running: umount /var/tmp/mylvmbbackup/mnt/backup
Info: DONE: Unmounting /var/tmp/mylvmbbackup/mnt/backup
Info: LVM Usage stats:
Info:   LV                               VG      Attr   Lsize
Origin          Snap% Move Log Copy% Convert
Info:   lv_mysql_sandbox_snapshot vadmin swi-a- 1.00G
lv_mysql_sandbox  0.01
Info: Running: lvremove -f
/dev/vadmin/lv_mysql_sandbox_snapshot
Info: DONE: Removing snapshot

```

Como hemos podido ver en este ejemplo, el script automatiza todas las operaciones a cambio de un gran número de argumentos. A continuación, le ofrecemos la descripción de los más importantes:

- `--vgname`: indica el nombre del grupo de volumen que contiene el volumen lógico que almacena los datos MySQL que se van a respaldar (por defecto, `mysql`).
- `--lvname`: indica el nombre del volumen lógico que almacena los datos MySQL que se van a respaldar (por defecto, `data`).
- `--lvsize`: indica el tamaño del volumen lógico que se utiliza para almacenar la instantánea (snapshot). Debemos saber que solo los bloques modificados en el respaldo necesitarán almacenarse. Si tenemos pocos cambios en curso, podemos especificar un tamaño pequeño, como 100 MB.
- `--backuptype`: indica el tipo de respaldo que se desea efectuar (`tar`, `rsync`, `rsnap`, `none`). Veremos un caso de uso de `none` en un segundo ejemplo.
- `--user`: especifica el login necesario para conectarse al servidor MySQL.

- `--password`: especifica la contraseña necesaria durante la conexión al servidor MySQL.
- `--port`: especifica el puerto que se debe utilizar para conectarse al servidor MySQL por TCP.
- `--socket`: indica el archivo `socket` que se debe utilizar para conectarse al servidor MySQL en lugar de conectarse por TCP.
- `--innodb_recover`: puede usarse si solo se alojan tablas de tipo InnoDB en el servidor MySQL.
- `--skip_flush_tables`: fuerza la creación de un nuevo archivo de registro binario después de haber bloqueado todas las tablas.
- `--skip_hooks`: desactiva el uso de `hooks`, que son los comandos o scripts externos que se puede añadir en el proceso de almacenamiento en diferentes fases (antes de la conexión a la base, después de la creación del snapshot, antes de montar el volumen lógico del snapshot, etc.).
- `--mysqld_safe`: indica la ruta al script `mysqld_safe` necesaria para forzar la recuperación con la opción `innodb_recover` sin utilizar la ruta por defecto (busca en la variable `$PATH`).
- `--mycnf`: indica el archivo de configuración que se debe utilizar (por defecto es `/etc/my.cnf`).
- `--keep_snapshot`: solicita que la instantánea creada no sea destruida al final del respaldo.

Al terminar la ejecución del comando `mylvmbackup` que hemos visto antes, dispondremos del respaldo almacenado en el directorio `/var/tmp/mylvmbackup/backup`. En él, encontraremos el árbol completo del volumen lógico `lv_mysql_sandbox`, el archivo de configuración `my.sandbox.cnf` y un archivo que contiene la posición de los registros binarios (que podrá utilizarse en caso de una restauración de tipo PITR):

```
shell> find /var/tmp/mylvmbackup/backup -maxdepth 2
/var/tmp/mylvmbackup/backup
/var/tmp/mylvmbackup/backup/backup-20100103_064647_mysql
```

```
/var/tmp/mylvmbbackup/backup/backup-  
20100103_064647_mysql/backup-20100103_064647_mysql.pos  
/var/tmp/mylvmbbackup/backup/backup-  
20100103_064647_mysql/backup-  
20100103_064647_mysql_my.sandbox.cnf  
/var/tmp/mylvmbbackup/backup/backup-  
20100103_064647_mysql/msb_5_5_25
```

Otro ejemplo para desactivar el guardado e inhibir la supresión de la instantánea:

```
Shell> echo "se hace en el directorio raíz de MySQL 5.5 para  
no tener errores en la ejecución del respaldo"  
shell> echo "Se inicia el respaldo"  
shell> mylvmbbackup --vgname=vadmin --lvname=lv_mysql_sandbox  
--lvsize=1G --backuptype=none --user=msandbox  
--password=msandbox --socket=/tmp/mysql_sandbox5500.sock  
--mycnf=/opt/sandboxes/msb_5_5_25/my.sandbox.cnf  
--innodb_recover --skip_flush_tables --skip_hooks  
--keep_snapshot  
--mysqld_SAFE=/usr/local/mysql/5.5.25/bin/mysqld_safe
```

Conservar varias instantáneas permite mantener distintas versiones de la base de datos y, como las instantáneas solo consumen espacio cuando los bloques de datos son modificados, podemos, cuando hay pocos datos que se modifiquen (o a menudo los mismos), mantener varios respaldos sin necesitar (número de respaldos * tamaño de la base) espacio en bytes.

b. mysqlpump

Todas las distribuciones de MySQL 5.7 incluyen una herramienta llamada `mysqlpump`, que con el tiempo podría sustituir a `mysqldump`. `mysqlpump` proporciona un archivo SQL, al igual que `mysqldump`, pero ofrece funcionalidades adicionales, tales como el respaldo de varias tablas en paralelo o la compresión.

La herramienta no ha recibido mucha publicidad por parte de Oracle, y en el momento de escribir este libro es difícil saber si `mysqlpump` es lo suficientemente estable para poder

utilizarla en producción.

Observe que, aunque `mysqlpump` puede ser interesante para acelerar los respaldos lógicos mediante el paralelismo, la restauración se realiza siempre de manera secuencial ejecutando comandos SQL unos a continuación de otros. Y como el principal problema de los respaldos lógicos es el tiempo necesario para las restauraciones, `mysqlpump` no siempre permite usar con facilidad copias de seguridad lógicas para las bases de datos voluminosas.

La documentación está disponible en la página siguiente:

<https://dev.mysql.com/doc/refman/5.7/en/mysqlpump.html>

Hardware y sistema operativo

1. Procesador

Las arquitecturas con varios procesadores o con varios núcleos son las más comunes; cabe preguntarse si es mejor favorecer la velocidad o el número de procesadores.

Antes de la versión 5.5, MySQL adolecía de fuertes limitaciones cuando muchos threads estaban en ejecución concurrente, limitaciones que le impedían funcionar de forma correcta con los equipos que disponen de muchos núcleos. Era corriente tener que limitarse a arquitecturas con un máximo de 4 núcleos con MySQL 5.1 para evitar degradaciones de rendimiento significativas.

Estos problemas se han corregido permitiendo el uso de un número alto de núcleos con MySQL 5.6 y aún más con MySQL 5.7. Sin embargo, es importante no descuidar la velocidad de los procesadores. En efecto, no podemos paralelizar la ejecución de una petición en varios procesadores, haciendo que la velocidad del procesador sea muchas veces determinante en la rapidez de la respuesta. Como ejemplo, en un entorno de replicación para un servidor esclavo, esta limitación puede aparecer rápidamente: aunque la mayor parte de la carga procede de las peticiones replicadas, solo el thread de replicación estará activo y solo podrá ocupar un procesador. Esto nos llevará en este caso a un procesador cargado al 100 %, mientras que los demás no están prácticamente utilizados (si configuramos la replicación multithread, quizás no tengamos nunca este problema).

Recuerde que, para un servidor esclavo, tendremos todo el interés en favorecer la velocidad del procesador antes que el número de núcleos, de modo que la replicación siga siendo lo más síncrona posible, y en un servidor maestro, más núcleos implican una mayor capacidad de tratar conexiones simultáneas.

¿Por qué no buscar en los servidores esclavos a la vez la velocidad y el número de

procesadores? Simplemente porque tenemos en general varios servidores esclavos, lo que permitirá distribuir las peticiones, pero todas las peticiones procedentes de la replicación deberán ejecutarse en todos los esclavos. La cuestión del coste entra, por supuesto, también en juego.

Un último punto para los usuarios de Linux: en algunas distribuciones, la frecuencia del procesador se modula en función de la carga de la CPU. El objetivo es reducir el consumo eléctrico de los procesadores, haciéndolos correr al ralenti si no se utilizan de forma intensa. Esta característica es muy útil en un equipo portátil para aumentar la autonomía, pero resulta muy mala en un servidor para el que buscamos el máximo rendimiento. Piense en comprobar la frecuencia real de los procesadores.

En el siguiente ejemplo, el procesador funciona en teoría a 2,5 GHz, pero la realidad es muy diferente.

```
# cat /proc/cpuinfo | grep MHz
cpu MHz          : 499.992
cpu MHz          : 499.992
cpu MHz          : 499.992
cpu MHz          : 499.992
```

Si la frecuencia real es inferior a la frecuencia nominal, nos interesará cambiar el CPU governor que modula la frecuencia.

Retomando el ejemplo anterior, podemos ver que el governor tipo `powersave` está activado. Es mejor activar el governor `performance`:

```
# cpufreq-info
[...]
hardware limits: 400 MHz - 3.10 GHz
  available cpufreq governors: performance, powersave
# cpufreq-set -g performance
# cat /proc/cpuinfo | grep MHz
cpu MHz          : 3053.375
cpu MHz          : 3079.781
```

cpu MHz	: 3040.679
cpu MHz	: 3011.125

☐ Tenga en cuenta que los comandos pueden variar según las distribuciones.

2. Memoria RAM

La memoria RAM desempeña un papel crucial en el rendimiento de un servidor de base de datos como MySQL. En efecto, MySQL pasa gran parte de su tiempo leyendo datos, a los que se tiene un acceso mucho más rápido si se encuentran en la memoria y no en el disco.

Con InnoDB, la memoria RAM tiene también su utilidad para las escrituras porque, cuando es posible conservar durante algún tiempo las modificaciones de datos en una caché de memoria, el servidor puede optimizar las escrituras de disco, agrupando todas aquellas que tienen lugar en la misma zona del disco o solo ejecutando la modificación más reciente si el mismo dato se actualiza varias veces. Este motor, cuyo diseño se remonta al período en que los discos eran muy lentos, recurre a estrategias complejas utilizando cachés de memoria para agrupar al máximo el acceso al disco.

Para estimar mejor la cantidad de memoria RAM necesaria, debemos tener en cuenta más parámetros: el tamaño de la base de datos (datos e índice), la evolución previsible de la volumetría y la cantidad de datos útiles en un instante t.

Los ejemplos siguientes nos darán una idea de la influencia de estos parámetros.

Si la base de datos tiene un tamaño pequeño (algunos GB, por ejemplo) y su tamaño debe permanecer estable en el tiempo, entonces es fácil elegir una cantidad de RAM que permita a la totalidad de la base de datos estar en memoria.

Si la base de datos tiene un tamaño pequeño pero debe crecer 1 GB por semana, es necesario saber qué parte de estos datos es útil en un momento dado. Si solo se consultan los datos de la última semana, se puede considerar que solo necesitan estar en memoria 1 GB de datos. Sin embargo, si el 100 % de los datos son necesarios, la base de datos no podrá mantenerse en memoria mucho tiempo y la cuestión del rendimiento de los discos se volverá rápidamente crucial.

Por último, si la base de datos es muy voluminosa (1 TB, por ejemplo), la cuestión de la cuota de datos útiles es de nuevo crucial. Si solo utilizamos 10 GB, 16 GB de memoria serán sin duda suficientes, pero si requerimos 500 GB, la elección de los discos será aún más crucial.

Si usamos en exclusiva InnoDB, he aquí un procedimiento sencillo para determinar si necesitamos más memoria. Observe la evolución de la variable de estado `Innodb_buffer_pool_reads` cuando el servidor está cargado: si los valores están cerca de 0, casi todas las lecturas de datos se hacen en memoria, mientras que si los valores están cerca del número de lecturas/s que el disco permite, aumentar la memoria y el tamaño del buffer pool sin duda nos proporcionará una mejora de rendimiento.

El siguiente ejemplo se extrae de un servidor en el que los discos pueden realizar alrededor de 2000 operaciones/s. La línea de comandos siguiente examina la variable `Innodb_buffer_pool_reads` cada segundo (`-i 1`) y muestra solo la diferencia con respecto al resultado anterior (`-r`). Ignore el primer valor que devuelve el número de lecturas desde el arranque del servidor:

```
# mysqladmin ext -ri1 | grep Innodb_buffer_pool_reads
| Innodb_buffer_pool_reads          | 1969213816      |
| Innodb_buffer_pool_reads          | 141             |
| Innodb_buffer_pool_reads          | 150             |
| Innodb_buffer_pool_reads          | 157             |
| Innodb_buffer_pool_reads          | 118             |
| Innodb_buffer_pool_reads          | 134             |
| Innodb_buffer_pool_reads          | 122             |
| Innodb_buffer_pool_reads          | 163             |
| Innodb_buffer_pool_reads          | 166             |
```

Como podemos constatar, las lecturas en disco son significativas, pero están muy lejos de la capacidad máxima del disco: añadir memoria no es crítico en este caso.

3. Disco duro

a. Elementos de elección

Sin entrar en detalles sobre las ventajas o desventajas de tal o cual modelo de disco duro (información que se vuelve obsoleta con rapidez), el objetivo de esta sección es proporcionar información sobre las características que deben tenerse en cuenta para realizar una elección informada.

Los resultados de un disco duro se miden con dos parámetros: el tiempo de acceso y el tráfico de datos. Según nuestra aplicación, uno de estos factores será más importante. Si realizamos peticiones cortas principalmente, como es el caso en la mayoría de las aplicaciones transaccionales web, el tiempo de acceso a los datos tendrá una gran importancia. Sin embargo, si la aplicación demanda recorridos completos de tablas voluminosas, como será el caso con una base de datos que sirve de almacén para una aplicación estadística, el tráfico de datos será el factor esencial que se deberá considerar.

Por otra parte, en las mismas condiciones, un disco duro de 15 000 revoluciones por minuto será más rápido que un disco duro de 10 000 revoluciones por minuto. Por otra parte, puede ser interesante la posibilidad de seleccionar discos cuyo tamaño sea muy superior al tamaño de los datos que se van a almacenar, ya que esto mejorará la ubicación de los datos.

b. RAID

Descargado en: www.detodoprogramacion.org

Para proporcionar redundancia o mejores resultados, con frecuencia podremos configurar los discos duros en RAID (*Redundant Array of Inexpensive Disks*, matriz redundante de discos independientes), bien por software o mediante un controlador RAID hardware. En general, se prefiere un hardware RAID, que proporciona mejores resultados. Habrá que pensar en configurar la política de escritura en modo Write Back, es decir, utilizando la caché integrada en el controlador RAID, y verificar que esta caché esté protegida por una UPS (a riesgo de perder los datos en caso de un error fatal).

Estos son los principales niveles RAID utilizados:

- RAID 0: los datos se distribuyen por entrelazado (*striping* en inglés) en el conjunto de los discos. De esta forma, las prestaciones aumentan notablemente en lectura y escritura. Los datos solo se encuentran en una ubicación, así que, si fallase un disco, el conjunto de datos se perdería. Esta configuración no ofrece redundancia, aunque se clasifica entre los niveles de RAID. Observe que la probabilidad de fallo de una

configuración RAID 0 es mayor que la probabilidad de fallo de un solo disco, y que esta probabilidad aumenta con el número de discos usados.

- RAID 1: los datos se encuentran en discos de tamaños idénticos (dos discos como mínimo). Cada uno de los discos es una copia exacta de todos los demás, lo que explica el nombre de *mirroring* dado en inglés a este tipo de configuración; en caso de avería de uno de los discos, los datos permanecen disponibles en la otra unidad de disco. Esta configuración proporciona una muy buena fiabilidad, puesto que hasta N-1 discos pueden fallar sin causar pérdidas de datos. Sin embargo, la capacidad de almacenamiento total es igual a la capacidad de almacenamiento de un solo disco: esta configuración se vuelve muy costosa con rapidez cuando el número de discos es alto.
- RAID 5: los datos se distribuyen en todos los discos, pero a diferencia de RAID 0, los bloques de paridad se reparten a su vez, lo que permite reconstruir los datos en caso de avería de un disco. El equivalente de la capacidad de almacenamiento de un disco se pierde para permitir la escritura de los bloques de paridad. Los rendimientos son mejores para las lecturas, pero no para las escrituras. En caso de avería del disco, las operaciones necesarias para la reconstrucción de los datos son muy largas.
- RAID 10: este nivel combina las técnicas de RAID 0 y RAID 1 y disfruta de las ventajas de estos dos niveles, es decir, de muy buenos resultados en lectura y escritura, así como una buena tolerancia a fallos. Los datos son mucho más fáciles de reponer que en el caso de RAID 5, pero solo puede utilizarse la mitad de la capacidad total de almacenamiento. El coste de esta configuración puede ser problemático, al igual que para RAID 1.

c. SSD

Desde hace algunos años, otra categoría de discos ha hecho su aparición: los discos SSD (*Solid-State Drive*). Estos discos tienen la característica de ser casi tan rápidos como la memoria RAM almacenando los datos de manera permanente. Se está adoptando con rapidez, a medida que sus principales desventajas se eliminan gradualmente: su coste ha disminuido de forma notable y los problemas de tiempo de vida característicos de los primeros modelos han desaparecido casi por completo.

Los SSD se dividen en dos categorías:

- Los discos con una interfaz SATA (*Serial Advanced Technology Attachment*), que

pueden servir de sustitutos directos de los discos duros convencionales. Aquí también se pueden distinguir dos categorías: SSD SLC (*Single Level Cell*) con un precio elevado y capacidad relativamente baja, pero con muy buenos rendimientos, y SSD MLC (*Multi Level Cell*), a un precio más asequible y con mayor capacidad, pero con menores rendimientos.

- Los discos con una interfaz PCIe (*Peripheral Component Interconnect Express*), que requieren la instalación de controladores dedicados para poder ser reconocidos por el sistema operativo. Estos discos tienen un precio que los reserva para las aplicaciones de gama alta, aunque sus resultados son excepcionales y la capacidad de almacenamiento, elevada. Para ser sinceros, en la actualidad, MySQL aún tiene dificultades para aprovechar todo el rendimiento de estos discos.

Cabe señalar también que utilizar RAID es tan importante para los SSD SATA como para discos duros tradicionales, ya que el riesgo de fallo es significativo. Sin embargo, para discos PCIe, su duración es a priori por lo menos tan alta como la de las placas base de los servidores, lo que hace innecesario el uso de RAID (RAID se utiliza por lo general de manera interna).

4. Sistema operativo

MySQL está disponible para muchas plataformas, siendo las más comunes Linux para producción y Windows para desarrollo. La elección de un sistema operativo estará más vinculada a las competencias técnicas del equipo profesional, a opciones económicas o a las características deseadas, como un sistema de archivos en particular, que a las restricciones impuestas por el servidor MySQL.

- ☐ Observe que el rendimiento en Windows ha mejorado de forma significativa a partir de la versión 5.5. Si por lo general encontraba MySQL lento en Windows, este ya no es el caso.

Observe también que, gracias al modelo cliente-servidor de MySQL, el sistema operativo de los clientes puede ser diferente del sistema del servidor. De esta forma, una aplicación hospedada en un servidor Windows puede interrogar a un servidor MySQL que funciona en Linux.

Optimización del esquema

1. Tipos de datos

a. Principios generales

La búsqueda del mejor tipo de datos posible es una tarea que a menudo se descuida durante la fase de diseño del esquema de la base de datos, ya que es a la vez difícil y tedioso preguntarse, para cada campo, cuáles son los valores mínimos y máximos que se han de almacenar. Y al poner la aplicación en producción, la modificación de los campos suele verse como una operación arriesgada, con riesgos de pérdidas de datos, y sin impacto significativo sobre el rendimiento. Sin embargo, una elección informada de los tipos de datos será siempre beneficiosa. Esta sección tiene por objeto establecer algunas reglas simples que conviene seguir para seleccionar de manera eficaz los tipos de datos.

En primer lugar, tenga en cuenta que, cuanto más simple y compacto sea el tipo de datos, más ligero y eficiente resultará. Así, un entero es más sencillo que una cadena de caracteres, ya que los conceptos de juegos de caracteres y conjuntos no existen para los enteros. Del mismo modo, entre los diferentes tipos de números enteros, un `TINYINT` (un byte por valor) es más compacto que un `BIGINT` (ocho bytes por valor) y por lo tanto más rápido de procesar.

En segundo lugar, debemos evitar al máximo las columnas `NULL` o, lo que es lo mismo, declarar todas las columnas `NOT NULL`, salvo en caso de expresa necesidad. Las columnas que pueden ser `NULL` requieren un trabajo adicional que es mejor evitar en el servidor. A menudo, el valor `NULL` utilizado como valor por defecto puede ser sustituido por `0` o una cadena vacía.

b. Números

Los datos numéricos se dividen en dos categorías bien diferenciadas: los enteros y los números reales.

Para los enteros, el tipo más corriente es `INT` (cuatro bytes por valor almacenado), pero existe también el tipo `TINYINT` (un byte), `SMALLINT` (dos bytes), `MEDIUMINT` (tres bytes) y `BIGINT` (ocho bytes). El número de bytes ocupados por valor almacenado devuelve la ventana de valores de cada uno de los tipos: un byte para representar 256 valores, N bytes para representar 256^N valores. La optimización consiste en elegir el tipo más pequeño posible, con un rango de valores suficiente para almacenar todos los valores posibles.

- ☐ Cuando solo necesitamos manejar valores positivos, debemos considerar declarar la columna `UNSIGNED`, lo que elimina la posibilidad de almacenar valores negativos y duplica la capacidad de los números positivos.

He aquí algunos ejemplos:

- Para una edad, `TINYINT UNSIGNED` conviene (0 a 255).
- Para la población de un país, `UNSIGNED INT` es una buena elección (0 a más de 4 millardos).
- Imaginemos una tabla con 10 campos `INT`: cada fila consume 40 bytes. Si podemos utilizar, en lugar de un campo `INT`, seis campos `MEDIUMINT` y tres campos `TINYINT`: cada fila no ocupará más de 25 bytes o una ganancia de 40 %. Para decenas de tablas y millones de filas, conseguiremos ganar mucho espacio en disco.

- ☐ A veces se pasa un parámetro entre paréntesis para los campos de tipo `INT` o incluidos (`INT (4)`), por ejemplo). Solo se utiliza para el formateo de los valores en la pantalla para algunos clientes, pero no cambia la ventana de valores posibles ni el número de bytes que necesita un valor para ser almacenado. De esta forma, un `INT (4)` toma siempre cuatro bytes de espacio en disco y puede almacenar valores entre aproximadamente -2 millardos y 2 millardos, lo mismo que un `INT` o un `INT (11)`.

Para los números reales, podemos elegir entre `DECIMAL` por una parte, `FLOAT` y `DOUBLE` por otra parte. `DECIMAL` ofrece una ventana de valores más pequeña, pero almacena estos de manera exacta: este tipo de datos será esencial cuando deseemos poder realizar operaciones sin error de redondeo, como es el caso para el almacenamiento de datos financieros. `FLOAT` y `DOUBLE` almacenan por el contrario siempre valores aproximados.

- ☐ Para estos tres tipos de datos, podemos especificar el número de cifras significativas y el número de cifras decimales. De esta forma, `DECIMAL (5, 2)` permite almacenar números de cinco cifras significativas y dos decimales, es decir, los números comprendidos entre -999,99 y 999,99.

c. Cadenas de caracteres

Existen dos tipos de cadenas de caracteres: los tipos `CHAR` y `VARCHAR` y las variantes del tipo `TEXT`.

Los campos de estas dos categorías poseen todos en su definición un juego de caracteres y una colación (*collation* en inglés). Un juego de caracteres es solo una tabla de conversión entre un símbolo y su representación numérica para el almacenamiento. La elección del juego de caracteres se realiza teniendo en cuenta dos factores: el tamaño en bytes de un carácter y los símbolos que un juego de caracteres puede representar. De forma típica, podemos preferir `latin1` para aplicaciones en español o inglés y `utf8` para las aplicaciones que requieren el soporte de varios idiomas con caracteres muy diferentes, como el francés, japonés y el árabe.

- ☐ Elegir `utf8` por defecto no es siempre una buena idea. Por ejemplo, para una aplicación que solo debe almacenar caracteres rusos, `cp866` será una mejor opción (un byte por carácter en lugar de tres para `utf8`).

La colación corresponde al orden de los diferentes símbolos de un juego de caracteres. Un mismo juego de caracteres puede tener varias colaciones, lo que significa que los símbolos están ordenados de manera diferente. Las colaciones se refieren con mayor frecuencia al

nombre del juego de caracteres al que están vinculadas y llevan también un sufijo que permite conocer el tipo de clasificación proporcionado. Los sufijos que encontrará son:

- `_cs` para las colaciones sensibles a mayúsculas y minúsculas (*case sensitive*).
- `_ci` para las colaciones no sensibles a mayúsculas y minúsculas (*case insensitive*).
- `_bin` para las colaciones binarias (se considerará que la cadena no cuenta con más juegos de caracteres ni colaciones).

☐ Un VARCHAR que tiene una colación binaria es equivalente a un VARBINARY y un TEXT con colación binaria es equivalente a un BLOB.

He aquí algunos ejemplos de la influencia de la colación:

```
mysql> SELECT 'A' = 'a' COLLATE latin1_general_cs;
```

```
+-----+
| 'A' = 'a' COLLATE latin1_general_cs |
+-----+
|                                     0 |
+-----+
```

```
mysql> SELECT 'A' = 'a' COLLATE latin1_general_ci;
```

```
+-----+
| 'A' = 'a' COLLATE latin1_general_ci |
+-----+
|                                     1 |
+-----+
```

Los campos de tipo CHAR tienen una longitud fija. Esto significa que, si se inserta una cadena más corta que el tamaño del campo, la cadena se completará con espacios, y si tratamos de introducir una cadena más larga que el tamaño del campo la cadena se truncará y el servidor devolverá un error (el comportamiento depende del SQL_MODE elegido, ver el capítulo Configuración del servidor).

- ☐ El servidor elimina siempre los espacios finales de un CHAR en una petición SELECT:

```
mysql> CREATE TABLE t (col CHAR(5));

mysql> INSERT INTO t (col) VALUES ('xxx ');

mysql> SELECT LENGTH(col) FROM t;
+-----+
| LENGTH(col) |
+-----+
|           3 |
+-----+
```

Los campos de tipo VARCHAR tienen por el contrario una longitud dinámica y utilizan solo el tamaño de la cadena insertada más uno o dos bytes para almacenar el tamaño.

Durante la creación de un campo CHAR o VARCHAR, el parámetro pasado (VARCHAR (20) , por ejemplo) representa el número de caracteres que puede almacenar el campo. En función del juego de caracteres, el número de bytes podrá ser diferente del número de caracteres. En latin1, por ejemplo, todos los caracteres ocupan un byte, mientras que en utf8 los caracteres ocupan entre uno y tres bytes.

De forma general, los campos VARCHAR ocupan menos espacio que los campos CHAR, pero serán menos rápidos. ¿Cómo decidir entonces entre uno u otro tipo? Cuando las longitudes de las cadenas son muy variables, VARCHAR es probablemente una buena elección, mientras que, cuando las longitudes se aproximan entre sí, CHAR es sin duda mejor.

- ☐ Tenga en cuenta que con InnoDB se aconseja utilizar siempre los VARCHAR.

Observe que, con VARCHAR, muchas personas tienden a crear de forma

automática VARCHAR (255) , aunque los campos no almacenarán cadenas más largas que veinte o treinta caracteres. Se trata de una mala práctica, ya que MySQL utiliza el tamaño máximo para algunas operaciones, como la clasificación. Si por casualidad el espacio adicional necesario no permite lograr la clasificación en la memoria y demanda la creación de una tabla temporal en el disco, ¡la disminución del rendimiento puede ser catastrófica!

En el mismo orden de ideas, el juego de caracteres puede también afectar el rendimiento, ya que, para esas mismas clasificaciones, MySQL toma de manera automática el número máximo de bytes para cada carácter. De esta forma, una selección de un VARCHAR (20) en latin1 requerirá 20 bytes por valor, mientras que una selección de un VARCHAR (255) en utf8 ¡requerirá $255 \times 3 = 765$ bytes!

Observe también que las tablas Memory tratan los VARCHAR como CHAR.

El tipo TEXT y sus variantes permiten almacenar cadenas variables, posiblemente más largas con VARCHAR para MEDIUMTEXT y LONGTEXT. Los campos TEXT tienen algunas características que hacen preferir el tipo VARCHAR cuando ambos tipos pueden utilizarse:

- Un índice puede colocarse en un prefijo del campo, pero no en el campo entero.
- Estos campos suelen almacenarse en un espacio especial, aislado del resto de los campos.
- Para todas las operaciones que se efectúan con una tabla temporal, la tabla temporal será creada de forma automática en el disco y no en la memoria (solo porque las tablas MEMORY no permiten la creación de campos TEXT).

d. Datos binarios

MySQL también permite almacenar datos binarios, considerados cadenas sin juego de caracteres ni colación: se trata del tipo VARBINARY, así como las variaciones del tipo BLOB. Con la excepción de todo lo que afecte a los juegos de caracteres y colaciones, lo que se ha dicho en el párrafo anterior para VARCHAR es válido para VARBINARY y lo que se ha dicho para las variantes de TEXT es válido para las variantes de BLOB.

e. Fechas y horas

MySQL dispone de tipos para almacenar los años (YEAR), las fechas (DATE), las horas (TIME; la granularidad más fina es el segundo) o las fechas y horas (DATETIME y TIMESTAMP; la granularidad más fina es también el segundo para ambos tipos).

Al estar limitadas las posibilidades, la elección del tipo adecuado será, pues, siempre evidente, salvo en tres situaciones.

Primer caso: queremos guardar la fecha y la hora, ¿es mejor elegir DATETIME o TIMESTAMP?

Las diferencias de rango de valores posibles entre DATETIME y TIMESTAMP suelen imponer uno u otro: un campo DATETIME puede almacenar valores entre '1000-01-01 00:00:01' y '9999-12-31 23:59:59', mientras que un campo TIMESTAMP puede almacenar valores entre '1970-01-01 00:00:00' y una fecha establecida en 2038. Cuando el intervalo de fechas previstas permite utilizar los dos tipos, preferiremos TIMESTAMP, cuyos campos solo ocupan cuatro bytes por valor en lugar de cinco bytes de DATETIME (ocho bytes antes de MySQL 5.6).

Observe que los tipos TIMESTAMP y DATETIME disponen de dos atributos opcionales que pueden prestar grandes servicios. Con `DEFAULT CURRENT_TIMESTAMP`, cuando una línea se inserta, el campo TIMESTAMP de esta línea se actualiza de forma automática con la fecha y hora de la inserción. Y con `ON UPDATE CURRENT_TIMESTAMP`, el mismo campo se actualiza cada vez que un UPDATE tiene lugar. Solo un campo por tabla puede llevar estos dos atributos.

- ☐ Puede encontrar campos INT que contienen el número de segundos desde el 1 de enero de 1970. Estos campos se reemplazan con ventajas por campos TIMESTAMP, que son capaces de almacenar la misma gama de valores, ocupan el mismo espacio en disco y son mucho más fáciles de leer.

Segundo caso: ¿cómo almacenar datos temporales con una granularidad inferior al segundo?

Antes de MySQL 5.6, no existía ningún tipo adecuado; estábamos, por lo tanto, obligados a pasar por soluciones poco prácticas, como guardar en un campo BIGINT el número de microsegundos transcurridos desde una fecha determinada.

MySQL 5.6 ahora puede almacenar una precisión hasta el microsegundo para los tipos de datos TIME, DATETIME y TIMESTAMP.

Último caso: ¿cómo almacenar datos temporales que excedan la capacidad de los tipos existentes?

Una vez más, hay que recurrir a soluciones de compromiso, como el uso de un campo BIGINT.

f. ENUM y SET

ENUM y SET son dos tipos de datos muy prácticos, que toman como parámetro una lista de cadenas de caracteres aceptadas y almacenan los resultados en forma de números enteros. ENUM permite almacenar un valor de la lista de parámetros, mientras que SET permite grabar uno o varios valores.

Esta forma de trabajar permite obtener campos compactos y rápidos: un campo ENUM acepta hasta 65 535 parámetros y requiere uno o dos bytes según el número de parámetros, mientras que SET acepta hasta 64 parámetros y requiere de uno a ocho bytes según el número de parámetros.

ENUM o SET sustituyen de forma ventajosa una clave externa a una tabla de referencia cuando se trata de almacenar en un campo los valores de una lista que no evoluciona a lo largo del tiempo.

Tomemos el caso de la base de datos de ejemplo world:

```
mysql> SHOW CREATE TABLE Country\G
CREATE TABLE Country (
  Code char(3) NOT NULL default '',
  Name char(52) NOT NULL default '',
  Continent enum('Asia','Europe','North
America','Africa','Oceania','Antarctica','South America') NOT NULL
default 'Asia',
```

```
...  
PRIMARY KEY (Code)  
)ENGINE=MyISAM;
```

El motor de almacenamiento de esta tabla es MyISAM; no es posible crear una clave externa para una tabla que contiene la lista de los continentes. Tampoco sería mejor cambiar el campo al tipo VARCHAR (¿qué ocurre si alguien inserta Europa en lugar de Europe?).

La inserción se realiza simplemente como si se tuviera una cadena de caracteres, salvo que cualquier valor que no forme parte de la lista que se describe en la definición del campo será rechazado (o convertido en una cadena vacía en SQL_MODE):

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

```
mysql> INSERT INTO Country (Code,Name,Continent) VALUES  
( 'SAK', 'Sakila', 'Europe' );  
Query OK, 1 row affected (0,00 sec)
```

```
mysql> INSERT INTO Country (Code,Name,Continent) VALUES  
( 'FLI', 'Flipper', 'Europa' );  
ERROR 1265 (01000): Data truncated for column 'Continent' at row 1
```

2. Normalización

a. Papel de la normalización

Para almacenar los datos de una aplicación, hay muchas maneras de estructurar las tablas. Sin embargo, no todas las maneras son equivalentes desde el punto de vista del rendimiento o de la integridad de datos. Nacida a comienzos de la década de 1970, la teoría relacional tiene como objetivo presentar un conjunto de normas que permitan obtener un modelo de datos eficiente y seguro. Estas normas están agrupadas en diferentes categorías y definen los niveles de normalización. Un ejemplo nos mostrará qué problemas pueden surgir para

un modelo no normalizado, así como los pasos que hay que seguir para normalizar un modelo en las tres primeras formas normales, que son las más comunes.

b. Primera forma normal

El ejemplo se basa en la siguiente tabla, que contiene las competencias de los empleados de una empresa:

```
mysql> CREATE TABLE empleado (  
  id int(11) NOT NULL,  
  nombre varchar(30) NOT NULL DEFAULT '',  
  ciudad varchar(30) NOT NULL DEFAULT '',  
  competencia1 varchar(30) NOT NULL DEFAULT '',  
  niv1 tinyint(4) NOT NULL,  
  competencia2 varchar(30) NOT NULL DEFAULT '',  
  niv2 tinyint(4) NOT NULL,  
  PRIMARY KEY (id)  
) ENGINE=InnoDB;  
  
mysql> SELECT * FROM empleado;  
+-----+-----+-----+-----+-----+-----+-----+  
+  
| id | nombre | ciudad | competencia1 | niv1 | competencia2 | niv2 |  
+-----+-----+-----+-----+-----+-----+-----+  
+  
| 1 | Arthur | Paris | Linux | 2 | MySQL | 1 |  
+-----+-----+-----+-----+-----+-----+-----+  
| 2 | Bob | Burdeos | Apache | 3 | MySQL | 2 |  
+-----+-----+-----+-----+-----+-----+-----+  
| 3 | Charlie | Estrasburgo | MySQL | 3 | PHP | 1 |  
+-----+-----+-----+-----+-----+-----+-----+  
--+
```

Este modelo adolece de muchas deficiencias, entre ellas:

- La dificultad de recuperar cierta información: si queremos conocer todos los ámbitos de competencia de la empresa, la petición que nos dará este resultado dista de ser sencilla de escribir.
- La rigidez: en efecto, si desea agregar nuevos conocimientos a un empleado, hay que modificar la estructura de la tabla y por lo tanto algunas peticiones.
- Los riesgos de incoherencia de los datos: los nombres de competencias se almacenan de forma directa como cadenas de caracteres, lo que puede dar lugar a errores (PHP, php y P.H.P. ¿son la misma competencia o no?).

Para resolver estos problemas, vamos a normalizar la tabla, comenzando por la primera forma normal. La condición es no tener ningún grupo de columnas que se repita. Aquí, dos grupos de columnas se repiten: (competencia1, niv1) y (competencia2, niv2). Basta, para corregir el problema, con tener una pareja (conocimiento, nivel) y escribir para cada empleado tantas filas en la tabla como competencias tenga.

Obtenemos la siguiente tabla:

```
mysql> CREATE TABLE empleado2 (
  id int(11) Not NULL,
  nombre varchar(30) Not NULL DEFAULT '',
  ciudad varchar(30) Not NULL DEFAULT '',
  competencia varchar(30) Not NULL DEFAULT '',
  nivel tinyint(4) Not NULL
) ENGINE=InnoDB;
```

```
mysql> select * from empleado2;
```

id	nombre	ciudad	competencia	nivel
1	Arthur	Paris	Linux	2
1	Arthur	Paris	MySQL	1
2	Bob	Burdeos	Apache	3
2	Bob	Burdeos	MySQL	2
3	Charlie	Estrasburgo	MySQL	3
3	Charlie	Estrasburgo	PHP	1

c. Segunda forma normal

La tabla empleado2 elimina los problemas de rigidez de la estructura de datos y los problemas para recuperar cierta información. Sin embargo, los riesgos de incoherencia debidos a la redundancia de datos están todavía presentes en las columnas nombre, ciudad y competencia. Vamos a dividir la tabla en dos partes:

```
Mysql> CREATE TABLE empleado3 (  
  id_emp int(11) Not NULL,  
  nombre varchar(30) Not NULL DEFAULT '',  
  ciudad varchar(30) Not NULL DEFAULT '',  
  PRIMARY KEY (id_emp)  
) ENGINE=InnoDB;
```

```
mysql> CREATE TABLE competencia (  
  id_saber int(11) Not NULL,  
  id_emp int(11) Not NULL,  
  saber varchar(30) Not NULL DEFAULT '',  
  nivel tinyint(4) Not NULL,  
  PRIMARY KEY (id_saber,id_emp)  
) ENGINE=InnoDB;
```

```
mysql> select * from empleado3;  
+-----+-----+-----+  
| id_emp | nombre | ciudad |  
+-----+-----+-----+  
| 1      | Arthur | Paris  |  
| 2      | Bob    | Burdeos|  
| 3      | Charlie| Estrasburgo|  
+-----+-----+-----+
```

```
mysql> select * from competencia;  
+-----+-----+-----+-----+  
| id_saber | id_emp | competencia | nivel |
```

1	1	Linux	2		
2	1	MySQL	1		
2	2	MySQL	2		
2	3	MySQL	3		
3	2	Apache	3		
4	3	PHP	1		

Ahora la tabla empleado3 no presenta más redundancias y se encuentra en la segunda forma normal. En efecto, una tabla está en la segunda forma normal si está en primera forma normal y todos los campos fuera de la clave primaria dependen totalmente de la clave primaria.

Para la tabla competencia, obtendrá siempre redundancias en la columna competencia. Esta tabla no respeta todavía la segunda forma normal, ya que la columna competencia solo depende de la columna id_competencia y no de toda la clave primaria, formada por la pareja (id_competencia, id_emp).

Debemos, por tanto, dividir la tabla competencia:

```

Mysql> CREATE TABLE competencia2 (
  id_saber int(11) Not NULL,
  saber varchar(30) Not NULL,
  PRIMARY KEY (id_saber)
) ENGINE=InnoDB;

mysql> CREATE TABLE competencia_empleado (
  id_competencia int(11) Not NULL,
  id_emp int(11) Not NULL,
  nivel tinyint(4) Not NULL,
  PRIMARY KEY (id_saber, id_emp)
) ENGINE=InnoDB;

mysql> select * from competencia2;
+-----+

```


id_competencia	competencia
1	Linux
2	MySQL
3	Apache
4	PHP

```
mysql> select * from competencia_empleado;
```

id_competencia	id_emp	nivel
1	1	2
2	1	1
2	2	2
2	3	3
3	2	3
4	3	1

Las tablas `competencia2` y `competencia_empleado` ya no tienen redundancias y cumplen con la segunda forma normal.

d. Tercera forma normal

Para que una tabla respete la tercera forma normal, debe estar en segunda forma normal y todos los campos fuera de la clave primaria deben depender solo de la clave primaria y no de otros campos que no forman parte de la clave primaria.

La tabla `competencia2` está en tercera forma normal, ya que la columna `competencia` depende solo de la clave primaria `id_competencia`. Es también el caso de la tabla `competencia_empleado`, cuya columna `nivel` depende solo de la clave primaria (`id_competencia`, `competencia`).

Al contrario, la tabla `empleado3` no está en la tercera forma normal, ya que el campo `ciudad` depende en primer lugar del campo `nombre` antes de depender de la clave primaria. Para hacer esta constatación evidente, basta con imaginar un cuarto registro en la

tabla cuya ciudad sea Estrasburgo. Tendríamos entonces una redundancia del nombre de la ciudad, y por tanto un riesgo de incoherencia de los datos. Debemos dividir la tabla empleado3 para producir el modelo final siguiente:

```
mysql> CREATE TABLE empleado4 (  
    id_emp int(11) NOT NULL,  
    nombre varchar(30) NOT NULL,  
    id_ciudad int(11) NOT NULL,  
    PRIMARY KEY (id_emp)  
) ENGINE=InnoDB;
```

```
mysql> SELECT * FROM empleado4;  
+-----+-----+-----+  
| id_emp | nombre  | id_ciudad |  
+-----+-----+-----+  
|      1 | Arthur  |      1    |  
|      2 | Bob     |      2    |  
|      3 | Charlie |      3    |  
+-----+-----+-----+
```

```
mysql> CREATE TABLE ciudad (  
    id_ciudad int(11) NOT NULL,  
    ciudad varchar(30) NOT NULL,  
    PRIMARY KEY (`id_ciudad`)  
) ENGINE=InnoDB;
```

```
mysql> SELECT * FROM ciudad;  
+-----+-----+  
| id_ciudad | ciudad    |  
+-----+-----+  
|      1    | Paris     |  
|      2    | Burdeos   |  
|      3    | Estrasburgo |  
+-----+-----+
```

```
mysql> CREATE TABLE competencia2 (  
    id_competencia int(11) NOT NULL,
```

```
competencia varchar(30) NOT NULL,
PRIMARY KEY (id_competencia)
) ENGINE=InnoDB;
```

```
mysql> SELECT * FROM competencia2;
```

```
+-----+-----+
| id_competencia | competencia |
+-----+-----+
|          1 | Linux      |
|          2 | MySQL      |
|          3 | Apache     |
|          4 | PHP        |
+-----+-----+
```

```
mysql> CREATE TABLE competencia_empleado(
  id_competencia int(11) NOT NULL,
  id_emp int(11) NOT NULL,
  nivel tinyint(4) NOT NULL,
  PRIMARY KEY (id_competencia,id_emp)
) ENGINE=InnoDB;
```

```
mysql> SELECT * FROM competencia_empleado;
```

```
+-----+-----+-----+
| id_competencia| id_emp | nivel |
+-----+-----+-----+
|          1 |      1 |      2 |
|          2 |      1 |      1 |
|          2 |      2 |      2 |
|          2 |      3 |      3 |
|          3 |      2 |      3 |
|          4 |      3 |      1 |
+-----+-----+-----+
```

Al estar el esquema normalizado, ¿podemos acceder a la misma información que antes de la normalización? Sí, la siguiente petición obtiene por ejemplo la información disponible de todos los empleados:

```
mysql> SELECT e.id_emp,e.nombre,v.ciudad,c.competencia,ce.nivel
        FROM empleado4 e INNER JOIN ciudad v USING(id_ciudad)
        INNER JOIN competencia_empleado ce USING(id_emp)
        INNER JOIN competencia2 c USING(id_competencia);
```

id_emp	nombre	ciudad	competencia	nivel
1	Arthur	Paris	Linux	2
1	Arthur	Paris	MySQL	1
2	Bob	Burdeos	MySQL	2
2	Bob	Burdeos	Apache	3
3	Charlie	Estrasburgo	MySQL	3
3	Charlie	Estrasburgo	PHP	1

Y ahora es muy fácil encontrar el conjunto de competencias de la empresa:

```
Mysql> SELECT conocimiento FROM competencia2;
```

conocimiento
Linux
MySQL
Apache
PHP

- La normalización sola no protege de todos los problemas relacionados con las incoherencias de datos. Con este esquema, por ejemplo, podemos eliminar una línea de la tabla `ciudad`, lo que tendrá como consecuencia dejar como información de la ciudad de un usuario un identificador que apunta hacia un dato vacío en la tabla `empleado4`. Para eliminar este riesgo, definimos las

claves externas (descritas en la siguiente sección dedicada a los índices) para cada una de las relaciones.

e. Resumen de las ventajas de la normalización

La normalización exige un esfuerzo de reflexión para tener éxito en el diseño de tablas sin errores. A continuación hay que hacer un esfuerzo adicional para escribir los joins que permiten recuperar los datos que nos interesen. Pero el balance sigue siendo muy positivo, ya que la normalización de un modelo proporciona los beneficios siguientes:

- Los datos ya no tienen redundancia. Así, el riesgo de incoherencia en caso de modificación o supresión se limitará al máximo. Además, se reduce el espacio en disco necesario para el almacenamiento, lo que permite a las tablas y las filas ser más compactas; por lo tanto, tenerlas en memoria con mayor facilidad y estar bloqueadas menos tiempo.
- La estructura de datos es más flexible. Esto permite más posibilidades para combinar los datos, eliminando el uso de las cláusulas como `DISTINCT` o `GROUP BY` para filtrar las repeticiones.

f. Inconvenientes de la normalización

Un esquema normalizado tiene un gran inconveniente: aparte de algunas peticiones muy simples, la gran mayoría de las peticiones deberán escribirse en forma de joins entre tablas atómicas, es decir, desglosadas al máximo. En algunos casos, la descomposición produce el efecto de colocar en tablas distintas las columnas que hubiéramos deseado agrupar para los índices de forma eficaz. Además, ejecutar joins significa que el servidor accederá más a menudo a los datos mediante accesos aleatorios, el tipo de acceso más lento (ver al respecto la sección dedicada a los accesos a datos en este mismo capítulo). Por eso a veces es necesario no seguir la teoría y desnormalizar.

3. Desnormalización

Si los datos se mantienen en la misma tabla, los posibles problemas de indexación podrían eliminarse y los joins se convertirían, en el peor de los casos, en un recorrido secuencial de toda la tabla, lo que puede ser más rápido que muchos accesos aleatorios.

La desnormalización consiste en restablecer las tablas que hemos desglosado en el proceso de normalización con el fin de evitar uniones. El nivel de desnormalización dependerá de la aplicación y las peticiones que tendremos que ejecutar.

También existe una posibilidad intermedia, que consiste en generar tablas que agregan una serie de datos, lo que permite ahorrar uniones llamando directamente a las tablas así generadas. Las actualizaciones de estas tablas pueden hacerse mediante un proceso periódico si los datos pueden estar ligeramente desfasados o por triggers si los resultados siempre deben estar al día.

La desnormalización solo es beneficiosa cuando estamos seguros de que los problemas de rendimiento proceden de las tablas normalizadas y hemos optimizado al máximo todos los parámetros bajo nuestro control.

- ☐ Observe que desnormalizar nos expone a todos los riesgos de inconsistencias debido a la redundancia de datos descritos en la sección anterior sobre la normalización.

En caso de problemas de rendimiento, es necesario siempre comenzar por normalizar el modelo de datos y optimizar al máximo las peticiones, en particular, colocando buenos índices. Si después de todas estas optimizaciones los joins siguen siendo lentos, solo entonces habrá que plantearse la cuestión de la desnormalización. Este punto es crucial porque la necesidad de desnormalización debería ser excepcional: hay muchos más casos en los que la solución pasa por normalizar que casos en los que la solución consiste en desnormalizar. Por ejemplo, desnormalizar para tratar de mejorar un join entre tres tablas de 100 000 filas cada una es sin duda una mala idea.

4. Modificación del esquema en producción

Históricamente, siempre fue difícil modificar en producción el esquema de tablas voluminosas. La razón es sencilla: durante mucho tiempo, todas las peticiones `ALTER TABLE` producían un bloqueo de lectura sobre la tabla y, por lo tanto, la tabla no podía estar accesible en escritura durante toda la duración de la operación. Este bloqueo era aceptable cuando la modificación del esquema duraba unos segundos o posiblemente unos

cuantos minutos, pero en tablas con varias decenas o cientos de GB el tiempo de espera podía ser con facilidad de varias horas.

La mayoría de los administradores debían recurrir a trucos para poder realizar estas operaciones, como poner la aplicación en mantenimiento o realizar la operación durante las horas muertas. Pero, evidentemente, este tipo de estrategia no siempre funciona, por ejemplo si la aplicación no tiene ningún periodo muerto. Por esta razón, era corriente utilizar la replicación para facilitar el cambio de esquema:

- El esquema se modificaba en cada uno de los esclavos, uno por uno.
- Luego, uno de los esclavos era promovido a nuevo maestro.
- Posteriormente, el cambio de esquema era realizado en el antiguo maestro.
- Eventualmente, el antiguo maestro era de nuevo promovido a maestro.

Esta estrategia funciona muy bien porque el tiempo de interrupción es bajo (solo el tiempo de promoción de un esclavo) y, sobre todo, porque el tiempo de interrupción es totalmente independiente del tiempo tomado por el comando `ALTER TABLE`.

La mayoría de los administradores experimentados han recurrido alguna vez a esta estrategia, pero no solo tiene beneficios:

- A menos que utilicemos herramientas para automatizar el proceso, la promoción de un esclavo no es fácil, sobre todo cuando el sistema funciona con muchos esclavos.
- El cambio de esquema se efectuará esclavo por esclavo, lo que es muy costoso en términos de tiempo si tenemos muchos esclavos. Por ejemplo, si tenemos una veintena de esclavos y el comando `ALTER TABLE` toma varias horas, sin duda actualizaremos un solo esclavo cada día. En total, la operación demandará veinte días, lo que es una eternidad en ciertos entornos.

Por todas estas razones, se desarrollaron herramientas que permiten emular comandos `ALTER TABLE` no bloqueantes. Las dos más conocidas son la herramienta de Facebook `OnlineSchemaChange` y la herramienta de Percona `pt-online-schema-change`. Su funcionamiento es similar: estas herramientas crean una nueva tabla vacía con el nuevo esquema, insertan los datos de la antigua tabla en la nueva, y registran todos los cambios en la antigua tabla para poder volcarlos en la nueva, y finalmente intercambian los nombres de las tablas y eliminan la antigua tabla.

La herramienta de Percona es la más popular; aquí le explicamos cómo usarla:

Imaginemos que deseamos agregar un índice sobre la tabla `my_app.t`. El comando `ALTER TABLE` correspondiente podría ser:

```
mysql> ALTER TABLE my_app.t ADD INDEX idx_col1_col2 (col1, col2);
```

Con `pt-online-schema-change`, el comando que se ejecutará será:

```
# pt-online-schema-change --alter=' ADD INDEX idx_col1_col2 (col1, col2)' D=my_app,t=t u=root,p=xyz,h=localhost
```

Las ventajas de `pt-online-schema-change` son las siguientes:

- Ejecutamos el script en el maestro de replicación, el cambio de esquema no es bloqueante y será automáticamente replicado sobre todos los esclavos.
- El script supervisa la carga del maestro y se pone en pausa si se crece demasiado (ver las opciones `--max-load` y `--critical-load`).
- El script supervisa el retraso de replicación de los esclavos y se pone en pausa si se supera cierto umbral (ver la opción `--max-lag`).

Observe que, por definición, este script modifica sus datos, ya que cambia el esquema de una tabla. Por consiguiente, es muy recomendable leer la documentación (<https://www.percona.com/doc/percona-toolkit/2.2/pt-online-schema-change.html>) y probar la herramienta antes de usarla en producción. Tenga en cuenta la opción `--dry-run`, que permite simular la ejecución del script. Se trata de una opción muy útil para informar de un problema potencial antes de comenzar a modificar los datos.

Observe también que la herramienta no carece de inconvenientes:

- Ya que el script intenta no aumentar demasiado la carga de los servidores, los cambios de esquema son mucho más largos que con un `ALTER TABLE` clásico.
- Nunca hay que ejecutar `pt-online-schema-change` en una tabla en la que

una petición muy larga está en ejecución, incluso un `SELECT`. Dicha petición genera, en efecto, un bloqueo especial (*metadata lock*) que impide la modificación de la tabla. Asistiremos entonces a un desastroso efecto bola de nieve: una petición bloquea `pt-online-schema-change`, el cual bloquea todas las peticiones que modifican la tabla... y, después de un momento, el servidor alcanza el máximo de conexiones permitidas.

- `pt-online-schema-change` utiliza triggers para capturar los cambios. Antes de MySQL 5.7, si la tabla ya contaba con triggers, no podíamos utilizar la herramienta, ya que no se puede tener varios triggers por acción.

A partir de MySQL 5.6, y si usamos InnoDB, `pt-online-schema-change`, ya no es tan necesaria como antes, ya que la mayoría de los cambios de esquema pueden efectuarse de forma nativa de manera no bloqueante ejecutando un comando `ALTER TABLE` estándar. Algunas operaciones siguen siendo bloquantes. Se recomienda consultar la documentación en línea para saber exactamente qué cambios son bloqueantes y cuáles no lo son:

- Para MySQL 5.6: <https://dev.mysql.com/doc/refman/5.6/en/innodb-create-index-overview.html>
- Para MySQL 5.7: <https://dev.mysql.com/doc/refman/5.7/en/innodb-create-index-overview.html>

Esta característica de cambio de esquema no bloqueante es muy interesante, ya que evita tener que invertir tiempo en el aprendizaje de un script externo. Pero `pt-online-schema-change` sigue siendo a menudo una mejor elección en los entornos que utilizan la replicación. En efecto, las escrituras procedentes del maestro se efectúan en serie sobre los esclavos. Por lo tanto, aun cuando el maestro puede llevar a cabo registros en paralelo del cambio de esquema, los esclavos no pueden. Si el cambio de esquema dura, por ejemplo, una hora, los esclavos tomarán todos como mínimo una hora de retraso de replicación, lo que puede plantear problemas si la aplicación lanza peticiones de lectura sobre los esclavos, suponiendo que el retraso de replicación no supere unos segundos. La replicación multithread resuelve en principio este problema con MySQL 5.7, pero todavía no se cuenta con mucha experiencia al respecto.

Indexación

1. Aspectos generales de los índices

a. Rol de un índice

Cuando las tablas crecen mucho, el servidor emplea cada vez más tiempo en recuperar los datos que los clientes solicitan, y esta hiperactividad se traduce, entre otras cosas, en peticiones largas de ejecutar. Para encontrar un buen rendimiento, una solución usual consiste en añadir uno o varios índices a la tabla. Un índice es una estructura de datos relacionada con una tabla y cuya función es comparable al índice en un libro: si quiere buscar una palabra en un libro, es más rápido buscar esa palabra en el índice, donde encontrará directamente el número de todas las páginas donde aparece, que no leer todo el libro de comienzo a fin.

Existen muchos tipos de índice; algunos tienen más limitaciones sobre los datos de la tabla, pero en todos los casos la finalidad es la misma: recuperar cuanto antes una referencia a los datos deseados.

La utilización de un índice para resolver una petición la decide el servidor durante la fase de optimización de la petición: antes de realizar una petición, el servidor intenta determinar cuál va a ser el medio más rápido de buscar los resultados utilizando un subprograma especializado denominado optimizador de peticiones.

b. Claves e índice

Algunos sistemas de bases de datos distinguen claramente las claves y los índices. Una clave es una limitación sobre los datos, mientras que un índice es una estructura que tiene por objeto permitir buscar rápidamente un conjunto de datos. Así, se puede definir en estos sistemas una limitación de unicidad o una limitación para obligar a todos los valores de una

columna a ser superiores a 10.

Con MySQL, el concepto de limitación también existe, pero de una manera más limitada. Y, sobre todo, todas las limitaciones se implementan empleando un índice. Por esta razón, en la continuación de este libro, las palabras clave e índice se considerarán a veces equivalentes, aun cuando en algunos casos será habitual que se emplee una en lugar de la otra.

c. Columnas que pueden beneficiarse de un índice

Todas las columnas de una tabla pueden, en teoría, beneficiarse de un índice. En la práctica, las candidatas son mucho menos numerosas: se trata principalmente de las columnas utilizadas para los joins, para los filtros (cláusula `WHERE`), para la ordenación (cláusula `ORDER BY`) o para las agregaciones (cláusula `GROUP BY`). Los criterios de selección de las columnas que se han de indexar se explicarán en detalle en la sección Optimización de las peticiones de este capítulo.

Recuerde que la colocación de un índice no es gratuita: todas las modificaciones realizadas en una columna indizada requerirán un mayor trabajo por parte del servidor.

Necesitaremos, por lo tanto, siempre medir la ganancia que puede aportar un índice con respecto a su coste: si un índice permite dividir el tiempo de ejecución de una petición por diez, entonces es útil y podemos mantenerlo. Pero si no aporta ningún beneficio medible y empeora las tareas del servidor, será interesante eliminarlo.

- ☐ MySQL, con muy pocas excepciones, solo puede usar un índice por tabla para una petición determinada. Esto explica por qué las columnas candidatas a la indización en última instancia son muy pocas.

d. Creación y eliminación de un índice

Puede crear los índices al mismo tiempo que la tabla.

Por ejemplo, el siguiente comando crea una tabla simple con un índice:

```
mysql> CREATE TABLE t(  
      id INT NOT NULL,
```

```
col1 VARCHAR(30) NOT NULL,  
col2 VARCHAR(30) NOT NULL,  
KEY idx_col1 (col1)  
);
```

☐ La palabra clave KEY podrá sustituirse por su equivalente INDEX.

El nombre del índice (aquí `idx_col1`) es opcional.

También puede agregar un índice con el comando `CREATE INDEX`:

```
mysql> CREATE INDEX idx_id ON t (id);
```

O con `ALTER TABLE`:

```
mysql> ALTER TABLE t ADD INDEX (col2);
```

Estos dos comandos presentan algunas diferencias sintácticas:

- Con `CREATE TABLE`, el nombre del índice será obligatorio, mientras que con `ALTER TABLE` es opcional.
- Si no se especifica un nombre para el índice con `ALTER TABLE`, MySQL le asigna uno de forma automática.
- `ALTER TABLE` permite crear varios índices a la vez separando las cláusulas `ADD INDEX` por una coma, lo que no es posible con `CREATE TABLE`.

Para borrar un índice, podemos utilizar bien `DROP INDEX` o `ALTER TABLE`. En ambos casos, debemos conocer el nombre del índice (y no el nombre de la columna indexada):

```
mysql> SHOW CREATE TABLE t\G
***** 1. row *****

Table: t
Create Table: CREATE TABLE t (
  id int(11) NOT NULL,
  col1 varchar(30) NOT NULL,
  col2 varchar(30) NOT NULL,
  KEY idx_id (id),
  KEY col2 (col2),
  KEY idx_col1 (col1)
) ENGINE=MyISAM DEFAULT CHARSET=latin1

mysql> DROP INDEX idx_col1 ON t;
Query OK, 0 rows affected (0,02 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE t DROP INDEX col2;
Query OK, 0 rows affected (0,04 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

- ☐ Si trabaja con tablas voluminosas, utiliza una versión antigua de MySQL (5.1 sin plug-in InnoDB o más antigua) y debe hacer varias operaciones simultáneas en los índices, prefiera ALTER TABLE a CREATE INDEX/DROP INDEX. En efecto, cada creación de índice iniciará la reconstrucción completa de la tabla y ALTER TABLE, agrupando las operaciones, permitirá reconstruir la tabla una sola vez. ¡Muy útil si la reconstrucción de una tabla toma dos horas!

e. ¿Qué columnas indexar?

Este punto se detalla en la continuación de este capítulo (sección Optimización de las peticiones). El papel principal de un índice es permitir al servidor leer el mínimo de datos posible para satisfacer una petición. Utilizará, por lo tanto, un índice cada vez que una petición requiera un filtrado, es decir, una condición WHERE.

Pero, como veremos en la parte de nociones avanzadas de este capítulo, un índice también puede ayudar a las peticiones con una selección. Un índice puede asimismo evitar en algunos casos tener que leer los datos. Es entonces un índice de cobertura.

2. Tipos de índice

a. Índices únicos

Un índice único en una columna es un índice que lleva una limitación: todos los valores no NULL de la columna deben ser diferentes. Si la columna acepta NULL, pueden existir varios valores NULL.

Creamos un índice único precisando `UNIQUE KEY` o `UNIQUE INDEX` al crear el índice.

El ejemplo siguiente muestra la creación de un índice y el efecto de la limitación:

```
mysql> CREATE TABLE t(
      id INT,
      UNIQUE KEY (id)
);

mysql> INSERT INTO t (id) VALUES (1);
Query OK, 1 row affected (0,08 sec)

mysql> INSERT INTO t (id) VALUES (1);
ERROR 1062 (23000): Duplicate entry '1' for key 'id'

mysql> INSERT INTO t (id) VALUES (NULL);
Query OK, 1 row affected (0,03 sec)

mysql> INSERT INTO t (id) VALUES (NULL);
Query OK, 1 row affected (0,00 sec)

mysql> SELECT * FROM t;
+-----+
| id    |
```

```
+-----+
| NULL |
| NULL |
|    1 |
+-----+
```

b. Claves primarias

Una clave primaria se asemeja a un índice único porque se trata de un índice que lleva también una limitación de unicidad de los valores indexados. Pero existen algunas diferencias esenciales entre estos dos tipos de índices:

- Solo puede existir una clave primaria por tabla, mientras que puede crear tantas claves únicas como desee.
 - Una clave primaria no puede contener valores NULL.
- ☐ Si colocamos un índice único en una columna definida como NOT NULL, el índice funcionará de forma equivalente a una clave primaria, con la diferencia de que, una vez más, podemos definir varios índices únicos en columnas NOT NULL.

La sintaxis para manipular las claves primarias es ligeramente diferente a la de otros índices. En particular, no podremos usar `CREATE INDEX`. Hay que saber también que una clave primaria siempre lleva el nombre `PRIMARY`:

```
mysql> CREATE TABLE t (id int);
```

```
mysql> CREATE INDEX `PRIMARY` ON t;
```

```
ERROR 1064 (42000): You have an error in your SQL syntax; check
the manual that corresponds to your MySQL server version for the
right syntax to use near '' at line 1
```

```
mysql> ALTER TABLE t ADD PRIMARY KEY (id);
```

```
Query OK, 0 rows affected (0,04 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> DROP INDEX `PRIMARY` ON t;
Query OK, 0 rows affected (0,09 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

c. Índices no únicos

Un índice no único no tiene ninguna restricción concreta, su utilidad es solo facilitar las búsquedas. No existe ninguna limitación específica para la creación de tales índices.

La creación de un índice no único se hace con la palabra clave INDEX o KEY, como hemos visto antes en la sección dedicada a la creación y la eliminación de un índice.

d. Índice en varias columnas

Para todos los tipos de índices antes descritos, podemos crear un índice sobre varias columnas al mismo tiempo, conocido también como índice compuesto o índice multicolumna:

```
mysql> CREATE TABLE t (id int, col1 VARCHAR(30), col2
VARCHAR(30));

mysql> CREATE INDEX idx_col12 ON t (col1,col2);
Query OK, 0 rows affected (0,11 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

¿Cuál es el interés de establecer un índice sobre varias columnas en comparación con un índice para cada columna? Cuando MySQL ejecuta una petición sobre una tabla, solo puede usar un índice. Sin embargo, en algunas situaciones, es conveniente aprovechar un filtrado simultáneo en varias columnas de la misma tabla: en este caso, solo es posible con un índice compuesto.

Los índices multicolumna también tienen una característica interesante: MySQL es capaz

de explotar un prefijo a la izquierda del índice. Así, cuando un índice se coloca en las columnas (A, B), si solo la columna A es útil para optimizar la petición, MySQL sabe considerar este índice como un índice en A solamente. El orden escogido para las columnas es muy importante, ya que un índice sobre (B, A) no puede cumplir la misma función.

- ☐ El concepto de prefijo izquierdo solo es válido en realidad para los índice b-tree (los algoritmos disponibles para representar a los índices se examinarán más adelante en esta sección).

e. Índice sobre un prefijo de columna

Para los campos que almacenan cadenas de caracteres, podemos no indexar los N primeros caracteres del campo: eso es lo que se conoce como un índice de un prefijo de columna; no confundir con el concepto de prefijo de índice descrito en el apartado anterior.

Para las columnas de tipo BLOB y afines, un índice solo puede crearse si especificamos un tamaño, para solo indexar un prefijo:

```
mysql> CREATE TABLE t(col TEXT);

mysql> CREATE INDEX idx_t ON t(col);
ERROR 1170 (42000): BLOB/TEXT column 'col' used in key
specification without a key length

mysql> CREATE INDEX idx_t ON t(col(50));
Query OK, 0 rows affected (0,03 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Este tipo de índice es útil sobre todo cuando tratamos cadenas potencialmente largas: eligiendo bien la longitud del prefijo, podemos obtener un índice mucho más compacto, con casi la misma capacidad de filtrado, que un índice sobre la totalidad de la columna.

Para encontrar qué tamaño debe utilizarse para el prefijo, podemos calcular la selectividad de diferentes prefijos y compararla con la selectividad del índice sobre la columna completa.

Tomemos como ejemplo la tabla `City`. Esta tabla tiene un campo `Name` definido como `CHAR(35)`. Las peticiones siguientes permiten determinar que 20 es un buen tamaño para el prefijo:

```
mysql> SELECT COUNT(distinct left(name,5))/COUNT(*) AS res5 FROM City;
+-----+
| res5  |
+-----+
| 0.8713 |
+-----+
```

```
mysql> SELECT COUNT(distinct left(name,10))/COUNT(*) AS res10
FROM City;
+-----+
| res10 |
+-----+
| 0.9708 |
+-----+
```

```
mysql> SELECT COUNT(distinct left(name,15))/COUNT(*) AS res15
FROM City;
+-----+
| res15 |
+-----+
| 0.9792 |
+-----+
```

```
mysql> SELECT COUNT(distinct left(name,20))/COUNT(*) AS res20
FROM City;
+-----+
| res20 |
+-----+
| 0.9801 |
+-----+
```

```
mysql> SELECT COUNT(distinct name)/COUNT(*) AS resTot FROM City;
```

```
+-----+
| resTot |
+-----+
| 0.9801 |
+-----+
```

- ☐ El cálculo presentado aquí es un cálculo de selectividad. Los conceptos de selectividad media y selectividad en el peor de los casos se presentarán más adelante en esta sección.

f. Índices redundantes

Cuando, a raíz de una posible mala manipulación, hemos creado varios índices sobre la misma columna, obtendremos índices redundantes. Antes de la versión 5.6, MySQL no alertaba nunca de este problema, ya que no se trata de un error bloqueante. A pesar de todo, tener índices repetidos no aporta ninguna ganancia en rendimiento al ejecutar las lecturas y retrasa la ejecución de las actualizaciones. Será, pues, beneficioso eliminar todos esos índices innecesarios.

Identificar los índices redundantes no es siempre evidente a primera vista. Tomemos un ejemplo para ver los diferentes casos:

```
mysql> CREATE TABLE t (
    col1 INT,
    col2 INT,
    PRIMARY KEY (col1,col2),
    KEY (col1),
    KEY (col2),
    UNIQUE KEY (col1)
);
```

Una clave primaria definida sobre (col1, col2), col1 es un prefijo izquierdo: el índice no único sobre col1 es innecesario.

`col2` no es un prefijo izquierdo de `(col1, col2)`: el índice no único sobre `col2` es útil.

La clave primaria indica que cada pareja `(col1, col2)` debe ser única, pero no impide tener varias veces el mismo valor para `col1`: las parejas `(1, 1)` y `(1, 2)` pueden existir con la única limitación de la clave primaria. El índice único sobre `col1` es útil.

- ☐ Además, la redundancia depende del tipo de índice (concepto que abordamos a continuación). Por ejemplo, con un índice b-tree, un índice sobre `(col1,col2)` es diferente de un índice sobre `(col2,col1)`, mientras que con un índice hash ambos serían redundantes.
- ☐ MySQL 5.6 es capaz de detectar si hemos definido dos veces un índice sobre la misma columna, pero no que hemos creado un índice que es un prefijo de otro índice.

Percona Toolkit puede ayudarle a detectar índices redundantes mediante `pt-duplicate-key-checker`. Esta herramienta presenta cada uno de los índices redundantes y ofrece una petición para eliminar cada índice innecesario.

- ☐ Para la instalación de Percona Toolkit, consulte el capítulo Instalación del servidor.

Ejemplo de uso:

```
mysql> CREATE TABLE tm(  
    col1 INT,  
    col2 INT,  
    KEY idx_c1 (col1),  
    KEY idx_c1_bis (col1),  
    KEY idx_c1c2 (col1,col2),  
    KEY idx_c2c1 (col2,col1),
```

```
KEY idx_c2(col2));
```

```
shell> pt-duplicate-key-checker -uroot --ask-pass --databases=test
--tables=tm
```

```
Enter password:
```

```
#
#####
#####
# test.tm
#
#####
#####
```

```
# idx_c2 is a left-prefix of idx_c2c1
# Key definitions:
#   KEY `idx_c2` (`col2`)
#   KEY `idx_c2c1` (`col2`,`col1`),
# Column types:
#   `col2` int(11) DEFAULT NULL
#   `col1` int(11) DEFAULT NULL
# To remove this duplicate index, execute:
ALTER TABLE `test`.`tm` DROP INDEX `idx_c2`;
```

```
# idx_c1 is a left-prefix of idx_c1c2
# Key definitions:
#   KEY `idx_c1` (`col1`),
#   KEY `idx_c1c2` (`col1`,`col2`),
# Column types:
#   `col1` int(11) DEFAULT NULL
#   `col2` int(11) DEFAULT NULL
# To remove this duplicate index, execute:
ALTER TABLE `test`.`tm` DROP INDEX `idx_c1`;
```

```
# idx_c1_bis is a left-prefix of idx_c1c2
# Key definitions:
#   KEY `idx_c1_bis` (`col1`),
#   KEY `idx_c1c2` (`col1`,`col2`),
# Column types:
```

```

# `col1` int(11) DEFAULT NULL
# `col2` int(11) DEFAULT NULL
# To remove this duplicate index, execute:
ALTER TABLE `test`.`tm` DROP INDEX `idx_c1_bis`;

#
#####
#####
# Summary of indexes
#
#####
#####

# Size Duplicate Indexes    0
# Total Duplicate Indexes   3
# Total Indexes             5

```

En este ejemplo, `pt-duplicate-key-checker` detecta los índices definidos varias veces, y también aquellos que han sido definidos como un prefijo izquierdo de otro índice.

Si estamos de acuerdo con lo que propone `pt-duplicate-key-checker`, no queda más que aplicar los comandos `ALTER TABLE`:

```

shell > pt-duplicate-key-checker -uroot --ask-pass
--databases=test --tables=TM | grep ALTER | mysql -uroot -p

```

- ☐ `pt-duplicate-key-checker` es también capaz de detectar para InnoDB cuándo un sufijo de un índice secundario es un prefijo de la clave primaria. Para entender este caso, consulte la sección sobre los index clusters InnoDB.

Observe que el script se basa en el comando `SHOW CREATE TABLE` y solo

accede en lectura al servidor. La aplicación de los cambios, por lo tanto, siempre queda a cargo del usuario.

g. Claves externas

Las claves externas representan el mecanismo preferido para garantizar la integridad referencial entre varias tablas. El servidor MySQL no soporta por sí mismo las claves externas: la implementación se deja a los motores de almacenamiento. Hoy en día, InnoDB es el único motor actual que proporciona esta funcionalidad.

He aquí un ejemplo de creación de una clave externa entre el campo `ref_nom` de una tabla `t2` (tabla hijo) y el campo `nom` de una tabla `t1` (tabla padre):

```
mysql> CREATE TABLE t1 (  
    id int(11) NOT NULL AUTO_INCREMENT,  
    nom varchar(20) NOT NULL DEFAULT '',  
    PRIMARY KEY (id),  
    KEY idx_nom (nom)  
) ENGINE=InnoDB;  
  
mysql> CREATE TABLE t2 (  
    id int(11) NOT NULL AUTO_INCREMENT,  
    ref_nom varchar(20) NOT NULL DEFAULT '',  
    PRIMARY KEY (id),  
    CONSTRAINT fk_nom FOREIGN KEY fk_ref_nom (ref_nom) REFERENCES t1  
(nom)  
) ENGINE=InnoDB;
```

- ☐ En este ejemplo, la clave externa solo referencia a una clave primaria o un índice único, lo que está permitido con InnoDB (extensión del estándar SQL).

Si no existe un índice sobre el campo correspondiente (aquí `t1.nom`) o un índice cuyo campo de referencia es un prefijo, entonces la instrucción `CREATE TABLE` de la tabla `t2` regresará un error del tipo:

ERROR 1005 (HY000): Can't create table 'test.t2' (errno: 150)

- ☐ La cláusula CONSTRAINT es opcional y el nombre de la limitación debe ser único en toda la base de datos.

El nombre de índice después de FOREIGN KEY (aquí `fk_ref_nom`) es opcional.

- ☐ Las claves externas pueden añadirse o eliminarse con ALTER TABLE.

Una clave externa puede referenciar varios campos al mismo tiempo, pero siempre en una misma tabla.

La creación de una clave externa indica al servidor que debe rechazar cualquier adición o modificación en la tabla hijo si el valor de la clave no tiene correspondencia en la tabla padre.

El comportamiento en una actualización o eliminación en la tabla padre depende de modificadores adicionales añadidos en la cláusula FOREIGN KEY:

- RESTRICT (o NO ACTION): cualquier intento de actualización o eliminación en la tabla padre provoca un error si una o varias filas correspondientes existen en la tabla hijo. Se trata de la acción por defecto si no se especifica nada en la cláusula FOREIGN KEY.
- CASCADE: cuando una fila se actualiza o elimina en la tabla padre, la fila o las filas correspondientes son actualizadas o eliminadas de forma automática en la tabla hijo.
- SET NULL: cuando se actualiza una fila o se elimina en la tabla padre, la fila o las filas correspondientes tomarán el valor NULL en la tabla hijo. Las columnas no deben ser NOT NULL en la tabla hijo, si no se inicia un error al definir la clave externa.

- El modificador SET DEFAULT que existe en otros SGBD no está autorizado con MySQL.

Algunos SGBD distinguen entre RESTRICT y NO ACTION; este no es el caso de MySQL.

Para añadir el modificador CASCADE a la tabla t2 del ejemplo anterior, podemos proceder así:

```
mysql> ALTER TABLE t2 DROP FOREIGN KEY fk_nom;

mysql> ALTER TABLE t2 ADD CONSTRAINT fk_nom FOREIGN KEY (ref_nom)
REFERENCES t1(nom) ON DELETE CASCADE ON UPDATE CASCADE;
```

3. Conceptos avanzados

a. Index b-tree

Con MySQL, la implementación física más frecuente de un índice se realiza en forma de un b-tree (*Balanced Tree* o árbol equilibrado en español, o árbol B). Podemos obtener esta información con el comando SHOW INDEX:

```
mysql> SHOW INDEX FROM world.City\G
***** 1. row *****
      Table: City
    Non_unique: 0
      Key_name: PRIMARY
Seq_in_index: 1
  Column_name: ID
    Collation: A
  Cardinality: 4079
      Sub_part: NULL
         Packed: NULL
```

Null:

Index_type: BTREE

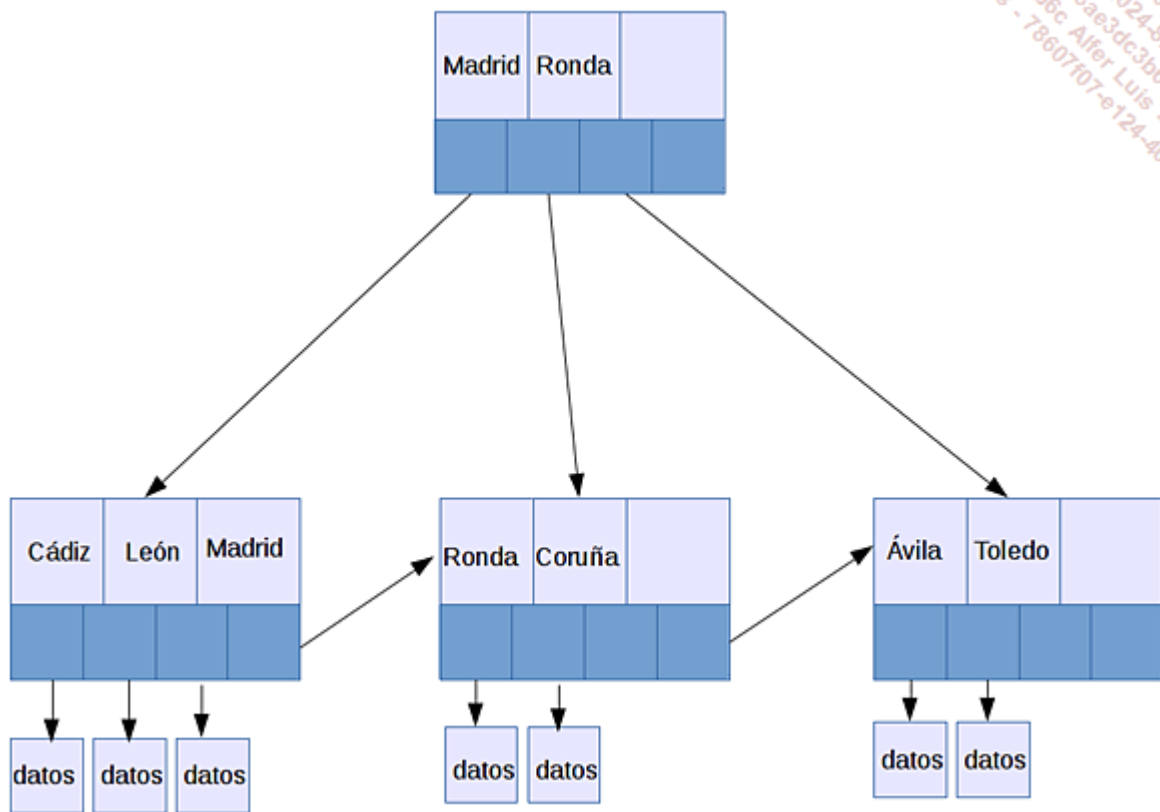
Comment:

Este tipo de árbol se encuentra equilibrado por naturaleza, lo que significa que todas las hojas se encuentran a la misma distancia de la raíz. Tiene características muy interesantes para representar un índice: las operaciones de inserción, borrado y de búsqueda se optimizan, los datos almacenados se seleccionan y la estructura es compacta y poco costosa en espacio en disco.

No todos los diferentes motores implementan necesariamente la estructura exacta de un árbol B. Así, InnoDB no utiliza árboles B, sino los árboles B+ que poseen algunas propiedades especiales, como el hecho de solo almacenar los datos en las hojas o de tener un puntero entre cada hoja, permitiendo recorridos secuenciales rápidos de las hojas en lugar de tener que remontar hasta la raíz para ir de hoja en hoja. Del mismo modo, MySQL utiliza árboles bicolors en lugar de árboles B.

Para evitar añadir complejidad, solo los términos árbol B e index b-tree se utilizarán a partir de ahora, con el objetivo de hacerle comprender los conceptos de la implementación de índice en forma de árboles, más que darle detalles muy técnicos con poco interés para el marco de este libro.

El diagrama siguiente resume de manera simplificada el principio de un árbol B, con un índice en la columna `ciudad` de una tabla.



La manera de apuntar a los datos dependerá del motor de almacenamiento. Por ejemplo, con MySQL, las hojas de un índice contendrán un puntero hacia la posición del dato en el archivo .MYD correspondiente. Con InnoDB, el mecanismo es diferente y se describe con más detalle a continuación en esta sección porque tiene implicaciones importantes sobre la manera de crear las tablas y los índices.

Un árbol B que representa un índice en una columna es capaz de responder a muchos criterios de búsqueda. Refiriéndonos al esquema anterior, veremos que es posible responder a las búsquedas siguientes con el índice:

- Buscar un dato completo: para encontrar si la ciudad de Ronda existe en la tabla (`ciudad = 'Ronda'`). Es posible que la petición retorne varias filas.
- Búsqueda en un prefijo de columna: para encontrar todas las ciudades que comienzan por 'Ro' (`ciudad LIKE 'Ro%'`).
- Búsqueda en un intervalo de valores: para encontrar todas las ciudades cuyo nombre comienza con una letra comprendida entre M y R (`ciudad BETWEEN 'R' AND 'V'`).
- Clasificación: los datos en el índice son clasificados por ciudad, la clasificación por ciudad es inmediata.

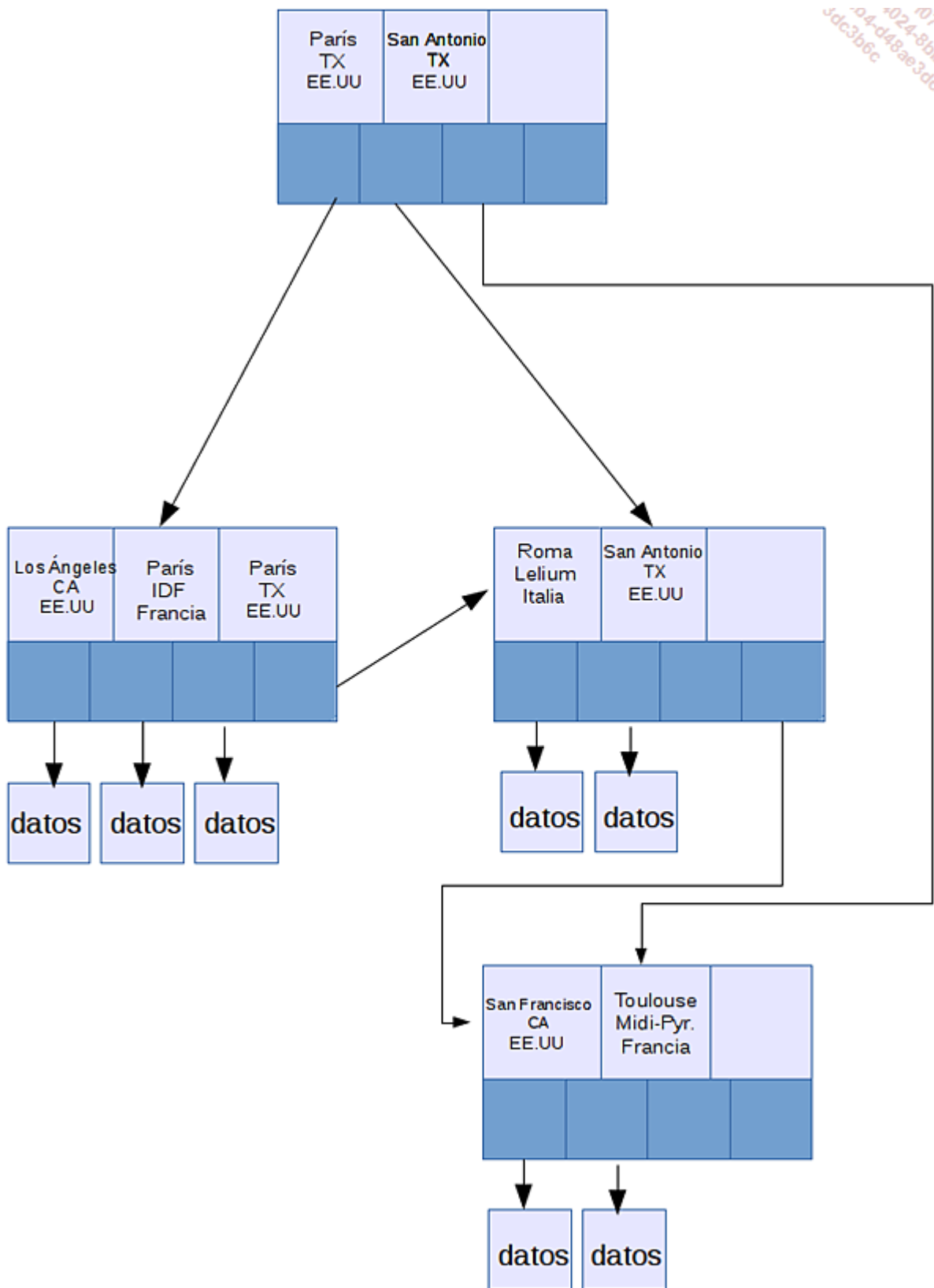
También es posible combinar varios criterios con el mismo índice. De esta forma, la siguiente petición puede satisfacerse con el mismo índice:

```
mysql> SELECT * FROM v WHERE ciudad BETWEEN 'R' AND 'V' AND ciudad  
LIKE 'SAN%' ORDER BY ciudad;
```

- ☐ En algunas situaciones, aunque en teoría es posible utilizar un índice, el optimizador de peticiones no hará esta elección: decidirá usar otro índice que considere más eficaz para la petición, aunque deba seleccionar un recorrido completo de la tabla.

Siempre consultando el esquema, notará que algunas operaciones no son a pesar de todo posibles con tal índice, como por ejemplo, la petición sobre un valor que no es un prefijo: el índice no es útil para buscar las ciudades que terminan por 'es' (`ciudad LIKE ' %es'`).

El diagrama siguiente representa, siempre de manera simplificada, la apariencia de un árbol B, con un índice en tres columnas (ciudad, provincia, país):



Las operaciones siguientes son posibles:

- Petición sobre un valor completo (ciudad = 'París' AND provincia = 'IDF' AND país = 'Francia').
- Petición sobre un valor exacto para un prefijo izquierdo del índice (ciudad = 'París').

Petición sobre un valor exacto para un prefijo izquierdo del índice y un intervalo de valores en la columna siguiente (`ciudad = 'París' AND provincia BETWEEN 'A' AND 'K'`).

- Petición sobre un valor exacto para un prefijo izquierdo del índice y un prefijo de la columna siguiente (`ciudad = 'París' AND provincia LIKE 'IDF%'`).
- Ordenación sobre todas las columnas, en orden ascendente (`ORDER BY ciudad, provincia, país`).
- Ordenación sobre un prefijo de columna, en orden ascendente (`ORDER BY ciudad, provincia, país`).
- Clasificación en la primera columna, en cualquier orden (`ORDER BY ciudad DESC`).

Todas las demás operaciones son imposibles de realizar con el índice, en particular:

- Búsqueda en un prefijo izquierdo del índice y varios intervalos de valor (`ciudad = 'París' AND provincia LIKE 'I%' AND país LIKE 'F%'`). En este caso, el índice puede usarse hasta el primer intervalo, es decir, aquí para las columnas `ciudad` y `provincia`.
- Búsqueda sobre las columnas más allá del primer intervalo de valor (`ciudad LIKE 'París' AND provincia LIKE 'I%' AND país = 'Francia'`). En este caso, el índice puede usarse hasta el primer intervalo.
- Búsqueda sobre otra cosa diferente de un prefijo izquierdo del índice (`provincia = 'TX'`).
- Selección distinta de las mencionadas antes (`ORDER BY país, provincia`).

☐ Los índices b-tree autorizan los índices de cobertura, tema detallado en la continuación de esta sección.

El conocimiento de las operaciones posibles o imposibles con los árboles B permitirá comprender mejor en qué ocasiones un índice puede ser útil.

He aquí un ejemplo:

```
mysql> SELECT * FROM City WHERE Name LIKE '%illa';
```

Ahora sabemos que, incluso si se añade un índice sobre la columna `Name`, MySQL no lo utilizará para esta petición, ya que la cláusula `LIKE` tal como está escrita no representará un prefijo del valor indexado.

He aquí un pequeño truco: podemos transformar un intervalo de números enteros en una cláusula `IN ()` que enumera los diferentes valores. Esta transformación permite ahora al índice ser utilizado en la columna que lleva el `IN ()`.

El siguiente ejemplo pondrá en práctica esta técnica.

A partir de la base de ejemplo `Country`, deseamos conocer los países que se independizaron entre 1965 y 1970 y cuya `Population` excede los 2 millones de habitantes; vamos a escribir una petición que no utiliza el índice por defecto.

El comando `EXPLAIN`, que permite obtener información sobre el plan de ejecución y que se detallará más adelante en este capítulo, indica que ningún índice es utilizable y, por lo tanto, ningún índice se utilizará.

```
mysql> EXPLAIN SELECT Name FROM Country WHERE IndepYear BETWEEN
1965 AND 1970 AND Population > 2000000\G
***** 1. row *****
...
possible_keys: NULL
key: NULL
...
```

Si añade un índice a `(IndepYear,Population)`, este no se utilizará por completo. Para darse cuenta, basta con crear también un índice sobre `IndepYear` solo y volver a ejecutar el comando `EXPLAIN`:

```
mysql> ALTER TABLE Country ADD INDEX idx_ind (IndepYear), ADD
INDEX idx_ind_pop (IndepYear,Population);

mysql> EXPLAIN SELECT Name FROM Country WHERE IndepYear BETWEEN
1965 AND 1970 AND Population > 2000000\G
***** 1. row *****

...
possible_keys: idx_ind,idx_ind_pop
key: idx_ind
...
```

Si convierte el BETWEEN en IN:

```
mysql> EXPLAIN SELECT Name FROM Country WHERE IndepYear
IN(1965,1966,1967,1968,1969,1970) AND Population > 2000000\G
***** 1. row *****

...
possible_keys: idx_ind,idx_ind_pop
key: idx_ind_pop
...
```

El índice de (IndepYear,Population) se utiliza ahora.

- ☐ Este truco resulta especialmente interesante cuando la lista de valores de la cláusula IN no es demasiado larga.

b. Índice hash

Un índice hash consiste en una estructura mucho más simple que un índice b-tree: se trata de una tabla de control que permite almacenar por un lado el resultado del control de los valores de las columnas indexadas y, por otro, un puntero hacia la fila de datos.

En una tabla de hash, se emplea por estadística el mismo tiempo para acceder a un

elemento, sea cual sea su posición en la tabla. Además, la función hash se elige para dar resultados compactos, sea cual sea el tamaño del dato inicial que se ha de trocear. Por estas razones, un índice hash suele ser muy eficaz cuando puede usarse.

Sin embargo, solo existe un caso para el empleo de un índice hash: la búsqueda de un valor concreto. En efecto, los hash de dos valores cercanos no tienen ninguna relación entre ellos, la tabla de control no llevará ninguna indicación sobre el orden de los valores: las peticiones por intervalo de valores o las clasificaciones son imposibles utilizando el índice. Del mismo modo, el hash de un prefijo no es el prefijo de un hash; por lo tanto, todas las peticiones por prefijo que eran viables con los índices b-tree no son posibles con un índice hash.

En la actualidad, este tipo de índice solo está disponible para el motor `Memory`, por defecto. Tenga en cuenta que también puede configurar los índices b-tree con las tablas `Memory`, para lo que emplearemos una de las sintaxis siguientes:

- Al crear la tabla:

```
mysql> CREATE TABLE t(  
    id INT,  
    INDEX idx_id USING BTREE(id)  
    ) ENGINE = MEMORY;
```

- Después de haber creado la tabla con `CREATE INDEX`:

```
mysql> CREATE INDEX idx_id USING BTREE ON t (id);
```

- Después de haber creado la tabla con `ALTER TABLE`:

```
mysql> ALTER TABLE t ADD INDEX idx_id USING BTREE (id);
```

- Los demás motores de almacenamiento aceptan la sintaxis con `USING` para especificar un algoritmo de indexación, pero no toman en cuenta la instrucción.

MySQL permite la creación de índices hash para valores no únicos. En este caso, los punteros hacia las diferentes filas se almacenarán en forma de una lista encadenada.

c. Otros algoritmos de indexación

Además de los índices de tipo árbol B y hash que hemos descrito, MySQL utiliza algunos otros algoritmos para sus índices:

- El motor NDB utiliza árboles T, una variante del árbol B.
- El motor MyISAM utiliza árboles R para los índices espaciales, otra variante de árbol.
- El motor MyISAM utiliza un doble árbol B para los índices FULLTEXT, que veremos en la continuación de este capítulo.
- Algunos motores de almacenamiento pueden utilizar otros algoritmos. Es el caso, por ejemplo, de Tokudb con sus árboles fractales.

Por último, con MySQL no existe la posibilidad de crear un índice de mapa de bits o un índice de una función.

d. Selectividad y distribución de valores

Cuando desee construir un índice en un prefijo de columna, puede calcular la selectividad media para diferentes tamaños de prefijo, como hemos descrito en el apartado sobre los índices en un prefijo de columna. Las peticiones presentadas permiten calcular la selectividad de un índice en toda la columna y la selectividad de este mismo índice en un prefijo de la columna. Cuando ponemos un índice, el objetivo debe ser siempre poder filtrar el máximo de datos con este índice; por tanto, tener la mejor selección posible. Cuando buscamos el tamaño ideal para un prefijo, buscamos el mejor compromiso entre un prefijo largo y selectivo, pero que tendrá un alto coste en disco (y por tanto en tiempo de acceso), o un prefijo más corto y, por lo tanto, menos selectivo, pero más eficiente en espacio en disco.

Esta noción de selectividad tiene también sentido cuando se quiere poner un índice en una columna entera: si este índice es poco selectivo, solo filtrará unos pocos datos y el optimizador de MySQL estimará con certeza que no es interesante utilizarlo. La clave primaria representa el mejor de los casos: un valor en el índice corresponde a un único dato. Un ejemplo típico de un índice poco selectivo, y por lo tanto poco útil, es un índice sobre el sexo (hombre/mujer), ya que en la mayoría de las situaciones dispondremos de aproximadamente 50 % de mujeres y 50 % de hombres, lo que no ofrecerá un filtro eficaz.

Otro concepto importante es la distribución de valores: en un juego de datos reales, es frecuente que los datos no estén distribuidos de forma uniforme en el conjunto de valores posibles, pero existe una o varias zonas que concentran la mayoría de los datos. Cuando desee calcular el tamaño de prefijo para un índice, el conocimiento de la distribución de los valores permite participar en la selectividad en el peor de los casos, y no solo en la selectividad media, como ha sido el caso hasta ahora. Si tenemos una tabla que contiene información acerca de películas y deseamos añadir un índice sobre el título, un prefijo de 3 dará quizá una selectividad media correcta, pero todas las películas cuyo título comienza por "The" serán consideradas iguales por el índice, dando una selectividad muy mala para el peor de los casos. Más vale en este caso añadir algunos caracteres en el índice para mejorar la selectividad.

Si volvemos a tomar el ejemplo del campo Name de la tabla City, podemos comparar la selectividad en el peor de los casos para toda la columna y para un prefijo de cinco caracteres:

```
mysql> SELECT Name AS prefijo,COUNT(name) AS total
        FROM City
        GROUP BY prefijo
        ORDER BY total DESC
        LIMIT 5;
```

prefijo	total
San José	4
La Paz	3
San Fernando	3
Richmond	3

Valencia	3
----------	---

```
mysql> SELECT left(Name,5) AS prefijo,COUNT(left(Name,5)) AS total
        FROM City
        GROUP BY prefijo
        ORDER BY total DESC
        LIMIT 5;
```

prefijo	total
Santa	26
Saint	13
San F	12
Ciuda	10
San J	10

Estas peticiones muestran que, tomando toda la columna, no tendremos ningún valor indexado que corresponda a más de cuatro valores diferentes, mientras que si tomamos un prefijo de cinco caracteres, ciertos valores indexados corresponderán a más de una decena de valores. Si usamos la misma petición para un prefijo de 20 caracteres, obtendremos un resultado comparable al resultado para el conjunto de la columna.

Sobre las tablas más representativas (la tabla `City` solo tiene 4000 registros), este efecto puede ser aún mayor, y es muy posible que un prefijo de un valor pueda corresponder a cientos de registros, mientras que los valores completos tienen pocas ocurrencias.

- ☐ Los juegos de datos generados de forma aleatoria reproducen en raras ocasiones la distribución de valores de la realidad: nada reemplaza, por lo tanto, los datos reales para hacer hipótesis correctas sobre la colocación de índices pertinentes.

El optimizador de peticiones es capaz de analizar la selectividad de un índice, y podrá decidir, para dos peticiones similares, utilizar un índice en un caso u otro dependiendo de si

la distribución de los valores para el índice es muy eficaz en un caso y poco eficaz en otro.

Podrá, por ejemplo, verificar este efecto con la tabla `film_Text` de la base `sakila`, añadiendo un índice a la columna `description` y ejecutando las peticiones siguientes (consulte la sección siguiente sobre el comando `EXPLAIN` si no conoce el concepto de plan de ejecución de una petición):

```
mysql> ALTER TABLE film_text ADD INDEX idx_desc
(description(50));
```

```
mysql> EXPLAIN SELECT * FROM film_text WHERE description LIKE 'A S
%\G
```

```
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: film_text
         type: range
possible_keys: idx_desc
           key: idx_desc
      key_len: 153
         ref: NULL
        rows: 41
   Extra: Using where
```

```
mysql> EXPLAIN SELECT * FROM film_text WHERE description LIKE 'A A
%\G
```

```
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: film_text
         type: ALL
possible_keys: idx_desc
           key: NULL
      key_len: NULL
         ref: NULL
        rows: 1000
   Extra: Using where
```

La primera petición utiliza como estaba previsto el índice de la columna `description`, lo que se indica por `key: idx_desc`. Pero en la segunda, aunque el optimizador considere el índice (`possible_keys: idx_desc`), lo descarta, ya que le parece demasiado costoso (`key: NULL`). Esto significa que el optimizador consideró que era más rápido recorrer las 1 000 filas de la tabla que recuperar directamente las filas que se distribuyen de forma aleatoria.

- A veces podemos encontrar información que indica que tan pronto como un valor indexado corresponde a, como mínimo, el 30 % de las filas de la tabla, MySQL prefiere recorrer toda la tabla en vez de utilizar el índice. Este umbral del 30 % es una leyenda y no se encuentra codificado en MySQL. En el ejemplo anterior, el valor 'A' está presente en 207 líneas de 1 000, lo que supone un porcentaje del 20 %. Sin embargo, el optimizador prefiere un recorrido completo de la tabla.

Tenga en cuenta que también hay casos en los que los valores están distribuidos de forma muy desigual y donde el interés de colocar un índice dependerá de la gama de valores sobre los que se centrarán sus peticiones. Por ejemplo, en una tabla de pedidos, si tenemos un campo de estado, es muy probable que al cabo de unos pocos meses el 99 % de los pedidos tengan la condición de archivado. Si en las peticiones queremos encontrar los pedidos en curso, es muy interesante tener un índice en la columna `estado`. Sin embargo, si deseamos solo calcular las estadísticas sobre los pedidos archivados, este índice es inútil por completo.

e. Index cluster InnoDB

InnoDB utiliza un método de almacenamiento de datos en el cual los datos se almacenan con uno de los índices de la tabla, que se denomina `index cluster`. Algunos SGBD permiten la elección del índice que se designa como clúster, pero este no es el caso de InnoDB: se trata por necesidad de la clave primaria.

Esta configuración tiene consecuencias importantes para todas las peticiones efectuadas

sobre las tablas InnoDB:

- Para los datos almacenados con la clave primaria, toda petición que utilice la clave primaria será muy eficiente.
- Toda petición que utilice un índice secundario será menos eficaz, ya que requerirá dos peticiones en los índices (una búsqueda en el índice secundario que da la clave primaria, y luego una búsqueda en la clave primaria para encontrar el dato).
- La elección de una clave primaria compacta es fundamental, ya que esta clave se añade de forma implícita al final de cada uno de los índices secundarios.
- La actualización del index cluster tiene un alto coste, ya que el sistema se actualiza al mismo tiempo que el índice y los datos asociados, y deber reorganizar una parte del índice para mantener la clasificación de los valores indexados.

InnoDB utiliza siempre un índice cluster. ¿Qué ocurre si no tenemos definida una clave primaria? El motor intentará encontrar un índice único que no tenga ningún valor NULL, y si no lo encuentra creará una clave primaria oculta. Más vale siempre definir una clave primaria con las tablas InnoDB.

El index cluster tiene una consecuencia importante sobre los índices secundarios: cualquier índice secundario que contenga la clave primaria como sufijo podrá reducirse eliminando todas las columnas de la clave primaria. Incluso es posible ir un poco más allá: cualquier prefijo izquierdo de la clave primaria que aparezca como sufijo de un índice secundario puede ser eliminado del índice secundario.

La detección de este caso no es siempre evidente. Por fortuna, Percona Toolkit puede ayudar con `pt-duplicate-key-checker`, una herramienta que fue presentada en el apartado sobre los índices redundantes.

El siguiente ejemplo muestra la ayuda que presta `pt-duplicate-key-checker`:

```
mysql> CREATE TABLE ti(  
    col1 INT PRIMARY KEY,  
    col2 INT,  
    col3 INT,  
    KEY idx_c2 (col2),  
    KEY idx_c1c2 (col1,col2),
```

```
        KEY idx_c3 (col3),
        KEY idx_c2c1 (col2,col1)
    ) ENGINE=INNODB;
```

```
shell> pt-duplicate-key-checker -uroot --ask-pass --databases=test
--tables=ti
```

```
Enter password:
```

```
#
#####
#####
# test.ti
#
#####
#####
```

```
# idx_c2 is a left-prefix of idx_c2c1
```

```
# Key definitions:
```

```
# KEY `idx_c2` (`col2`),
# KEY `idx_c2c1` (`col2`,`col1`)
```

```
# Column types:
```

```
# `col2` int(11) DEFAULT NULL
# `col1` int(11) NOT NULL
```

```
# To remove this duplicate index, execute:
```

```
ALTER TABLE `test`.`ti` DROP INDEX `idx_c2`;
```

```
# Key idx_c2c1 ends with a prefix of the clustered index
```

```
# Key definitions:
```

```
# KEY `idx_c2c1` (`col2`,`col1`)
# PRIMARY KEY (`col1`),
```

```
# Column types:
```

```
# `col2` int(11) DEFAULT NULL
# `col1` int(11) NOT NULL
```

```
# To shorten this duplicate clustered index, execute:
```

```
ALTER TABLE `test`.`ti` DROP INDEX `idx_c2c1`, ADD INDEX
`idx_c2c1` (`col2`);
```

```
#
#####
```



```
#####
# Summary of indexes
#
#####
#####

# Size Duplicate Indexes    14
# Total Duplicate Indexes   2
# Total Indexes             5
```

En primer lugar, `pt-duplicate-key-checker` indica que el índice `idx_c2` puede ser eliminado, quedando un prefijo izquierdo del índice `idx_c2c1`. El motor `InnoDB` añade la clave primaria a todos los índices secundarios, el índice `idx_c2c1` puede reducirse al índice sobre (`col2`), `pt-duplicate-key-checker` propone en segunda instancia el cambio de índice.

- ☐ Tenga en cuenta que la herramienta no comprueba si un índice reducido debido a un índice cluster `InnoDB` es redundante en comparación con otro índice. Para comprobar el resultado, es mejor volver a ejecutar la herramienta después de haber realizado las modificaciones sugeridas.

f. Índice de cobertura

Puede haber observado como un índice hash no almacena el valor indexado, sino solo un hash del valor referenciado, mientras que un índice b-tree registra realmente los valores indexados. Si por casualidad escribe una petición utilizando un índice b-tree para que todas las columnas mencionadas en la petición se encuentren en el índice, ¿es realmente indispensable ir a leer los datos en la tabla? Toda la información para resolver la petición se encuentra, en efecto, en el índice: hablamos entonces de índice de cobertura.

El optimizador de MySQL sabe reconocer este tipo de situación y se contentará con recuperar los datos en el índice, sin tomarse la molestia de leer la tabla.

Tomando el ejemplo de una tabla `t` con un índice en las columnas (`ciudad`, `provincia`, `país`), la siguiente petición utilizará un índice de cobertura:

```
mysql> SELECT ciudad, provincia FROM t WHERE ciudad LIKE 'A%';
```

Podrá saber si su petición está cubierta por un índice al examinar la columna `Extra` del resultado del comando `EXPLAIN`. La mención `Using index` es el signo de un índice de cobertura:

```
mysql> EXPLAIN SELECT CountryCode FROM CountryLanguage WHERE
Language = 'Spanish'\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: CountryLanguage
         type: index
possible_keys: NULL
          key: PRIMARY
       key_len: 33
         ref: NULL
        rows: 984
      Extra: Using where; Using index
```

- ☐ No confunda `Using index` en la columna `Extra`, que señala un índice de cobertura, con `index` en la columna `type`, que señala que la petición se resuelve navegando un índice.

El uso de un índice de cobertura ahorra muchas operaciones de lectura en disco, pues en lugar de leer tanto los datos en el índice como las filas de la tabla (que pueden estar unos u otros en el disco, posiblemente los dos), eliminamos una etapa al solo leer los datos del índice. Como además los índices son normalmente más compactos y fáciles de mantener en memoria que los datos, tenemos más posibilidades de resolver la petición sin necesitar lecturas en el disco. Los índices de cobertura constituyen una solución ideal cuando el disco

es el factor limitante a nivel de rendimiento.

Tenga en cuenta que existen limitaciones:

- Solo los índices b-tree permiten índices de cobertura.
- Un índice de cobertura no podrá utilizarse en el caso de un `SELECT *`, simplemente porque no tendrá, por lo general, un índice que cubra todas las columnas. ¡He aquí otra buena razón para desterrar los `SELECT *` de nuestras aplicaciones!
- Añadir columnas a un índice con el fin de convertirlo en índice de cobertura para una petición no es eficaz si el índice es pesado de mantener.

Por último, debe saber que con las tablas InnoDB es posible aprovechar el index cluster para utilizar índices de cobertura con mayor frecuencia. En efecto, cualquier índice secundario contiene la clave primaria. Si escribimos una petición que llama a las columnas del índice secundario y la clave primaria, el índice será cubierto (lo que no hubiera sido el caso con MyISAM).

El ejemplo siguiente muestra este mecanismo:

```
mysql> SHOW CREATE TABLE customer\G
***** 1. row *****

      Table: customer
Create Table: CREATE TABLE customer (
  customer_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,
  store_id tinyint(3) unsigned NOT NULL,
  ...
  PRIMARY KEY (customer_id),
  KEY idx_fk_store_id (store_id),
  ...
) ENGINE=InnoDB

mysql> EXPLAIN SELECT customer_id FROM customer WHERE store_id =
1\G
***** 1. row *****

      id: 1
select_type: SIMPLE
```

```
table: customer
  type: ref
possible_keys: idx_fk_store_id
  key: idx_fk_store_id
key_len: 1
  ref: const
rows: 270
Extra: Using index
```

EXPLAIN indica un índice de cobertura, mientras que, después de la salida de SHOW CREATE TABLE, la columna `customer_id` no forma parte del índice utilizado.

4. Indexación FULLTEXT

a. Nociones básicas sobre la indexación FULLTEXT

Los diferentes tipos de índices descritos hasta ahora (clave primaria, única, externa, índices no únicos) permitían, según los casos, buscar un valor, una gama de valores o una parte del valor. Pero a veces será necesario hacer peticiones sobre trozos de texto teniendo en cuenta un criterio adicional: la pertinencia. En esta situación, solo un índice FULLTEXT le podrá ayudar.

- ☐ Presentamos aquí el funcionamiento de los índices FULLTEXT para MySQL. A partir de la versión 5.6, también podemos definir índices FULLTEXT para InnoDB. Consulte la documentación en línea para obtener más detalles.

En muchas aplicaciones, el motor de búsqueda interno es una parte crítica y los índices FULLTEXT no cuentan con frecuencia con suficientes funcionalidades. Por ello, estos índices fueron prácticamente abandonados en beneficio de programas informáticos dedicados a la búsqueda, como Lucene, Sphinx o Solr.

Un índice FULLTEXT puede crearse al mismo tiempo que la tabla con la instrucción

CREATE TABLE o más tarde con CREATE INDEX o ALTER TABLE. Solo las columnas de tipo TEXT o derivadas, así como CHAR y VARCHAR, aceptan los índices FULLTEXT:

```
mysql> CREATE TABLE t(  
    col1 varchar(50),  
    col2 varchar(50),  
    col3 varchar(50),  
    FULLTEXT idx_col1_col2 (col1,col2)  
);  
Query OK, 0 rows affected (0,08 sec)  
  
mysql> CREATE FULLTEXT INDEX idx_col2 ON t (col2);  
Query OK, 0 rows affected (0,08 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
mysql> ALTER TABLE t ADD FULLTEXT idx_col3 (col3);  
Query OK, 0 rows affected (0,08 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

Como hemos constatado en este ejemplo, podemos crear un índice FULLTEXT multicolumna. Sin embargo, no podemos efectuar peticiones FULLTEXT sin utilizar un prefijo izquierdo del índice. Así, un índice FULLTEXT en col1 no es redundante con un índice FULLTEXT en (col1,col2).

Los comandos DROP INDEX o ALTER TABLE permiten la eliminación de un índice FULLTEXT:

```
mysql> DROP INDEX idx_col2 ON t;  
Query OK, 0 rows affected (0,07 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
mysql> ALTER TABLE t DROP INDEX idx_col3;  
Query OK, 0 rows affected (0,09 sec)
```

Records: 0 Duplicates: 0 Warnings: 0

Si queremos iniciar una búsqueda FULLTEXT, debemos obligatoriamente utilizar una sintaxis especial. Especifique las columnas sobre las que se refiere la petición en la cláusula MATCH () y los términos buscados en la cláusula AGAINST () .

Obtendrá la pertinencia de los resultados, indicando la cláusula MATCH () . . .
AGAINST () en las columnas que devolverá el SELECT.

Para ilustrar todos estos temas, he aquí un ejemplo de la tabla film_text, que se encuentra en la base de datos test de sakila. Esta tabla contiene un índice FULLTEXT en las columnas (title, description):

```
mysql> SELECT film_id,LEFT(description,50)
        FROM film_text
        WHERE MATCH(title,description)
              AGAINST('Database Administrator')
        LIMIT 5;
```

```
+-----+-----+
| film_id | LEFT(description,50) |
+-----+-----+
|      113 | A Thrilling Yarn of a Database Administrator And a |
|      962 | A Fanciful Story of a Database Administrator And a |
|       27 | A Amazing Reflection of a Database Administrator A |
|      363 | A Fast-Paced Display of a Car And a Database Admin |
|      372 | A Taut Display of a Cat And a Girl who must Overco |
+-----+-----+
5 rows in set (0,00 sec)
```

```
mysql> SELECT film_id,
        LEFT(description,30),
        MATCH(title,description)
              AGAINST('Database Administrator') AS pertinence
        FROM film_text
        WHERE MATCH(title,description)
              AGAINST('Database Administrator')
```

LIMIT 5;

```
+-----+-----+-----+
| film_id | LEFT(description,30) | pertinence |
+-----+-----+-----+
|      113 | A Thrilling Yarn of a Database | 6.66279077529907 |
|      962 | A Fanciful Story of a Database | 6.66279077529907 |
|       27 | A Amazing Reflection of a Data | 6.66279077529907 |
|      363 | A Fast-Paced Display of a Car | 4.52737426757812 |
|      372 | A Taut Display of a Cat And a | 4.52737426757812 |
+-----+-----+-----+
5 rows in set (0,00 sec)
```

- ☐ La cláusula MATCH () . . . AGAINST () está presente dos veces en la última petición, pero MySQL hace el cálculo una sola vez: por lo tanto, no se penaliza en términos de rendimiento consultar al servidor la pertinencia de cada resultado.

Aquí los resultados se ordenan por relevancia, pero este no es el caso de todos los modos de búsqueda FULLTEXT.

b. Búsqueda en lenguaje natural

Por defecto, las peticiones FULLTEXT son búsquedas en lenguaje natural. Esto significa que cada palabra de la cláusula AGAINST se busca en el índice y la pertinencia se calculará en función de la frecuencia de aparición de cada palabra. Si una palabra es poco frecuente en el conjunto de filas indexadas, pero forma parte de los términos buscados, la pertinencia de la línea será elevada. En cambio, las palabras frecuentes darán una pertinencia baja. Por último, si una o más palabras están presentes varias veces en la misma fila, la pertinencia de esta línea aumenta.

En este modo de búsqueda, los resultados se ordenan de forma automática por importancia decreciente.

- ☐ Las palabras que están presentes en al menos el 50 % de los documentos son de

forma automática ignoradas en una búsqueda en lenguaje natural.

He aquí otro ejemplo tomado de la tabla `film_text`:

```
mysql> SELECT film_id,
              title,
              description,
              MATCH(title,description)
                AGAINST('Database Administrator Anaconda')
              AS pertinence FROM film_text
              WHERE MATCH(title,description)
                AGAINST('Database Administrator Anaconda')
              LIMIT 6\G
***** 1. row *****
    film_id: 315
      title: FINDING ANACONDA
description: A Fateful Tale of a Database Administrator And a Girl
who must Battle a Squirrel in New Orleans
  pertinence: 10.0525226593018
***** 2. row *****
    film_id: 113
      title: CALIFORNIA BIRDS
description: A Thrilling Yarn of a Database Administrator And a
Robot who must Battle a Database Administrator in Ancient India
  pertinence: 6.66279077529907
***** 3. row *****
    film_id: 27
      title: ANONYMOUS HUMAN
description: A Amazing Reflection of a Database Administrator And
a Astronaut who must Outrace a Database Administrator in A Shark
Tank
  pertinence: 6.66279077529907
***** 4. row *****
    film_id: 962
      title: WASTELAND DIVINE
description: A Fanciful Story of a Database Administrator And a
Womanizer who must Fight a Database Administrator in Ancient China
```



```

pertinence: 6.66279077529907
***** 5. row *****
    film_id: 23
    title: ANACONDA CONFESSIONS
description: A Lacklusture Display of a Dentist And a Dentist who
must Fight a Girl in Australia
    pertinence: 5.23557186126709
***** 6. row *****
    film_id: 363
    title: GO PURPLE
description: A Fast-Paced Display of a Car And a Database
Administrator who must Battle a Woman in A Balloon
    pertinence: 4.52737426757812
6 rows in set (0,00 sec)

```

Tenga en cuenta que solo el primer resultado contiene las tres palabras. A continuación encontrará tres líneas donde los términos Database y Administrator son cada uno repetido dos veces. Luego encontrará una línea donde no aparece más que la palabra Anaconda y finalmente todas las líneas que contienen una vez las palabras Database y Administrator. En esta tabla, la palabra Anaconda es menos frecuente que las palabras Database y Administrator (que aparecen solo en dos filas), lo que explica por qué la fila que contiene la palabra Anaconda se devuelve con una relevancia tan alta.

- ☐ Si desea hacer explícito el modo de búsqueda por defecto, puede modificar la cláusula AGAINST de la siguiente manera:

```

mysql> SELECT ... FROM ... WHERE MATCH(...) AGAINST('Mi texto' IN
NATURAL LANGUAGE MODE);

```

c. Búsqueda booleana

-

-

```
film_id: 23
title: ANACONDA CONFESSIONS
description: A Lacklusture Display of a Dentist And a Dentist who
must Fight a Girl in Australia
***** 2. row *****
film_id: 315
title: FINDING ANACONDA
description: A Fateful Tale of a Database Administrator And a Girl
who must Battle a Squirrel in New Orleans
```

Esta petición tiene solo dos resultados, pues solo las filas que contienen la palabra Anaconda se tomarán en cuenta (solo dos líneas contienen la palabra Anaconda).

- ☐ Los resultados de una búsqueda en modo booleano no se ordenan por relevancia.

d. Búsqueda con expansión de la petición

La última posibilidad es iniciar una búsqueda con expansión de la petición. En este caso, la petición se ejecuta dos veces: una primera vez con los términos de la búsqueda, y luego una segunda vez con los términos de la búsqueda con los resultados más pertinentes.

Este tipo de búsqueda se ejecuta, una vez más, modificando la cláusula AGAINST:

```
mysql> SELECT ... FROM ... WHERE MATCH(...) AGAINST('Mi texto'
WITH QUERY EXPANSION);
```

Las búsquedas con expansión de la petición son útiles cuando conocemos los términos relacionados con lo que estamos buscando. Por ejemplo, si deseamos encontrar documentos relativos a los políticos influyentes en el tiempo de Luis XIV sin conocer su nombre, una búsqueda booleana o en lenguaje natural no devolverá nada, pero una búsqueda con expansión de petición nos permitirá encontrar documentos con los nombres de Colbert o de

Fouquet.

- ☐ La ampliación de la petición puede fácilmente dar resultados sin ninguna relevancia. Es mejor limitarse a palabras clave cortas para evitar este problema.

e. Configuración de la búsqueda

Solo algunos parámetros sirven para modificar el comportamiento de una búsqueda FULLTEXT. Estos son las principales:

- `ft_min_word_len`: define la longitud mínima de las palabras que se han de indexar.
- `ft_max_word_len`: define la longitud máxima de las palabras que se han de indexar.
- `ft_stopword_list`: define la lista de palabras que no serán indexadas.

Si cambia cualquiera de estas variables, es necesario reconstruir todos los índices FULLTEXT, ya sea suprimiendo y regenerando los índices, ya sea con el comando `REPAIR TABLE`:

```
mysql> REPAIR TABLE t QUICK;
```

f. Rendimiento

Para comprender las implicaciones de los índices FULLTEXT sobre los resultados, es necesario pasar revista al funcionamiento de estos índices. MySQL almacena los índices FULLTEXT con la forma de un doble árbol B. El primer nivel del árbol almacena todas las palabras clave indexadas, mientras que el segundo nivel ofrece, para cada palabra clave, una lista de punteros a las filas de la tabla que contienen la palabra clave. Por cada modificación de una fila, el índice se modificará para cada uno de los términos indexados, lo que significa que, para una fila compuesta de 20 palabras, 20 operaciones de actualización se efectuarán en el índice (para un índice clásico, una sola operación tiene lugar de forma independiente del tamaño del campo).

Este funcionamiento explica por qué es pesado de mantener un índice FULLTEXT en una tabla que se modifica con frecuencia. En particular, si necesitamos recargar una tabla desde una copia de seguridad, es muy interesante no crear el índice FULLTEXT hasta después de la carga de todos los datos, para evitar que cada fila insertada provoque múltiples modificaciones del índice.

La complejidad de la estructura de doble árbol B suele ser el origen de una fragmentación considerable, comparada con la fragmentación de los índices convencionales. Con frecuencia tendrá la necesidad de defragmentar los índices FULLTEXT empleando el comando `OPTIMIZE TABLE` para mantener un buen rendimiento (ver la sección Mantenimiento de las tablas para más detalles sobre este comando).

Los parámetros `ft_min_word_len`, `ft_max_word_len` y `ft_stopword_list` permiten reducir el índice filtrando los términos menos relevantes. No hay, por lo tanto, que vacilar en configurar estas variables para obtener un índice más compacto y lo más exacto posible. En particular, la lista de palabras excluidas debe estar en relación con el tema de los documentos indexados: si indexamos documentos relacionados con la arquitectura LAMP, palabras como Linux, Apache, MySQL o PHP no deberían incluirse en el índice, ya que es casi seguro que estos términos se encuentren en todas las filas.

Recuerde mantener los índices FULLTEXT en memoria siempre que sea posible. A menudo, las peticiones FULLTEXT pasan a ser lentas a partir del momento en que el servidor se ve obligado a leer el índice del disco. A veces será necesario crear una caché específica para los índices FULLTEXT.

g. Limitaciones y puntos que se deben conocer

La indexación FULLTEXT con MySQL sufre una serie de limitaciones. He aquí una lista; algunos puntos ya han sido abordados en los párrafos anteriores:

- Es obligatorio escribir una cláusula `MATCH AGAINST` para que el servidor utilice un índice FULLTEXT.
- Tan pronto como indique una cláusula `MATCH AGAINST` para la que pueda utilizarse un índice FULLTEXT, el optimizador optará por usar este índice aunque otro índice sea mejor (caso de una cláusula `WHERE` con condiciones más restrictivas

que la condición MATCH AGAINST).

- Un índice FULLTEXT solo puede servir para una búsqueda FULLTEXT. Si tenemos en la petición otras condiciones WHERE, el filtrado tendrá lugar después de que el índice FULLTEXT haya remitido todas las filas correspondientes a la cláusula MATCH AGAINST.
- Los resultados de búsquedas en lenguaje natural se ordenan por relevancia, pero este no es el caso de los resultados de búsqueda en modo booleano.
- El umbral del 50 % para las búsquedas en lenguaje natural causa a menudo problemas durante las pruebas: para comprobar que un índice FULLTEXT funciona bien, llenaremos una fila o dos con unas palabras y efectuaremos una búsqueda para cualquiera de estas palabras. Sin embargo, la palabra tiene todas las posibilidades de superar el umbral del 50 % de frecuencia de aparición y, por tanto, de ser excluida de la petición, lo que induce a pensar que el índice no funciona.
- No es posible cambiar por simple configuración el método de cálculo de la pertinencia, basado en la frecuencia de aparición. Por ejemplo, no podremos garantizar de manera sencilla que un término encontrado en el título tenga un peso dos veces más elevado que el término encontrado en el cuerpo del texto, ni tampoco dar una importancia mayor a una fila donde varios términos buscados estén cerca unos de otros.
- Un índice FULLTEXT no almacena el texto indexado, no puede utilizarse como índice de cobertura.
- Un índice FULLTEXT no contiene la columna en la que aparece una palabra clave. Por ello, las columnas de una cláusula MATCH deben ser exactamente las del índice y encontrarse en el mismo orden. La búsqueda sobre un prefijo izquierdo del índice no funciona por la misma razón.

El comando EXPLAIN

1. Rol

Cuando una petición se ejecuta con lentitud, necesitará información sobre lo que hace MySQL de forma interna a fin de poder influir en su comportamiento, por ejemplo, añadiendo un índice, modificando la petición o modificando uno o varios parámetros a nivel del servidor.

El comando EXPLAIN desempeña un papel especialmente importante, ya que es gracias a este que MySQL comunica el plan de ejecución de la petición. ¿Qué es un plan de ejecución? Se trata simplemente de la estrategia adoptada por el servidor para determinar el resultado de una petición. Recuerde que SQL es un lenguaje en el que indicamos los criterios de búsqueda o la acción (por ejemplo: añadir 1 a la edad de todas las personas nacidas el 1 de marzo), pero en el que no especificamos cómo encontrar los registros correspondientes a la búsqueda.

2. Acceso a los datos

Como veremos a continuación en este capítulo, EXPLAIN nos informará más o menos directamente sobre la forma en que el servidor va a acceder a los datos para encontrar los resultados de una petición. Por consiguiente, es interesante conocer los métodos de acceso más eficaces y los que hay que intentar evitar.

a. Acceso secuencial o aleatorio

Sea cual sea el medio de almacenamiento (memoria RAM, disco duro tradicional o SSD), los datos siguen organizándose en una estructura como de árbol o lista encadenada. Cuando

desea acceder a varios datos en una estructura, hay dos posibilidades: o bien los datos están situados unos al lado de otros o están diseminados a través de la estructura.

En el primer caso, tan pronto como se determina la posición del primer dato, basta con leer los siguientes datos para recuperar toda la información necesaria. Hablamos entonces de acceso secuencial.

En el segundo caso, la posición de cada dato debe ser calculada. Hablamos entonces de acceso aleatorio. Como las operaciones que se han de realizar son más complejas, un acceso aleatorio es más lento que un acceso secuencial.

- ☐ Esta visión es simplista, ya que la fragmentación de una estructura puede provocar accesos aleatorios donde solo deberíamos tener accesos secuenciales. Pero en el caso de una estructura de acceso secuencial, los accesos aleatorios son por lo general poco frecuentes.

b. Acceso en memoria o en disco

No se trata de dar cifras sobre los tiempos de acceso a un disco tradicional o en memoria, ya que estas cifras dependen mucho del hardware elegido, sino de dar tendencias.

De manera general, acceder a los datos en memoria resulta mucho más rápido que en un disco duro, lo que no es sorprendente. Pero también es interesante tener en cuenta los siguientes puntos:

- Un acceso secuencial es solo en torno a 10 veces más rápido en memoria que en disco.
- Un acceso aleatorio es más de 1 000 veces más rápido en memoria que en disco.
- Un acceso aleatorio en disco es muy lento.
- El acceso a memoria sigue siendo rápido, sea secuencial o aleatorio.
- Los accesos con SSD tienden a acercarse a los accesos a memoria, pero las diferencias entre los modelos son considerables: no ceda a la tentación de considerar que cualquier SSD dará resultados similares a los obtenidos en memoria.

c. En resumen

El cuadro siguiente resume los diferentes medios de acceso a los datos, con su coste

asociado. La columna SSD nos da una idea de lo que podemos esperar, en comparación con los discos duros tradicionales o la memoria RAM:

	Disco	Memoria	SSD
Acceso aleatorio	Muy lento	Rápido	Rápido
Acceso secuencial	Rápido	Muy rápido	Rápido
Coste	Barato	Muy caro	Barato(pero capacidad limitada)

Cuando trate de optimizar sus peticiones, tenga en cuenta algunos de estos principios:

- Prefiera los accesos a memoria sobre los accesos a disco.
- Para los accesos a disco, prefiera accesos secuenciales a los accesos aleatorios.
- Para el acceso a la memoria, el carácter aleatorio o secuencial pierde su importancia.
- La sustitución de discos duros tradicionales por SSD puede ser una buena solución cuando no se puede eliminar los accesos aleatorios.

3. Leer el plan de ejecución

a. Ejemplo simple

Podemos obtener el plan de ejecución que el servidor prevé emplear para resolver una petición de tipo `SELECT` añadiendo simplemente la palabra clave `EXPLAIN` antes de la petición:

```
mysql> EXPLAIN SELECT COUNT(*) FROM City\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: NULL
         type: NULL
```

```
possible_keys: NULL
              key: NULL
              key_len: NULL
              ref: NULL
              rows: NULL
              Extra: Select tables optimized away
1 row in set (0,02 sec)
```

- Cuando utilice EXPLAIN con el cliente en línea de comandos `mysql`, es preferible terminar la petición con `\G` en lugar de `;` a fin de mejorar la legibilidad de la pantalla producida. Observe: ¡en este caso, las líneas aparecen de forma vertical!

Sin entrar todavía en el detalle del plan de ejecución, EXPLAIN produce un cierto número de líneas (aquí, una sola), cada línea cuenta con 10 campos (de `id` a `Extra`). Una línea en la pantalla de EXPLAIN describe el trabajo realizado por el servidor sobre una tabla real, una tabla temporal, una vista, una tabla creada por una petición o una tabla llamada derivada (el concepto de tabla derivada aparece más adelante).

Leyendo la información contenida en cada línea, podemos reconstruir paso a paso la ruta empleada por el servidor para ejecutar la petición.

b. Peticiones diferentes de SELECT

Antes de MySQL 5.6, EXPLAIN solo podía utilizarse en las peticiones de tipo SELECT. Por lo tanto, a veces era necesario comprender cómo el servidor trataba un UPDATE o un DELETE. En este caso, la solución consistía en tratar de construir un SELECT equivalente a su UPDATE o DELETE para poder utilizar EXPLAIN. En la mayoría de los casos, esta técnica es fiable y proporciona información significativa.

He aquí un ejemplo con la base `sakila`:

```
UPDATE rental SET return_date = NULL WHERE YEAR(rental_date) <
```

Para construir un `SELECT` equivalente, debe mantener intacta la cláusula `WHERE` y pasar a la cláusula `SELECT` todos los campos afectados por la cláusula `SET`. Obtendrá la siguiente petición:

```
SELECT return_date FROM rental WHERE YEAR(rental_date) < 2008
```

A partir de MySQL 5.6, puede llamar directamente a `EXPLAIN` para una petición que no es un `SELECT`:

```
mysql> EXPLAIN UPDATE rental SET customer_id = 1 WHERE rental_id =
500\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: rental
         type: range
possible_keys: PRIMARY,rental_date,idx_fk_inventory_id,
idx_fk_customer_id,idx_fk_staff_id
          key: PRIMARY
      key_len: 4
         ref: NULL
        rows: 1
     Extra: Using where
```

c. Joins (uniones)

`EXPLAIN` utilizado con un join entre N tablas producirá N líneas, una para cada tabla. El orden de las líneas procedentes de `EXPLAIN` corresponde al orden de join elegido por MySQL y no al orden de escritura del join. El optimizador de peticiones analizará las

diferentes posibilidades de ordenar las tablas para efectuar los joins y elegirá el orden más eficaz.

El siguiente ejemplo muestra que el orden de escritura de las tablas no es siempre el orden de join elegido por el servidor:

```
mysql> EXPLAIN SELECT City.Name FROM City INNER JOIN Country ON ID
= Capital WHERE Country.Name = 'Francia'\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: Country
...
***** 2. row *****
      id: 1
  select_type: SIMPLE
        table: City
...
```

d. Uniones

MySQL ejecuta las uniones (UNION o UNION ALL), creando una tabla temporal y reemplazándola a lo largo del tiempo con los resultados de las peticiones que componen la unión. El comando EXPLAIN muestra una línea adicional correspondiente a esta tabla temporal e indica de qué tablas se obtienen los resultados.

El siguiente ejemplo extraído de la base `sakila` servirá para ilustrar este principio:

```
mysql> EXPLAIN SELECT first_name, last_name FROM customer
UNION ALL
SELECT first_name, last_name FROM staff\G
***** 1. row *****
      id: 1
  select_type: PRIMARY
        table: customer
```

```

...
***** 2. row *****
      id: 2
  select_type: UNION
      table: staff
  ...
***** 3. row *****
      id: NULL
  select_type: UNION RESULT
      table: <union1,2>
...

```

La primera tabla de una unión tiene como valor PRIMARY en la columna `select_type`, luego UNION para todas las tablas siguientes. En este ejemplo de una unión de dos tablas, EXPLAIN devuelve tres líneas porque la tercera línea corresponde a la tabla temporal utilizada para construir el resultado. El valor de la columna `select_type` es entonces UNION RESULT, y en la columna `table` encontraremos los identificadores (columna `id`) de las tablas sobre las cuales la unión se realiza (aquí 1 y 2).

- ☐ Cuando sea posible, es aconsejable usar UNION ALL, que regresa todas las líneas, mientras que UNION filtra las repeticiones. El filtrado se realiza comparando todas las columnas de cada fila, lo que puede volverse con rapidez muy pesado.

e. Subconsultas

Las subconsultas añaden un nivel de complejidad no desdeñable en el análisis de la salida del comando EXPLAIN, en especial para las subconsultas en la cláusula FROM. Las subconsultas introducen nuevos valores para la columna `select_type`, así como notaciones específicas para la columna `table`.

La columna `select_type` tomará el valor DERIVED en el caso de una petición en la cláusula FROM y el valor SUBQUERY en otros casos. El valor SUBQUERY puede darse

con dos calificativos: `UNCACHEABLE`, que indica que la petición no puede ponerse en caché, y `DEPENDENT`, que indica que la petición depende de datos de una petición externa.

Para las subconsultas en la cláusula `FROM`, la columna `table` indica un valor del tipo `<derivedX>`, donde `X` representa el valor de la columna `id` de la subconsulta. Observe que, al contrario de la noción `<unionM, N>`, donde `M` y `N` son referencias hacia atrás (es decir, referencias a las filas situadas antes en la salida de `EXPLAIN`), el `X` de `<derivedX>` es siempre una referencia hacia adelante. La lectura del plan de ejecución se efectúa entonces saltándose las líneas y luego volviendo hacia atrás, ¡lo que explica por qué es difícil de entender los resultados del comando `EXPLAIN` en las peticiones complejas que mezclan joins, subconsultas e incluso uniones!

- Tenga en cuenta que, en todos los casos, la primera tabla listada en el resultado del comando `EXPLAIN` tiene siempre por valor `PRIMARY` en la columna `select_type`.

Los dos ejemplos siguientes, procedentes de la base `world`, ilustrarán estos principios:

```
mysql> EXPLAIN SELECT DISTINCT Language
        FROM CountryLanguage
        WHERE CountryCode IN (
        SELECT Code
        FROM Country
        WHERE GovernmentForm = 'Monarchy')
        ORDER BY Language\G
***** 1. row *****
      id: 1
  select_type: PRIMARY
        table: CountryLanguage
...
***** 2. row *****
      id: 2
  select_type: DEPENDENT SUBQUERY
```

table: Country

...

Este ejemplo muestra un caso donde el optimizador gestiona muy mal la subconsulta. El objetivo de esta petición es encontrar las lenguas de los países que tienen un régimen monárquico. La idea lógica es buscar en primer lugar la lista de países con un régimen monárquico, lo que puede hacerse en la subconsulta; luego, inyectar esta lista de constantes en la cláusula `IN` de la petición externa. Se llegaría entonces a realizar dos peticiones independientes.

```
mysql> SELECT Code FROM Country WHERE GovernmentForm = 'Monarchy';
```

Si el resultado de esta petición es 10, 35, 72, la segunda petición que debería llevarse a cabo sería:

```
mysql> SELECT DISTINCT Language FROM CountryLanguage WHERE  
ContryCode IN (10, 35, 72);
```

Ahora bien, este no es del todo el plan de ejecución presentado por el comando `EXPLAIN` (examen de la tabla `CountryLanguage` primero y luego subconsulta con dependencia externa para la tabla `Country`). El optimizador considera que es más pertinente añadir una condición adicional en la subconsulta correlacionando la subconsulta con la petición externa, lo que equivale a ejecutar la siguiente petición:

```
mysql> SELECT DISTINCT Language  
      FROM CountryLanguage  
      WHERE EXISTS (  
          SELECT *  
          FROM Country  
          WHERE GovernmentForm = 'Monarchy'
```

```
AND CountryLanguage.CountryCode = Country.Code)
ORDER BY Language;
```

¿En qué medida es mala la solución elegida por el optimizador? Entre las líneas del resultado de EXPLAIN que no han sido reproducidas, figura la forma en que las tablas serán examinadas. Sin embargo, con el plan elegido por el optimizador, la tabla CountryLanguage será recorrida por completo, lo que penalizará en gran medida si esta tabla es de gran tamaño.

El segundo ejemplo muestra una subconsulta en la cláusula FROM:

```
mysql> EXPLAIN SELECT Name
      FROM (
        SELECT Name, COUNT(*) AS nlanguages,
        GROUP_CONCAT(Language) as languages
        FROM Country c, CountryLanguage cl
        WHERE c.Code = cl.CountryCode
        AND cl.IsOfficial = 'T'
        GROUP BY Name
        ORDER BY nlanguages DESC
        LIMIT 1
      ) AS tmp\G
***** 1. row *****
      id: 1
  select_type: PRIMARY
      table: <derived2>
...
***** 2. row *****
      id: 2
  select_type: DERIVED
      table: cl
...
***** 3. row *****
      id: 2
  select_type: DERIVED
      table: c
```


...

Aquí podemos ver el mecanismo de las tablas DERIVED. La primera línea indica que la petición se hace sobre la tabla <derived2>, lo que corresponde al join entre las tablas cl (CountryLanguage) y c (Country), cuyas filas tienen las dos el valor 2 en la columna id. Observe la lógica del plan de ejecución tal como se presenta: la petición se realiza sobre una tabla (línea 1), que en realidad es una tabla derivada del join de otras dos tablas (líneas 2 y 3).

- ☐ ¡Tome el tiempo de entender estos ejemplos, ya que, a pesar de su aparente complejidad, estas peticiones son relativamente simples!

4. Columnas principales

a. Tipos de acceso a los datos

La columna type indica cómo el optimizador prevé el acceso a los datos.

Los métodos siguientes son los que encontraremos más a menudo, ordenados de peor a mejor:

- ALL: se requiere un recorrido completo de la tabla para encontrar las filas, ya que no hay ningún índice disponible o satisfactorio.

Ejemplo:

```
mysql> EXPLAIN SELECT * FROM film_text WHERE title LIKE 'The%'\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: film_text
         type: ALL
...

```

- `index`: se requiere un recorrido completo de un índice.

Ejemplo:

```
mysql> EXPLAIN SELECT category_id FROM category\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: category
        type: index
...

```

- `range`: basta solo con examinar una parte de un índice.

Ejemplo:

```
mysql> EXPLAIN SELECT * FROM category WHERE category_id > 50\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: category
        type: range
...

```

- `index_merge`: varios índices se emplean simultáneamente. Se trata de una excepción al principio de que MySQL solo utiliza un índice por tabla y por petición. Este caso se describe de forma detallada en la siguiente sección sobre la optimización de las peticiones.

Ejemplo:

```
mysql> EXPLAIN SELECT * FROM rental WHERE rental_id > 10 OR
inventory_id < 100\G

```

```
***** 1. row *****
      id: 1
select_type: SIMPLE
      table: rental
      type: index_merge
...
```

- `ref`: se examina un índice no único para recuperar las filas correspondientes a un valor. Varias filas pueden devolverse para el mismo valor. Existe una variante, `ref_or_null`, que indica el mismo tipo de acceso con una contraseña adicional para recuperar los valores NULL.

Ejemplo:

```
mysql> EXPLAIN SELECT rental_date
      FROM rental INNER JOIN customer
      USING (customer_id)
      WHERE customer_id=1\G
***** 1. row *****
      id: 1
select_type: SIMPLE
      table: customer
...
***** 2. row *****
      id: 1
select_type: SIMPLE
      table: rental
      type: ref
...
```

- `eq_ref`: se examina un índice único o una clave primaria para recuperar la fila correspondiente a un valor. En general, es el mejor tipo de acceso posible; los tres tipos siguientes son casos particulares más raros.

Ejemplo:

```
mysql> EXPLAIN SELECT email
      FROM customer INNER JOIN rental
      USING (customer_id)
      WHERE rental_date > '2006-04-10 00:00:00'
      AND rental_date < '2006-05-24 00:00:00'\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
      table: rental
...
***** 2. row *****
      id: 1
  select_type: SIMPLE
      table: customer
      type: eq_ref
...
```

- `const`: solo se requiere la lectura de una única fila de la tabla.

Ejemplo:

```
mysql> EXPLAIN SELECT * FROM category WHERE category_id=1\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
      table: category
      type: const ...
```

- `System`: idéntico al caso anterior, salvo que la tabla es una tabla con una única fila.
- `NULL`: no se requiere ningún recorrido de la tabla o índice.

Ejemplo:

```
mysql> EXPLAIN SELECT COUNT(*) FROM film_text\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: NULL
         type: NULL
...

```

b. Índices examinados

Es muy importante examinar las dos columnas `possible_keys` y `key`, ya que se refieren a los índices utilizados por la petición. En concreto, `possible_keys` indica el conjunto de índices que el optimizador va a considerar como candidatos durante la elaboración del plan de ejecución, y `key` indica el índice usado finalmente.

- ☐ Recordemos que MySQL solo puede utilizar un índice por tabla, salvo en el caso de que la columna `type` indique `index_merge`. Este caso particular se detalla más adelante en este capítulo.

El valor `NULL` para `possible_keys` es señal de que no existe ningún índice utilizable, y el valor `NULL` para `key` se encuentra cuando ningún índice candidato se considera pertinente por el optimizador.

`key_len` devuelve el tamaño en bytes del índice seleccionado en la columna `key`. Esta información permite en primer lugar saber cuántas de las columnas de un índice compuesto se utilizan.

Tomemos un ejemplo con la base `world`:

```
mysql> ALTER TABLE City ADD INDEX
      idx_name_district(Name,District);

mysql> EXPLAIN SELECT * FROM City WHERE Name LIKE 'San%\G
***** 1. row *****

```

```
...
    key: idx_name_district
key_len: 35
...
```

El índice `idx_name_district` se utiliza (columna `key`) y, ya que `key_len` vale 35, o sea el tamaño del campo `Name`, podemos deducir que solo la columna `Name` se utiliza en el índice, lo que es lógico, ya que la cláusula `WHERE` solo hace aparecer esta columna.

- ☐ Para un campo de tipo `VARCHAR` o asimilado, según el juego de caracteres elegido, el tamaño en bytes podrá ser diferente del tamaño en número de caracteres.

Última columna en relación con los índices, `ref` muestra con qué se compara el índice seleccionado. En el caso de un recorrido de tabla o índice, es decir, cuando la columna `type` del resultado de `EXPLAIN` vale `ALL`, `INDEX` o `range`, esta columna valdrá siempre `NULL`. En el caso de un `join`, esta columna toma el valor de los campos de `join` de la tabla adjunta.

Siempre con la base `world`:

```
mysql> EXPLAIN SELECT City.Name FROM City INNER JOIN Country ON ID
= Capital WHERE Country.Name = 'Francia'\G
***** 1. row *****
    table: Country
    type: ALL
    ref: NULL
...
***** 2. row *****
    table: City
    type: eq_ref
    key: PRIMARY
    ref: world.Country.Capital
```

...

EXPLAIN nos muestra que el optimizador examinará en primer lugar la tabla `Country`, para la que se llevará a cabo un recorrido completo. Posteriormente, cada valor del campo `Capital` será comparado con la clave primaria de la tabla `City`.

c. Número de filas recorridas

La columna `rows` proporciona una estimación del número de filas que deberán ser recorridas a fin de llegar al resultado. Se trata de una métrica aproximada para determinar la eficacia de una petición: cuanto mayor es el número en esta columna, mayor es el número de filas que hay que leer y mayor el riesgo de que la petición sea lenta al ejecutar.

En el caso de una petición que haga aparecer varias filas en el resultado de EXPLAIN (por ejemplo, para un join), tendremos una estimación del número total de filas leídas por el servidor multiplicando el valor de esta columna para cada fila.

He aquí un ejemplo con un producto cartesiano:

```
mysql> EXPLAIN SELECT * FROM City CROSS JOIN Country\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: Country
...
      rows: 238
    Extra:
***** 2. row *****
      id: 1
  select_type: SIMPLE
        table: City
...
      rows: 4079
    Extra: Using join buffer
2 rows in set (0,00 sec)
```

Esta petición recorre toda la tabla `Country` (238 registros) y, para cada registro, se recorre el conjunto de la tabla `City` (4079 registros). En total, se leen más de 800 000 filas. ¡Sin duda, la petición será lenta!

Entienda bien que el número de filas indicadas por la columna `rows` no indica ni el número de filas del resultado, ni el número de filas que serán efectivamente leídas por el servidor para resolver la petición, solo el número de filas que el servidor considera que debe leer.

El siguiente ejemplo demuestra que la columna `rows` es simplemente una estimación:

```
mysql> EXPLAIN SELECT * FROM City WHERE Name like 'San%'\G
***** 1. row *****
...
      rows: 141
      Extra: Using where
1 row in set (0,00 sec)

mysql> SELECT COUNT(*) FROM City WHERE Name like 'San%';
+-----+
| COUNT(*) |
+-----+
|      110 |
+-----+
```

La precisión del valor indicado por esta columna depende de la calidad de las estadísticas mantenidas por el motor de almacenamiento sobre la tabla. Si encontramos que tienen lugar errores graves, puede resultar beneficioso actualizar estas estadísticas con el comando `ANALYZE TABLE` (ver la sección Mantenimiento de las tablas para más información sobre el tema). Con `MyISAM`, es posible tener estadísticas actualizadas y bastante precisas, lo que no suele ser el caso con `InnoDB`.

☐ Una cláusula `LIMIT` nunca se toma en cuenta en la columna `rows`:


```
mysql> EXPLAIN SELECT * FROM City WHERE Name like 'San%' LIMIT
10\G
***** 1. row *****
...
      rows: 141
Extra: Using where
```

d. La columna Extra

La columna `Extra` proporciona información adicional sobre la ejecución de la petición. Siempre es interesante tener en cuenta esta columna, ya que podemos considerar que todas las indicaciones que sean dignas de interés sobre la ejecución de una petición y que no tienen cabida en las columnas restantes se encuentran en la columna `Extra`.

A continuación describimos solo los valores que veremos aparecer de forma regular:

- `Using where`: encontraremos esta indicación cuando algunas de las condiciones de la cláusula `WHERE` no puedan ser resueltas por un índice. El motor de almacenamiento devuelve entonces todas las filas correspondientes a las condiciones estudiadas, y el servidor debe hacer por sí mismo un nuevo paso de filtrado para eliminar las filas que no cumplen las condiciones restantes. Atención, hay bugs que hacen que esta información aparezca por error en el resultado del comando `EXPLAIN`.
- `Using index`: es la señal de que MySQL va a utilizar un índice de cobertura (ver la sección de indexación de este capítulo). Salvo excepciones, un índice de cobertura mejora el rendimiento de una petición.
- `Using temporary`: MySQL va a utilizar una tabla temporal, por lo general para realizar una ordenación o para almacenar los resultados de una petición `UNION`. No podremos saber si esta tabla temporal se crea en memoria o en disco. En general, intentaremos construir los índices para evitar las tablas temporales y, si no es posible evitar una tabla temporal, habrá que tratar de asegurarse de que se mantiene en memoria, so pena de obtener muy malos resultados.

- Using filesort: MySQL indica que la clasificación no ha podido realizarse utilizando un índice. Las filas son devueltas sin ordenar por el motor de almacenamiento y el servidor debe recurrir a un mecanismo de ordenación para devolver las filas en el orden correcto al cliente. De forma regular, esta ordenación hecha por el servidor requerirá de una tabla temporal y veremos en la columna Extra las menciones: Using temporary; Using filesort.

5. EXPLAIN EXTENDED

Este nuevo comando se utiliza exactamente de la misma manera que el EXPLAIN clásico, pero añadiendo dos funcionalidades adicionales.

La primera diferencia con un EXPLAIN clásico es una columna adicional llamada `filtered`, disponible a partir de la versión 5.1. Esta columna da, con relación al número de filas leídas (columna `rows`), una aproximación del porcentaje de filas que serán devueltas por el servidor una vez que se hayan aplicado todas las condiciones a la tabla. Multiplicando los valores de las columnas `rows` y `filtered`, obtenemos una estimación del número de filas del resultado de la petición. La calidad de esta estimación varía mucho según las peticiones.

He aquí un ejemplo donde la estimación es buena (50 filas estimadas para 49 filas reales):

```
mysql> EXPLAIN EXTENDED SELECT ID FROM City WHERE ID < 50\G
***** 1. row *****

...

      rows: 50
  filtered: 100.00

mysql> SELECT COUNT(ID) FROM City WHERE ID < 50;
+-----+
| COUNT(ID) |
+-----+
|         49 |
+-----+
```

He aquí un ejemplo donde la estimación es mala (4079 filas estimadas para 1 fila real):

```
mysql> EXPLAIN EXTENDED SELECT Name FROM City WHERE CountryCode
LIKE 'F%' AND district LIKE 'F%' AND population > 100000\G
***** 1. row *****
...
      rows: 4079
    filtered: 100.00

mysql> SELECT Name FROM City WHERE CountryCode LIKE 'F%' AND
District LIKE 'F%' AND population > 100000;
+-----+
| Name      |
+-----+
| Besançon  |
+-----+
1 row in set (0,00 sec)
```

Otra posibilidad de EXPLAIN EXTENDED: si está ejecutando el comando SHOW WARNINGS inmediatamente después de EXPLAIN EXTENDED, verá la petición reescrita a partir de la información del plan de ejecución, lo que a veces puede ser útil cuando se está buscando optimizar la ejecución.

He aquí un ejemplo:

```
mysql> EXPLAIN EXTENDED SELECT COUNT(*) FROM Country\G
***** 1. row *****
      id: 1
...
1 row in set, 1 warning (0,00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
      Level: Note
      Code: 1003
```

```
Message: select count(0) AS `COUNT(*)` from `world`.`Country`  
1 row in set (0,00 sec)
```

Vemos aquí como, desde el punto de vista del optimizador, COUNT (*) y COUNT (0) son idénticos.

Optimización de las peticiones

En esta sección, veremos una serie de técnicas para mejorar el tiempo de ejecución de sus peticiones. Lea estas propuestas como consejos que pueden ayudarle, no como recetas absolutas que funcionan en todos los casos. La optimización sigue siendo un arte y la experiencia sigue siendo la cualidad esencial para encontrar la mejor manera de acelerar una petición lenta.

1. Aislamiento de las columnas

La encapsulación de una columna en una función prohíbe la utilización de un índice. Si es posible, asegúrese, por tanto, de reescribir su petición de manera que no necesite de una función. Este caso se presenta a menudo con las fechas.

Ejemplo de una petición que no puede usar un índice:

```
mysql> EXPLAIN SELECT * FROM rental WHERE TO_DAYS(CURRENT_DATE())
- TO_DAYS(rental_date) < 10\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: rental
         type: ALL
possible_keys: NULL
          key: NULL
      ...
```

Al tratar de aislar la columna `rental_date` sobre la cual existe un índice, el resultado

es mejor:

```
mysql> EXPLAIN SELECT * FROM rental WHERE rental_date >
CURRENT_DATE() + INTERVAL 10 DAY\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: rental
         type: range
possible_keys: rental_date
          key: rental_date
      ...
```

- Si hemos leído bien la sección sobre la caché de peticiones, sabremos que el uso de `CURRENT_DATE()` impedirá la puesta en caché; por tanto, es preferible calcular la fecha actual en la aplicación e inyectar la constante así obtenida en la petición.

Sería mejor especificar la lista de los campos necesarios antes que pasar por `SELECT *`, que impide, por ejemplo la utilización de un posible índice de cobertura.

2. Joins

MySQL solo conoce un método para realizar joins: las tablas se adjuntan una a una y de forma sucesiva. Esto significa que un join entre cuatro tablas `t1` a `t4` se llevará a cabo, por ejemplo, uniendo `t1` a `t2` (lo que nos da una tabla `t2'`), luego `t2'` a `t3` (dando `t3'`), luego `t3'` a `t4`. El plan de ejecución no será nunca adjuntar `t1` a `t2` (dando `t2'`), luego `t3` a `t4` (dando `t4'`), y por último `t2'` a `t4'`.

En el caso de joins internos (`INNER JOIN`), el orden de join no tiene ninguna importancia y el optimizador podrá decidir no presentar las tablas en el orden de escritura.

He aquí un ejemplo de join donde el optimizador decide no respetar el orden de escritura:

```
mysql> EXPLAIN SELECT email
      FROM customer INNER JOIN rental
      ON customer.customer_id = rental.customer_id
      WHERE rental_date > '2006-04-10 00:00:00'
      AND rental_date < '2006-05-24 00:00:00'\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: rental
        type: range
possible_keys: rental_date,idx_fk_customer_id
          key: rental_date
      key_len: 8
        ref: NULL
       rows: 10
      Extra: Using where; Using index
***** 2. row *****
      id: 1
    select_type: SIMPLE
      table: customer
        type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 2
        ref: sakila.rental.customer_id
       rows: 1
      Extra:
```

¿Por qué eligió el optimizador ese orden? Tenga en cuenta que aquí el optimizador estima que habrá que examinar $10 * 2 = 20$ filas. Para forzar al optimizador a efectuar el join en el orden de escritura y determinar cuántas filas deberán leerse en este caso, podemos usar `STRAIGHT_JOIN`:

```

mysql> EXPLAIN SELECT email
      FROM customer STRAIGHT_JOIN rental
      ON customer.customer_id = rental.customer_id
      WHERE rental_date > '2006-04-10 00:00:00'
      AND rental_date < '2006-05-24 00:00:00'\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: customer
        type: ALL
possible_keys: PRIMARY
      key: NULL
    key_len: NULL
      ref: NULL
     rows: 541
    Extra:
***** 2. row *****
      id: 1
    select_type: SIMPLE
      table: rental
        type: range
possible_keys: rental_date,idx_fk_customer_id
      key: rental_date
    key_len: 8
      ref: NULL
     rows: 10
    Extra: Using where; Using index; Using join buffer

```

Si el join se efectúa en el orden de escritura, el servidor deberá leer $541 * 10 = 5410$ filas. Por eso el optimizador prefirió invertir el orden del join.

Para los joins externos (LEFT JOIN o RIGHT JOIN), el orden de join suele ser importante y no puede modificarse. Esto ayuda al optimizador a encontrar el mejor plan de ejecución eliminando muchas posibilidades.

☐ El algoritmo utilizado por MySQL para unir las tablas se basa simplemente en

bucles anidados. Podrá encontrar quizás el término inglés: nested-loop join. Este algoritmo no permite tratar los joins de tipo FULL OUTER JOIN.

Observe que hay algunos frameworks de desarrollo que generan de forma automática las peticiones SQL: sucede que todos los joins se escriben con un LEFT JOIN en lugar de un INNER JOIN. El optimizador de peticiones se encuentra limitado en los planes de ejecución posibles, lo que causa a veces importantes problemas de rendimiento.

Debe saber también que es posible dar indicaciones al optimizador sobre el uso de un índice con las cláusulas USE INDEX, FORCE INDEX e IGNORE INDEX:

- USE INDEX indica a MySQL que creemos que el índice pasado como parámetro es un buen índice. El optimizador puede, a pesar de todo, optar por no usarlo.

Ejemplo con la petición anterior:

```
mysql> EXPLAIN SELECT email
        FROM customer INNER JOIN rental
        USE INDEX (idx_fk_customer_id)
        ON customer.customer_id = rental.customer_id
        WHERE rental_date > '2006-04-10 00:00:00'
        AND rental_date < '2006-05-24 00:00:00'\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: customer
         type: ALL
possible_keys: PRIMARY
          key: NULL
...
***** 2. row *****
      id: 1
  select_type: SIMPLE
        table: rental
         type: ref
possible_keys: idx_fk_customer_id
```

`key: idx_fk_customer_id`

...

El índice indica que ha sido utilizado y provocó una inversión del orden de join.

- `FORCE INDEX` impone al optimizador el uso del índice pasado como parámetro. Sin embargo, no se tendrá en cuenta si no es del todo utilizable para la petición. El ejemplo anterior funcionará sustituyendo `USE` por `FORCE`.
- `IGNORE INDEX` indica al optimizador no tener en cuenta los índices pasados como parámetro.

Ejemplo:

```
mysql> EXPLAIN SELECT email
        FROM customer INNER JOIN rental
        IGNORE INDEX (idx_fk_customer_id,rental_date)
        ON customer.customer_id = rental.customer_id
        WHERE rental_date > '2006-04-10 00:00:00'
        AND rental_date < '2006-05-24 00:00:00'\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: rental
         type: ALL
possible_keys: NULL
         key: NULL
...
***** 2. row *****
      id: 1
  select_type: SIMPLE
        table: customer
         type: eq_ref
possible_keys: PRIMARY
         key: PRIMARY
...
```

3. Filtrados

Las columnas de las cláusulas `WHERE` figuran entre las primeras que deben considerarse cuando la petición es lenta. En efecto, la colocación de un índice puede disminuir significativamente el número de filas para leer, acelerando así la ejecución. Recuerde todo lo que se ha dicho sobre la selectividad de los índices para determinar si es pertinente implementar un índice: un índice poco filtrante no mejorará la velocidad de ejecución, añadirá trabajo para cada modificación de la tabla y demandará trabajo adicional al optimizador creando nuevos planes de ejecución que deben examinarse.

Recuerde también que MySQL solo utiliza un índice por tabla: si en una cláusula `WHERE` aparecen varias columnas, es más beneficioso crear un índice sobre todas las columnas. De esta manera, se podrá realizar un filtro de forma simultánea en todas las columnas.

4. Ordenación

MySQL dispone de dos maneras para realizar una ordenación: utilizar un índice o recuperar las filas sin clasificar y realizar una operación adicional para ordenarlas.

La utilización de un índice es a priori interesante, pero resulta a veces frustrante. En efecto, el recorrido del índice es rápido, pero cada valor del índice apuntará a datos distribuidos de forma aleatoria y, si los datos se encuentran en disco, las operaciones tomarán mucho tiempo. Para evitar este tipo de problema, pruebe lo antes posible a ubicar un índice que permita a la vez filtrar y ordenar o coloque un índice de cobertura.

Por lo general, un índice solo puede ser útil para una ordenación si las columnas de selección forman un prefijo izquierdo de las columnas del índice. Si este no es el caso pero las columnas en la cláusula `WHERE` complementan las columnas que faltan, la clasificación podrá ser efectuada por el índice y obtendremos un índice que filtra y ordena simultáneamente. Esto no es válido si la condición de la columna de la cláusula `WHERE` utilizada para completar el índice es una igualdad y no un intervalo de valores.

Así, en la tabla `rental` de la base `sakila`, existe un índice de columnas (`rental_date`, `inventory_id`, `customer_id`). Las condiciones siguientes

podrán utilizar el índice:

```
mysql> ... ORDER BY rental_date, inventory_id;

mysql> ... WHERE rental_date = '2006-04-10 00:00:00' ORDER BY
inventory_id;
```

pero no estas condiciones:

```
mysql> ... WHERE customer_id = 1 ORDER BY inventory_id;

mysql> ... WHERE rental_date > '2006-05-24 00:00:00' ORDER BY
inventory_id;
```

5. Agregaciones

Las funciones de agregación (COUNT () , SUM () , MAX () , MIN () , etc.) son difíciles de optimizar porque, por lo general, hay que leer toda la tabla para obtener el resultado. Sin embargo, existe al menos un caso concreto simple de optimizar: se trata de COUNT (*) , sin requisito adicional para las tablas MyI SAM, que es muy rápido, simplemente porque el motor de almacenamiento mantiene entre sus metadatos el número de registros de la tabla.

Así, el comando EXPLAIN sobre la siguiente petición:

```
mysql> EXPLAIN SELECT COUNT(*) FROM film_text\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: NULL
         type: NULL
possible_keys: NULL
```

```
key: NULL
key_len: NULL
ref: NULL
rows: NULL
Extra: Select tables optimized away
```

nos enseña que no es necesario ningún acceso a la tabla (la columna `type` es `NULL`), ya que la petición pudo simplificarse durante la fase de optimización (columna `Extra`).

En los otros casos, la mejor solución consiste por lo general en implementar un índice de cobertura o generar una tabla de caché (ver el apartado dedicado a este tema más adelante en este capítulo).

- Observe que solo las peticiones con un `COUNT (*)` sin condición `WHERE` adicional son muy rápidas con las tablas `MyISAM`; otras `COUNT ()` no podrán beneficiarse de la misma optimización.

6. Reescritura de peticiones

Las mejores peticiones candidatas para una reescritura son las subconsultas, que MySQL ejecuta a veces muy lentamente. Si es el caso, tenga en cuenta que algunas subconsultas pueden transformarse en joins.

Por ejemplo, con la base `world`, si deseamos conocer todos los idiomas hablados en Francia, podemos escribir una subconsulta:

```
mysql> SELECT Language
      FROM CountryLanguage
     WHERE CountryCode =
           (SELECT Code
            FROM Country
           WHERE Name='Francia');
```

Pero también podemos escribir un join equivalente:

```
mysql> SELECT language
      FROM CountryLanguage
     INNER JOIN Country
      ON Code=CountryCode
     WHERE Name='Francia';
```

Tenga en cuenta que, a partir de MySQL 5.6 (también es el caso de MariaDB 5.5), la ejecución de las subconsultas ha sido mejorada, en particular para subconsultas con la cláusula FROM. ¡No debemos creer que es imperativo evitar las subconsultas con MySQL!

7. Utilización de varios índices

A partir de la versión 5.0, MySQL es capaz, en algunas situaciones, de utilizar varios índices en la misma tabla. El caso más frecuente es el de una petición con OR en la cláusula WHERE y para la que ninguno de los índices de cada una de las condiciones es muy selectivo. MySQL podrá utilizar varios índices para filtrar de forma simultánea en varias columnas.

Ejemplo:

```
mysql> EXPLAIN SELECT * FROM rental WHERE customer_id = 100 OR
staff_id = 100\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: rental
         type: index_merge
possible_keys: idx_fk_customer_id,idx_fk_staff_id
          key: idx_fk_customer_id,idx_fk_staff_id
       key_len: 2,1
         ref: NULL
        rows: 25
```

```
Extra: Using union(idx_fk_customer_id,idx_fk_staff_id);  
Using where
```

La columna `type` indica `index_merge`, y la columna `key` demuestra que el índice de cada una de las columnas de la cláusula `WHERE` se utiliza.

8. Otras características

Algunos administradores de bases de datos nos dirán que las peticiones más rápidas son las que el servidor no tiene necesidad de ejecutar... Detrás de este tópico se esconde una realidad: si podemos evitar la realización de algunas peticiones, en particular aquellas pesadas, el servidor MySQL solo podrá comportarse mejor.

Las peticiones que realicen cálculos de agregados (`SUM ()`, `COUNT ()` ...) son a menudo a la vez lentas y difíciles de optimizar. Estas peticiones son, por lo general, muy buenas candidatas para establecer tablas de caché, o tablas de valores precalculados, ya que el resultado no tiene la mayoría del tiempo la necesidad de ser muy exacto. Si queremos conocer el número de visitantes a la página de inicio y los visitantes se cuentan por centenares de miles, un error 500 o 1 000 en el recuento será con toda probabilidad irrelevante. En este caso, podemos contentarnos con hacer el cálculo de vez en cuando y almacenar el resultado en una tabla. Entre ambos cálculos, solo tendremos que hacer una petición `SELECT` sencilla sobre la tabla que contiene el resultado precalculado.

La actualización de la tabla de caché puede hacerse mediante una tarea programada o, si estamos usando MySQL 5.1, con el planificador de eventos. La frecuencia de actualización dependerá de nuestra aplicación: cuanto menos actualicemos la tabla de caché, menos tiempo de cálculo, pero un mayor nivel de desvío de los resultados. En algunos casos, podremos arrancar una reconstrucción de la tabla de caché cada 24 horas, mientras que en otras situaciones será necesario rehacer el cálculo cada 30 minutos.

Tenga en cuenta, a su vez, que si necesitamos tener los resultados siempre al día, podemos intentar utilizar un trigger para mantener actualizada la tabla de caché.

☐ Todas estas técnicas permiten construir el equivalente de las vistas

materializadas que ofrecen otros SGBD. Para más información sobre las vistas, consulte el capítulo Otras funcionalidades - secciones Rutinas almacenadas, Disparadores (triggers), Eventos.

Optimizaciones para MySQL 5.6/5.7

MySQL introdujo muchos cambios en el optimizador de peticiones. El objetivo es lograr encontrar nuevas maneras de realizar peticiones que eran muy lentas en las versiones anteriores, es decir, no se ejecutaban de forma óptima. Cabe señalar que MariaDB 5.5 implementa casi los mismos cambios, aunque el código es diferente.

Las técnicas que presentamos a continuación son seleccionadas de forma automática por el optimizador cuando aportan un beneficio. En teoría no tenemos, por lo tanto, que hacer nada para aprovecharlas. Tenga presente que solo la experiencia permite confirmar en qué casos estas mejoras son interesantes y en qué casos es mejor evitarlas.

1. Index Condition Pushdown

Ya hemos mencionado el hecho de que MySQL no puede filtrar empleando las columnas de un índice más allá de la primera diferencia. Por ejemplo, si una tabla contiene un índice de columnas (a, b) y la petición se refiere a la condición `WHERE a > 5 AND b = 10`, solo la primera columna del índice puede utilizarse para el filtrado.

A partir de MySQL 5.6, una optimización llamada Index Condition Pushdown permite poner en el índice las condiciones que no pueden, por norma, tenerse en cuenta; así se evita al motor de almacenamiento tener que buscar la fila correspondiente y rechazarla si no cumple los requisitos.

Veamos el funcionamiento en un ejemplo. Vamos primero a modificar un poco la tabla `film` para probar un ejemplo sencillo sobre una versión 5.5:

```
mysql> UPDATE film SET release_year = release_year - FLOOR(RAND()*10);  
Query OK, 893 rows affected, 0 warning (0.18 sec)
```

Rows matched: 1000 Changed: 893 Warnings: 0

mysql> ALTER TABLE film ADD INDEX release_length_idx
(release_year,length);

Query OK, 0 rows affected (0.33 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> EXPLAIN SELECT * FROM film WHERE release_year > 2005 AND length

=
100\G

***** 1. row *****

id: 1

select_type: SIMPLE

table: film

type: range

possible_keys: release_length_idx

key: release_length_idx

key_len: 2

ref: NULL

rows: 107

Extra: Using where

Después de haber hecho la misma operación en una versión 5.6, he aquí la pantalla que produce EXPLAIN:

mysql> EXPLAIN SELECT * FROM film WHERE release_year > 2005 AND length

=
100\G

***** 1. row *****

id: 1

select_type: SIMPLE

table: film

type: range

possible_keys: release_length_idx

key: release_length_idx

key_len: 2

ref: NULL

rows: 115

¿Cuál es la diferencia? Todo está en la columna `Extra:` en lugar de `Using where` para la versión 5.5, que indica un posfiltrado por el servidor una vez que el motor de almacenamiento ha recuperado las filas, tenemos `Using index condition`, que indica que se ha puesto en el índice la condición de que forme parte del índice, pero que no pueda usarse de forma directa.

Por supuesto, sobre una tabla pequeña, no verá una diferencia importante a nivel del tiempo de ejecución, pero, para las peticiones que requieran muchas lecturas en disco (lecturas lentas en esencia), la mejora puede ser considerable.

2. Multi Range Read

Al utilizar un índice secundario, si las columnas que deben recuperarse no se encuentran en el índice, deben leerse al acceder a la clave primaria (estamos hablando de `InnoDB` y el concepto de índice clúster. Para `MyISAM`, esto es cierto para cualquier tipo de índice). Sin embargo, si los datos se leen de forma secuencial en el índice secundario, hay muy pocas posibilidades de que los datos se lean de forma secuencial en la clave primaria, solo porque los valores en el índice secundario no están ordenados de la misma manera que en la clave primaria.

¿Cuál es la consecuencia? Siendo rápida la lectura del índice al ser secuencial, las lecturas en la clave primaria pueden ser muy lentas porque son aleatorias, como acabamos de explicar.

El objetivo de la optimización llamada `Multi Range Read` es recoger en una caché las tuplas procedentes de un índice secundario, ordenar estas tuplas por el valor de su clave primaria y luego recorrer la clave primaria de forma secuencial para recuperar las filas correspondientes.

Si la caché es demasiado pequeña para almacenar todas las tuplas, se rellena y ordena en primer lugar, luego se buscan las filas correspondientes. Esto permite limpiar la caché para reiniciar tantas veces como sea necesario. El interés es siempre transformar las lecturas aleatorias lentas en lecturas lo más secuenciales posible, mucho más rápidas.

La caché utilizada para estas operaciones puede, por supuesto, configurarse. Se trata de la variable `read_rnd_buffer_size`.

Una vez más, esta optimización es especialmente interesante cuando la lectura de datos requiere de acceso a disco.

Mantenimiento de las tablas

1. Actualización de las estadísticas del índice

Disponer de estadísticas de índice actualizadas es tener la garantía de que el optimizador de peticiones seleccionará el mejor plan de ejecución. Por lo tanto, le asegura que el rendimiento del servidor no se degrada de forma aleatoria solo porque algunas peticiones van a resolverse con un costoso recorrido completo de la tabla, cuando con un índice se podría haber limitado el número de filas para leer.

En general, notaremos que no es necesario forzar un nuevo cálculo de estas estadísticas porque InnoDB las vuelve a calcular cada cierto tiempo mediante la opción `innodb_stats_auto_recalc`, activada por defecto. Este cálculo se realiza principalmente cuando el número de registros de la tabla ha variado de forma significativa. Para calcular las estadísticas, InnoDB no tiene necesidad de desplazarse por la tabla completa; recorre solo una muestra de páginas y deduce las estadísticas que supone correctas para toda la tabla.

Esta forma de proceder tiene la ventaja de hacer la operación de actualización de las estadísticas poco costosa en tiempo y carga del servidor. Pero si algunas tablas tienen distribuciones de datos muy heterogéneas, este muestreo puede ocasionar estadísticas erróneas. El número de páginas para la toma de muestras puede modificarse si es necesario (ver más adelante).

Para volver a calcular de forma manual las estadísticas de una tabla, usamos el comando `ANALYZE TABLE`. Este comando es bloqueante, pero solo suele durar unos segundos. Es, por consiguiente, bastante sencillo realizar los trabajos de producción, incluso en tablas de gran tamaño.

Por ejemplo, veamos el tiempo de ejecución de una tabla de gran tamaño en un entorno de

producción.

Hasta MySQL 5.5, las estadísticas se perdían en cada reinicio del servidor. Esto planteaba algunas veces problemas de rendimiento después de un reinicio, ya que las nuevas estadísticas debían ser calculadas, lo que podría dar lugar a planes de ejecución de poca calidad. A partir de MySQL 5.6, las estadísticas de índice son persistentes por defecto (opción `innodb_stats_persistent` activada). Entonces podemos obtener información sobre estas estadísticas en las tablas `innodb_table_stats` e `innodb_index_STATS` de la base de datos de sistema `mysql`.

Usted puede influir en la rapidez y precisión del comando `ANALYZE TABLE` (y los recálculos automáticos de estadísticas) mediante una de las dos variables:

- **Innodb_stats_persistent_sample_pages** (valor por defecto: 20) si usamos las estadísticas persistentes.
- **Innodb_stats_sample_pages** (valor por defecto: 8) si no usamos estadísticas persistentes.

2. Defragmentación de las tablas

Imaginemos la situación siguiente: el espacio en disco del servidor está ocupado en un 99 % y sabemos que puede purgar el 80 % de los datos de una tabla de mucho volumen. Puede así recuperar unos valiosos gigabytes. Por desgracia, incluso con la opción `innodb_file_per_table`, el archivo de datos `.idb` no cambiará de tamaño. Tendremos que reconstruir la tabla con el comando `OPTIMIZE TABLE` para recuperar el espacio correspondiente a los datos borrados.

Para InnoDB, `OPTIMIZE TABLE` es equivalente a un `ALTER TABLE` vacío, puede utilizar cualquiera de estos dos comandos:

```
mysql> OPTIMIZE TABLE t ;  
mysql> ALTER TABLE t ENGINE=INNODB ;
```

- Puede que se pregunte cuál es la utilidad de tener dos comandos con un resultado idéntico. `OPTIMIZE TABLE` es en realidad un comando que puede usarse con cualquier motor de almacenamiento. InnoDB no tiene soporte para este comando, y los desarrolladores han decidido que sería equivalente a ejecutar un comando `ALTER TABLE` vacío.

A diferencia de `ANALYZE TABLE`, que es muy rápido, `OPTIMIZE TABLE` es siempre una operación pesada. En InnoDB, ya que el comando es equivalente a un `ALTER TABLE`, puede disfrutar de la funcionalidad de cambio de esquema no bloqueante a partir de MySQL 5.6. Y en todas las versiones de MySQL, `pt-online-schema-change` (o equivalente) puede usarse para hacer la operación de defragmentación no bloqueante.

3. Otros comandos

También existen los comandos `CHECK TABLE` y `REPAIR TABLE`. Estos dos comandos solo son útiles para las tablas MyISAM. `CHECK TABLE` comprobará si los datos son incoherentes y `REPAIR TABLE` intentará reparar las incoherencias.

InnoDB, con su mecanismo de recuperación automática de datos en caso de fallo inesperado, no requiere de estos comandos.

Aspectos generales sobre la replicación

MySQL dispone de un sistema nativo de replicación, fácil de implementar y que puede ser útil en muchas situaciones. Cuando un servidor (el esclavo) replica los datos de otro servidor (maestro), el esclavo se sincroniza de forma automática con respecto a su maestro. La mayoría de las aplicaciones que emplean MySQL como SGBD utilizan la replicación para ayudar a mantener la carga, aumentar la disponibilidad de la base de datos, simplificar la creación de copias, descargar el maestro de grandes peticiones consumidoras de recursos y muchas otras razones.

Este capítulo va a explicar, entre otras cosas, cuáles son los problemas que la replicación puede resolver, cómo configurar un maestro y un esclavo y lo que hay que hacer si la replicación no funciona de forma adecuada.

1. Utilidad de la replicación

La replicación le puede ayudar para un gran número de problemas con las bases de datos:

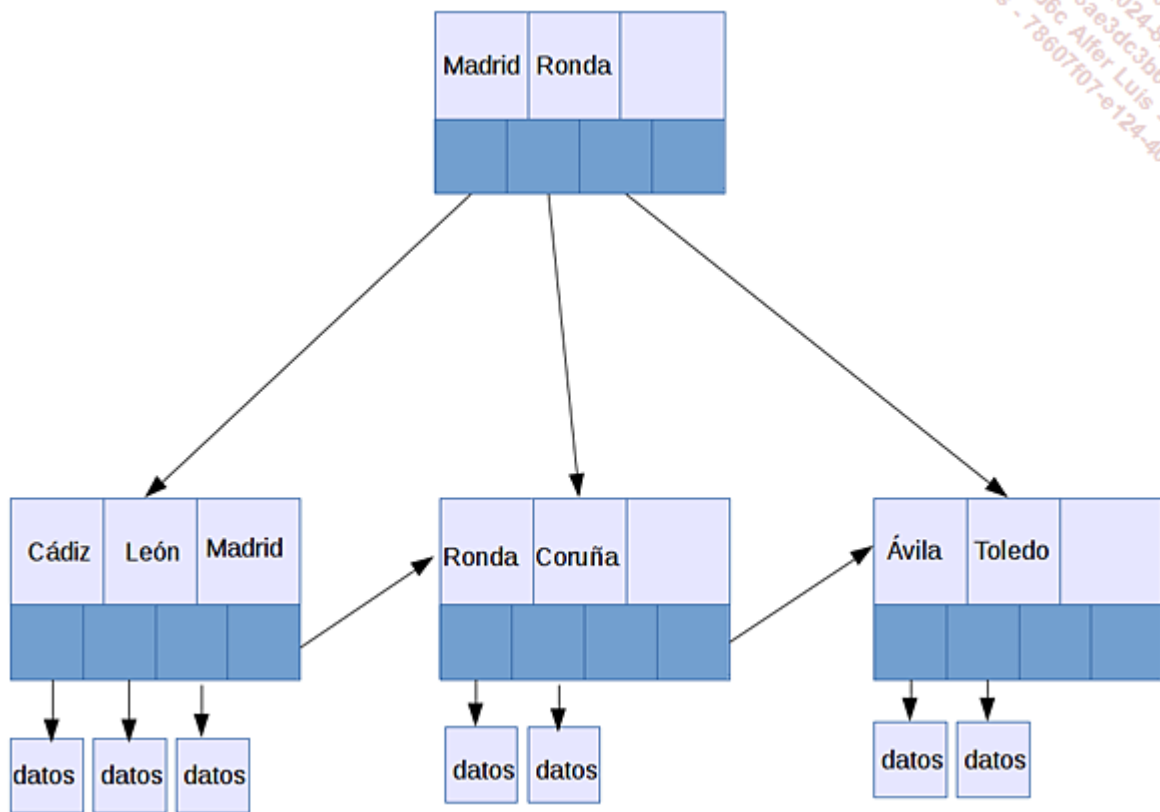
- El conjunto en el aumento de la carga de las lecturas: a menudo, una aplicación solo funciona en un principio con un servidor de bases de datos. Pero si el tráfico aumenta, se llega rápidamente a un punto en el que el servidor no tiene la capacidad de hacer frente a todas las lecturas y todas las escrituras. En las aplicaciones web, las lecturas son por lo general mayores en relación con las escrituras. En este caso, implementando servidores esclavos, podemos mover las escrituras al maestro y las lecturas a uno de los esclavos. Observe que, para las aplicaciones donde hay mayoría de lecturas, este modelo no funciona.
- Una ayuda con los respaldos: respaldar los datos, evidentemente, es indispensable,

pero el impacto en el servidor a menudo está lejos de ser insignificante. Mediante el respaldo en un esclavo, que contiene una copia de los datos del maestro, es mucho más simple disminuir el impacto de las copias de seguridad sobre la aplicación. Preste atención: hacer una copia de seguridad en un esclavo significa que hay que estar seguro de que los datos son los mismos que en el maestro. Recuerde también que la replicación no constituye en sí misma una copia de seguridad (¡de la misma manera que configurar sus discos en RAID no aporta absolutamente ningún respaldo!).

- Una mejora de la disponibilidad: ¿qué ocurre con una aplicación monoservidor cuando este se cae? ¡La aplicación deja de funcionar! Cuando disponemos de un esclavo con los datos del maestro, podemos hacer que este esclavo se convierta en el nuevo maestro el tiempo necesario para poner en marcha el antiguo maestro (esto es lo que se conoce como la promoción de un esclavo).
- Una redundancia geográfica: puede ser útil tener una copia de los datos en un centro de hospedaje remoto. Por un lado, en caso de problema en su centro de alojamiento habitual, dispondremos de una solución de emergencia, y por otra parte, para las aplicaciones de los usuarios en todo el mundo, se obtiene un mejor rendimiento a una petición con un servidor geográficamente cercano.
- Una ayuda para las operaciones pesadas: es difícil optimizar un servidor simultáneamente para los procesos transaccionales (muchas peticiones simples) y para procesos estadísticos (pocas peticiones pero muy pesadas). La replicación permite dedicar uno o varios esclavos a la ejecución de estos tratamientos pesados.

2. Funcionamiento de la replicación

El siguiente esquema resume el funcionamiento de la replicación:



Todo empieza en el servidor maestro con el registro de cambios en los registros binarios (binlogs). Estos cambios serán recuperados por los esclavos y luego ejecutados. Más en concreto, cuando la replicación se pone en marcha, dos threads se ejecutan en el esclavo: el primero (IO_THREAD) copia la información de los registros binarios del maestro en un registro especial del esclavo (relay log), y el segundo (SQL_THREAD) ejecutará las modificaciones incluidas en los relay logs. Se abre un tercer thread en el maestro por el IO_THREAD para la transferencia del contenido de los registros binarios. Esta operación se realiza mediante un proceso particular llamado binlog dump, para el cual no existe nada equivalente en SQL.

Este funcionamiento tiene una consecuencia muy importante: aunque en el maestro las modificaciones podrán realizarse en paralelo cuando varias conexiones simultáneas modifican los datos, los mismos cambios se aplicarán de forma secuencial en los esclavos, ya que solo el SQL_THREAD está a cargo de la ejecución de los cambios presentados por la replicación. Es, por tanto, bastante fácil engordar la replicación: si las escrituras son muchas y con un paralelismo elevado sobre el maestro, los esclavos pueden tener muchas dificultades para seguir el ritmo. Por esta razón, la replicación no es una buena solución cuando se trata de tener un aumento de carga relacionado con las escrituras.

A partir de MySQL 5.6, la replicación multithread está disponible para



disminuir esta limitación. El funcionamiento ha sido mejorado de forma significativa con MySQL 5.7. Se dedica una sección en la continuación de este capítulo.

También es útil saber que un esclavo no puede tener más que un maestro, pero que un maestro puede tener varios esclavos. Veremos más adelante en este capítulo cuáles son las configuraciones comunes cuando se dispone de varios servidores.

3. Formatos de replicación

MySQL puede utilizar uno de los tres formatos siguientes para los registros binarios: STATEMENT, ROW o MIXED. Consulte la sección sobre el registro en el capítulo Configuración del servidor si estos términos no le resultan familiares.

Hasta MySQL 5.6, STATEMENT era el formato por defecto y también el más utilizado. Su gran ventaja es que las peticiones se almacenan en los registros binarios fáciles de leer, pues son una copia exacta de las peticiones presentadas al maestro.

Pero este formato tiene dos inconvenientes importantes:

- El primero es el derroche de recursos. Imaginemos un maestro con veinte esclavos y una petición de actualización que toma unos segundos para ser ejecutada cuando una sola fila se modifica. ¿Qué ocurrirá con el formato STATEMENT? La demanda será ejecutada en el maestro, escrita en los registros binarios y luego transferida a cada uno de los esclavos. Y cada esclavo tendrá que ejecutar esta pesada petición. Sería mucho más inteligente escribir en los registros binarios no la petición original, sino solo los elementos modificados por la petición: de esta manera, el proceso será muy rápido en los esclavos. Recuerde que, por defecto, los esclavos tienen una desventaja en comparación con el maestro, ya que ejecutan las peticiones procedentes de la replicación de forma secuencial, mientras que el maestro puede llevar a cabo las escrituras en paralelo. Además, cualquier optimización de los esclavos es bienvenida.
- El segundo es que abre la posibilidad de crear incoherencias de datos entre maestro y esclavo. Imaginemos una petición de actualización que termina en la siguiente cláusula: `ORDER BY RAND() LIMIT 3`. A causa del `ORDER BY`

RAND () , esta petición no es determinista, ya que, si la ejecutamos en dos servidores con los mismos datos, tendremos seguramente dos resultados diferentes. Así puede ocurrir, ya que la replicación no muestra un error, aunque una petición modifique filas diferentes en el maestro y el esclavo. Se trata de un problema en especial irritante, porque en general suponemos que los datos son idénticos entre maestro y el esclavo.

Para estas dos razones, el formato ROW se volvió popular y se trata ahora del formato por defecto con MySQL 5.7. El registro binario solo contiene las peticiones originales y no una representación de los cambios, lo que permite eliminar los dos problemas antes mencionados. El principal inconveniente del formato ROW es que no podemos reconstruir con facilidad las verdaderas peticiones ejecutadas en el maestro.

Otro problema puede ocurrir si tenemos columnas del tipo BLOB/TEXT que no modificamos: por defecto, el formato ROW almacena en el registro binario todo el contenido de las columnas modificadas. Si solo cambiamos un número entero en una fila que contiene también un BLOB de 500 MB, el BLOB de 500 MB se encontrará en el registro binario, lo que es muy pesado e ineficaz.

MySQL ofrece, sin embargo, una variable de configuración, `binlog_row_image`, que permite controlar mejor los datos escritos en los registros binarios cuando la replicación RBR está activada.

- Con `binlog_row_image = full` (valor por defecto), todas las columnas de las filas modificadas se graban en los registros binarios. Se trata del único comportamiento posible hasta MySQL 5.5.
- Con `binlog_row_image = minimal`, solo se graban las columnas modificadas.
- Con `binlog_row_image = noblob`, todas las columnas salvo los campos BLOB y TEXT se grabarán. Los campos BLOB y TEXT no se grabarán en los registros binarios, salvo que sean modificados.

Con el fin de obtener lo mejor, se desarrolló el formato MIXED: las peticiones están escritas en formato STATEMENT por defecto, pero el formato cambia de forma automática a ROW para todas las peticiones no deterministas. Como se mencionó en el

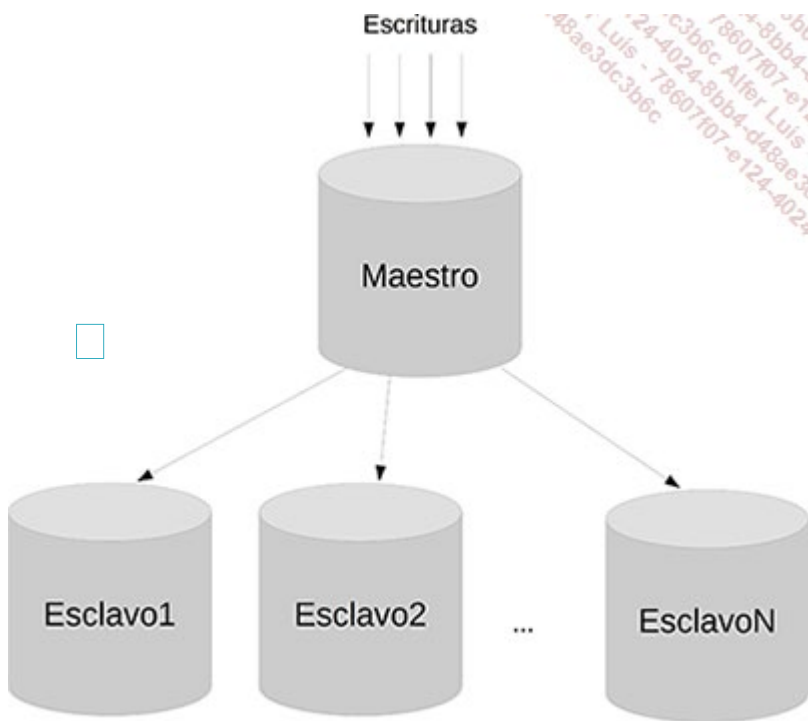
capítulo Configuración del servidor, este modo no es, sin embargo, ideal, ya que hay varios bugs importantes y no corregidos.

En general, con frecuencia interesará utilizar el formato ROW.

Puesta en marcha de la replicación

1. Replicación maestro-esclavo(s)

a. Configuración



Comenzaremos por el caso más simple: aquel en el que el maestro y el esclavo están recién instalados, es decir, sin ningún dato. Deberá realizar los siguientes pasos:

- Crear un usuario para la replicación en el maestro:

El esclavo debe poder conectarse con el maestro (IO_THREAD). Es necesario crear una cuenta dedicada en el maestro con el derecho REPLICATION SLAVE. En realidad, muchas veces, es conveniente crear también una cuenta simétrica en el esclavo (útil si el esclavo puede ser ascendido a maestro; en este caso, la replicación tendrá que ejecutarse en el sentido opuesto) y añadir el derecho REPLICATION CLIENT, que permite comandos externos para el control de la replicación.

```
mysql> GRANT REPLICATION SLAVE, REPLICATION CLIENT on *.* to  
'repli_user'@'IP_esclavo' IDENTIFIED BY 'mi_cont';
```

- Configurar el maestro:

Hay que activar los registros binarios en el maestro y declarar un usuario llamado `server_id`, que deberá ser único entre todos los servidores asociados por la replicación. Estos cambios se hacen en el archivo `my.cnf/my.ini` y requieren un reinicio del servidor:

```
[mysqld]  
log_bin = /var/lib/mysql/mysql-bin  
server_id = 100
```

- Configurar el esclavo:

Cada esclavo también debe tener un `server_id` único. Es inútil activar los registros binarios, salvo si el esclavo es él mismo maestro de otros esclavos:

```
[mysqld]  
server_id = 101
```

- Introducir las coordenadas de replicación:

Obtendrá las coordenadas de replicación en el maestro con la instrucción siguiente:

```
mysql> SHOW MASTER STATUS;  
  
+-----+-----+-----+-----+  
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |  
+-----+-----+-----+-----+  
| mysql-bin.004  | 106      |              |                  |  
+-----+-----+-----+-----+
```

A continuación debemos utilizar estas coordenadas en el esclavo:

```
mysql> CHANGE MASTER TO MASTER_HOST = 'IP_maestro',  
MASTER_USER = 'repli_user',  
MASTER_PASSWORD = 'mi_cont',  
MASTER_LOG_FILE = 'mysql-bin.004',  
MASTER_LOG_POS = 106;
```

- Arrancar la replicación en el esclavo:

Basta con ejecutar el comando `START SLAVE` y luego `SHOW SLAVE STATUS` para comprobar que todo es correcto:

```
mysql> START SLAVE;  
mysql> SHOW SLAVE STATUS\G  
[...]  
Slave_IO_Running: Yes  
Slave_SQL_Running: Yes
```

En la realidad, con frecuencia tendremos la necesidad de configurar un esclavo, mientras que el maestro ya tiene los datos. Si nos contentamos con iniciar la replicación en la posición más reciente de los registros binarios del maestro, obtendremos un error de forma inmediata. En efecto, el esclavo no recupera los datos del maestro antes de comenzar a responder a las peticiones. Debemos disponer de una copia de seguridad. Observe que necesita imperativamente la posición en los registros binarios en el momento del respaldo. Esta posición será la que se utilizará para comenzar la replicación.

Si utiliza `mysqldump`, esta posición se encuentra al principio del dump `mysqldump` si se inició con la opción `--master-data` en el maestro o `--dump-slave` en un esclavo:


```
# head -25 dump.sql
[...]
--
-- Position to start replication or point-in-time recovery from
--
CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000003',
MASTER_LOG_POS=154;
```

Si utiliza XtraBackup, usará la información del archivo

`xtrabackup_binlog_info` si el respaldo se realizó en el maestro o la información del archivo `xtrabackup_slave_info` si el respaldo se llevó a cabo en un esclavo:

```
# cat xtrabackup_binlog_info
mysql-bin.000001 257
```

En ambos casos, comenzaremos por restaurar el respaldo en el esclavo. La posición en los registros binarios definida en el respaldo proporcionará los parámetros para el comando `CHANGE MASTER TO`, por ejemplo:

```
mysql> CHANGE MASTER TO MASTER_LOG_FILE=' mysql-bin.000001',
MASTER_LOG_POS=257, [...] ;
```

Un maestro puede tener múltiples esclavos: si desea por ejemplo obtener una configuración con un maestro y diez esclavos, habrá que repetir los pasos anteriores para cada esclavo (el maestro solo necesita configurarse una sola vez, luego puede aceptar varios esclavos).

b. Puntos fuertes y débiles de esta configuración

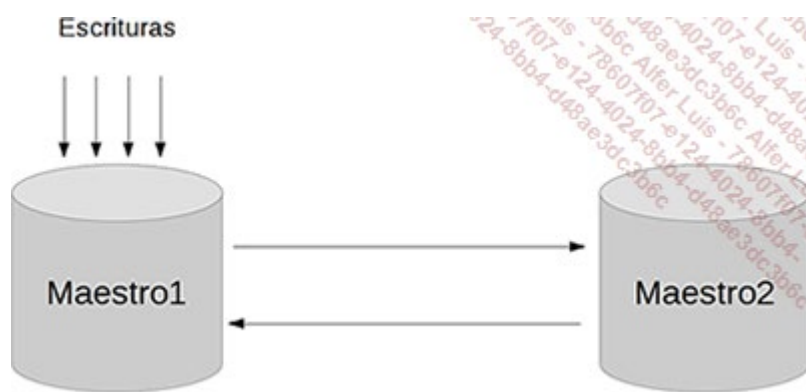
La configuración maestro-esclavo permite ser escalable en lectura, es decir, se puede aumentar el número de lecturas que pueden soportarse, añadiendo esclavos. Esta configuración es simple de implementar, pero es necesario mandar las escrituras al maestro y las lecturas a los esclavos. Por defecto, MySQL no impide las escrituras sobre los

esclavos, pero es posible y aconsejable ubicar la variable `read-only` para impedir los cambios, excepto las enviadas por los usuarios que tengan el privilegio `SUPER` o los procedentes de la replicación.

La replicación asíncrona no certifica que los datos de los esclavos están sincronizados con los datos del maestro; la lectura también debe hacerse en el maestro cuando es necesario leer los datos que acaban de ser modificados. En general, en los casos en que deseemos dirigir el mayor número de lecturas posibles en los esclavos, su aplicación deberá determinar si una lectura dada tiene gran necesidad de leer los datos más recientes. En caso afirmativo, deberá ejecutarse en el maestro, si no puede enviarse a un esclavo.

Otra problemática puede plantearse en el momento de la pérdida del maestro. Los esclavos pueden, por ejemplo, haber alcanzado posiciones diferentes en los registros binarios del maestro. La promoción de un esclavo puede en este caso ser compleja (ver la parte replicación y alta disponibilidad más adelante en este capítulo).

2. Replicación maestro-maestro



a. Configuración

La configuración de una replicación bidireccional se hace simplemente configurando cada servidor a la vez como un maestro y como un esclavo. Se recomienda añadir la opción `log_slave_updates = 1` en el archivo de configuración de cada uno de los servidores. El objetivo de esta opción es asegurarse de que el servidor que tiene el papel de esclavo escribe en sus registros binarios las peticiones provenientes del servidor que tiene el papel de maestro. Con esta opción, podemos añadir esclavos a cualquiera de los dos servidores.

Cuando la replicación está establecida en un sentido, es fácil encontrar las coordenadas de

replicación para configurar la replicación en otro sentido. Basta con ejecutar el comando `SHOW MASTER STATUS` en el esclavo y luego utilizar esa información para escribir el comando `CHANGE MASTER` correspondiente en el maestro.

- ☐ Tenga en cuenta que, si los servidores A y B están configurados con una replicación bidireccional, una escritura ejecutada sobre A será replicada hacia B, pero no será replicada por segunda vez de B a A. En efecto, cada petición almacena en los registros binarios de su servidor de origen y la replicación no se ejecutará nunca por la replicación de una escritura procedente de B pero marcada como ejecutada en primer lugar por A.
- ☐ Mediante este mecanismo, no es necesario recortar las escrituras en el maestro antes del comando `SHOW MASTER STATUS`.

b. Puntos fuertes y débiles de esta configuración

El caso más simple es el modo activo-pasivo, en el que un maestro se utiliza para las lecturas/escrituras, mientras que el segundo maestro se utiliza como maestro de seguridad (pero también puede usarse en modo de solo lectura). Esta configuración permite evitar cualquier conflicto en el plano de la replicación, ya que los cambios solo se hacen en un servidor. También permitiría facilitar las operaciones de mantenimiento (actualización de hardware, defragmentación de las tablas con los comandos `ALTER TABLE`, actualizar las estadísticas con el comando `ANALYZE TABLE`, etc.) que puede hacerse a su vez en cada uno de los maestros cambiando su papel. Resulta más fácil realizar operaciones de *failover* (cambio de maestro activo) o de *fallback* (cambio de maestro activo para volver a la configuración de origen) simplemente desviando las lecturas/escrituras sin intervenir en los servidores MySQL.

También es posible utilizar una configuración multimaestro (a dos maestros) en modo activo-activo, permitiendo que las lecturas/escrituras se hagan sobre cada uno de los maestros. Sin embargo, en esta configuración, es necesario gestionar los conflictos. Antes de la versión 5.0, los campos autoincrementados planteaban problemas, ya que dos órdenes de inserción en la misma tabla que se ejecutan en paralelo en ambos servidores generaban el mismo valor.

```
mysql> INSERT INTO
tabla_replicada(col1_no_autoincrementada,col2_no_autoincrementada)
VALUES(ROUND(RAND()*10),ROUND(RAND()*10));
```

En el primer maestro:

```
mysql> select * from tabla_replicada;
+-----+-----+-----+
| col1_no_autoincrementada | col2_no_autoincrementada | col3 |
+-----+-----+-----+
| 1 | 8 | 1 |
+-----+-----+-----+
```

En el segundo maestro:

```
mysql> select * from tabla_replicada;
+-----+-----+-----+
| col1_no_autoincrementada | col2_no_autoincrementada | col3 |
+-----+-----+-----+
| 4 | 6 | 1 |
+-----+-----+-----+
```

Sin embargo, durante la replicación del orden de inserción entre los maestros, esto generaba un error porque un campo `autoincrement` es por definición único. Para resolver este tipo de problema, están disponibles las variables:

- `auto_increment_increment`, que indica el paso que hay que utilizar para generar los valores de los campos `autoincrementados`.
- `auto_increment_offset`, que define el valor inicial de los campos `autoincrementados`.

Mediante estas variables, ahora es posible dejar de encontrar el error descrito utilizando configuraciones diferentes para cada maestro.

- `auto_increment_offset=1` y `auto_increment_increment=2` en el primer maestro.
- `auto_increment_offset=2` y `auto_increment_increment=2` en el segundo maestro.

Con esta configuración, los campos autoincrementados tomarán los valores impares en el primer maestro y pares en el segundo, lo que soluciona el problema.

Por desgracia, siempre pueden ocurrir otros errores. Es el caso de los campos únicos. Si dos transacciones ejecutadas en paralelo se basan en el último valor de un campo único para generar el valor siguiente y el algoritmo es determinista, entonces el mismo valor único se generará en los dos servidores y obstaculizará nuevamente la replicación.

También es el caso si dos operaciones no conmutativas (multiplicación y eliminación, por ejemplo) se realizan en un mismo campo de forma paralela en los dos maestros:

En el primer maestro:

```
mysql> UPDATE tabla_replicada SET a=a*2;
```

En el segundo maestro:

```
mysql> UPDATE tabla_replicada SET a=a-10;
```

Una vez efectuada la replicación de esta orden, tendremos dos valores diferentes en los dos maestros:

- $A = A * 2 - 10 = 2A - 10$ en el primer maestro.
- $A = (A - 10) * 2 = 2A - 20$ en el segundo maestro.

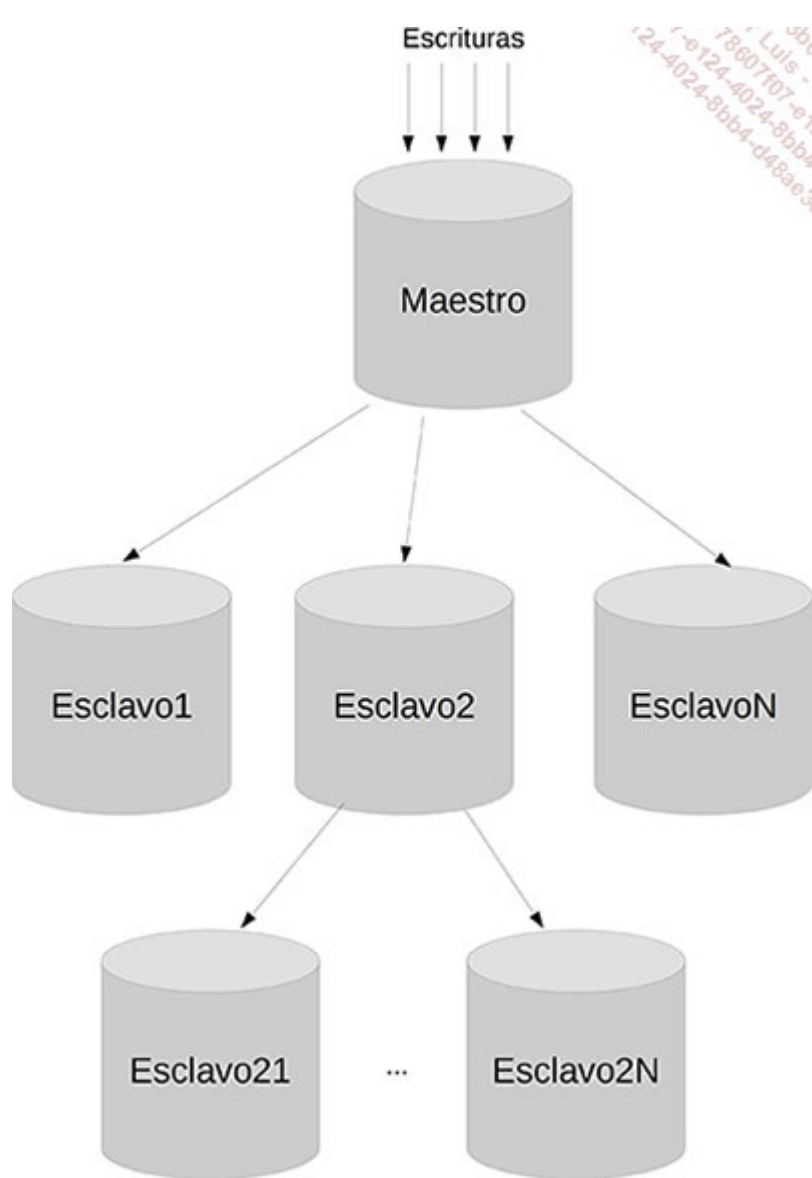
Este tipo de situación conduce a incoherencias de datos entre servidores sin que MySQL emita el menor error. La configuración activo-activo es en particular peligrosa y nunca

debe utilizarse.

Algunos lectores pueden imaginar que la configuración activo-activo puede mejorar el rendimiento de escritura, ya que se puede escribir en ambos servidores al mismo tiempo. Por desgracia, esta solución no es buena: si el primer servidor puede paralelizar la ejecución de las peticiones que recibe de forma directa, la ejecución de las demandas del segundo servidor recibidas a través de la replicación será de forma serial. Los resultados no son tan buenos como en el caso de que el 100 % de las escrituras pueden hacerse en paralelo.

En resumen, la configuración activo-pasivo es interesante para facilitar las operaciones de mantenimiento y el cambio de un servidor a otro. Por el contrario, salvo algunos casos excepcionales, la configuración activo-activo se debe evitar, ya que no es posible asegurar que ningún conflicto existirá entre las transacciones realizadas por los dos servidores.

3. Replicación en varios niveles



a. Configuración

A partir del momento en que un esclavo tiene los parámetros `log_bin` y `log_slave_updates` activados, el esclavo escribe en sus propios registros binarios todas las peticiones recibidas por la replicación. Puede, por lo tanto, servir como maestro a uno o varios esclavos.

Para establecer una configuración de replicación en varios niveles, comience por configurar los esclavos directos del maestro, asegurándose de que los esclavos que van a tener el papel de maestros estén configurados correctamente (`log_bin` y `log_slave_updates`). Luego, podremos configurar los esclavos de los niveles siguientes como esclavos de un esclavo de otro nivel.

b. Puntos fuertes y débiles de esta configuración

Esta configuración puede ser interesante cuando tiene muchos esclavos. En efecto, cada

esclavo añade una cierta carga sobre el maestro para la transmisión de los registros binarios. Si tiene diez esclavos, esto no suele ser un problema. Sin embargo, si tenemos cien esclavos, el maestro puede en algunos momentos pasar todo su tiempo sirviendo a los esclavos. Tener maestros intermedios permite limitar el número de esclavos por maestro.

Otra posibilidad es la distribución geográfica: si tenemos un centro de datos en Europa con el maestro y un centro de datos en Estados Unidos, puede ser interesante tener un único servidor que haga transitar los datos entre Europa y Estados Unidos, servidor que será el propio maestro para otros esclavos en Estados Unidos.

En general, sin embargo, es preferible evitar tal configuración, ya que todos los servidores que son a la vez esclavos y maestros se convierten en servidores críticos. Si uno de ellos deja de funcionar, varios esclavos cesarán de forma inmediata de recibir las actualizaciones del maestro principal. Habrá que reconfigurar los esclavos para que se conecten a otro servidor, pero esta operación es compleja si no usamos identificadores de transacción.

4. Principales variables

Variable	Descripción	Rol	Indispensable
server-id	Identificador del servidor.	Maestro/esclavo	Sí
log-bin	Fuerza la escritura en los registros binarios de órdenes ejecutadas en el servidor MySQL (sin contar con las replicadas a partir de un maestro).	Maestro (posible en un esclavo)	Sí
log-slave-updates	Fuerza la escritura en los registros	Esclavo	No, salvo en una replicación

	binarios de las órdenes recibidas de un maestro y aplicadas.		circular a partir de tres nodos o cuando un servidor esclavo debe propagar los cambios que recibe a sus propios esclavos.
expire_log_days	Especifica la retención en número de días de los registros binarios. Si esta variable es 0, la retención es infinita y es necesario purgar los registros de forma manual (PURGE BINARY LOG).	Maestro (posible en un esclavo si genera registros binarios)	Recomendado
binlog-do-db, binlog-do-table, binlog-ignore-db, binlog-ignore-table	Especifica el nombre de una base/tabla para ¡Atención! Las	Maestro (posible en un esclavo)	No

	base que alberga a la tabla modificada.		
<p>replicate-do-db, replicate-ignore-db, replicate-do-table, replicate-ignore-table, replicate-wild-do-table, replicate-wild-ignore-table</p>	<p>Especifica el nombre de una base/tabla o el modelo de nombre de tabla (utilizando los mismos metacaracteres que la instrucción LIKE, como ' % ' y ' _ ') para replicar o ignorar en el esclavo.</p> <p>¡Atención! Las reglas replicate-*-db prueban la base actual y no la base que alberga a la tabla modificada.</p>	Esclavo	No
innodb_ support_xa	Además del hecho de que activa el soporte de transacciones distribuidas, esta opción fuerza el flush de los registros binarios	Maestro	Altamente recomendable

	cada COMMIT.		
<code>sync_binlog=1</code>	Fuerza la escritura en los registros binarios para cada instrucción «auto commit» o cada transacción validada.	Maestro	No
<code>sync_master_info</code> <code>sync_relay_log</code> <code>sync_relay_log_info</code>	Fuerza la sincronización en disco de los distintos registros relacionados con la replicación en el esclavo.	Esclavo	No
<code>Slave-parallel-workers</code>	Indica el número de threads que ejecutan en paralelo las transacciones (sobre bases diferentes para MySQL 5.6).	Esclavo	No

Resolución de problemas de operación frecuentes

1. Impedir la replicación de algunas peticiones

Por defecto, todas las escrituras del maestro se escriben en los registros binarios. Estos registros binarios se copian en los esclavos; por lo tanto, los esclavos replican todas las peticiones del maestro. Sin embargo, en algunos casos, podemos no desear replicar algunas peticiones; por ejemplo, si queremos tener un esclavo que no contiene uno de los esquemas del maestro. En este caso, deberemos usar un filtro de replicación.



Se puede filtrar en el maestro empleando las opciones `binlog-do-db` y `binlog-ignore-db`, o en los esclavos empleando las opciones `replicate-do-db` y `replicate-ignore-db`. Salvo excepciones, se prefieren siempre los filtros sobre los esclavos. En efecto, un filtro en el maestro impedirá la copia de algunas peticiones en los registros binarios. Sin embargo, en el peor de los casos, podemos necesitar recuperar los datos restaurando primero una copia de seguridad y luego aplicando todas las escrituras almacenadas en los registros binarios. Si las peticiones no se han registrado en los registros binarios, las escrituras correspondientes se perderán.

Otro punto importante es que las reglas de filtrado no funcionan de la misma manera, según el formato de los registros binarios.

Con el formato SBR (*Statement-Based Replication*), solo se verifica la base actual. Por ejemplo, si utilizamos `binlog-do-db=db1`, el comando `INSERT` siguiente se replicará del mismo modo:

```
mysql> USE db1;
```

```
Database changed
mysql > INSERT INTO db2.t1 values(10);
```

A la inversa, la siguiente petición no se replicará, aunque la tabla `t2` pertenezca a la base `db1` que replicamos:

```
mysql> USE db2;
Database changed
mysql > INSERT INTO db1.t2 values(10);
```

Con el formato RBR (*Row-Based Replication*), las órdenes de modificación serán replicadas si los objetos a los que se aplican conciernen a las reglas. Si la tabla modificada no pertenece a la base `db1`, entonces no será replicada, incluso en el caso de `UPDATE` múltiples:

```
mysql> USE db2;
Database changed
mysql> UPDATE db1.t1,db2.t2 SET i=1232,j=4310;
```

Así, durante la replicación del `UPDATE` anterior, solo las modificaciones de la tabla `t1` se replicarán con el modo RBR.

Puede ser difícil recordar de forma exacta qué reglas se aplican para un determinado formato. En este caso, recuerde solo que con el formato de RBR los filtros de replicación funcionan de manera intuitiva, mientras que este no es el caso con el formato SBR.

También es posible solicitar expresamente al maestro no registrar una escritura en los registros binarios utilizando la siguiente orden en la sesión que realiza las peticiones que no se han de replicar:

```
mysql> SET SQL_LOG_BIN=0;
```

Es el caso, por ejemplo, cuando se quiere reconstruir una tabla con el comando `ALTER TABLE`. También podemos querer actualizar las estadísticas sobre las tablas que han sido muy modificadas, o crear informes y almacenarlos en tablas temporales. Si, luego, las órdenes que se han de replicar deben ejecutarse en la misma sesión, basta con cambiar el valor de la variable a 1.

2. No-replicación de una petición

Este problema puede provenir de las reglas de replicación aplicadas en el maestro (con las variables `binlog-do-db` y `binlog-ignore-db`) y en los esclavos (con las variables `replicate-do-db`, `replicate-ignore-db`, `replicate-do-table` y `replicate-ignore-table`). Es importante saber que todas estas opciones solo tienen un valor como argumento y que es necesario repetir la opción si deben usarse varios valores. Por ejemplo, si solo las órdenes sobre las bases `db1` y `db2` deben escribirse en los registros binarios del maestro, usaremos las opciones siguientes en el archivo de configuración:

```
binlog-do-db = db1
binlog-do-db = db2
```

Si hubiese utilizado la opción `binlog-do-db=db1,db2`, esto no habría funcionado como se esperaba y ¡MySQL no habría devuelto ningún error ! De hecho, MySQL permite las comas en los nombres de base, y por lo tanto MySQL buscará ejecutar las órdenes sobre la base `db1,db2` que no existe.

Otro motivo de error es el hecho de que MySQL solo replica en modo SBR las órdenes SQL que se refieren a la base actual, sea cual sea la base sobre la cual la sesión está conectada. Por ejemplo, la siguiente orden no sería replicada:

```
mysql > USE db1;
```

```
mysql > UPDATE db2.t1 SET i=10;
```

El orden UPDATE que modifica la tabla `t1` de la base `db2` no responde porque la base actual es la base `db1`.

3. Evitar el retraso de replicación

Ya se ha mencionado que, por defecto, la replicación trabaja con un solo hilo (thread) en los esclavos, mientras que el maestro puede llevar a cabo escrituras en paralelo. Un esclavo sufre, por lo tanto, de una limitación, y si el tráfico de escritura es elevado en el maestro, el esclavo podría no poder ejecutar las escrituras con suficiente rapidez. Veremos aparecer un retraso de replicación. Este retraso es visible con el comando `SHOW SLAVE STATUS:`

```
Seconds_Behind_Master: 0
```

Para tratar de reducir esta demora, existen varias opciones:

- Cambiar el equipo del esclavo: se puede optar por utilizar un servidor con menos procesadores, pero cuya frecuencia es mayor, con discos más rápidos o con SSD en lugar de discos duros tradicionales. La elección debe hacerse en función de las estadísticas de sistema detectadas.
- Configurar los esclavos para favorecer el rendimiento en lugar de la seguridad de los datos, por ejemplo con `innodb_flush_log_at_trx_commit = 2` en lugar de 1 para el maestro.
- Utilizar el formato ROW en lugar de STATEMENT (atención, el formato ROW no es siempre el más rápido).
- Optimizar las peticiones para que se ejecuten de manera más rápida.
- Utilizar la replicación multithread con MySQL 5.6 y 5.7.

4. Corregir un error de replicación

Para comprobar si un esclavo replica siempre los datos de su maestro, se usa el comando `SHOW SLAVE STATUS`. Si el `IO_THREAD` o el `SQL_THREAD` no funcionan, hay que verificar el contenido de las columnas `Last_Error`, `Last_IO_Error` y `Last_SQL_Error`:

```
mysql> SHOW SLAVE STATUS\G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
        Master_Host: 127.0.0.1
        Master_User: msandbox
        Master_Port: 11806
        Connect_Retry: 60
        Master_Log_File: mysql-bin.000001
    Read_Master_Log_Pos: 1128
        Relay_Log_File: mysql_sandbox11807-relay-
bin.000002
        Relay_Log_Pos: 1091
    Relay_Master_Log_File: mysql-bin.000001
      Slave_IO_Running: Yes
      Slave_SQL_Running: No
        Replicate_Do_DB:
    Replicate_Ignore_DB:
        Replicate_Do_Table:
    Replicate_Ignore_Table:
    Replicate_Wild_Do_Table:
    Replicate_Wild_Ignore_Table:
          Last_Errno: 1062
          Last_Error: Error 'Duplicate entry '23' for
key 'id' on query. Default database: 'test'. Query: 'insert
into t3 values(23)'

[...]
```

En este ejemplo, una petición SQL no ha podido replicarse a causa de un error de unicidad porque el `SQL_THREAD` está detenido. A continuación hay que buscar la causa de este

error, corregirlo y reiniciar la replicación con el comando `START SLAVE SQL_THREAD`.

A veces, después de haber corregido el error, es conveniente anular la petición que había provocado el error. Para ello, existe el comando `SET GLOBAL SQL_SLAVE_SKIP_COUNTER`:

```
mysql > SET GLOBAL SQL_SLAVE_SKIP_COUNTER=1;
...
mysql > START SLAVE;
```

En algunos casos, será preferible evitar de forma directa la petición que plantea un problema sin tratar de corregir el error. Usaremos de forma directa `SET GLOBAL SQL_SLAVE_SKIP_COUNTER`. Este método es peligroso, ya que, a partir del momento en que solicitamos de forma expresa no realizar una petición, hay muchas probabilidades de que los datos del esclavo se desincronicen en comparación con los del maestro. Pero si el problema se produce en momentos de mucha carga, a veces es más ventajoso simplemente ignorar el error y reconstruir el esclavo más tarde.

Si no es posible corregir los errores, será necesario reconstruir el esclavo a partir de una copia de seguridad. Si la base de datos es voluminosa, esta operación puede tomar varias horas fácilmente.

Si el error es redundante y no hay ningún otro medio de solventarlo más que con el comando `SET GLOBAL SQL_SLAVE_SKIP_COUNTER`, es posible utilizar la variable `slave-skip-errors` (que requiere un reinicio del servidor) para forzar el servidor a pasarlo de forma automática.

```
mysql [esclavo] > show global variables like 'slave_skip_errors';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| slave_skip_errors | 1062 |
+-----+-----+
```

1 row in set (0.00 sec)

☐ Este método no se recomienda en el caso de que no se domine el error.

En contraposición, esto puede ser interesante en algunos casos, como por ejemplo cuando se necesita volver a ejecutar los registros binarios que contienen solo las órdenes de inserción en las tablas con limitaciones de singularidad y para los cuales no se conoce la posición en la que los registros deberán releerse. De esta forma, las órdenes con error se pasarán y, una vez alcanzadas las nuevas inserciones, ya no se recuperará ningún error. Se tendrá cuidado en este caso de desactivar esta variable, ya que no seguirá siendo útil.

5. Recuperar el espacio en disco de los registros binarios

Por defecto, los registros binarios nunca se eliminan de los discos. Sin acción por nuestra parte, los discos terminarán por llenarse. Para evitar este problema, lo más simple es configurar una purga automática de los registros binarios después de N días, con la opción `expire_logs_days`. Por ejemplo, `expire_logs_days = 7` se asegurará de borrar los archivos de más de 7 días.

Encontraremos que el valor correcto para este parámetro requiere un poco de experimentación. Se aconseja tener al menos una semana si es posible, pero, si los discos son pequeños o si escribe mucho en los registros binarios, solo podremos quizás conservar dos o tres días de historia.

También existe el comando `PURGE BINARY LOG`. Este comando permite eliminar todos los registros binarios anteriores al archivo o la fecha proporcionada como parámetro.

Por ejemplo, si el servidor dispone de los siguientes archivos en su índice:

```
mysql> SHOW BINARY LOGS;
+-----+-----+
| Log_name          | File_size |
+-----+-----+
```

log-bin.000001	155
log-bin.000002	155
log-bin.000003	155
log-bin.000004	155
log-bin.000005	107
+-----+	

y solo queremos conservar el archivo `log-bin.000005`, se utilizará el comando:

```
mysql> PURGE BINARY LOGS TO 'log-bin.000005';
```

```
mysql> SHOW BINARY LOGS;
```

+-----+	
Log_name	File_size
+-----+	
log-bin.000005	107
+-----+	

Un segundo ejemplo puede ser la eliminación de la última semana de archivos de registros binarios utilizando el siguiente comando:

```
mysql> PURGE BINARY LOGS BEFORE NOW()-INTERVAL 7 DAY;
```

6. Eliminar la configuración de replicación

Para asegurar que un esclavo anula su configuración de replicación, bastará con ejecutar los siguientes comandos:

```
mysql> STOP SLAVE;
mysql> RESET SLAVE ALL;
```

Y podemos confirmar con `SHOW SLAVE STATUS` que el servidor ya no es un esclavo.

Esta operación es útil, por ejemplo, si queremos promocionar un esclavo a maestro. Para evitar confusiones más tarde, es recomendable ejecutar los comandos anteriores con objeto de que quede bien claro que el servidor ya no es un esclavo.

7. Verificar la coherencia de los datos entre un maestro y un esclavo

Hemos visto antes que algunas situaciones pueden crear incoherencias de datos entre maestro y esclavo, mientras que la replicación no reporta ningún error. Esto puede ser el caso si estamos usando una configuración maestro-maestro en modo activo-activo o si usamos el formato SBR para los registros binarios. Por esta razón, puede ser interesante comprobar de forma periódica que los datos son idénticos en todos los servidores.

La única herramienta utilizable en producción en la actualidad es `pt-table-checksum`, de Percona Toolkit. El interés de esta herramienta es que permite una comparación sin bloqueo de las tablas. Ejecutaremos `pt-table-checksum` en el maestro, por ejemplo de la siguiente manera:

```
# pt-table-checksum --replicate=percona.checksums --max-lag=60  
h=localhost,u=root,p=cont_xyz
```

El comando de arriba verificará la coherencia de los datos entre el maestro y todos sus esclavos. El resultado de la comparación se almacenará en la tabla `percona.checksums`. Como parámetro opcional, el retraso máximo de réplicas autorizado durante la operación será de 60 s.

Cuando la comparación ha terminado, lo que puede tomar varias horas para bases de datos voluminosas, podremos realizar la siguiente petición para determinar qué tablas presentan incoherencias:

```

SELECT db, tbl, SUM(this_cnt) AS total_rows, COUNT(*) AS chunks
FROM percona.checksums
WHERE (
    master_cnt <> this_cnt
    OR master_crc <> this_crc
    OR ISNULL(master_crc) <> ISNULL(this_crc))
GROUP BY db, tbl;

```

La documentación está disponible en la página <https://www.percona.com/doc/percona-toolkit/2.2/pt-table-checksum.html>. Es muy completa, pero no fácil de seguir, ya que la herramienta es compleja y tiene múltiples opciones.

Si `pt-table-checksum` encuentra diferencias, una manera de volver a sincronizar los esclavos es reconstruirlos. Es una solución segura, pero muy lenta si tenemos varios cientos de GB de datos y varios esclavos para reconstruir. Una alternativa es utilizar `pt-table-sync`, que es otra herramienta del Percona Toolkit capaz de generar peticiones que corrigen las incoherencias.

Es posible, por ejemplo, recuperar las órdenes SQL que permiten sincronizar con rapidez una tabla de 20 000 registros con seis líneas diferentes entre el maestro y el esclavo con el siguiente comando:

```

shell > pt-table-sync
--print -d test --sync-to-master localhost
REPLACE INTO `test`.`t1`(`id`, `c`, `pkid`) VALUES (1,
'cyril1', 1);
REPLACE INTO `test`.`t1`(`id`, `c`, `pkid`) VALUES (10,
'cyril10', 10);
REPLACE INTO `test`.`t1`(`id`, `c`, `pkid`) VALUES (43,
'cyril43', 43);
REPLACE INTO `test`.`t1`(`id`, `c`, `pkid`) VALUES (55,
'cyril55', 55);
REPLACE INTO `test`.`t1`(`id`, `c`, `pkid`) VALUES (76,
'cyril76', 76);
REPLACE INTO `test`.`t1`(`id`, `c`, `pkid`) VALUES (123,

```

```
'cyril123', 123);
```

Basta después con ejecutar esas órdenes sobre el maestro para restablecer la sincronización entre el maestro y su esclavo. También se puede solicitar a la herramienta ejecutar de forma directa las peticiones utilizando la opción `--execute` en lugar de `--print`.

8. Algunos comandos útiles

a. SHOW SLAVE STATUS

Este comando ejecutado en un esclavo muestra varios datos interesantes, como:

- La posición actual del maestro en su registro binario (`Master_Log_file` y `READ_Master_Log_Pos`).
- La posición actual del esclavo en los registros binarios de su maestro (`Relay_Master_Log_File` y `Exec_Master_Log_Pos`).
- El último error encontrado (`Last_Error`).
- El estado de los procesos ligeros `SQL_THREAD` y `IO_THREAD` (`Slave_IO_Running` y `Slave_SQL_Running`: YES para arrancar y NO para detener).
- El retraso de replicación estimado en segundos, que se basa en el diferencial entre el timestamp (`date`) del último evento escrito en el relay log y el timestamp de la petición en curso de ejecución (`Seconds_Behind_Master`).

Se recomienda encarecidamente poner el sufijo `\G` a este comando para obtener una visualización en modo columna, como en el siguiente ejemplo:

```
mysql> SHOW SLAVE STATUS\G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: 127.0.0.1
      Master_User: msandbox
      Master_Port: 11605
```

Connect_Retry: 60
Master_Log_File: mysql-bin.000001
Read_Master_Log_Pos: 2728
Relay_Log_File: mysql_sandbox11606-relay-bin.000002
Relay_Log_Pos: 2624
Relay_Master_Log_File: mysql-bin.000001
Slave_IO_Running: Yes
Slave_SQL_Running: No
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 2478
Relay_Log_Space: 3043
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: NULL
Master_SSL_Verify_Server_Cert: No
Last_IO_Errno: 0
Last_IO_Error:
Last_SQL_Errno: 0
Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 1

1 row in set (0.00 sec)

En el ejemplo anterior, el thread SQL fue intencionalmente detenido para observar el desfase en el plano de las diferentes variables de posición. El campo `Seconds_Behind_Master` es NULL, ya que el thread SQL está parado; no hay ninguna petición en curso de ejecución, lo que no permite efectuar el cálculo mencionado más arriba. Esta variable es útil para saber si un esclavo está retrasado con relación a su maestro. Debe saber que existen situaciones en las que esta variable no es muy fiable. Es el caso, por ejemplo, si tenemos varios niveles de replicación: si C es esclavo de B que es a su vez esclavo de A, `SHOW SLAVE STATUS` sobre C reflejará el retraso de replicación de C con respecto a B y no en relación con A. Si está buscando un método fiable, use `pt-heartbeat` del Percona Toolkit.

b. `START/STOP SLAVE {IO_THREAD|SQL_THREAD}`

Este comando, como su nombre indica, se utiliza para iniciar o detener la replicación. Controla el estado de los procesos `SQL_THREAD` y `IO_THREAD`, y es posible arrancarlo o detenerlo de forma independiente de uno y otro.

```
mysql> START SLAVE;
```

```
mysql> SHOW SLAVE STATUS\G
```

```
***** 1. row *****
```

```
....
```

```
Slave_IO_Running: Yes
```

```
Slave_SQL_Running: Yes
```

```
....
```

```
mysql> STOP SLAVE SQL_THREAD;
```

```
mysql> SHOW SLAVE STATUS\G
```

```
***** 1. row *****
```

```
....
```

```
Slave_IO_Running: Yes
```

```
Slave_SQL_Running: No
```

```
....
```



```
mysql> STOP SLAVE IO_THREAD;
```

```
mysql> SHOW SLAVE STATUS\G
```

```
***** 1. row *****
```

```
....
```

```
Slave_IO_Running: No
Slave_SQL_Running: No
```

```
....
```

```
mysql> START SLAVE IO_THREAD;
```

```
mysql> SHOW SLAVE STATUS\G
```

```
***** 1. row *****
```

```
....
```

```
Slave_IO_Running: Yes
Slave_SQL_Running: No
```

```
....
```

```
mysql> START SLAVE SQL_THREAD;
```

```
mysql> SHOW SLAVE STATUS\G
```

```
***** 1. row *****
```

```
....
```

```
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
```

```
....
```

c. RESET MASTER

Ejecutado en el maestro, este comando elimina todos los registros binarios que se encuentran en el archivo `master.index`. Los podemos listar con el comando `SHOW BINARY LOGS`. El comando termina por reposicionar el maestro a la posición 0 en el primer registro binario:

```
mysql> SHOW MASTER STATUS;
```

```
+-----+-----+-----+-----+
```

File	Position	Binlog_Do_DB	Binlog_Ignore_DB
log-bin.000005	107		

```
mysql> SHOW BINARY LOGS;
```

Log_name	File_size
log-bin.000001	155
log-bin.000002	155
log-bin.000003	155
log-bin.000004	155
log-bin.000005	107

```
mysql > RESET MASTER;
```

```
mysql> SHOW MASTER STATUS;
```


File	Position	Binlog_Do_DB	Binlog_Ignore_DB
log-bin.000001	107		

```
mysql> SHOW BINARY LOGS;
```

Log_name	File_size
mysql-bin.000001	107

Replicación y alta disponibilidad

1. Promoción de un esclavo

La replicación es muy útil para garantizar la disponibilidad de servidores MySQL: si el maestro se vuelve inservible (crash software, problema de hardware, por ejemplo), a menudo es mucho más rápido sustituir el maestro por un esclavo en lugar de tratar de reparar el maestro. Las operaciones necesarias para la promoción no son, en teoría, demasiado complejas, pero deberán efectuarse minuciosamente, en particular para no cometer errores en las coordenadas de replicación. Además, estas operaciones suelen ser necesarias solo en caso de crisis, y es bien sabido que, bajo presión, es mucho más fácil  cometer errores. Por eso recomendamos dominar las técnicas explicadas en esta sección y tratar de probarlas, para sentirnos cómodos el día en que las tengamos que usar.

Antes de comenzar la promoción de un esclavo, se aconseja suspender las escrituras en el antiguo maestro. He aquí la lista de los pasos que es preciso realizar:

- Identificar el esclavo que vamos a promover si están disponibles varios esclavos. Basta con utilizar el comando `SHOW SLAVE STATUS` en cada esclavo para ver cuál es el más avanzado y verificar los valores de los contadores `Master_log_file` y `Exec_Master_log_Pos`.
- Activar los registros binarios en el futuro maestro, si no es ya el caso.
- Aumentar la posición de los registros binarios con el comando `SHOW MASTER STATUS` en el futuro maestro (¡se puede utilizar este comando en un esclavo una vez que los registros binarios están activados!).
- Configurar la replicación en el antiguo maestro con el comando `CHANGE MASTER TO`.
- Arrancar la replicación en el antiguo maestro (START SLAVE).

- Detener la replicación en el nuevo maestro (STOP SLAVE).

El procedimiento siguiente funcionará si todos los esclavos han podido recibir todas las peticiones del antiguo maestro antes de que deje de estar disponible. Si un esclavo tiene un retardo en la replicación, solo hay que esperar a que haya recorrido todos sus relay logs antes de volver a configurarla.

El procedimiento se vuelve más complejo si uno de los esclavos no ha recibido todas las peticiones del antiguo maestro. En este caso, hay que encontrar a partir de los relay logs del esclavo la posición correspondiente en los registros binarios del nuevo maestro. Se trata de una operación manual que puede ser especialmente complicada si se encuentran con frecuencia las mismas peticiones. Si no es posible determinar con precisión a partir de qué posición el esclavo puede comenzar la replicación en el nuevo maestro, lo mejor es reconstruir el esclavo a partir de una copia de seguridad.

- ☐ También puede intentar arrancar la replicación con una posición «al azar» y utilizar `pt-table-checksum/pt-table-sync` para resincronizar el esclavo sin tener que reconstruirlo. Esta técnica puede ser práctica si tenemos grandes cantidades de datos, pero no siempre funciona.

2. Automatización de la promoción

Una promoción se vuelve complicada con rapidez en cuanto se tienen más de dos o tres esclavos. En este caso, será muy interesante invertir tiempo en una herramienta que permita automatizar la promoción. En general, una herramienta puede funcionar bien en modo automático (la herramienta detecta por sí misma que el maestro no está disponible y desencadena la promoción de forma automática), o bien en modo semiautomático (la detección de la indisponibilidad es manual, ejecutar una línea de comandos es suficiente para desencadenar la promoción).

Una posibilidad consiste en desarrollar nuestros propios scripts para tener en cuenta las particularidades de nuestro sistema. De esta forma, estamos seguros de dominar las etapas de la promoción. El principal inconveniente es que resulta difícil prever todos los casos.

Otra posibilidad es utilizar una herramienta ya disponible. MHA

(<https://code.google.com/archive/p/mysql-master-ha>) es una de estas herramientas para automatizar el conjunto de estas operaciones. MHA es un proyecto libre y fue diseñado para funcionar incluso en los casos más complejos. Por ejemplo, es capaz de conectarse al maestro para descargar los registros binarios que contienen las peticiones pendientes realizadas por los esclavos.

Las MySQL Utilities incluyen también dos scripts: `mysqlrpladmin` para promociones en modo semiautomático y `mysqlfailover` para promociones en modo automático. La única restricción con respecto a MHA es que estas dos herramientas no funcionan si seleccionamos los identificadores de transacción (GTID, ver más adelante en este capítulo).

Replicación y escalabilidad

1. Escalabilidad en lectura

Uno de los usos más comunes de la replicación es permitir ofrecer a una aplicación varios servidores para efectuar las lecturas. En efecto, si todas las grabaciones deben ejecutarse en el maestro, las lecturas pueden, en teoría, efectuarse indistintamente en el maestro o sobre cualquier esclavo, y en la mayoría de las aplicaciones que utilizan MySQL hay más lecturas que escrituras.

Como ya se ha mencionado, la realidad es diferente: la replicación MySQL es asíncrona, no es posible saber si un esclavo tiene los mismos datos que el maestro. Tenga en cuenta que el retraso de replicación dado por la variable `Seconds_behind_master` en el resultado de `SHOW SLAVE STATUS` no es un indicador fiable, puesto que la granularidad del contador es el segundo.

Si el maestro ejecuta 1 000 grabaciones por segundo, un retraso de 100 ms (que aparecerá como 0 con `Seconds_behind_master`) significa un retraso de 100 peticiones.

Por lo tanto, la aplicación deberá contar sin duda con una lógica lo suficientemente específica como para saber qué lecturas se pueden enviar a un esclavo y qué lecturas deberán enviarse al maestro. Tenemos varias estrategias posibles. Por ejemplo, podemos decidir que algunas lecturas necesitan leer la última versión de los datos (que se realizará en el maestro), mientras que otras no tienen limitaciones en cuanto a la frescura de los datos (se realizarán en un esclavo). También es posible decidir de forma arbitraria que, si el retraso de replicación es de 0 o 1 segundo, cualquier lectura puede enviarse a un esclavo, pero si el retraso es superior, las lecturas deben efectuarse en el maestro.

Sea cual sea su estrategia, necesitará siempre tratar de pensar en el caso de que no esté disponible ningún esclavo: ¿debe la aplicación enviar todas las lecturas al maestro, a riesgo

de saturarlo, o debe bloquear las lecturas, con el riesgo de volver inservible la aplicación?

2. Escalabilidad en escritura

Si su aplicación ejecuta muchas más escrituras que lecturas, la replicación no será una buena solución para que su sistema pueda absorber más peticiones. En efecto, con la replicación MySQL, todas las peticiones se realizan en el maestro y luego se replican sobre los esclavos. En otras palabras, todos los servidores ejecutan todas las peticiones. Añadir esclavos no puede, en ningún caso, disminuir el número de escrituras.

Como se mencionó en la parte sobre la replicación maestro-maestro, una configuración activo-activo, donde los dos servidores reciben las escrituras, no permite tampoco aumentar el número de grabaciones total que el sistema puede absorber. Una vez más, los dos servidores deberán llevar a cabo todas las peticiones, y la replicación no permite aumentar el rendimiento de escritura.

El único mecanismo capaz de garantizar un aumento de las escrituras es el *sharding*, es decir, la separación de sus datos en varias instancias que funcionen de forma independiente. Cada parte de los datos tendrá su propio maestro y sus propios esclavos, y, como ahora tenemos varios maestros, podemos aumentar con facilidad el número total de escrituras aceptadas por el sistema. Construir este tipo de sistema sigue siendo difícil con MySQL, ya que no existe una solución de *sharding* automática (como en MongoDB, por ejemplo). Por lo tanto, habrá que construir todos los elementos y adaptar el código para que siga funcionando.

Pero, a menudo, no es necesario recurrir al *sharding* para mantener hasta cierto punto el aumento de la carga de escritura. Podemos invertir en equipos más potentes (más memoria para minimizar el acceso a disco o discos más rápidos), podemos tratar de optimizar las peticiones de escritura, podemos usar un caché como Redis para escribir con menos frecuencia en MySQL o podemos cambiar la configuración de MySQL para favorecer el rendimiento.

Es difícil saber cuándo un servidor MySQL puede absorber escrituras de una manera general, pero he aquí algunas órdenes de magnitud:

- Por debajo de 1 000 escrituras/s, no deberíamos tener ningún problema ni en el maestro ni en los esclavos.

- Alrededor de 10 000 escrituras/s, el maestro puede seguir, pero los esclavos pueden tener retrasos regulares en la replicación. La replicación paralela puede ser útil.
- A partir de 15 000 registros/s, comenzarán a aparecer muchos problemas. Por ejemplo, habrá que prestar atención al espacio en disco, porque sin duda escribimos varios cientos de GB en los registros binarios cada día.

Funcionalidades avanzadas

1. Identificadores de transacción

A partir de la versión 5.6, podemos configurar MySQL para que cada transacción tenga un identificador único (GTID - *Global Transaction Identifier*). Este identificador es mucho más interesante que las clásicas coordenadas de replicación, ya que es el mismo en todos los servidores involucrados en un sistema de replicación. Hemos visto que la reconstrucción de un esclavo a partir de otro esclavo puede ser difícil por el hecho de que una misma transacción no tiene las mismas coordenadas de replicación en el maestro y en los esclavos (dos esclavos tienen, por otra parte, posiciones diferentes para una misma transacción). Los GTID solucionan este problema.

Para poder utilizar los GTID, es necesario primero activar algunas opciones en cada uno de los servidores:

```
[mysqld]
log-bin
log_slave_updates
gtid_mode = ON
```

Luego hay que reiniciar cada uno de los servidores. La replicación se configura ahora de forma ligeramente diferente:

```
mysql> CHANGE MASTER TO MASTER_HOST = 'ip_master', MASTER_USER =
'mi_usuario', MASTER_PASSWORD = 'mi_cont', MASTER_AUTO_POSITION = 1;
```

Tenga en cuenta que con MySQL 5.6 es obligatorio que reinicie el conjunto de los servidores al mismo tiempo, ya que la replicación no puede funcionar si algunos servidores usan los GTID y otros no. Esta limitación se eliminó con MySQL 5.7. Existe un procedimiento disponible para actualizar la configuración servidor por servidor. Este procedimiento es bastante complejo; lo mejor es consultar la documentación en línea: <https://dev.mysql.com/doc/refman/5.7/en/replication-mode-change-online-enable-gtids.html>

☐ Existe un procedimiento similar para la versión 5.6 con Percona Server.

Otra mejora de MySQL 5.7: no es necesario activar los registros binarios en todos los servidores. Podemos mantener los registros binarios solo en los esclavos susceptibles de ser promovidos a maestros.

La administración de replicación con GTID puede ser desconcertante si no estamos preparados. El principal escollo aparecerá si deseamos ignorar una transacción con `SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1`. En efecto, este comando está prohibido con GTID; obtendremos un error si intentamos ejecutarlo. En realidad, no es posible ignorar una transacción con los GTID.

Sin embargo, siempre puede saltar una transacción con los GTID, pero inyectando una transacción vacía. Imaginemos que desea saltar la transacción cuyo identificador es A-B-C:10. Entonces ejecutaremos los siguientes comandos:

```
mysql> STOP SLAVE;  
mysql> SET GTID_NEXT=' A-B-C:10' ;  
mysql> BEGIN; COMMIT;  
mysql> SET GTID_NEXT=AUTOMATIC;
```

Descargado en: www.detodoprogramacion.org

☐ MariaDB 10 dispone también de esta funcionalidad, pero la implementación es diferente: el código es diferente, los parámetros tienen nombres diferentes. Y,

sobre todo, la replicación con GTID entre MySQL MariaDB no funciona.

2. Replicación paralela

Por defecto, los esclavos únicamente aplican las transacciones del maestro con un solo thread: tan pronto como el maestro tiene un tráfico de escritura importante con varias conexiones que realizan transacciones en paralelo, los esclavos corren el riesgo de no poder seguir el ritmo. Puede aparecer un retraso en la replicación. Para solucionar este problema frecuente, es posible, a partir de MySQL 5.6, utilizar varios threads para ejecutar las transacciones del maestro.

La configuración se hace simplemente especificando `slave_parallel_workers = N`, donde N es el número de threads sobre el esclavo que puedan aplicar las transacciones del maestro.

Con MySQL 5.6, la funcionalidad está limitada: solo las transacciones realizadas en esquemas diferentes pueden hacerse en paralelo. Por lo tanto, si las tablas se encuentran en el mismo esquema, la replicación en paralelo de MySQL 5.6 no será de utilidad alguna.

Con MySQL 5.7, se introdujo una nueva variable (`slave_parallel_type`). El valor por defecto es `DATABASE`, lo que conduce al paralelismo por esquema de MySQL 5.6. Otro valor disponible es `LOGICAL_CLOCK`, que permite hacer paralelas cualquier tipo de transacciones, incluidas las transacciones en las tablas en el mismo esquema.

Preste atención: para poder utilizar el valor `LOGICAL_CLOCK`, los registros binarios deben ser generados por un maestro que funcione con MySQL 5.7. En efecto, la información de `group commit` es la base del procesamiento paralelo y solo está disponible si el maestro utiliza MySQL 5.7.

La replicación paralela puede crear agujeros en la ejecución de las transacciones que se encuentran en los relay logs de los esclavos. Estos agujeros son naturalmente resueltos porque tenemos la garantía de que todas las transacciones de los relay logs son correctamente ejecutadas. Sin embargo, las posiciones legibles con `SHOW SLAVE STATUS` pueden ser engañosas. En efecto, estas posiciones solo indicaban el lugar más reciente en los relay logs para el cual no hay ningún agujero. Es perfectamente posible que puedan haberse ejecutado transacciones adicionales.

Esta característica puede causar problemas si necesitamos conocer la posición exacta de la replicación; por ejemplo, si efectuamos una copia de seguridad. Este problema solo puede eludirse si implementamos los GTID. En efecto, en este caso, el conjunto de identificadores de transacciones ya ejecutados están disponibles.

3. Replicación multifuente

a. Introducción

En circunstancias normales, un esclavo solo puede tener un único maestro. Sin embargo, en algunos casos, esta limitación nos penaliza. Por ejemplo, si usamos el *sharding*, podemos querer configurar un esclavo que contendrá los datos procedentes de todos los maestros para facilitar algunos comandos estadísticos. O podemos necesitar reunir datos procedentes de diversas fuentes en un mismo servidor.

Antes de la versión 5.7, este tipo de configuración era complicado de poner en marcha. La opción más extendida era escribir un script que reconfiguraba la replicación cada N minutos, por ejemplo:

- Durante un minuto, replicar desde el maestro 1.
- Detener la replicación, configurar la replicación desde el maestro 2.
- Durante un minuto, replicar desde el maestro 2.
- Detener la replicación, configurar la replicación desde el maestro 3.
- Durante un minuto, replicar desde el maestro 3.
- Detener la replicación, configurar la replicación desde el maestro 1.
- Repetir el ciclo.

Este script es frágil y poco productivo; se trata más de una forma de eludir el problema que de una verdadera solución.

MySQL introdujo el concepto de canal de replicación, lo que permite a un esclavo recibir las actualizaciones desde varios maestros.

b. Implementación

Para entender cómo configurar la replicación multifuente, imaginemos que tenemos dos

servidores maestros cuyas direcciones IP son 1.2.3.4 y 5.6.7.8. Para que su esclavo pueda replicar las escrituras procedentes de estos dos servidores, he aquí los pasos que debemos seguir:

- Restaurar una copia de seguridad de cada uno de los maestros en el esclavo observando las posiciones correspondientes en los registros binarios de los maestros. En general, cada uno de los maestros operará en diferentes esquemas. Observe que MySQL no propone ninguna resolución de conflicto de los registros. Asegúrese de que los maestros no modifican los mismos datos; si no, la replicación no funcionará con toda probabilidad.
- Activar las dos opciones siguientes en el esclavo:
master_info_repository = TABLE y
relay_log_info_repository = TABLE
- Configurar la replicación desde el maestro 1 utilizando la posición en los registros binarios encontrada en el respaldo del maestro 1.

```
mysql> CHANGE MASTER TO MASTER_HOST= ' 1.2.3.4' , MASTER_USER=' ...' ,  
MASTER_PASSWORD=' ...' , MASTER_LOG_FILE=' ...' , MASTER_LOG_POS=...  
FOR CHANNEL=' maestro1';
```

- Configurar la replicación desde el maestro 2 utilizando la posición en los registros binarios encontrada en el respaldo del maestro 2:

```
mysql> CHANGE MASTER TO MASTER_HOST= ' 5.6.7.8' , MASTER_USER=' ...' ,  
MASTER_PASSWORD=' ...' , MASTER_LOG_FILE=' ...' , MASTER_LOG_POS=...  
FOR CHANNEL=' maestro2';
```

- Reiniciar la replicación para cada uno de los canales de replicación:

```
mysql> START SLAVE FOR CHANNEL=' maestro' ;  
mysql> START SLAVE FOR CHANNEL=' maestro2' ;
```

La utilización de los GTID no es obligatoria sí aconsejable, ya que la gestión de las posiciones puede volverse muy complicada con rapidez.

Cada thread de replicación puede ser detenido o iniciado de forma independiente de los demás. Con las notaciones del ejemplo anterior, podemos, pues, por ejemplo, ejecutar los siguientes comandos:

```
mysql> STOP SLAVE IO_THREAD FOR CHANNEL=' maestro1';  
mysql> STOP SLAVE SQL_THREAD FOR CHANNEL=' maestro2';  
mysql> START SLAVE IO_THREAD FOR CHANNEL=' maestro1';
```

4. Replicación semisíncrona

a. Introducción

La replicación MySQL es normalmente asíncrona, lo que permite al maestro ejecutar las escrituras lo antes posible, sin tener que preocuparse por el rendimiento de los esclavos. Sin embargo, es imposible para los esclavos asegurarse de que no tienen ningún retraso de replicación. Si queremos compartir las lecturas entre maestro y esclavos, este comportamiento puede ser molesto porque a veces necesitamos estar seguros de que un esclavo tiene la última versión de los datos antes de realizar una lectura. El carácter asíncrono de la replicación también puede ser un problema al promover un esclavo: si el maestro tiene transacciones que no se han enviado a ningún esclavo, los datos correspondientes se pierden.

A partir de MySQL 5.5, podemos configurar la replicación en modo semisíncrono para uno o varios esclavos. En este caso, las escrituras en el maestro no serán realmente validadas hasta que hayan alcanzado los esclavos semisíncronos. Esta característica tiene la finalidad de permitir resolver los dos problemas que mencionamos a continuación.

La realidad no es, sin embargo, tan simple. En primer lugar, el esclavo devuelve una respuesta al maestro no al llevarse a cabo la transacción procedente del maestro, sino cuando la transacción se escribe en los relay logs. No hay ninguna garantía de que el retraso

de replicación sea nulo.

El valor era para paliar los riesgos de pérdidas al promover un esclavo: puesto que al menos se garantiza un esclavo que haya recibido todas las transacciones del maestro, no pueden producirse pérdidas de datos en caso de desaparición del maestro. Pero hasta MySQL 5.7, el maestro esperaba una respuesta de los esclavos semisíncronos solo después de haber validado la transacción en InnoDB. El resultado de la transacción es visible por otras conexiones al maestro. Si el maestro desaparece después de haber validado la transacción en InnoDB pero antes de haber recibido una respuesta de los esclavos semisíncronos, los datos se perderán.

b. Implementación

Para implementar una replicación semisíncrona, no se requiere recompilar el código. El código necesario se entrega en forma de librerías que debemos instanciar utilizando el comando `INSTALL PLUGIN`. De esta forma, el maestro añade el plug-in `rpl_semi_sync_master`:

```
mysql>INSTALL PLUGIN rpl_semi_sync_master SONAME  
'semisync_master.so';
```

Y en uno o varios esclavos, se añade el plug-in `rpl_semi_sync_slave`:

```
mysql>INSTALL PLUGIN rpl_semi_sync_slave SONAME  
'semisync_slave.so';
```

Luego, comprobamos que el plug-in esté correctamente instalado y activado.

```
mysql> SHOW PLUGINS\G  
[...]
```

```
***** 17. row *****
```

```
Name: MyISAM
Status: ACTIVE
Type: STORAGE ENGINE
Library: NULL
License: GPL
***** 18. row *****
Name: rpl_semi_sync_master
Status: ACTIVE
Type: REPLICATION
Library: semisync_master.so
License: GPL
```

Por supuesto, el nombre del plug-in y de la librería difiere en el esclavo.

A continuación hay que activar el modo semisíncrono. Podemos hacerlo en línea, sin interrumpir el servicio MySQL en los servidores, pero esto requiere interrumpir puntualmente la replicación si esta se encuentra activa. Para activarla en línea, se modifica la configuración del maestro:

```
mysql> SET GLOBAL rpl_semi_sync_master_enabled=1;
```

Y se modifica la configuración en el esclavo o los esclavos escogidos para el modo semisíncrono:

```
mysql> STOP SLAVE IO_THREAD;
mysql> SET GLOBAL rpl_semi_sync_slave_enabled=1;
mysql> START SLAVE IO_THREAD;
```

No se debe, por el contrario, olvidar cargar estos cambios en los archivos de configuración para evitar perder esta configuración en el próximo reinicio.

Cuando se ejecuta una orden en el maestro, la siguiente petición:


```
mysql> INSERT INTO t1 values(1231);
```

puede comprobar que la replicación se llevó a cabo correctamente, vigilando los contadores MySQL siguientes en el maestro:

- `Rpl_semi_sync_master_no_tx`, que indica el número de transacciones que han fracasado en modo semisíncrono.
- `Rpl_semi_sync_master_yes_tx`, que indica el número de transacciones efectuadas en modo semisíncrono.

Así, en nuestro caso, no se ha producido ningún error:

```
mysql> SHOW GLOBAL STATUS LIKE 'rpl%tx';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Rpl_semi_sync_master_no_tx | 0     |
| Rpl_semi_sync_master_yes_tx | 5     |
+-----+-----+
```

También podemos probar el caso de aparición de un error deteniendo la `IO_THREAD` en el esclavo y repitiendo una orden de inserción. En este caso, la orden ocurre, pero solo se validará al cabo de 10 segundos. El error es visible en el contador

`Rpl_semi_sync_master_no_tx` y la replicación vuelve al modo asíncrono:

```
mysql> INSERT INTO t1 values(1232);

mysql> SHOW GLOBAL STATUS LIKE 'rpl%tx';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Rpl_semi_sync_master_no_tx | 1     |
| Rpl_semi_sync_master_yes_tx | 5     |
+-----+-----+
```

```
mysql> SHOW GLOBAL STATUS LIKE 'Rpl_semi_sync_master_status';
```

Variable_name	Value
Rpl_semi_sync_master_status	OFF

Dado que el esclavo muestra la forma que desea utilizar cuando se conecta a su maestro, es necesario reiniciar la replicación cuando un error se produce y la replicación ha pasado a forma asíncrona para activar el modo semisíncrono.

Por último, con respecto a la supervisión de la replicación semisíncrona, diversas variables de estado están disponibles en el maestro:

- `Rpl_semi_sync_master_clients`: número de esclavos conectados en forma semisíncrona.
- `Rpl_semi_sync_master_no_times`: número de veces que el maestro ha desactivado el modo semisíncrono.
- `Rpl_semi_sync_master_tx_wait_time`: tiempo total en microsegundos que el maestro esperó a que una transacción fuera validada.
- `Rpl_semi_sync_master_tx_waits`: número total de veces que el maestro esperó a que la transacción fuera validada.
- `Rpl_semi_sync_master_wait_sessions`: número de sesiones actuales en espera de respuestas del esclavo.
- `Rpl_semi_sync_master_net_wait_time`: tiempo total en microsegundos durante el cual el maestro espera la respuesta de los esclavos.

c. Novedades con MySQL 5.7

Una de las novedades importantes de MySQL 5.7 para la replicación semisíncrona es la mejora significativa de los rendimientos. Antes, solo podía validarse una transacción a la vez en el maestro cuando se activaba la replicación semisíncrona. Para los servidores cuyo tráfico de escritura es elevado, esta limitación tenía a menudo un gran impacto. Ahora, se

pueden validar varias transacciones de forma simultánea.

- ☐ No confié, sin embargo, en el rendimiento de la replicación asíncrona, ya que para cada transacción, será necesario un tiempo de espera.

La segunda novedad de MySQL 5.7 se refiere a las posibilidades de pérdidas de datos: tenemos una nueva variable, `RPL_semi_sync_master_wait_point`, cuyo valor por defecto es `AFTER_SYNC`. La diferencia con respecto a MySQL 5.5/5.6 es la siguiente: cuando una transacción se valida en el maestro, la espera se hace antes de que la transacción sea validada en InnoDB. Si el maestro desaparece antes de haber alcanzado un esclavo semisíncrono, no habrá pérdida de datos, ya que la transacción no se habrá validado en el maestro. Si queremos conservar el comportamiento de las antiguas versiones, podemos configurar `rpl_semi_sync_master_wait_letra` en `AFTER_COMMIT`.

5. Replicación retrasada

Aunque en general queramos que los esclavos tengan el menor retardo de replicación posible, hay casos donde forzar un retraso en la replicación es útil:

- Es el caso si deseamos probar el comportamiento de la aplicación cuando los esclavos van retrasados, por ejemplo, 10 segundos.
- Un esclavo con retraso también puede ayudarnos a recuperar de forma rápida los datos después de un `DROP TABLE` que se ha llevado a cabo por error: en lugar de reconstruir toda la base a partir de una copia de seguridad, podemos simplemente avanzar la replicación justo antes del comando `DROP TABLE`, respaldar la tabla y volverla a importar en el maestro. Se trata de una técnica muy útil cuando la base es muy voluminosa; pero cuando la mayoría de las tablas son pequeñas, recuperar una tabla es mucho más rápido que recuperar la base de datos entera. En este caso, desearemos por ejemplo forzar a un retraso de replicación de una hora.

A partir de MySQL 5.6, forzar a un esclavo a mantener un retraso de N segundos se consigue simplemente con el comando `CHANGE MASTER`:

```
mysql> CHANGE MASTER TO MASTER_DELAY = N
```

Antes de MySQL 5.6, la solución más común era utilizar `pt-slave-delay` del Percona Toolkit, por ejemplo:

```
shell> pt-slave-delay --delay=12h u=mi_usuario,p=mi_cont,h=ip_esclavo
```

Particionado

1. Interés y limitaciones

El particionado horizontal permite dividir una tabla en función de los datos que contiene, de forma transparente para el usuario. Este proceso nos puede permitir una gestión más eficaz de los datos almacenados agrupándolos en diferentes lugares (particiones) según determinados criterios. El particionado de una tabla se define en su creación, indicando el tipo y la clave de las particiones, así como otros parámetros, como el número de particiones que se van a generar.



□ Particionado manual: podemos particionar de forma manual una tabla MyISAM utilizando el motor MERGE. Sin embargo, este motor solo es compatible con el motor MyISAM, y no es tan transparente como el particionado y requiere en todos los casos escanear todas las tablas para la lectura de los registros. Este tipo de particionado casi ha desaparecido hoy desde que InnoDB se convirtió en el motor de almacenamiento más corriente.

a. Gestión del incremento de carga

Cuanto más solicitada sea una tabla, mayor el riesgo de sufrir problemas de rendimiento. El particionado puede ayudar a reducir la contención de una tabla distribuyéndola en el conjunto de las particiones, gracias al uso de «partition pruning» (implementado a partir de MySQL 5.6.6). Por ejemplo, durante la ejecución de una petición SELECT sobre una tabla con el motor de almacenamiento MyISAM, solo las particiones necesarias serán bloqueadas en lectura en lugar de la tabla entera, como es el caso con versiones anteriores de MySQL 5.6.6.

El particionado también puede permitir aislar los registros a los que se accede de forma ocasional. Esto permite disminuir el tamaño de los índices a los que se accederá con mayor frecuencia y, como efecto, disminuir también el tamaño de la memoria necesaria para su almacenamiento en caché. Por ejemplo, para un índice de tipo b-tree, esto permite disminuir el número de hojas, así como la profundidad, lo que produce una aceleración de las operaciones de búsqueda, inserción y borrado. Además, los datos relacionados con el índice deben tener una mayor probabilidad de encontrarse en memoria, lo que tiene como consecuencia reducir la cantidad de entradas/salidas que se han de realizar y con ello mejorar los tiempos de ejecución.

Por último, podemos distribuir la carga de entradas/salidas utilizando varios discos. Para esto hay que utilizar un sistema de software o hardware de tipo RAID (1 o 5, por ejemplo), o bien solicitar a MySQL hacerlo, especificando las ubicaciones diferentes para almacenar los datos (cláusula `DATA DIRECTORY`), es decir, los índices (cláusula `DATA DIRECTORY`) igualmente con el motor de almacenamiento MyISAM.

b. Gestión de grandes volúmenes

Con algunos sistemas de archivos, el límite del tamaño máximo de un archivo puede ser un problema. El particionado puede utilizarse para evitar este límite, repartiendo el contenido de una tabla en varios archivos.

c. Partition pruning

El «partition pruning» permite al optimizador evitar analizar las particiones que no contienen los datos necesarios para la ejecución de la petición. Se utiliza en los casos siguientes:

- `columna_particionada = constante`,
- `columna_particionada IN (constante1, constante2, constante3...)`,
- `columna_particionada BETWEEN constante1 AND constante2`,
- `columna_particionada [op] constante` con `op`, que es un operador de la desigualdad (`<`, `>`, `<>`, `<=`, `>=`).
- con las funciones `YEAR()`, `TO_DAYS()`, `TO_SECONDS()` en las columnas de tipo `DATE` y `DATETIME`.

d. Eliminación rápida de un gran volumen de datos

Borrar un gran volumen de datos de una tabla con una orden SQL DELETE es a veces una operación muy pesada. El particionado puede permitir en este caso reducir el conjunto de estas operaciones a la eliminación de archivos y a una modificación de estructura; en otras palabras, a tener que usar un comando DDL (ALTER) en lugar de un comando DML (DELETE).

En el siguiente ejemplo, la tabla t1 contiene cuatro particiones:

```
mysql> SHOW CREATE TABLE t1\G
***** 1. row *****

Table: t1
Create Table:
CREATE TABLE `t1` (
  `valor` tinyint(3) unsigned NOT NULL,
  `cuando` date NOT NULL
)
PARTITION BY RANGE (YEAR(cuando))
(
  PARTITION p0 VALUES LESS THAN (2000),
  PARTITION p1 VALUES LESS THAN (2002),
  PARTITION p2 VALUES LESS THAN (2004),
  PARTITION p3 VALUES LESS THAN MAXVALUE
);

mysql > SELECT count(*),YEAR(cuando) an FROM t1 GROUP BY an ORDER BY
an DESC;
***** 1. row *****
count(*): 12642912
an: 2008
***** 2. row *****
count(*): 20000
an: 2004
***** 3. row *****
count(*): 3670016
an: 2002
```

```
***** 4. row *****
count(*): 524288
      an: 2000
4 rows in set (15,80 sec)
```

La eliminación de todos los datos cuyo año equivale a 2008 se realiza con la petición:

```
mysql> DELETE FROM t1 WHERE YEAR(cuando)=2008;
```

Esta petición no utiliza el «partition pruning» debido a la utilización de la función YEAR () en el campo cuando, como puede verse usando el comando EXPLAIN:

```
mysql> EXPLAIN PARTITIONS SELECT count(*) FROM t1 WHERE
YEAR(cuando)=2008;
+----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys |
key | key_len | ref | rows | Extra |
+----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t1 | p0,p1,p2,p3 | ALL | NULL |
NULL | NULL | NULL | 4214304 | Using where |
+----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
```

Para poder obtener beneficio del «partition pruning», hay que liberarse de la función YEAR () :

```
mysql> EXPLAIN PARTITIONS SELECT count(*) FROM t1 WHERE cuando
between '2008-01-01' and '2008-12-31';
+----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
```


id	select_type	table	partitions	type	possible_keys
1	SIMPLE	t1	p3	ALL	NULL

key | key_len | ref | rows | Extra

NULL | NULL | NULL | 4214304 | Using where

Sin embargo, el comando `DELETE`, incluso con el uso del «partition pruning», tomará tiempo y recursos en el servidor MySQL y tendrá un impacto significativo sobre el rendimiento.

```
mysql> DELETE FROM t1 WHERE cuando BETWEEN '2008-01-01' AND
'2008-12-31';
Query OK, 12642912 rows affected (2 min 5,82 sec)
```

En el caso de operaciones de gran volumen, como las purgas de datos, es importante tener en cuenta el impacto que esto puede tener en el servidor, la replicación, si se utiliza, etc. El particionado, si es pertinente, permite minimizar la carga del servidor usando el comando `ALTER TABLE ... DROP PARTITION`:

```
mysql> ALTER TABLE t1 TRUNCATE PARTITION p3;
Query OK, 0 rows affected (0,00 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

e. Limitaciones

El particionado se encuentra limitado por una serie de restricciones. Este apartado enumera las más importantes. La lista completa está disponible en la documentación en línea:

<http://dev.mysql.com/doc/refman/5.7/en/partitioning-limitations.html>

Excepto los modos de particionado para KEY, COLUMNS RANGE y COLUMNS LIST, la clave o expresión de particionado debe necesariamente devolver un valor de tipo entero o nulo.

Ejemplo

```
CREATE TABLE part1 (  
    id int(11) NOT NULL,  
    dateCreation date NOT NULL  
)  
PARTITION BY KEY(dateCreation)  
    partitions 10  
;
```

Query OK, 0 rows affected (3,59 sec)

```
mysql> CREATE TABLE part2 (  
    id int(11) NOT NULL,  
    dateCreation date NOT NULL  
)  
PARTITION BY HASH(dateCreation)  
    partitions 10  
;
```

ERROR 1659 (HY000): Field 'dateCreation' is of a not allowed type
for this type of partitioning

Todas las particiones de una misma tabla deben utilizar el mismo motor de almacenamiento. No es posible tener una mezcla de motores de almacenamiento en una misma tabla. No podremos, por ejemplo, crear una tabla para la cual algunas particiones usan el motor InnoDB y otras particiones usan el motor Archive.

Ejemplo

```
CREATE TABLE part1 (  
    id int(11) NOT NULL,  
    hired date DEFAULT NULL  
) ENGINE=InnoDB
```

```
PARTITION BY RANGE (YEAR(hired))
(
    PARTITION p0 VALUES LESS THAN (2000) ENGINE = ARCHIVE,
    PARTITION p1 VALUES LESS THAN (2010) ENGINE = InnoDB,
    PARTITION p3 VALUES LESS THAN MAXVALUE ENGINE = InnoDB
)
;
ERROR 1497 (HY000): The mix of handlers in the partitions is not
allowed in this version of MySQL
```

Cada índice único (incluida la clave primaria) de la tabla particionada deberá contener todas las columnas utilizadas en los términos de las claves de particionado.

Ejemplo

```
mysql> CREATE TABLE t (
    id int(11) NOT NULL,
    age tinyint unsigned NOT NULL,
    primary key (id)
)
PARTITION BY KEY(age)
    partitions 10
;
ERROR 1503 (HY000): A PRIMARY KEY must include all columns
in the table's partitioning function
```

El particionamiento es, en particular, incompatible con el uso de claves externas, índices FULLTEXT, columnas espaciales y tablas temporales:

```
CREATE TEMPORARY TABLE tmp_part1 (
    id int(11) NOT NULL,
    dateCreation date NOT NULL
)
PARTITION BY KEY()
```

```

        partitions 10
    ;
ERROR 1562 (HY000): Cannot create temporary table with partitions

```

El número de particiones (incluyendo las subparticiones) se limita a 8192. Para superar este límite, podemos utilizar disparadores (triggers) en una tabla usando el motor de almacenamiento BlackHole (que no almacena ningún dato), que redistribuirá los datos sobre distintas tablas que serán, a su vez, particionadas. El límite será de no más de 8192 particiones, pero de (8192*número de tablas) particiones. Será necesario efectuar los SELECT directamente sobre las tablas finales y no sobre la tabla del tipo BlackHole para leer los registros.

2. Tipos de particiones

Para utilizar el particionado, debemos verificar previamente que esta opción está activada.

```

mysql> SHOW PLUGINS;
+-----+-----+-----+
+-----+-----+
| Name                               | Status   | Type                               |
Library | License |
+-----+-----+-----+
+-----+-----+
...
| ARCHIVE                               | ACTIVE   | STORAGE ENGINE                   |
NULL    | GPL    |
| partition                             | ACTIVE   | STORAGE ENGINE                   |
NULL    | GPL    |
+-----+-----+-----+
+-----+-----+

```

Si la opción no está activada, debe primero recuperar los binarios que tienen el soporte del particionado integrado.

MySQL soporta cuatro tipos de particiones: por intervalo (RANGE), por lista (LIST) y por hash (HASH o KEY).

a. El particionado de tipo RANGE

Esta configuración permite separar físicamente el contenido de una tabla en intervalos de valores. La sintaxis de MySQL para crear una tabla particionada en RANGE es similar a la siguiente:

```
CREATE TABLE table_partition_range (  
  ID INT NOT NULL,  
  fechaNacimiento date NOT NULL  
)  
PARTITION BY RANGE (Expression) (  
  PARTITION nom_particion0 VALUES LESS THAN(1990),  
  PARTITION nom_particion01 VALUES LESS THAN(2000),  
  PARTITION nom_particion02 VALUES LESS THAN(2010),  
  PARTITION nom_particion03 VALUES LESS THAN MAXVALUE  
);
```

La expresión que proporciona la clave de particionado debe regresar un valor entero para que la orden de creación sea válida. En este ejemplo, la expresión puede ser la función YEAR():

```
PARTITION BY RANGE (YEAR(fechaNacimiento))  
(  
  PARTITION p1990 VALUES LESS THAN (1990),  
  PARTITION p2000 VALUES LESS THAN (2000),  
  PARTITION p2010 VALUES LESS THAN (2010),  
  PARTITION pmaxvalue VALUES LESS THAN MAXVALUE  
)  
  
INSERT INTO `table_partition_range` VALUES (1, '0000-00-00'),  
(4, '2000-10-24'), (5, '2001-10-24'), (6, '2003-10-24'), (7, '2007-10-24'),  
(9, '2005-10-24'), (10, '2002-10-24'), (2, '2012-10-24'), (3, '2010-10-24'),
```

```
(8, '2015-10-24');
```

En el disco, en el directorio donde se almacena la tabla `table_partition_range`, se crean los siguientes archivos:

```
$ ls -l
-rw-rw----  1 daz daz  8604 oct.  24 22:22
table_partition_range.frm
-rw-rw----  1 daz daz   92 oct.  24 22:22
table_partition_range.par
-rw-rw----  1 daz daz 114688 oct.  24 22:22
table_partition_range#P#p1990.ibd
-rw-rw----  1 daz daz 114688 oct.  24 22:22
table_partition_range#P#p2000.ibd
-rw-rw----  1 daz daz 114688 oct.  24 22:22
table_partition_range#P#p2010.ibd
-rw-rw----  1 daz daz 114688 oct.  24 22:22
table_partition_range#P#pmaxvalue.ibd
...
```

Para cada una de las particiones, el valor insertado debe ser estrictamente inferior al rango. Por ejemplo, el valor de 2000 se almacena en la tercera partición (P2010). El orden de las particiones es importante. Solo pueden definirse desde el valor más pequeño hasta el valor más grande, más la cláusula opcional `MAXVALUE`, que permite especificar el rango máximo de las particiones de la tabla, que debe especificarse en última posición. En otras palabras, la partición `VALUES LESS THAN MAXVALUE (pmaxvalue)` almacenará todos los valores superiores o iguales a 2010.

La tabla `partitions`, que pertenece al esquema `information_schema`, permite tener información vinculada a las tablas particionadas:

```
mysql> SELECT PARTITION_NAME, PARTITION_ORDINAL_POSITION,
PARTITION_METHOD, PARTITION_EXPRESSION, PARTITION_DESCRIPTION,
```

```

TABLE_ROWS FROM information_schema.partitions WHERE
TABLE_NAME='table_partition_range'\G
***** 1. row *****
      PARTITION_NAME: p1990
PARTITION_ORDINAL_POSITION: 1
      PARTITION_METHOD: RANGE
      PARTITION_EXPRESSION: YEAR(fechaNacimiento)
PARTITION_DESCRIPTION: 1990
      TABLE_ROWS: 1
***** 2. row *****
      PARTITION_NAME: p2000
PARTITION_ORDINAL_POSITION: 2
      PARTITION_METHOD: RANGE
      PARTITION_EXPRESSION: YEAR(fechaNacimiento)
PARTITION_DESCRIPTION: 2000
      TABLE_ROWS: 0
***** 3. row *****
      PARTITION_NAME: p2010
PARTITION_ORDINAL_POSITION: 3
      PARTITION_METHOD: RANGE
      PARTITION_EXPRESSION: YEAR(fechaNacimiento)
PARTITION_DESCRIPTION: 2010
      TABLE_ROWS: 6
***** 4. row *****
      PARTITION_NAME: pmaxvalue
PARTITION_ORDINAL_POSITION: 4
      PARTITION_METHOD: RANGE
      PARTITION_EXPRESSION: YEAR(fechaNacimiento)
PARTITION_DESCRIPTION: MAXVALUE
      TABLE_ROWS: 3

```

La distribución de los 10 registros de la tabla `table_partition_range` es la siguiente:

- uno con p1990.
- seis con p2010.

tres con pmaxvalue.

El particionado RANGE es, en particular, interesante con las peticiones SQL que se aplican sobre un intervalo de valores que se refiere solo a un pequeño porcentaje de particiones, como la siguiente petición:

```
mysql> UPDATE table_partition_range SET col=col+1 WHERE
fechaNacimiento BETWEEN 2000 AND 2009;
```

El comando EXPLAIN PARTITIONS permite saber qué particiones seleccionó el optimizador para poder procesar los datos solicitados.

```
EXPLAIN PARTITIONS UPDATE table_partition_range SET col=col+1 WHERE
fechaNacimiento BETWEEN '2000-01-01' AND '2009-0-01'\G
***** 1. row *****

      id: 1
select_type: SIMPLE
      table: table_partition_range
  partitions: p2010
        type: ALL
possible_keys: NULL
          key: NULL
       key_len: NULL
          ref: NULL
         rows: 6
      Extra: Using where
```

Gracias al particionado de la tabla table_partition_range, MySQL solo escaneará la partición P2010 para poder aplicar los cambios.

b. El particionado de tipo RANGE COLUMNS

A partir de MySQL 5.5, podemos efectuar el particionado de una tabla en función de una o varias columnas, que pueden ser de tipo entero, cadena de caracteres (CHAR, VARCHAR,

BINARY, VARBINARY) o temporal (DATE, DATETIME) con el particionado RANGE COLUMNS. Pero, a diferencia del particionado RANGE, no es posible tener una expresión como criterio de particionado. La sintaxis de MySQL para crear una tabla particionada en RANGE COLUMNS es similar a la siguiente:

```
mysql>CREATE TABLE table_partition_range_columns (  
valor TINYINT UNSIGNED NOT NULL,  
cuando DATE NOT NULL  
)  
PARTITION BY RANGE COLUMNS(valor,cuando) (  
    PARTITION p0 VALUES LESS THAN (10,'2006-10-02'),  
    PARTITION p1 VALUES LESS THAN (10,'2008-04-12'0),  
    PARTITION p2 VALUES LESS THAN (100,MAXVALUE),  
    PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE)  
);
```

Con el particionado RANGE COLUMNS, la comparación durante la asignación de los datos en una partición no solo se hace sobre un valor, sino en el conjunto de los valores de la partición:

```
mysql> CREATE TABLE tpr_multi_col (  
a SMALLINT UNSIGNED NOT NULL,  
b SMALLINT UNSIGNED NOT NULL,  
c SMALLINT UNSIGNED NOT NULL  
)  
PARTITION BY RANGE COLUMNS(a,b) (  
    PARTITION p0 VALUES LESS THAN (120,60),  
    PARTITION p1 VALUES LESS THAN (270,120),  
    PARTITION p2 VALUES LESS THAN (530,MAXVALUE),  
    PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE)  
);
```

Para determinar en qué partición se va a recuperar un registro, MySQL comparará la tupla

(a, b) del valor del par correspondiente a la definición de la primera partición (p0). Si el test es negativo, la tupla se compara con el valor del par correspondiente a la definición de la segunda partición (p1) y así sucesivamente. En otras palabras, la tupla (n1, n2) se almacenará en la primera partición donde la expresión (n1 < 120 OR (n1 = 120 AND n2 < 60)) devuelve verdadero.

Por ejemplo, con el registro (120, 130, 30), el servidor prueba la tupla (120, 130) con la partición p0, es decir, con la pareja (120, 60). La prueba es negativa porque 120 es igual a 120, pero 130 es superior a 60. La tupla es comparada con la siguiente partición y allí la prueba es positiva porque 120 es inferior a 270.

El registro (120, 130, 30) se almacena en la segunda partición (p1).

c. El particionado de tipo LIST

El tipo LIST permite definir una lista de números enteros única para cada partición. MySQL abrirá un error para todos los intentos de inserción de un entero que no está en la lista de valores particionados. Cabe señalar que el valor NULL se considera como un valor diferente de entero.

La sintaxis de MySQL para crear una tabla partitionada por listas es similar a la siguiente:

```
CREATE TABLE table_partition_list (  
  ID INT NOT NULL,  
  valid Tinyint unsigned  
)  
PARTITION BY LIST (Expression) (  
  PARTITION pUndef VALUES IN (NULL),  
  PARTITION pFalso VALUES IN (0),  
  PARTITION pVerdadero VALUES IN (1)  
);
```

La expresión que proporciona la clave de particionado debe retornar un valor entero o NULL. La partición pUndef almacenará todos los registros para los cuales la expresión devolverá un valor NULL, pFalso el valor 0 y pVerdadero el valor 1.

Este tipo de particionado permite optimizar las peticiones SQL que tengan como filtro una

lista de valores:

```
PARTITION BY LIST (valid) (  
PARTITION pUndef VALUES IN (NULL),  
PARTITION pFalso VALUES IN (0),  
PARTITION pVerdadero VALUES IN (1)  
);
```

```
mysql> SELECT valid, count(*) FROM table_partition_list GROUP BY  
valid;
```

+-----+-----+	
valid	count(*)
+-----+-----+	
NULL	700
0	2499
1	880
+-----+-----+	

```
mysql> SELECT PARTITION_NAME, PARTITION_ORDINAL_POSITION,  
PARTITION_METHOD, PARTITION_EXPRESSION, PARTITION_DESCRIPTION,  
TABLE_ROWS FROM information_schema.partitions WHERE  
table_name='table_partition_list'\G
```

```
***** 1. row *****  
PARTITION_NAME: pUndef  
PARTITION_ORDINAL_POSITION: 1  
PARTITION_METHOD: LIST  
PARTITION_EXPRESSION: valid  
PARTITION_DESCRIPTION: NULL  
TABLE_ROWS: 885  
***** 2. row *****  
PARTITION_NAME: pFalso  
PARTITION_ORDINAL_POSITION: 2  
PARTITION_METHOD: LIST  
PARTITION_EXPRESSION: valid  
PARTITION_DESCRIPTION: 0  
TABLE_ROWS: 2785  
***** 3. row *****
```

```
        PARTITION_NAME: pVerdadero
PARTITION_ORDINAL_POSITION: 3
        PARTITION_METHOD: LIST
        PARTITION_EXPRESSION: valid
PARTITION_DESCRIPTION: 1
        TABLE_ROWS: 1035
```

```
mysql> DELETE FROM table_partition_list WHERE valid=1;
Query OK, 880 rows affected (0,00 sec)
```

```
mysql> EXPLAIN PARTITIONS DELETE FROM table_partition_list
WHERE valid=1\G
```

```
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: table_partition_list
    partitions: pVerdadero
         type: ALL
possible_keys: NULL
          key: NULL
       key_len: NULL
         ref: NULL
        rows: 1176
   Extra: Using where
```

Solo la partición pVerdadero se analiza en la ejecución del comando DELETE.

d. El particionado de tipo LIST COLUMNS

También desde MySQL 5.5, es posible el particionado de una tabla en función de una o varias columnas que pueden ser de tipo entero, cadena de caracteres (CHAR, VARCHAR, BINARY, VARBINARY) o temporal (DATE, DATETIME) con el particionado LIST COLUMNS. Sin embargo, a diferencia del particionado LIST, no es posible tener una expresión como criterio de particionado. La sintaxis de MySQL para crear una tabla particionada LIST COLUMNS es similar a la siguiente:

```
mysql> CREATE TABLE table_partition_list_columns (
TypeBalance TINYINT NOT NULL,
NumCuenta SMALLINT UNSIGNED NOT NULL,
quand DATE NOT NULL
)
PARTITION BY LIST COLUMNS(cuando) (
    PARTITION p0 VALUES IN ( '2009-10-02','2008-06-10','2005-02-12'),
    PARTITION p1 VALUES IN ( '2002-04-04')
);
```

Podemos incluir los siguientes registros y comprobar que MySQL realiza el particionado esperado:

```
mysql> INSERT INTO table_partition_list_columns VALUES (1,60210,
'2002-04-04'),(1,61230,'2002-04-04'),(-1,60210,'2009-10-02');
Query OK, 3 rows affected (0,04 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT PARTITION_NAME, PARTITION_ORDINAL_POSITION,
PARTITION_METHOD, PARTITION_EXPRESSION, PARTITION_DESCRIPTION,
TABLE_ROWS FROM information_schema.partitions WHERE
table_name='table_partition_list_columns'\G
***** 1. row *****
    PARTITION_NAME: p0
PARTITION_ORDINAL_POSITION: 1
    PARTITION_METHOD: LIST COLUMNS
    PARTITION_EXPRESSION: `cuando`
PARTITION_DESCRIPTION: '2009-10-02','2008-06-10','2005-02-12'
    TABLE_ROWS: 1
***** 2. row *****
    PARTITION_NAME: p1
PARTITION_ORDINAL_POSITION: 2
    PARTITION_METHOD: LIST COLUMNS
    PARTITION_EXPRESSION: `cuando`
PARTITION_DESCRIPTION: '2002-04-04'
    TABLE_ROWS: 2
```

Como en todos los tipos de particionado, ¡si el registro no corresponde a ninguna partición simplemente no se almacenará y se devolverá un error!

```
mysql> INSERT INTO table_partition_list_columns VALUES (1,64210,  
'2010-12-02');  
ERROR 1526 (HY000): Table has no partition for value from column_list
```

En el caso de varias columnas, se utilizará la sintaxis siguiente:

```
mysql> CREATE TABLE tpl_multi_col1 (  
a SMALLINT UNSIGNED NOT NULL,  
b SMALLINT UNSIGNED NOT NULL,  
c SMALLINT UNSIGNED NOT NULL  
)  
PARTITION BY LIST COLUMNS(a,b) (  
    PARTITION p0 VALUES IN ( (120,60), (10,20), (1,2) ),  
    PARTITION p1 VALUES IN ( (270,120), (42,45), (34,36) ),  
    PARTITION p2 VALUES IN ( (530,236), (5,6) ),  
    PARTITION p3 VALUES IN ( (7,7), (0,0), (1,1) )  
);
```

Para cada registro, la pareja (a , b) se compara a las parejas contenidas en cada una de las particiones p0, p1, p2 y P3 para saber dónde se almacena el registro.

Por ejemplo, el registro (7 , 7 , 10) se almacenará en la partición p3, que contiene el par (7 , 7).

e. El particionado de tipo HASH

El tipo HASH permite distribuir de forma equilibrada los datos de una tabla sobre un número de particiones elegido. Es interesante para las tablas que no contienen criterios de particionado evidente (como un campo fecha, por ejemplo) o para los valores secuenciales,

como los enteros de tipo `AUTO_INCREMENT`. En efecto, permite distribuir los datos de forma homogénea en las diferentes particiones en función de una columna o una expresión.

- La distribución de los datos sobre las particiones es bastante simple, es el módulo (el resto de la división). Si tenemos tres particiones, `p0`, `p1`, `p2`, un registro irá en `p0`, el siguiente en `p1`, el siguiente en `p2`, el siguiente en `p0`, etc.

El cálculo hecho por MySQL es el siguiente:

```
IF(ISNULL(valor_partition), 0, ABS(valor_partition)) MOD  
num_de_particiones
```

La sintaxis de SQL para crear una tabla particionada en HASH es similar a la siguiente:

```
CREATE TABLE table_partition_hash (  
ID INT NOT NULL AUTO_INCREMENT,  
Nom CHAR(35) DEFAULT '',  
PRIMARY KEY (ID)  
)  
PARTITION BY HASH (Expression)  
PARTITIONS 10;
```

La expresión que proporciona la clave de particionado debe retornar un valor entero. Puede ser, por supuesto, el nombre de una columna de tipo entero:

```
mysql> CREATE TABLE table_partition_hash (  
ID INT NOT NULL AUTO_INCREMENT,  
Nom CHAR(35) DEFAULT '',  
PRIMARY KEY (ID)  
)  
PARTITION BY HASH (ID)
```

```
PARTITIONS 10;
```

Los datos serán distribuidos de forma homogénea en las 10 particiones:

```
mysql> SELECT count(*) FROM table_partition_hash;
```

```
+-----+
| count(*) |
+-----+
| 4079      326 |
+-----+
```

```
mysql> SELECT PARTITION_NAME, PARTITION_ORDINAL_POSITION,
PARTITION_METHOD, PARTITION_EXPRESSION, PARTITION_DESCRIPTION,
TABLE_ROWS FROM information_schema.partitions WHERE
table_name='table_partition_hash'\G
```

```
***** 1. row *****
```

```
      PARTITION_NAME: p0
PARTITION_ORDINAL_POSITION: 1
      PARTITION_METHOD: HASH
      PARTITION_EXPRESSION: ID
PARTITION_DESCRIPTION: NULL
      TABLE_ROWS: 407
```

```
***** 2. row *****
```

```
      PARTITION_NAME: p1
PARTITION_ORDINAL_POSITION: 2
      PARTITION_METHOD: HASH
      PARTITION_EXPRESSION: ID
PARTITION_DESCRIPTION: NULL
      TABLE_ROWS: 408
```

```
***** 3. row *****
```

```
...
```

```
***** 9. row *****
```

```
      PARTITION_NAME: p8
PARTITION_ORDINAL_POSITION: 9
```



```

PARTITION_METHOD: HASH
PARTITION_EXPRESSION: ID
PARTITION_DESCRIPTION: NULL
TABLE_ROWS: 408
***** 10. row *****
PARTITION_NAME: p9
PARTITION_ORDINAL_POSITION: 10
PARTITION_METHOD: HASH
PARTITION_EXPRESSION: ID
PARTITION_DESCRIPTION: NULL
TABLE_ROWS: 408

```

El optimizador sabe en qué particiones están los datos:

```

mysql> EXPLAIN PARTITIONS SELECT * FROM table_partition_hash WHERE
ID IN (1,11,21,31)\G

```

```

***** 1. row *****
id: 1
select_type: SIMPLE
table: table_partition_hash
partitions: p1
type: range
possible_keys: PRIMARY
key: PRIMARY
key_len: 4
ref: NULL
rows: 4
Extra: Using where

```

```

mysql> EXPLAIN PARTITIONS SELECT * FROM table_partition_hash WHERE
ID BETWEEN 5 AND 9\G

```

```

***** 1. row *****
id: 1
select_type: SIMPLE
table: table_partition_hash
partitions: p5,p6,p7,p8,p9

```

```
type: range
possible_keys: PRIMARY
  key: PRIMARY
key_len: 4
ref: NULL
rows: 5
Extra: Using where
```

Es importante tener en cuenta que, cuanto menos deba analizar MySQL las particiones (y por ende los datos), más rápida será la petición. El criterio de particionado debe elegirse en función del tipo de peticiones.

He aquí un ejemplo que muestra la utilización de una expresión para particionar los datos por lista:

```
mysql> CREATE TABLE cuenta (
Balance MEDIUMINT UNSIGNED NOT NULL,
NumCuenta INT UNSIGNED primary key
)
PARTITION BY HASH(NumCuenta)
PARTITIONS 10;
```

Ahora es posible recuperar la balanza de la cuenta 123456 recorriendo solo la partición que almacena el registro correspondiente o la partición p6, ya que `SELECT IF(ISNULL(123456), 0, ABS(123456) MOD 10 = 6:`

```
mysql> EXPLAIN PARTITIONS SELECT Balance FROM cuenta WHERE
NumCuenta=123456\G
***** 1. row *****
      id: 1
select_type: SIMPLE
      table: cuenta
  partitions: p6
      type: const
```

```
possible_keys: PRIMARY
    key: PRIMARY
    key_len: 4
    ref: const
    rows: 1
    Extra: NULL
```

El particionado de tipo HASH se basa en el resto de una división; es importante que este valor cambie para obtener una distribución equitativa de todas las particiones. Si en la tabla `cuenta` se almacenan solo los números de cuenta que terminan por el valor 2, solo se rellenará la partición `p2`.

```
mysql> SELECT numCuenta FROM cuenta;
```

```
+-----+
| numCuenta |
+-----+
| 2          |
| 12         |
| 22         |
| 32         |
| 42         |
| 142        |
| 212        |
| 242        |
| 342        |
| 123456     |
+-----+
```

```
10 rows in set (0,00 sec)
```

```
mysql> SELECT concat('p',mod(NumCuenta ,10)) `partition`,
count(*)
FROM cuenta GROUP BY `partition`;
```

```
+-----+-----+
| partition | count(*) |
+-----+-----+
| p2       | 9        |
```

```
| p6          | 1          |  
+-----+-----+
```

f. El particionado de tipo KEY

El particionado por KEY es casi idéntico al particionado de tipo HASH. Las diferencias son las siguientes:

- La clave de particionado facilitada puede estar vacía. En este caso, se elige la clave primaria o el primer índice único y distinto de cero (NOT NULL) si no existe ninguna clave primaria.
- La clave de particionado no debe ser necesariamente un número entero o una expresión que devuelva un entero.
- La función de hash es proporcionada por MySQL. Se basa en el mismo algoritmo que la función de control de las contraseñas, la función PASSWORD () .
- Los datos no necesariamente se distribuirán de forma homogénea en todas las particiones.

La sintaxis de MySQL para crear una tabla particionada por KEY es similar a la siguiente:

```
CREATE TABLE table_partition_key (  
  TextoComando CHAR(20) NOT NULL PRIMARY KEY,  
  Balance SMALLINT UNSIGNED NOT NULL  
)  
PARTITION BY KEY ([lista_de_columnas])  
PARTITIONS 4;
```

Un primer ejemplo puede ser querer particionar una tabla utilizando una clave primaria de tipo cadena de caracteres:

```
mysql> CREATE TABLE table_partition_key (  
  TextoComando CHAR(20) NOT NULL PRIMARY KEY,  
  Balance SMALLINT UNSIGNED NOT NULL
```

```
)  
PARTITION BY KEY ()  
PARTITIONS 4;
```

La tabla `table_partition_key` se compone de cuatro particiones. Estas se definen por el hash de los valores de la columna `TextoComando`, ya que es la clave primaria y ninguna lista de columna se da explícitamente como clave de particionado. Por supuesto, podemos especificar la columna `TextoComando` como clave explícita de particionado. El particionado de la siguiente tabla (`table_partition_key2`) es idéntico a la tabla `table_partition_key`.

```
mysql> CREATE TABLE table_partition_key2 (  
  TextoComando CHAR(20) NOT NULL PRIMARY KEY,  
  Balance SMALLINT UNSIGNED NOT NULL  
)  
PARTITION BY KEY (TextoComando)  
PARTITIONS 4;
```

Esta tabla también puede particionarse sobre la columna `Balance`. En este caso, es imperativo que se suprima la clave primaria en la columna `TextoComando` porque la columna `Balance` no forma parte de la clave primaria. Sin embargo, podemos poner un índice simple sobre `TextoComando`:

```
mysql> CREATE TABLE table_partition_key3 (  
  TextoComando CHAR(20) NOT NULL PRIMARY KEY,  
  Balance SMALLINT UNSIGNED  
)  
PARTITION BY KEY (Balance)  
PARTITIONS 4;  
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in  
the table's partitioning function
```

```
mysql> CREATE TABLE table_partition_key3 (  
  TextoComando CHAR(20) NOT NULL,  
  Balance SMALLINT UNSIGNED,  
  INDEX TextoComando (TextoComando)  
  )  
  PARTITION BY KEY (Balance)  
  PARTITIONS 4;  
Query OK, 0 rows affected (1,29 sec)
```

g. Las variantes LINEAR HASH/KEY

Para el particionado de tipo HASH y KEY, MySQL soporta otro algoritmo hash, el tipo LINEAR:

```
CREATE TABLE nombre_tabla (  
  col type,  
  ...  
  )  
  PARTITION BY LINEAR HASH (Expression) PARTITIONS N  
  );  
  
CREATE TABLE nombre_tabla (  
  col type,  
  ...  
  )  
  PARTITION BY LINEAR KEY (Expression) PARTITIONS N  
  );
```

Al añadir la cláusula LINEAR, MySQL utiliza un algoritmo basado en potencias de 2. Tiene como ventaja hacer más rápidas las operaciones de suma, eliminación, supresión, fusión y división de las particiones, limitando el número de registros/particiones que hay que manipular, pero también el inconveniente de una distribución de datos menos homogénea que el algoritmo convencional.

Es interesante utilizar el modo LINEAR en los casos de gestión de tablas de gran tamaño,

ya que los tiempos de tratamientos relacionados con la gestión de las particiones pueden reducirse de forma sustancial. El algoritmo LINEAR está detallado en la documentación oficial en la dirección: <http://dev.mysql.com/doc/refman/5.6/en/partitioning-linear-hash.html>

En el ejemplo siguiente, vamos a añadir varias particiones en una tabla con 10 millones de registros. En un primer momento esta prueba se realizará con el particionado no LINEAR HASH. Luego se repite la prueba, pero esta vez con un particionado LINEAR HASH.

He aquí la definición y el contenido de las dos tablas que nos servirán a lo largo de la prueba:

```
mysql> CREATE TABLE t10_hash(
        id int(10) unsigned NOT NULL AUTO_INCREMENT PRIMARY KEY,
        valor int(11) DEFAULT NULL
    )
    PARTITION BY HASH (id)
        PARTITIONS 6;

mysql> CREATE TABLE t10_linear_hash(
        id int(10) unsigned NOT NULL AUTO_INCREMENT PRIMARY KEY,
        valor int(11) DEFAULT NULL
    )
    PARTITION BY LINEAR HASH (id)
        PARTITIONS 6;
```

...después de haber insertado unos 10 millones de registros...

```
mysql> SELECT count(*) FROM t10_hash;
+-----+
| count(*) |
+-----+
| 10388608 |
+-----+
```

```
mysql> SELECT count(*) FROM t10_linear_hash;
```

```
+-----+  
| count(*) |  
+-----+  
| 10388608 |  
+-----+
```

Para agregar de forma sucesiva las particiones, serán suficientes los siguientes scripts:

```
$ cat /tmp/alter.sh
```

```
INICIO=$(date +%T)
```

```
J=0
```

```
for i in $(seq 1 20)
```

```
do
```

```
mysql -u mi_usuario -p mi_pass test -e "ALTER TABLE t10_hash ADD  
PARTITION PARTITIONS 1"
```

```
J=`expr $J + 1`
```

```
echo "$J partition(s) done"
```

```
done
```

```
echo "Inicio : " $INICIO
```

```
echo "FIN : " `date +%T`
```

```
shell> cat /tmp/alter_linear.sh
```

```
INICIO=$(date +%T)
```

```
J=0
```

```
for i in $(seq 1 20)
```

```
do
```

```
mysql -u mi_usuario -p mi_pass -e "ALTER TABLE t10_linear_hash ADD  
PARTITION PARTITIONS 1"
```

```
J=`expr $J + 1`
```

```
echo "$J partition(s) done"
```

```
done
```

```
echo "Inicio: " $INICIO
```

```
echo "FIN : " `date +%T`
```


La prueba consiste en añadir una partición 20 veces seguidas en la tabla y mostrar la hora de comienzo y de final de la prueba. El resultado es bastante impresionante, ya que ha tardado más de 6 horas 25 minutos para agregar los 20 particiones sobre la tabla `t10_hash` y solo 9 minutos para la tabla `t10_linear_hash`, es decir, ¡una reducción del 97,6 % del tiempo de procesamiento!

Reconstruir todas las particiones con cada agregar/eliminar/fusionar partición se vuelve muy costosa en tiempo, a medida que el tamaño de las tablas aumenta. Resulta, pues, aconsejable usar el particionado `LINEAR` en estos casos.

Por último, cabe señalar que, si la distribución no equitativa permite mejorar las operaciones de mantenimiento, es también una desventaja en los casos en los que se accede a los registros según una norma uniforme. En efecto, en este caso se accederá con más frecuencia a las particiones que contienen más registros que a las otras, lo que tiene su importancia si se considera que el particionado se eligió para repartir las entradas/salidas en varios ejes (RAID, discos).

h. Selección explícita de una partición

A partir de MySQL 5.6, podemos manipular los datos de una tabla particionada seleccionando de forma explícita la partición o las particiones (o subparticiones) en la petición. Solo los datos almacenados en las particiones seleccionadas se verán impactados durante la ejecución de la petición.

Esta funcionalidad es soportada por las siguientes peticiones:

- `SELECT`
- `INSERT`
- `UPDATE`
- `DELETE`
- `REPLACE`
- `LOAD DATA`
- `LOAD XML`

```
mysql> CREATE TABLE table_partition_range (
```

```

ID INT NOT NULL,
fechaNacimiento date NOT NULL
)
PARTITION nombre_particion BY RANGE (YEAR(fechaNacimiento)) (
PARTITION nombre_particion0 VALUES LESS THAN (1990),
PARTITION nombre_particion1 VALUES LESS THAN (2000),
PARTITION nombre_particion2 VALUES LESS THAN (2010),
PARTITION nombre_particion3 VALUES LESS THAN MAXVALUE
);

```

```
mysql> SELECT * FROM table_partition_range;
```

```

+-----+-----+
| ID | fechaNacimiento |
+-----+-----+
| 1 | 0000-00-00      |
| 4 | 2000-10-24      |
| 5 | 2001-10-24      |
| 6 | 2003-10-24      |
| 7 | 2007-10-24      |
| 9 | 2005-10-24      |
| 10| 2002-10-24      |
| 2 | 2012-10-24      |
| 3 | 2010-10-24      |
| 8 | 2015-10-24      |
+-----+-----+

```

```
10 rows in set (0,35 sec)
```

```
mysql> SELECT * FROM table_partition_range PARTITION (p2000);
Empty set (0,00 sec)
```

```
mysql> SELECT * FROM table_partition_range PARTITION (pmaxvalue);
```

```

+-----+-----+
| ID | fechaNacimiento |
+-----+-----+
| 2 | 2012-10-24      |
| 3 | 2010-10-24      |
| 8 | 2015-10-24      |
+-----+-----+

```

```
3 rows in set (0,00 sec)
```

```
mysql> DELETE FROM table_partition_range PARTITION(pmaxvalue)
WHERE ID=8;
Query OK, 1 row affected (0,30 sec)
```

```
mysql> SELECT * FROM table_partition_range PARTITION (pmaxvalue)
WHERE ID >= 3;
+-----+-----+
| ID | fechaNacimiento |
+-----+-----+
| 3  | 2010-10-24      |
+-----+-----+
1 row in set (0,00 sec)
```

i. Subparticionado

El subparticionado consiste en particionar una partición. Esta operación es posible con MySQL, pero solo para las particiones de tipo RANGE y LIST. Además, la partición no puede ser de tipo HASH o KEY.

La sintaxis SQL para crear una tabla subparticionada es la siguiente:

```
CREATE TABLE nombre_de_tabla (
col type,
...
)
PARTITION BY RANGE | LIST (Expression)
SUBPARTITION BY HASH | KEY (Expression)
SUBPARTITIONS N (
definicion_de_particiones
)
```

La definición de las particiones depende, evidentemente, de los tipos elegidos. He aquí un ejemplo de particionado RANGE, subparticionado para KEY:

```
mysql> CREATE TABLE customer (
  customer_id smallint(5) unsigned NOT NULL AUTO_INCREMENT,
  store_id tinyint(3) unsigned NOT NULL,
  first_name varchar(45) NOT NULL,
  last_name varchar(45) NOT NULL,
  create_date datetime NOT NULL,
  KEY (customer_id)
)
PARTITION BY RANGE (YEAR(create_date))
  SUBPARTITION BY KEY(customer_id)
  SUBPARTITIONS 10 (
    PARTITION p2008 VALUES LESS THAN (2008),
    PARTITION p2010 VALUES LESS THAN (2010),
    PARTITION p2012 VALUES LESS THAN (2012),
    PARTITION pMaxvalue VALUES LESS THAN MAXVALUE
  );
```

La tabla `customer` está compuesta por cuatro particiones de tipo `RANGE` (sobre la columna `create_date`) y cada una de esas particiones está compuesta de 10 particiones de tipo `KEY` (en la columna `customer_id`).

Esta tabla se compone de 40 particiones. Estas últimas son visibles en el disco duro:

```
$ ls -l customer*
-rw-rw---- 1 daz daz 8738 oct. 27 20:45 customer.frm
-rw-rw---- 1 daz daz 856 oct. 27 20:45 customer.par
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#p2008#SP#p2008sp0.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#p2008#SP#p2008sp1.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#p2008#SP#p2008sp2.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#p2008#SP#p2008sp3.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
```

```
customer#P#p2008#SP#p2008sp4.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#p2008#SP#p2008sp5.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#p2008#SP#p2008sp6.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#p2008#SP#p2008sp7.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#p2008#SP#p2008sp8.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#p2008#SP#p2008sp9.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#p2010#SP#p2010sp0.ibd
...
customer#P#p2012#SP#p2012sp9.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#pMaxvalue#SP#pMaxvaluesp0.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#pMaxvalue#SP#pMaxvaluesp1.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#pMaxvalue#SP#pMaxvaluesp2.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#pMaxvalue#SP#pMaxvaluesp3.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:45
customer#P#pMaxvalue#SP#pMaxvaluesp4.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#pMaxvalue#SP#pMaxvaluesp5.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#pMaxvalue#SP#pMaxvaluesp6.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#pMaxvalue#SP#pMaxvaluesp7.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:45
customer#P#pMaxvalue#SP#pMaxvaluesp8.ibd
-rw-rw---- 1 daz daz 114688 oct. 27 20:50
customer#P#pMaxvalue#SP#pMaxvaluesp9.ibd
```

Podemos nombrar las subparticiones, a condición de hacerlo con todas y nombrarlas de

manera única:

```
mysql> CREATE TABLE tsp_rh (
    id INT,
    hora TIME
)
PARTITION BY RANGE( HOUR(hora) )
    SUBPARTITION BY HASH( MINUTE(hora) ) (
        PARTITION p0 VALUES LESS THAN (8) (
            SUBPARTITION sp0,
            SUBPARTITION sp1
        ),
        PARTITION p1 VALUES LESS THAN (16) (
            SUBPARTITION sp2,
            SUBPARTITION sp3
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE
    )
;
ERROR 1064 (42000): Wrong number of subpartitions defined,
mismatch
with previous setting near ')' at line 16
```

La partición p2 debe definirse con dos particiones (subpartition) nombradas de forma explícita.

```
mysql> CREATE TABLE t4_with_subpartitions (id INT, d TIME)
PARTCREATE TABLE tsp_rh (
    id INT,
    hora TIME
)
PARTITION BY RANGE( HOUR(hora) )
    SUBPARTITION BY HASH( MINUTE(hora) ) (
        PARTITION p0 VALUES LESS THAN (8) (
```

```

        SUBPARTITION sp0,
        SUBPARTITION sp1
    ),
    PARTITION p1 VALUES LESS THAN (16) (
        SUBPARTITION sp2,
        SUBPARTITION sp4
    ),
    PARTITION p2 VALUES LESS THAN MAXVALUE (
        SUBPARTITION sp4
        SUBPARTITION sp5
    )
)
)

```

```
;
```

```
ERROR 1517 (HY000): Duplicate partition name sp4
```

```

mysql> CREATE TABLE tsp_rh (
    id INT,
    hora TIME
)
PARTITION BY RANGE( HOUR(hora) )
    SUBPARTITION BY HASH( MINUTE(hora) ) (
        PARTITION p0 VALUES LESS THAN (8) (
            SUBPARTITION sp0,
            SUBPARTITION sp1
        ),
        PARTITION p1 VALUES LESS THAN (16) (
            SUBPARTITION sp2,
            SUBPARTITION sp3
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION sp4,
            SUBPARTITION sp5
        )
    )
)

```

```
;
```

```
Query OK, 0 rows affected (1,78 sec)
```

j. Funciones del particionado

Solo se soportan las siguientes funciones en las expresiones de particionado:

- `abs()`
- `ceiling()`
- `day()`
- `dayofmonth()`
- `dayofweek()`
- `dayofyear()`
- `datediff()`
- `extract()`
- `floor()`
- `hour()`
- `microsecond()`
- `minute()`
- `mod()`
- `month()`
- `quarter()`
- `second()`
- `time_to_sec()`
- `to_days()`
- `to_seconds()`
- `unix_timestamp()`
- `weekday()`
- `year()`
- `yearweek()`

Sin embargo, el optimizador MySQL no soporta el «partition pruning» cuando las únicas funciones utilizadas en las expresiones de particionado son `to_days()`, `year()` y

`to_seconds` (a partir de la versión 5.5). Si se utilizan otras funciones diferentes de estas, el optimizador no podrá utilizar el «partition pruning» para evitar recorrer algunas particiones.

MySQL reacciona de manera diferente, según los tipos de particiones, cuando un registro insertado genera el retorno de un valor nulo en la expresión de particionado. De esta forma, para el tipo `RANGE`, los valores nulos son siempre incluidos en la partición que contiene los valores más bajos. Para el tipo `LIST`, el registro se almacena en la partición asociada a la lista que contiene el valor nulo o devolverá un error si no existe ninguna partición de este tipo. Por último, sobre los tipos `HASH` y `KEY`, el valor cero es tratado como el valor 0 y por lo tanto se almacenará en la misma partición que este valor.

k. Import y export de una partición en una tabla

Una de las novedades de MySQL 5.6 es la posibilidad de convertir una partición en una tabla. Este «truco de magia» es posible gracias a la orden `DDL, ALTER TABLE . . . EXCHANGE PARTITION`, que transferirá, a una tabla creada previamente por el DBA, el contenido de una partición o de una subpartición dentro de esta que deberá tener la misma estructura que la tabla particionada (sin las particiones).

Para que la operación pueda llevarse a cabo, se deben respetar algunas condiciones:

- La tabla que recibe los datos deberá tener la misma estructura que la tabla particionada.
- La tabla que recibe los datos no debe estar particionada.
- Si la tabla receptora ya contiene datos, estos últimos deben ser compatibles con el rango de datos de la partición (o subpartición) cuyos datos serán transferidos.
- La tabla que recibe los datos no debe tener claves externas ni ser remitida por la clave externa de otra tabla.
- Además de los derechos típicos para ejecutar el comando `ALTER TABLE` (`ALTER`, `INSERT`, `CREATE`), la cuenta de usuario utilizado debe tener el derecho `DROP`.
- Si la tabla receptora contiene un campo `AUTO_INCREMENT`, este último se restaurará durante el `ALTER TABLE . . . EXCHANGE PARTITION`.

Un largo ejemplo vale más que un largo discurso; he aquí la tabla `ciudad`, que contiene

4079 registros repartidos en cuatro particiones:

```
mysql> CREATE TABLE ciudad (  
    ID int(11) NOT NULL,  
    Name char(35) NOT NULL DEFAULT '',  
    CountryCode char(3) NOT NULL DEFAULT '',  
    District char(20) NOT NULL DEFAULT '',  
    Population int(11) NOT NULL DEFAULT '0'  
)  
PARTITION BY RANGE (Population) (  
PARTITION p10001 VALUES LESS THAN (10001),  
PARTITION p100001 VALUES LESS THAN (100001),  
PARTITION p1000001 VALUES LESS THAN (1000001),  
PARTITION pMaxvalue VALUES LESS THAN MAXVALUE  
);
```

```
mysql> SELECT count(*) FROM ciudad;
```

```
+-----+  
| count(*) |  
+-----+  
|      4079 |  
+-----+
```

La distribución de los registros es casi la siguiente (al estar la tabla en InnoDB, TABLE_ROWS devuelve una estimación del número de registros de la partición):

```
mysql> SELECT PARTITION_NAME, PARTITION_ORDINAL_POSITION,  
PARTITION_METHOD, PARTITION_EXPRESSION, PARTITION_DESCRIPTION,  
TABLE_ROWS FROM information_schema.partitions WHERE  
table_name='ciudad'\G
```

```
***** 1. row *****
```

```
    PARTITION_NAME: p10001
```

```
PARTITION_ORDINAL_POSITION: 1
```

```
    PARTITION_METHOD: RANGE
```

```
    PARTITION_EXPRESSION: Population
```

```

PARTITION_DESCRIPTION: 10001
TABLE_ROWS: 42
***** 2. row *****
PARTITION_NAME: p100001
PARTITION_ORDINAL_POSITION: 2
PARTITION_METHOD: RANGE
PARTITION_EXPRESSION: Population
PARTITION_DESCRIPTION: 100001
TABLE_ROWS: 437
***** 3. row *****
PARTITION_NAME: p1000001
PARTITION_ORDINAL_POSITION: 3
PARTITION_METHOD: RANGE
PARTITION_EXPRESSION: Population
PARTITION_DESCRIPTION: 1000001
TABLE_ROWS: 3190
***** 4. row *****
PARTITION_NAME: pMaxvalue
PARTITION_ORDINAL_POSITION: 4
PARTITION_METHOD: RANGE
PARTITION_EXPRESSION: Population
PARTITION_DESCRIPTION: MAXVALUE
TABLE_ROWS: 313

```

La tabla ciudad_menos_10000h recuperará los datos de la partición p10001. Debe tener la misma estructura que la tabla particionada, pero sin las particiones.

```

mysql> CREATE TABLE ciudad_menos_10000h LIKE ciudad;
Query OK, 0 rows affected (1,08 sec)

mysql> ALTER TABLE ciudad_menos_10000h REMOVE PARTITIONING;
Query OK, 0 rows affected (0,89 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> SHOW CREATE TABLE ciudad_menos_10000h;
CREATE TABLE `ciudad_menos_10000h` (

```

```

`ID` int(11) NOT NULL,
`Name` char(35) NOT NULL DEFAULT '',
`CountryCode` char(3) NOT NULL DEFAULT '',
`District` char(20) NOT NULL DEFAULT '',
`Population` int(11) NOT NULL DEFAULT '0'
)

```

```
mysql> SELECT count(*) FROM ciudad_menos_10000h;
```

```

+-----+
| count(*) |
+-----+
|         0 |
+-----+

```

El comando ALTER TABLE ... EXCHANGE PARTITION transfiere los datos de la partición a la tabla:

```
mysql> ALTER TABLE ciudad EXCHANGE PARTITION p10001 WITH TABLE
ciudad_menos_10000h;
```

Query OK, 0 rows affected (0,82 sec)

```
mysql> SHOW CREATE TABLE ciudad_menos_10000h;
```

```

CREATE TABLE `ciudad_menos_10000h` (
  `ID` int(11) NOT NULL,
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode` char(3) NOT NULL DEFAULT '',
  `District` char(20) NOT NULL DEFAULT '',
  `Population` int(11) NOT NULL DEFAULT '0'
)

```

```
mysql> SELECT count(*) FROM ciudad_menos_10000h;
```

```

+-----+
| count(*) |
+-----+
|        42 |
+-----+

```

La partición no se ha modificado, solo ha cambiado su contenido. Ahora está vacía:

```
mysql> SHOW CREATE TABLE ciudad;
CREATE TABLE `ciudad` (
  `ID` int(11) NOT NULL,
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode` char(3) NOT NULL DEFAULT '',
  `District` char(20) NOT NULL DEFAULT '',
  `Population` int(11) NOT NULL DEFAULT '0'
)
PARTITION BY RANGE (Population) (
  PARTITION p10001 VALUES LESS THAN (10001),
  PARTITION p100001 VALUES LESS THAN (100001),
  PARTITION p1000001 VALUES LESS THAN (1000001),
  PARTITION pMaxvalue VALUES LESS THAN MAXVALUE
)
```

```
mysql> SELECT count(*) FROM ciudad;
```

```
+-----+
| count(*) |
+-----+
|      4037 |
+-----+
```

```
mysql> SELECT PARTITION_NAME, PARTITION_ORDINAL_POSITION,
PARTITION_METHOD, PARTITION_EXPRESSION, PARTITION_DESCRIPTION,
TABLE_ROWS FROM information_schema.partitions WHERE
table_name='ciudad'\G
```

```
***** 1. row *****
      PARTITION_NAME: p10001
PARTITION_ORDINAL_POSITION: 1
      PARTITION_METHOD: RANGE
      PARTITION_EXPRESSION: Population
      PARTITION_DESCRIPTION: 10001
```

```

TABLE_ROWS: 0
***** 2. row *****
PARTITION_NAME: p100001
PARTITION_ORDINAL_POSITION: 2
PARTITION_METHOD: RANGE
PARTITION_EXPRESSION: Population
PARTITION_DESCRIPTION: 100001
TABLE_ROWS: 508
***** 3. row *****
PARTITION_NAME: p1000001
PARTITION_ORDINAL_POSITION: 3
PARTITION_METHOD: RANGE
PARTITION_EXPRESSION: Population
PARTITION_DESCRIPTION: 1000001
TABLE_ROWS: 3190
***** 4. row *****
PARTITION_NAME: pMaxvalue
PARTITION_ORDINAL_POSITION: 4
PARTITION_METHOD: RANGE
PARTITION_EXPRESSION: Population
PARTITION_DESCRIPTION: MAXVALUE
TABLE_ROWS: 313

```

3. Gestión de particionado

La adición, eliminación, fusión, explotación o la redefinición de una partición se efectúan con el comando `ALTER TABLE`.

Con los particionados tipo `[LINEAR] HASH/KEY`, podemos eliminar las particiones fusionando su contenido con las que van a sobrevivir. Por ejemplo, en una tabla que contiene 10 particiones, es posible no tener más que 6, migrando el contenido de las 10 particiones a las 6 restantes.

```

mysql> SHOW CREATE TABLE t10_key\G
***** 1. row *****

```

```
Table: t10_key
Create Table: CREATE TABLE `t10_key` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `valor` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
)
PARTITION BY KEY (id)
PARTITIONS 10
```

```
mysql>ALTER TABLE t10_key COALESCE PARTITION 4;
Query OK, 10388608 rows affected (7 min 30,14 sec)
Records: 10388608 Duplicates: 0 Warnings: 0
```

```
mysql> SHOW CREATE TABLE t10_key\G
***** 1. row *****

Table: t10_key
Create Table: CREATE TABLE `t10_key` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `valor` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
)
PARTITION BY KEY (id)
PARTITIONS 6
```

No obstante, no es posible eliminar por completo el comando COALESCE sin que esto equivalga a eliminar el particionado de la tabla:

```
mysql> ALTER TABLE t10_key COALESCE PARTITION 6;
ERROR 1508 (HY000): Cannot remove all partitions, use DROP TABLE
instead
```

En este último caso, existe el comando ALTER TABLE REMOVE PARTITIONING, que reconstruye la tabla eliminando el particionado.

```
mysql> ALTER TABLE t10_key REMOVE PARTITIONING;
Query OK, 10388608 rows affected (2 min 32,60 sec)
Records: 10388608 Duplicates: 0 Warnings: 0
```

Para añadir particiones en una tabla particionada, hay que utilizar la cláusula `ADD PARTITION` especificando su número:

```
mysql> ALTER TABLE t10_linear_hash ADD PARTITION PARTITIONS 2;
Query OK, 2597152 rows affected (1 min 10,61 sec)
Records: 2597152 Duplicates: 0 Warnings: 0
```

Con los particionados de tipo `[LINEAR] RANGE:LIST`, podemos añadir, suprimir, fusionar y dividir las particiones.

Vamos a estudiar un ejemplo que nos permitirá identificar el conjunto de estas operaciones. Supongamos que disponemos de la siguiente tabla:

```
mysql> SHOW CREATE TABLE t12_list\G
***** 1. row *****

Table: t12_list
Create Table: CREATE TABLE `t12_list` (
  `id` int(10) unsigned NOT NULL DEFAULT '0',
  `valor` tinyint(3) unsigned NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`,`valor`)
)
PARTITION BY LIST (valor) (
PARTITION p0 VALUES IN (1,10,27),
PARTITION p1 VALUES IN (13,45,67),
PARTITION p2 VALUES IN (32,76,22),
PARTITION p3 VALUES IN (11,4,7),
PARTITION p4 VALUES IN (16,5,9)
)
```


Las particiones p3 y P4 pueden fusionarse en una nueva partición p5, sin conservar los datos cuyo valor sea 9.

```
mysql> ALTER TABLE t12_list REORGANIZE PARTITION p3,p4 INTO
(PARTITION p5 VALUES IN(11,4,7,16,5));
Query OK, 7423 rows affected (0,19 sec)
Records: 7423 Duplicates: 1188 Warnings: 0
```

El campo Records especifica el número de registros que se reorganizan y el campo Duplicates indica el número de registros que no se almacenan tras la reorganización. Hay, por lo tanto, 1188 registros cuyo el valor equivalía a 9.

La tabla partitions del esquema information_schema permite obtener información detallada sobre el número de filas contenidas en cada partición y verificar que no se almacena ningún valor 9.

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS FROM
information_schema.partitions WHERE TABLE_NAME='t12_list';
+-----+-----+
| PARTITION_NAME | TABLE_ROWS |
+-----+-----+
| p0              |          41112 |
| p1              |           1212 |
| p2              |              0 |
| p5              |           6235 |
+-----+-----+

mysql> SELECT count(*) FROM t12_list WHERE valor=9;
+-----+
| count(*) |
+-----+
|          0 |
+-----+
```

Para añadir una nueva partición que almacene los registros que contengan el valor 17:

```
mysql> ALTER TABLE t12_list ADD PARTITION (PARTITION p6 VALUES  
IN(17));  
Query OK, 0 rows affected (0,02 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

Para eliminar la partición p1 con objeto de dejar de almacenar los valores 13,45,67.

```
mysql> ALTER TABLE t12_list DROP PARTITION p1;  
Query OK, 0 rows affected (0,01 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

Para dividir la partición p2, que no contiene aquí ningún registro, en dos particiones p7 y p8:

```
mysql> ALTER TABLE t12_list REORGANIZE PARTITION p2 INTO  
(PARTITION p7 VALUES IN(32), PARTITION p8 VALUES IN (76,22));  
Query OK, 0 rows affected (0.01 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

Cada una de estas operaciones realiza una modificación de estructura y plantea bloqueos exclusivos sobre las tablas. Efectuar estas operaciones en producción requiere, por lo tanto, pruebas, una preparación y una planificación esenciales a fin de tener el menor impacto posible en las aplicaciones y los usuarios que acceden a estas tablas.

4. Mantenimiento

Todas las operaciones de mantenimiento que se aplican a las tablas pueden aplicarse a las particiones y, al igual que para las tablas diferentes, las operaciones de mantenimiento en

una partición de una tabla no interfieren (excepto en lo que respecta al consumo de recursos, pero esto es propio del servidor) sobre las otras particiones de la misma tabla.

Las operaciones de mantenimiento se realizan utilizando el comando SQL ALTER TABLE y pueden aplicarse a una, varias o todas las particiones de una misma tabla en el mismo comando. Estas permiten defragmentar la tabla, recuperar espacio, recalcular las estadísticas, buscar errores o corregirlos.

He aquí la descripción de estos diferentes comandos:


- ALTER TABLE t1 REBUILD PARTITION p1, p3: reconstruye las particiones p1 y p3 de la tabla t1, eliminando los registros y volviéndolos a insertar. La partición no se vuelve a crear y el espacio no utilizado no se recupera.
- ALTER TABLE t1 OPTIMIZE PARTITION p2: elimina la partición p2, la vuelve a crear, inserta los antiguos registros y recalcula las estadísticas.
- ALTER TABLE t1 CHECK PARTITION ALL: busca posibles problemas de corrupción en los datos e índices de todas las particiones de la tabla t1.
- ALTER TABLE t2 REPAIR PARTITION p4: corrige la corrupción de datos e índices que hayan podido ser propagados por el comando CHECK PARTITION.
- ALTER TABLE t2 ANALYZE PARTITION p0, p2: actualiza las estadísticas respecto a la distribución de los valores indexados de las particiones p0 y p2 con el fin de mejorar los planes de ejecución que determine el optimizador MySQL.

Rutinas almacenadas

1. Rol

Las rutinas almacenadas son programas (procedimientos o funciones) creados por el usuario, precompilados y almacenados en el servidor MySQL. Permiten mover una parte de la lógica de negocio de una aplicación del cliente al servidor. El cliente no tiene la necesidad de volver a introducir todo el comando, sino solo hacer una simple referencia a la rutina.

Las rutinas almacenadas pueden tener varios usos:

-  Mejorar la seguridad: los programas cliente ya no acceden de forma directa a las tablas. En una API de este tipo, todas las operaciones de gestión de datos se realizan a través de rutinas almacenadas, lo que limita los privilegios de ejecución, sin dar acceso a las tablas que albergan la información.
- Centralizar las peticiones: distintas aplicaciones (que pueden utilizar lenguajes de programación diferentes) pueden acceder a los mismos datos y tener las mismas funcionalidades, lo que permite incluir el código SQL común e implica una disminución de la redundancia y mayor facilidad de mantenimiento del código.
- Aumentar el rendimiento: los comandos no tienen que analizarse varias veces y mucho menos la información enviada por la red, lo que permite limitar el tráfico y solicitar solo el servidor MySQL para determinadas operaciones.

El lenguaje para programar una rutina es bastante rudimentario, pero es posible utilizar:

- Las peticiones SQL (INSERT, UPDATE, CREATE...).
- Las variables definidas utilizando las palabras clave DECLARE y SET.
- Los operadores (=, AND, LIKE...) y las funciones nativas como el SQL (CEIL,

- Las funciones de control (IF, CASE, REPEAT, LOOP...).
- Los cursores que permiten recorrer las filas de salida de una petición SQL para realizar bucles de proceso.

Durante la creación de un programa almacenado (procedimiento, función, trigger...), es imprescindible modificar el delimitador, es decir, el símbolo que indica al intérprete que el comando ha terminado y que lo puede interpretar. Por defecto, el delimitador de MySQL es el carácter punto y coma «;», el mismo que se utiliza en el programa almacenado para definir el final de cada instrucción. En caso de que el delimitador no se cambie, la creación del programa almacenado fracasará en el primer punto y coma.

```
mysql> CREATE PROCEDURE `PS_verif_login`( IN p_mail varchar(50),  
IN p_contraseña varchar (45))  
-> BEGIN  
-> declare nb_user int;  
ERROR 1064 (42000): You have an error in your SQL syntax; check the  
manual that corresponds to your MySQL server version for the right  
syntax to use near '' at line 3
```

En el proceso de creación, debemos, antes de escribir el código del programa, definir el delimitador utilizando la palabra clave DELIMITER seguido del símbolo que elijamos. Al final del programa, es preciso poner el DELIMITER por defecto «;»:

```
mysql> DELIMITER //
```



```
mysql> CREATE PROCEDURE `PS_verif_login`( IN p_mail varchar(50), IN  
p_contraseña (45))  
BEGIN  
declare nb_user int;  
...  
END//
```

```
mysql> DELIMITER ;
```

2. Sintaxis

a. Procedimientos almacenados

Un procedimiento es una rutina que no tiene ningún valor de retorno; sin embargo, puede mostrar resultados utilizando peticiones `SELECT`. Para crearlo, hay que tener el permiso `CREATE ROUTINE`. La sintaxis es la siguiente:

```
CREATE
[DEFINER = { user | current_USER }]
PROCEDURE nombre_procedimiento([Lista de parámetros[...]]).
[Características]
[BEGIN] cuerpo [END]
```

La cláusula `DEFINER` (`[DEFINER = { user | current_USER }]`) asigna un creador al procedimiento almacenado. Se añadió en MySQL 5.0.20. El valor por defecto es `user`, es decir, el usuario que crea el procedimiento.

La lista de parámetros debe respetar el formato: `[IN | OUT | INOUT] nombre_parámetro type...`

Cada parámetro es un parámetro de tipo `IN` por defecto. Para especificar otro tipo, utilice las palabras `OUT` o `INOUT` antes del nombre del parámetro.

El tipo del parámetro corresponde a cualquier tipo de datos MySQL.

Las características se definen de esta manera:

```
LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
```

```
| SQL SECURITY { DEFINER | INVOKER }  
| COMMENT 'string'
```

- `LANGUAGE SQL`: este parámetro es ignorado por el servidor. En el futuro, podría estar disponible un framework para desarrollar procedimientos almacenados externos.
- `[NOT] DETERMINISTIC`: un procedimiento se considera como «determinista» si regresa siempre el mismo resultado para los mismos parámetros de entrada. De lo contrario, se considerará como «no determinista», el valor por defecto.
- `{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }`: proporciona información sobre la naturaleza de los datos manipulados por el procedimiento almacenado; a título indicativo:
 - `CONTAINS SQL` indica que la rutina contiene instrucciones SQL que no manejan datos procedentes de la base de datos.
 - `NO SQL` indica que la rutina no contiene instrucciones SQL.
 - `READS SQL DATA` indica que la rutina contiene instrucciones SQL que acceden en lectura a los datos de la base de datos (`SELECT`).
 - `MODIFIES SQL DATA` indica que la rutina contiene instrucciones SQL que modifican los datos de la base de datos (`INSERT`, `DELETE`, `UPDATE`).
- `SQL SECURITY`: este parámetro puede utilizarse para especificar si la rutina debe ejecutarse con los permisos del usuario que la creó (`DEFINER`) o con aquellos de quien llama la función (`INVOKER`). El valor por defecto es `DEFINER`.
- `COMMENT`: este parámetro permite añadir un comentario al procedimiento almacenado. Este comentario puede mostrarse con el comando `MySQL SHOW CREATE PROCEDURE`.

El cuerpo corresponde al comando o a los comandos SQL válidos.

He aquí algunos ejemplos de procedimientos almacenados:

```
mysql> DELIMITER //
```

```
mysql> DROP PROCEDURE IF EXISTS test.PS_actor//
```

```
Query OK, 0 rows affected (0,00 sec)
```

```
mysql> CREATE PROCEDURE test.PS_actor (IN fname char(30),  
IN lname char(30))
```

```
BEGIN
```

```
    SELECT * FROM test.actor WHERE first_name=fname AND  
last_name=lname;
```

```
END//
```

```
Query OK, 0 rows affected (0,05 sec)
```

```
mysql> DELIMITER ;
```

El procedimiento toma dos argumentos, debe llamarse con parámetros:

```
mysql> CALL test.PS_actor();
```

```
ERROR 1318 (42000): Incorrect number of arguments for PROCEDURE  
test.PS_actor; expected 2, got 0
```

```
mysql> CALL test.PS_actor('NICK', 'WAHLBERG')\G
```

```
***** 1. row *****
```

```
actor_id: 2
```

```
first_name: NICK
```

```
last_name: WAHLBERG
```

```
last_update: 2006-02-15 04:34:33
```

```
1 row in set (0,00 sec)
```

```
Query OK, 0 rows affected (0,00 sec)
```

Para recuperar información, hay que proporcionar parámetros modificables, ya que los procedimientos no devuelven un valor de retorno:


```
mysql> DELIMITER //
```

```
mysql> DROP PROCEDURE IF EXISTS PS_concatId//
```

```
Query OK, 0 rows affected, (0,00 sec)
```

```
mysql> CREATE PROCEDURE PS_concatId (IN fname char(20),OUT c  
char(100))
```

```
BEGIN
```

```
        SELECT GROUP_CONCAT(actor_id) INTO c FROM sakila.actor WHERE  
first_name=fname;
```

```
END//
```

```
Query OK, 0 rows affected (0,00 sec)
```

```
mysql> DELIMITER ;
```

```
mysql> CALL PS_concatId('NICK',@c);
```

```
Query OK, 1 row affected (0,05 sec)
```

```
mysql> SELECT @c;
```

```
+-----+
```

```
| @c      |
```

```
+-----+
```

```
| 2,44,166 |
```

```
+-----+
```

```
1 row in set (0,00 sec)
```

También puede utilizase una variable de tipo INOUT:

```
mysql> DELIMITER //
```

```
mysql> DROP PROCEDURE IF EXISTS PS_concatId//
```

```
Query OK, 0 rows affected (0,07 sec)
```

```
mysql> CREATE PROCEDURE PS_concatId (INOUT c char(20))
```

```
BEGIN
```

```
SELECT GROUP_CONCAT(actor_id) INTO c FROM sakila.actor WHERE
first_name=c;
END//
Query OK, 0 rows affected (0,00 sec)
```

```
mysql> DELIMITER ;
```

```
mysql> SELECT @c:='NICK';
```

```
+-----+
```

```
| @c:='NICK' |
```

```
+-----+
```

```
| NICK      |
```

```
+-----+
```

```
1 row in set (0,00 sec)
```

```
mysql> CALL PS_concatId(@c);
```

```
Query OK, 0 rows affected (0,00 sec)
```

```
mysql> SELECT @c;
```

```
+-----+
```

```
| @c      |
```

```
+-----+
```

```
| 2,44,166 |
```

```
+-----+
```

```
1 row in set (0,00 sec)
```

Los bloques BEGIN . . . END no son obligatorios si el procedimiento solo tiene una instrucción. El procedimiento PS_concatId puede escribirse según la siguiente sintaxis:

```
mysql> DROP PROCEDURE IF EXISTS PS_concatId;
```

```
Query OK, 0 rows affected (0,41 sec)
```

```
mysql> DELIMITER //
```

```
mysql> CREATE PROCEDURE PS_concatId (INOUT c char(20))
      SELECT GROUP_CONCAT(actor_id) INTO c FROM sakila.actor WHERE
first_name=c;//
Query OK, 0 rows affected (0,00 sec)

mysql> DELIMITER ;
```

He aquí un ejemplo más complejo que permite eliminar la última semana del archivo de registros binarios (binlog) utilizando un HANDLER de condiciones o excepciones. Se usa para modificar la variable done cuando el cursor llega al final de los resultados de la petición sobre la tabla `information_schema.TABLES`.

```
mysql> DELIMITER //

DROP PROCEDURE IF EXISTS test.proc1//

CREATE PROCEDURE proc1()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE e CHAR(25);
    DECLARE c INT DEFAULT 10;
    DECLARE cur CURSOR FOR SELECT engine, count(*) tables
FROM information_schema.TABLES GROUP BY engine;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;

    REPEAT
        FETCH cur INTO e,c;
        IF NOT done THEN
            SELECT e,c;
        END IF;
    UNTIL done END REPEAT;

    CLOSE cur;

END
```

```
//
```

```
DELIMITER ;
```

```
mysql> SHOW CREATE PROCEDURE test.proc1\G
```

```
***** 1. row *****
```

```
Procedure: proc1
```

```
sql_mode:
```

```
Create Procedure: CREATE DEFINER=`olivier`@`localhost`
```

```
PROCEDURE `proc1`()
```

```
BEGIN
```

```
DECLARE done INT DEFAULT 0;
```

```
DECLARE e CHAR(25); Descargado en: www.detodoprogramacion.org
```

```
DECLARE c INT DEFAULT 10;
```

```
DECLARE cur CURSOR FOR SELECT engine, count(*) tables
```

```
FROM information_schema.TABLES GROUP BY engine;
```

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
```

```
OPEN cur;
```

```
REPEAT
```

```
    FETCH cur INTO e,c;
```

```
    IF NOT done THEN
```

```
        SELECT e,c;
```

```
    END IF;
```

```
UNTIL done END REPEAT;
```

```
CLOSE cur;
```

```
END
```

```
character_set_client: utf8
```

```
collation_connection: utf8_general_ci
```

```
Database Collation: utf8_swedish_ci
```

b. Funciones almacenadas

Una función es una rutina almacenada que permite obtener un resultado de retorno al igual que las funciones internas como `RAND()`, `UPPER()`, `NOW()`, etc.

Las cláusulas de una función almacenada son prácticamente idénticas a las de un procedimiento almacenado. Acepta también parámetros, pero el tipo no puede especificarse, ya que solo puede ser de tipo IN:

```
mysql> DELIMITER //
```



```
mysql> DROP FUNCTION IF EXISTS confirmation//
```



```
mysql> CREATE FUNCTION confirmation(p_status CHAR(1))
RETURNS VARCHAR(20)
BEGIN
    DECLARE action VARCHAR(20);

    IF p_status = 'O' THEN
        SET action='Continuar';
    ELSEIF p_status = 'N' THEN
        SET action='Stop';
    END IF;

    RETURN(action);
END;
```



```
mysql> DELIMITER ;
```

Una función almacenada se usa de la misma forma que una función clásica:

```
mysql> SELECT confirmation('O');
+-----+
| confirmation('O') |
+-----+
| Continuar        |
+-----+
```

```
mysql> SELECT confirmation('o');
```

```
+-----+
| confirmation('o') |
+-----+
| Continuar          |
+-----+
```

```
mysql> SELECT confirmation('n');
```

```
+-----+
| confirmation('n') |
+-----+
| Stop              |
+-----+
```

En caso de que la letra no se corresponda o que el tipo no sea el correcto, el valor por defecto de la variable `action` será devuelto, en este caso el valor `NULL`, porque no se ha definido:

```
mysql> SELECT confirmation('A');
```

```
+-----+
| confirmation('A') |
+-----+
| NULL              |
+-----+
1 row in set (0,00 sec)
```

```
mysql> SELECT confirmation(42);
```

```
+-----+
| confirmation(42) |
+-----+
| NULL             |
+-----+
1 row in set, 1 warning (0,00 sec)
```

```
mysql> SHOW WARNINGS\G
```

```
***** 1. row *****
```

```
Level: Warning
```

```
Code: 1265
```

```
Message: Data truncated for column 'p_status' at row 1
```

3. Uso

Los procedimientos almacenados pueden servir para proteger determinadas operaciones al no dar, por ejemplo, acceso a tablas utilizadas por el usuario que invoca la rutina. Esto es posible porque MySQL permite especificar, en la definición de la rutina, con qué cuenta se ejecutará. Para ello, debemos usar la palabra clave `SQL SECURITY` e indicar si la ejecución se hará con la cuenta del usuario `DEFINER` (opción por defecto e indicada en la cláusula `DEFINER` precisamente) o del usuario `INVOKER`, es decir, la cuenta de usuario que hace la llamada a la rutina.

Pero he aquí algunos ejemplos que le permitirán comprender mejor el concepto.

Puede, en primer lugar, situar el cursor para saber con qué usuario y sobre qué base de datos está conectado en cualquier momento:

```
mysql> prompt mysql[\u@\d]>  
PROMPT set to 'mysql[\u@\d]>'  
mysql[root@test]>
```

Esta modificación solo es válida durante la sesión en la que el comando `prompt` se ha ejecutado, se recomienda encarecidamente colocar esta variable en el archivo de configuración `my.cnf` (o `my.ini` en MS Windows) en la sección `[client]`.

En el siguiente ejemplo, vamos a crear una tabla llamada `ban` y luego dos procedimientos almacenados que intentarán eliminar la tabla `ban`. Observaremos así la importancia de la cláusula `SQL SECURITY`:

```
mysql[root@test]> CREATE TABLE ban(id int);
```

```
Query OK, 0 rows affected (0,14 sec)
```

```
mysql[root@test]> INSERT INTO ban VALUES(1),(10);
```

```
Query OK, 2 rows affected (0,19 sec)
```

```
Records: 2 Duplicates: 0 Warnings: 0
```

Los dos procedimientos siguientes tienen el mismo funcionamiento, pero DEFINER distintos:

```
mysql[root@test]> DROP PROCEDURE IF EXISTS PS_test1;
```

```
Query OK, 0 rows affected, 1 warning (0,01 sec)
```

```
mysql[root@test]> DROP PROCEDURE IF EXISTS PS_test2;
```

```
Query OK, 0 rows affected, 1 warning (0,01 sec)
```

Cabe señalar que el warning muestra que el intento de eliminación del procedimiento ha fallado, simplemente porque no existe.

```
mysql> SHOW WARNINGS\G
```

```
***** 1. row *****
```

```
Level: Note
```

```
Code: 1305
```

```
Message: PROCEDURE test.PS_test2 does not exist
```

```
mysql[root@test]> DELIMITER //
```

```
mysql[root@test]> CREATE PROCEDURE PS_test1()
```

```
BEGIN
```

```
    SELECT * FROM test.ban;
```

```
END
```

```
//
```

```
Query OK, 0 rows affected (0,00 sec)
```



```

mysql[root@test]> CREATE PROCEDURE PS_test2()
SQL SECURITY INVOKER
BEGIN
    SELECT * FROM test.ban;
END
//

mysql[root@test]> DELIMITER ;

mysql[root@test]> SHOW PROCEDURE STATUS LIKE 'PS_test%'\G
***** 1. row *****
      Db: test
      Name: PS_test1
      Type: PROCEDURE
      Definer: root@localhost
      Modified: 2012-11-06 20:07:13
      Created: 2012-11-06 20:07:13
      Security_type: DEFINER
      Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
      Database Collation: utf8_general_ci
***** 2. row *****
      Db: test
      Name: PS_test2
      Type: PROCEDURE
      Definer: root@localhost
      Modified: 2012-11-06 20:07:13
      Created: 2012-11-06 20:07:13
      Security_type: INVOKER
      Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
      Database Collation: utf8_general_ci

```

Creación de un usuario `dev@localhost` que no tiene permisos de acceso a la tabla `ban`, pero que tiene permisos de ejecución sobre los procedimientos `PS_test1` y

PS_test2.

```
mysql[root@test]> CREATE USER dev@localhost IDENTIFIED BY 'Myp4s5)(';
```

```
mysql[root@test]> GRANT EXECUTE ON PROCEDURE test.PS_test1 TO  
dev@localhost;
```

```
mysql[root@test]> GRANT EXECUTE ON PROCEDURE test.PS_test2 TO  
dev@localhost;
```

```
mysql [localhost] {root} (test) > SHOW GRANTS FOR dev@localhost;
```

```
+-----+  
-----+  
| Grants for dev@localhost  
|  
+-----+  
-----+  
| GRANT USAGE ON *.* TO 'dev'@'localhost' IDENTIFIED BY PASSWORD  
'*B6784E76635EA54DBA72FBA49854AD3BE29C370F' |  
| GRANT EXECUTE ON PROCEDURE `test`.`ps_test2` TO 'dev'@'localhost'  
|  
| GRANT EXECUTE ON PROCEDURE `test`.`ps_test1` TO 'dev'@'localhost'  
|  
+-----+  
-----+
```

Conectándose con la cuenta de usuario dev@localhost, la ejecución de los dos procedimientos muestra el objetivo de la cláusula SQL SECURITY INVOKER:

```
mysql[dev@localhost]> CALL test.PS_test1();
```

```
+-----+  
| id    |  
+-----+  
|     1 |  
|    10 |
```

+-----+

```
mysql[dev@localhost]> CALL test.PS_test2();  
ERROR 1142 (42000): SELECT command denied to user 'dev'@'localhost'  
for table 'ban'
```

El procedimiento `PS_test1` ha sido definido con la cláusula `SQL SECURITY` por defecto: el `DEFINER`, que vale en este ejemplo `root@localhost`, ya que es la cuenta de usuario utilizada para la creación del procedimiento almacenado. Este superusuario tiene derecho de acceso en todas las tablas del servidor, incluida la tabla `ban`. Lo que explica la correcta ejecución del procedimiento `PS_test1`.

En lo que respecta al procedimiento `PS_test2`, es la cuenta del usuario que invoca la llamada al procedimiento, o en este caso `dev@localhost`, la que se utiliza. Sin embargo, este usuario no tiene acceso de lectura a la tabla `ban`. MySQL devuelve el mensaje de error apropiado. Un usuario debe tener los privilegios siguientes:

- `CREATE ROUTINE`, que permite crear rutinas almacenadas.
- `ALTER ROUTINE`, que se requiere para eliminar o modificar las rutinas almacenadas. Este privilegio se añade o retira de forma automática en la cuenta del autor de una rutina durante su creación o eliminación.

4. Metadatos

MySQL pone a disposición información sobre las rutinas almacenadas, como su definición, los privilegios, el nombre de la base de datos asociada, la fecha de creación, etc.

Para acceder a esta información, puede usar los comandos `SHOW PROCEDURE STATUS` y `SHOW FUNCTION STATUS` o bien consultar la tabla `INFORMATION_SCHEMA.ROUTINES`.

Por ejemplo, para obtener el número de procedimientos creados desde hace menos de una semana, usamos el siguiente comando:

```
mysql> SELECT COUNT(*) FROM information_schema.ROUTINES WHERE
CREATED>=NOW()-INTERVAL 1 WEEK AND ROUTINE_TYPE='PROCEDURE';
```

```
+-----+
| COUNT(*) |
+-----+
|         2 |
+-----+
```

Para obtener la información relativa a la función `confirmation()` antes creada, tiene varias posibilidades.

Empleando la tabla `ROUTINES` del esquema `information_schema`, he aquí lo que obtendrá:

```
mysql> SELECT * FROM information_schema.routines WHERE
ROUTINE_TYPE='FUNCTION' AND ROUTINE_NAME='confirmation'\G
```

```
***** 1. row *****
```

```
    SPECIFIC_NAME: confirmation
    ROUTINE_CATALOG: def
    ROUTINE_SCHEMA: test
    ROUTINE_NAME: confirmation
    ROUTINE_TYPE: FUNCTION
    DATA_TYPE: varchar
CHARACTER_MAXIMUM_LENGTH: 20
    CHARACTER_OCTET_LENGTH: 20
    NUMERIC_PRECISION: NULL
    NUMERIC_SCALE: NULL
    DATETIME_PRECISION: NULL
    CHARACTER_SET_NAME: utf8
    COLLATION_NAME: utf8_general_ci
    DTD_IDENTIFIER: varchar(20)
    ROUTINE_BODY: SQL
    ROUTINE_DEFINITION: BEGIN
    DECLARE action VARCHAR(20);
```

```

    IF p_status = '0' THEN
        SET action='Continuar'
    ELSEIF p_status = 'N' THEN
        SET action='Stop';
    END IF;

RETURN(action);
END

EXTERNAL_NAME: NULL
EXTERNAL_LANGUAGE: NULL
PARAMETER_STYLE: SQL
IS_DETERMINISTIC: NO
SQL_DATA_ACCESS: CONTAINS SQL
SQL_PATH: NULL
SECURITY_TYPE: DEFINER
CREATED: 2012-11-06 18:35:00
LAST_ALTERED: 2012-11-06 18:35:00
SQL_MODE: NO_ENGINE_SUBSTITUTION
ROUTINE_COMMENT:
DEFINER: root@localhost
CHARACTER_SET_CLIENT: utf8
COLLATION_CONNECTION: utf8_general_ci
DATABASE_COLLATION: utf8_general_ci
1 row in set (0,00 sec)

```

Empleando el comando `SHOW FUNCTION`, obtendremos esta información:

```

mysql> SHOW FUNCTION STATUS LIKE 'confirmation'\G
***** 1. row *****
      Db: test
     Name: confirmation
    Type: FUNCTION
   Definer: root@localhost
 Modified: 2012-11-06 18:35:00
    Created: 2012-11-06 18:35:00

```

```
Security_type: DEFINER
Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: utf8_general_ci
```

Empleando el comando SHOW CREATE FUNCTION, obtendremos el código de la función:

```
mysql> SHOW CREATE FUNCTION test.confirmation\G
***** 1. row *****
      Function: confirmation
      sql_mode: NO_ENGINE_SUBSTITUTION
Create Function: CREATE DEFINER=`root`@`localhost` FUNCTION
`confirmation`(p_status CHAR(1)) RETURNS varchar(20) CHARSET utf8
BEGIN
    DECLARE action VARCHAR(20);

    IF p_status = 'O' THEN
        SET action='Continuar';
    ELSEIF p_status = 'N' THEN
        SET action='Stop';
    END IF;

    RETURN(action);
END
character_set_client: utf8
collation_connection: utf8_general_ci
Database Collation: utf8_general_ci
```

5. Restricciones

He aquí algunos de los comandos prohibidos en las rutinas almacenadas:

- El (des)bloqueo de tablas (LOCK/UNLOCK TABLES).
- La modificación de vistas (ALTER VIEW).
- La carga de datos en una tabla con los comandos LOAD DATA/TABLE.
- Los comandos preparados en lo que respecta a las funciones almacenadas y los triggers (PREPARE, EXECUTE, DEALLOCATE PREPARE).

La lista completa está disponible en la documentación oficial de MySQL:

<http://dev.mysql.com/doc/refman/5.7/en/stored-program-restrictions.html>

Disparadores (triggers)

1. Rol

Los disparadores o triggers son objetos cuyo objetivo es ejecutar el código en respuesta a un evento que ocurre en una tabla. Los eventos pueden ser de tres tipos: INSERT, UPDATE o DELETE (o similares, por ejemplo: la instrucción REPLACE equivalente a INSERT, o INSERT y DELETE). La orden de activación se define antes (BEFORE) o después (AFTER) del evento. Por ejemplo, el DBA seleccionará BEFORE para un test de comprobación de datos y AFTER para el registro.



2. Sintaxis

La sintaxis que se debe observar durante la creación de un trigger es:

```
CREATE  
[DEFINER = { user | CURRENT_USER }]  
TRIGGER nombre_del_disparador momento_de_activación  
acción_ejecutada  
ON nombre_de_tabla FOR EACH ROW cuerpo_del_disparador
```

Con momento_de_activación que vale BEFORE o AFTER y acción_ejecutada que toma los valores INSERT, UPDATE o DELETE.

En el cuerpo del disparador, los alias OLD y NEW están disponibles en función del contexto (INSERT/UPDATE/DELETE). Permiten acceder, de forma respectiva, al valor de una

columna de la tabla que contiene el disparador, antes y después de la modificación:

- Durante un UPDATE, OLD.Columna hace referencia al valor de la columna antes de la modificación, mientras que NEW.Columna, al valor después de su modificación.
- Durante un INSERT, solo existe NEW.Columna.
- Durante un DELETE, solo existe OLD.Columna.

Ejemplo de un disparador que protege los datos borrados de la tabla City:

```
CREATE TABLE City (  
  id int(11) NOT NULL DEFAULT '0',  
  name char(35) NOT NULL DEFAULT ''  
) ENGINE=InnoDB
```

```
CREATE TABLE City_del (  
  id int(11) NOT NULL DEFAULT '0',  
  name char(35) NOT NULL DEFAULT ''  
) ENGINE=InnoDB
```

```
mysql> SELECT count(*) FROM City;
```

```
+-----+  
| count(*) |  
+-----+  
|      4079 |  
+-----+
```

```
mysql> SELECT count(*) FROM City_del;
```

```
+-----+  
| count(*) |  
+-----+  
|         0 |  
+-----+
```

```
DELIMITER //
```

```
CREATE TRIGGER papelera BEFORE DELETE ON City FOR EACH ROW
```

```

BEGIN
    INSERT INTO City_del (id, name) VALUES (OLD.id, OLD.name);
END //

DELIMITER ;

```

```

mysql> DELETE FROM City WHERE id = 1;
Query OK, 1 row affected (0,15 sec)

```

```

mysql> SELECT * FROM City_del;
+-----+-----+
| id | name |
+-----+-----+
| 1 | Kabul |
+-----+-----+

```

En caso de fallo de un disparador, si las tablas implicadas son transaccionales (como con InnoDB), todas las modificaciones se anularán. En caso contrario (como con MyISAM), las operaciones no continuarán, pero todas las modificaciones hechas hasta el fallo del trigger se mantienen.

Ejemplo de un trigger que protege algunos registros insertados en la tabla `journal`.

```

mysql> CREATE TABLE journal (
    ts timestamp,
    server
enum('host01','host02','host03','host04','host05','host06','host07',
'host08','host09','host10','host11','host12') DEFAULT NULL,
    program
enum('web','bd','appli1','appli2','appli3','appli4','appli5')
DEFAULT NULL,
    msg varchar(500) DEFAULT NULL
) ENGINE=InnoDB;

mysql> CREATE TABLE logs_history (

```

```

        ts timestamp,
        server
enum('host01','host02','host03','host04','host05','host06',
'host07',
'host08','host09','host10','host11','host12') DEFAULT NULL,
        msg varchar(500) DEFAULT NULL
) ENGINE=InnoDB;

```

```
mysql> DELIMITER //
```

```

mysql> CREATE TRIGGER history_log BEFORE INSERT ON journal FOR
EACH ROW
BEGIN
    IF (SELECT FIELD(NEW.program,'web','bd','appli3')) THEN
        INSERT INTO logs_history(server,msg) VALUES
(NEW.`server`,NEW.msg);
    END IF;
END //

```

```
mysql> DELIMITER ;
```

```

mysql> INSERT INTO journal (server,program,msg) VALUES
('host11','appli3','message 4');
Query OK, 1 row affected (0,04 sec)

```

```

mysql> SELECT * FROM logs_history;
+-----+-----+-----+
| ts                | server | msg          |
+-----+-----+-----+
| 2012-11-07 11:56:11 | host11 | message 4    |
+-----+-----+-----+

```

```

mysql> INSERT INTO journal (server,program,msg) VALUES
('host11','appli2','message 3');
Query OK, 1 row affected (0,04 sec)

```

```
mysql> SELECT * FROM logs_history;
```

ts	server	msg
2012-11-07 11:56:11	host11	message 4

Es posible utilizar en el cuerpo de los disparadores las mismas funciones que en una rutina almacenada para, como en el ejemplo, filtrar algunos elementos para el histórico de modificaciones. No existe ninguna instrucción para hacer fallar un disparador. Si debemos invalidar los datos, podemos hacerlo añadiendo una columna que servirá para informar al inicio del trigger los registros que se deben excluir. Luego, habrá que pensar de forma periódica en purgar estos registros o utilizar una tarea programada.

3. Restricciones

De manera general, las restricciones que se aplican a las rutinas almacenadas se aplican también a los disparadores.

Una importante limitación se eliminó con MySQL 5.7. Antes, no era posible crear más de un disparador para una misma acción y en la misma posición (BEFORE o AFTER). Con MySQL 5.7, también podemos especificar el orden de ejecución de los disparadores si varios están definidos para la misma acción.

El disparador se ejecuta para cada registro y no se puede activar una sola vez cuando se modifican varias filas (un concepto de disparador global como con otros SGBD). Tenga cuidado con las escrituras que modifican muchas filas en una tabla con un disparador: el riesgo de problemas de rendimiento es alto.

No es posible llamar a un procedimiento que devuelve registros al disparador.

No está permitido poner disparadores en las tablas del sistema del esquema `mysql`.

Los disparadores no son ejecutados por las acciones de claves externas.

Eventos

1. Rol

El planificador de eventos o «event scheduler» ofrece la posibilidad al administrador de bases de datos de iniciar la ejecución de programas almacenados directamente en el servidor MySQL. Este planificador de tareas interno (CRON-like) permite automatizar de forma sencilla las tareas a intervalos regulares, o a una hora fija, sin necesidad de configurar el sistema operativo, que alberga la base de datos. Para poder utilizarlo, debemos, en primer lugar, activarlo, ya que no es el caso por defecto:



```
mysql> SHOW VARIABLES LIKE 'event_scheduler';
```

+-----+	
Variable_name	Value
+-----+	
event_scheduler	OFF
+-----+	

```
mysql> SET GLOBAL event_scheduler = 1;
```

```
mysql> SHOW VARIABLES LIKE 'event_scheduler';
```

+-----+	
Variable_name	Value
+-----+	
event_scheduler	ON
+-----+	

Una vez activado, MySQL inicia un proceso ligero (thread) en segundo plano. Este proceso

se encarga de ejecutar los eventos cuando llega el momento. Puede comprobarlo utilizando el comando `SHOW PROCESSLIST`, para lo que debemos contar con el privilegio `SUPER`; en caso contrario, solo veremos los procesos que nos están asociados.

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
      Id: 12
     User: event_scheduler
    Host: localhost
       db: NULL
Command: Daemon
      Time: 58
   State: Waiting on empty queue
      Info: NULL
```

2. Sintaxis

La sintaxis para crear un evento es la siguiente:

```
CREATE
[DEFINER = { user | CURRENT_USER }]
EVENT [IF NOT EXISTS] nom_evento
ON SCHEDULE moment
[ON COMPLETION [NOT] PRESERVE]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'comentario']
DO
    cuerpo_del_evento;

moment:
AT timestamp [+ INTERVAL intervalo] ...
| EVERY interval
[STARTS timestamp [+ INTERVAL intervalo] ...]
```

```
[ENDS timestamp [+ INTERVAL intervalo] ...]
```

El nombre del evento es sensible a mayúsculas y minúsculas. Los parámetros disponibles son:

- `DEFINER` : el creador del evento.
- `IF NOT EXISTS`: elimina el evento si existe antes de crear el nuevo.
- `ON COMPLETION [NOT] PRESERVE`: inhibe la eliminación del evento cuando este expira.
- `ENABLE | DISABLE | DISABLE ON SLAVE`: activa/desactiva un evento. Preste atención, ¡un evento desactivado se borrará al reiniciar el servidor MySQL!
- `COMMENT`: añadir un comentario, lo que es, por lo demás, una excelente idea.
- `AT`: indica la hora en que un evento de tipo único debe ejecutarse (el tipo único se deduce de esta cláusula).
- `EVERY`: indica la frecuencia de ejecución del evento (en este caso, es de tipo recurrente).
- `STARTS`: indica la fecha de inicio del evento.
- `ENDS`: indica la fecha de parada del evento (en este caso, ha expirado: será eliminado de la base de datos si la opción `ON COMPLETION PRESERVE` no se utiliza).

Los eventos tienen las mismas limitaciones que las rutinas almacenadas con respecto a las instrucciones que podemos usar en el cuerpo de la definición.

He aquí un ejemplo para que el planificador de eventos permita purgar de forma regular una tabla con objeto de solo conservar un mes de datos y copiar los datos purgados en una tabla de respaldo:

```
mysql> CREATE TABLE bills (  
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
    d DATETIME NOT NULL,  
    price SMALLINT UNSIGNED NOT NULL,
```

```

        KEY idx_d (d)
    ) ENGINE=INNODB;

mysql> CREATE TABLE bills_archive LIKE bills;

mysql> DELIMITER //

mysql> CREATE EVENT archive_bills
ON SCHEDULE EVERY 1 MONTH STARTS '2012-11-01 03:00:00'
DO
BEGIN
    /* Copia de las facturas fechadas a más de un mes en
    la tabla bills_archive */
    INSERT INTO bills_archive(d,price) SELECT d,price FROM bills
    WHERE d<NOW()-INTERVAL 1 MONTH;
    /* Eliminación de las facturas de la tabla bills */
    DELETE FROM bills WHERE d<NOW()-INTERVAL 1 MONTH;
END
//

mysql> DELIMITER ;

```

El evento `archive_bills` se ejecutará el primer día del mes a las 3:00 de la mañana y se ocupará de archivar las facturas que datan de más de un mes copiándolas de la tabla `bills` a la tabla `bills_archive` y borrándolas luego de la tabla `bills`.

Antes del paso del evento:

```

mysql> SELECT count(*) FROM bills;
+-----+
| count(*) |
+-----+
|   300000 |
+-----+

```



```
mysql> SELECT count(*) FROM bills_archive;
+-----+
| count(*) |
+-----+
|         0 |
+-----+
```

Tras el primer paso de la tarea programada:

```
mysql> SELECT count(*) FROM bills;
+-----+
| count(*) |
+-----+
|  178513  |
+-----+

mysql> SELECT count(*) FROM bills_archive;
+-----+
| count(*) |
+-----+
|  121487  |
+-----+
```

Por convención, para modificar un evento, hay que utilizar el comando `ALTER EVENT`. Para eliminarlo, usamos el comando `DROP EVENT [IF EXISTS]`:

```
mysql> DELIMITER //

mysql> DROP EVENT IF EXISTS purge_bills_with_low_price //
Query OK, 0 rows affected, 1 warning (0,00 sec)

mysql> CREATE EVENT purge_bills_with_low_price
ON SCHEDULE EVERY 1 WEEK STARTS '2012-11-01 01:30:00'
```

```

DO
    /* eliminación de las facturas cuyo importe es inferior
a 100 EUR */
    DELETE FROM bills WHERE d<NOW()-INTERVAL 1 MONTH AND
price<100;
//

mysql> ALTER EVENT archive_bills
ON SCHEDULE EVERY 3 WEEK STARTS CURRENT_TIMESTAMP - INTERVAL
TIME_TO_SEC(TIME(CURRENT_TIMESTAMP())) SECOND + INTERVAL 4 HOUR ENDS
CURRENT_TIMESTAMP + INTERVAL 6 MONTH
DO
BEGIN
    /* Copia de las facturas con fecha anterior a tres semanas
en la tabla bills_archive */
    INSERT INTO bills_archive(d,price) SELECT d,price FROM bills
WHERE d<NOW()-INTERVAL 3 WEEK;
    /* Eliminación de las facturas de la tabla bills */
    DELETE FROM bills WHERE d<NOW()-INTERVAL 3 WEEK;
END
//

```

La visualización de la información de los eventos se hará al leer la tabla `mysql.event`, o bien utilizando el comando `SHOW CREATE EVENT`:

```

mysql> SELECT * FROM mysql.event WHERE name='archive_bills'\G
mysql> SHOW CREATE EVENT archive_bills\G

```

3. Restricciones

De forma general, las restricciones que se aplican a los disparadores se aplicarán también a los eventos.

Por otra parte, el nombre de los eventos es sensible a mayúsculas y minúsculas.

Los comandos DDL están prohibidos si la instrucción `LOCK TABLE` está activa.

No es posible garantizar que dos eventos se ejecuten a la misma hora.

La precisión de la ejecución de un evento es del orden de uno o dos segundos.

Vistas

1. Rol

Las vistas son tablas virtuales creadas a partir de una petición `SELECT`. No almacenan los datos que generan, sino solo la petición que permite crearlas. La petición `SELECT` que genera la vista hace referencia a una o varias tablas. La vista puede ser, por ejemplo, un `join` entre diferentes tablas, la agregación o la extracción de algunas columnas de una tabla. También puede crearse a partir de otra vista.

Las vistas son por lo general de solo lectura y solo permiten leer los datos. Sin embargo, MySQL permite la creación de vistas modificables en ciertas condiciones:

- La petición que genera la vista debe permitir a MySQL localizar la traza del registro que se va a modificar en la tabla o las tablas subyacentes, así como el de todos los valores de cada columna. La petición `SELECT` que crea la vista no debe contener la cláusula `DISTINCT`, `GROUP BY`, `HAVING` ni otras funciones de agregación.
- La cláusula `ALGORITHM` no debe ser de valor `TEMPTABLE`. Luego volveremos sobre este tema.
- La petición no debe acceder a vistas subyacentes no modificables.

Las vistas pueden utilizarse por diferentes razones; estas permiten:

- Comprobar la integridad al restringir el acceso a los datos para mejorar la privacidad con un particionado vertical y/u horizontal para ocultar los campos a los usuarios. Esto permite personalizar la visualización de información según el tipo de usuario.
- Ocultar la complejidad del esquema. La independencia lógica de los datos es útil para dar a los usuarios el acceso a un conjunto de relaciones representadas en la forma de una tabla. Los datos de la vista son entonces campos de distintas tablas agrupadas, o

los resultados de operaciones en estos campos.

- Modificar de forma automática los datos seleccionados (`sum()`, `avg()`, `max()` ...). Esto permite manipular los valores calculados a partir de otros valores del esquema.
- Conservar la estructura de una tabla si debe modificarse. El esquema puede modificarse sin que sea necesario cambiar las peticiones del lado de la aplicación.

☐ Ya que en general una vista sirve para ocultar la complejidad, no ayuda a mejorar el rendimiento. A menudo lo que ocurre es lo contrario: los desarrolladores pueden olvidar con facilidad que los registros de una vista son a veces difíciles de generar y que, por tanto, es preferible utilizarla lo menos posible.

Para crear una vista, el usuario debe tener el permiso `CREATE VIEW`. También es necesario que haya permiso para seleccionar todas las columnas que aparecen en el comando `SELECT` que permite crear la vista. Observe que, si se utiliza la cláusula `REPLACE`, también se requiere el permiso `DROP`.

El permiso `SHOW VIEW` ofrece la posibilidad de ejecutar el comando `SHOW CREATE VIEW`. Este comando permite obtener datos sobre la creación de una vista. Otra manera de obtener esta información es consultar la tabla `VIEWS` del esquema `information_schema`. Esta información es completa solo para las vistas que hemos creado nosotros.

2. Sintaxis

He aquí la sintaxis que debe observarse en la creación de una vista:

```
CREATE [OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { user | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW nombre_de_la_vista [(col1,col2,...)] AS consulta_SELECT
```

[WITH [CASCADED | LOCAL] CHECK OPTION]

Las cláusulas `DEFINER` y `SQL SECURITY` son las mismas que para los procedimientos almacenados.

La cláusula `OR REPLACE` borra una posible vista que ya existiese con el mismo nombre.

`ALGORITHM` es una cláusula no estándar, que toma los valores siguientes:

- `UNDEFINED`: es el valor por defecto. MySQL decide por sí mismo qué algoritmo elegir entre `MERGE` y `TEMPTABLE`.
- `MERGE`: utiliza la petición SQL que se empleó para la creación de la vista como punto de partida. En otras palabras, hacer una petición sobre la vista repite la misma petición sobre las tablas subyacentes (fusión de la vista y de la petición).
- `TEMPTABLE`: utiliza una tabla temporal creada para almacenar (de forma temporal) los resultados. Un punto interesante de este algoritmo es liberar de forma más rápida los bloqueos de las tablas subyacentes. Las otras peticiones se verán menos penalizadas.

Cabe señalar también que una vista con el valor `MERGE` podrá ser modificada, mientras que con el valor `TEMPTABLE` no lo será.

El algoritmo `MERGE` consiste en fusionar la instrucción que llama a la vista con el código de la vista.

```
mysql> CREATE TABLE ta1 (id INT, idg INT, ide INT);

mysql> INSERT INTO ta1 VALUES(1,1,4),(10,2,1),(3,4,2),(3,3,9),(3,5,8);

mysql> CREATE ALGORITHM=MERGE VIEW v1(id,idg) AS SELECT id,idg FROM
ta1 WHERE id<=idg;

mysql> SELECT * FROM v1 WHERE id=3;
+-----+-----+
| id    | idg   |
```

id	idg
3	4
3	3
3	5

La petición `SELECT * FROM v1 WHERE id=3` se modifica de hecho por MySQL y se ejecuta en la forma: `SELECT id, idg FROM ta1 WHERE id=3 AND id<=idg`:

```
mysql> EXPLAIN EXTENDED SELECT * FROM v1 WHERE id=3\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: ta1
         type: ALL
possible_keys: NULL
          key: NULL
       key_len: NULL
         ref: NULL
        rows: 5
   filtered: 100.00
      Extra: Using where
1 row in set, 1 warning (0,00 sec)
```

```
mysql> SHOW WARNINGS \G
***** 1. row *****
      Level: Note
       Code: 1003
  Message: /* select#1 */ select `test`.`ta1`.`id` AS
`id`, `test`.`ta1`.`idg` AS `idg` from `test`.`ta1` where
((`test`.`ta1`.`id` = 3) and (3 <= `test`.`ta1`.`idg`))
```

La vista `v1` puede modificarse:

```
mysql> SELECT TABLE_NAME,VIEW_DEFINITION,IS_UPDATABLE FROM
INFORMATION_SCHEMA.VIEWS WHERE TABLE_NAME LIKE 'v1'\G
***** 1. row *****

TABLE_NAME: v1
VIEW_DEFINITION: select `test`.`ta1`.`id` AS `id`,`test`.`ta1`.`idg`
AS `idg` from `test`.`ta1` where (`test`.`ta1`.`id` <= `test`.`ta1`.`
idg`)
IS_UPDATABLE: YES
```

El tipo MERGE no soporta tampoco que la vista solo retorne valores literales (que no proceden de ninguna tabla):

```
mysql> CREATE ALGORITHM=MERGE VIEW v3 AS SELECT 'Esta vista
no puede estar en MERGE';
Query OK, 0 rows affected, 1 warning (0,09 sec)
```

```
mysql> SHOW WARNINGS\G
***** 1. row *****

Level: Warning
Code: 1354
Message: View merge algorithm can't be used here for now (assumed
undefined algorithm)
```

```
mysql> SHOW CREATE VIEW v3\G
***** 1. row *****

View: v3
Create View: CREATE ALGORITHM=UNDEFINED DEFINER=
`root`@`localhost` SQL SECURITY DEFINER VIEW `v3` AS SELECT
'Esta vista no puede estar en MERGE' AS `esta vista no puede estar
en MERGE`
character_set_client: utf8
collation_connection: utf8_general_ci
```

```
mysql> SELECT * FROM v3\G
***** 1. row *****

Esta vista no puede estar en MERGE: Esta vista no puede estar en MERGE
```



```
1 row in set (0,00 sec)
```

Con el algoritmo TEMPTABLE, se crea una tabla temporal. La petición `SELECT * FROM v2 WHERE id=3` se modifica por MySQL y se ejecuta bajo la forma `SELECT id, idg FROM v2 WHERE id=3`.

```
mysql> CREATE ALGORITHM=TEMPTABLE VIEW v2(id,idg) AS SELECT id,idg
FROM ta1 WHERE id<=idg;
```

```
mysql> SELECT * FROM v2 WHERE id=3;
```

```
+-----+-----+
| id    | idg   |
+-----+-----+
| 3     | 4     |
| 3     | 3     |
| 3     | 5     |
+-----+-----+
```

```
mysql> EXPLAIN EXTENDED SELECT * FROM v2 WHERE id=3\G
```

```
***** 1. row *****
```

```
      id: 1
select_type: PRIMARY
      table: <derived2>
        type: ref
possible_keys: <auto_key0>
          key: <auto_key0>
        key_len: 5
          ref: const
         rows: 0
    filtered: 100.00
      Extra: NULL
```

```
***** 2. row *****
```

```
      id: 2
select_type: DERIVED
      table: ta1
        type: ALL
```

```
possible_keys: NULL
    key: NULL
    key_len: NULL
    ref: NULL
    rows: 5
    filtered: 100.00
    Extra: Using where
2 rows in set, 1 warning (0,00 sec)
```

```
mysql> SHOW WARNINGS \G
```

```
***** 1. row *****
Level: Note
Code: 1003
Message: /* select#1 */ select `v2`.`id` AS `id`,`v2`.`idg` AS `idg`
from `test`.`v2` where (`v2`.`id` = 3)
```

La vista v2 no puede modificarse:

```
mysql> SELECT TABLE_NAME,VIEW_DEFINITION,IS_UPDATABLE FROM
INFORMATION_SCHEMA.VIEWS WHERE TABLE_NAME LIKE 'v2'\G
***** 1. row *****
TABLE_NAME: v2
VIEW_DEFINITION: select `test`.`ta1`.`id` AS
`id`,`test`.`ta1`.`idg` AS `idg` from `test`.`ta1`
where (`test`.`ta1`.`id` <= `test`.`ta1`.`idg`)
IS_UPDATABLE: NO
```

El hecho de tratar de actualizar una vista no editable genera el mensaje de error número 1288:

```
mysql> UPDATE v2 SET idg=idg+10;
ERROR 1288 (HY000): The target Tabla V2 of the update is not updatable
```

La cláusula `WITH [CASCADED|LOCAL] CHECK OPTION` es útil cuando creamos vistas modificables que dependen de otras vistas. Permite indicar a MySQL si queremos, en una modificación, que se verifiquen solo las limitaciones de la vista accedida o también las de las vistas subyacentes.

Un ejemplo con la tabla `t_check` y las vistas `v_check1`, `v_check2` y `v_check3`:

```
mysql> CREATE TABLE t_check(id INT);
Query OK, 0 rows affected (0,01 sec)

mysql> INSERT INTO t_check VALUES (200),(50),(105);
Query OK, 3 rows affected (0,00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> CREATE VIEW v_check1 AS SELECT * FROM t_check WHERE id>105
WITH CHECK OPTION;
Query OK, 0 rows affected (0,00 sec)

mysql> CREATE VIEW v_check2 AS SELECT * FROM v_check1 WHERE id>100
WITH LOCAL CHECK OPTION;
Query OK, 0 rows affected (0,00 sec)

mysql> CREATE VIEW v_check3 AS SELECT * FROM v_check1 WHERE id>100
WITH CASCADED CHECK OPTION;
Query OK, 0 rows affected (0,00 sec)
```

La inserción del registro (101) no es aceptada por la vista `v_check1`, ya que no cumple la limitación `id>105` que se verifica a través de la cláusula `WITH CHECK OPTION`. Sin embargo, esta misma inserción pasa a la vista `v_check2`, donde solo la limitación local `id>100` se verifica a través de la cláusula `WITH LOCAL CHECK OPTION`. Sin embargo, la inserción fracasa en la vista `v_check3`. La cláusula `WITH CASCADED CHECK OPTION` fuerza la verificación de las limitaciones de todas las tablas subyacentes:

```
mysql> INSERT INTO v_check1 VALUES(101);  
ERROR 1369 (HY000): CHECK OPTION failed 'test.v_check1'
```

```
mysql> INSERT INTO v_check2 VALUES(101);  
Query OK, 1 row affected (0,00 sec)
```

```
mysql> INSERT INTO v_check3 VALUES(101);  
ERROR 1369 (HY000): CHECK OPTION failed 'test.v_check3'
```

Por supuesto, la estructura de una vista puede modificarse con el comando `ALTER VIEW` y eliminarse con `DROP VIEW`:

```
DROP VIEW v_check1, v_check2, v_check3;  
Query OK, 0 rows affected (0,00 sec)
```

Columnas generadas

1. Introducción

Las columnas generadas aparecieron con MySQL 5.7. La idea es muy simple: se trata de columnas cuyos valores son derivados de una o varias columnas de la misma tabla. Las columnas generadas pueden indexarse, lo que permite añadir índices funcionales para, por ejemplo:

- Indexar cadenas de caracteres de derecha a izquierda en lugar de izquierda a derecha.
- Indexar solo la fecha para una columna que contenga fecha y hora.

☐ ☐ • Indexar la suma de dos columnas.

MySQL ofrece dos tipos de columnas generadas: las columnas virtuales, cuyos valores se calculan sobre la marcha y nunca se almacenan en disco, y las columnas persistentes, que son columnas convencionales almacenadas en disco pero cuyo valor se calculará a partir de otras columnas.

- ☐ MariaDB también dispone de columnas generadas, pero la implementación es diferente. Por lo tanto, la sintaxis también es diferente y algunas funciones disponibles con MySQL 5.7 no están disponibles con MariaDB.

2. Columnas virtuales

Imaginemos que tenemos una tabla t con una columna que almacena un apellido. Para añadir a nuestra tabla una columna virtual que contenga el nombre de familia al revés, utilizaremos la siguiente sintaxis:

.

```
mysql> ALTER TABLE t ADD nombre_inverso VARCHAR(50) GENERATED ALWAYS  
AS REVERSE(nom) VIRTUAL;
```

Observe que, aunque la adición de una columna virtual se hiciera empleando el comando `ALTER TABLE`, la operación no es en realidad pesada, ya que se trata solo de una modificación de los metadatos de la tabla, pero no de los datos. Esto significa que la adición de una columna virtual será siempre muy rápida, incluso para una tabla de varias decenas o cientos de gigabytes.

- ☐ Este no es, por desgracia, el caso para MariaDB, que reconstruye por completo la tabla: la operación será larga en las tablas de gran volumen.

Una columna virtual puede ser indexada como una columna clásica, usando exactamente la misma sintaxis. Por ejemplo:

```
mysql> ALTER TABLE t ADD INDEX idx_nom_inv (nombre_inverso);
```

También es posible que una columna virtual sea una de las columnas de un índice sobre varias columnas. De esta forma, la siguiente sintaxis será válida:

```
mysql> ALTER TABLE t ADD INDEX idx_comp (edad, nombre_inverso,  
telefono);
```

Un índice sobre una columna virtual (o una combinación de columnas que contienen una columna virtual) no es virtual. Esto significa que el comando `ALTER TABLE` va a escribir esta vez realmente los datos en disco y que el tiempo de ejecución va a depender del tamaño de la tabla. Para las tablas de gran volumen, `pt-online-schema-change` puede ser útil, al igual que para añadir un índice clásico.

3. Columnas persistentes

Las columnas virtuales son tan flexibles que se puede preguntar quizás en qué situaciones se recomienda utilizar columnas persistentes. A decir verdad, las únicas dos situaciones corrientes para las que necesitaremos columnas persistentes son:

- La columna generada es la clave primaria de la tabla. Esta situación no está autorizada con columnas virtuales, es necesario utilizar una columna persistente.
- Los valores de la columna generada son pesados de calcular, por ejemplo, porque implican funciones matemáticas complejas. En este caso, una columna persistente será una caché materializada, y el espacio adicional requerido por la columna persistente se ve rentabilizado por la rapidez del acceso a los valores.

La sintaxis para crear una columna persistente es muy similar a la sintaxis para crear una columna virtual:

```
mysql> ALTER TABLE t ADD nombre_inverso VARCHAR(50) GENERATED ALWAYS  
AS  
REVERSE(nombre) STORED;
```

- ☐ Tenga en cuenta que añadir una columna persistente demanda una reconstrucción de la tabla, a diferencia de lo que sucede con una columna virtual. El comando `ALTER TABLE` será tan largo de ejecutar como el volumen de la tabla.

Soporte JSON

1. El tipo de datos JSON

Un nuevo tipo de datos hizo su aparición con MySQL 5.7: el tipo JSON. Era perfectamente posible almacenar en JSON en las versiones anteriores con columnas de tipo TEXT, pero el nuevo tipo de datos presenta varias ventajas:

- Durante su inserción, los documentos JSON se validan para asegurarse de que su sintaxis es correcta.
- El almacenamiento utiliza un formato binario adaptado a los documentos JSON, el uso del disco es menos pesado.
- Muchas funciones especiales permiten acceder a los elementos de un documento JSON.
- Los elementos de un documento JSON pueden indexarse mediante columnas generadas (más detalles a continuación).

Este nuevo tipo de datos marca el primer paso de una gran evolución de MySQL: en el futuro, MySQL será no solo capaz de interactuar con los datos a través de peticiones SQL, sino también a través de otro protocolo de comunicación inspirado por los sistemas NoSQL. En el momento de escribir este libro, Oracle ha publicado un nuevo plugin llamado Document Store que permite usar MySQL como un verdadero sistema NoSQL y que proporciona un resumen de las funcionalidades que estarán con toda probabilidad disponibles con MySQL 5.8.

He aquí un ejemplo de creación de una tabla con un campo de tipo JSON. Puede ver que la inserción de un document JSON no válido falla:

|


```
mysql> CREATE TABLE personas (details JSON);
mysql> INSERT INTO personas (details) VALUES ("nombre: Juan"); ERROR
3140 (22032): Invalid JSON text: "Invalid value." at position 1 in
value for column 'personas.details'.
mysql> INSERT INTO personas (details) VALUES
('{"id":1,"nombre":"Juan","ciudad":"Paris"}');
```

2. Ejemplo de operaciones con columnas JSON

Existen muchas funciones disponibles para trabajar con las columnas JSON. La lista completa de estas funciones puede consultarse en esta página:

<https://dev.mysql.com/doc/refman/5.7/en/json-functions.html>

Veamos solo algunos ejemplos simples de las posibilidades ofrecidas.

La operación más simple consiste en filtrar sobre un campo específico los documentos JSON. Con la tabla que hemos creado antes, si queremos contar el número de registros cuyo atributo nombre tenga el valor Juan, escribiremos la siguiente petición:

```
mysql> SELECT COUNT(*) FROM personas WHERE details->"$.nombre" =
'Juan';
+-----+
| count(*) |
+-----+
|          1 |
+-----+
```

Descargado en: www.detodoprogramacion.org

Imaginemos ahora que esperábamos recuperar la ciudad de todas las personas cuyo nombre es Juan. Un primer intento podría ser:

```
mysql > SELECT details->"$.ciudad" FROM personas
WHERE details->"$.nombre" = 'Juan';
+-----+
```

```
| details->"$.ciudad" |
+-----+
| "Paris" |
+-----+
```

El nombre de la ciudad se recupera de forma correcta, pero la presentación de comillas puede ser molesta. Puede utilizar la función `JSON_UNQUOTE`:

```
mysql> SELECT JSON_UNQUOTE(details->"$.ciudad") FROM personas
WHERE details->"$.nombre" = 'Juan';
+-----+
| JSON_UNQUOTE(details->"$.ciudad") |
+-----+
| Paris |
+-----+
```

3. Indexación

Gracias a su formato binario específico, el tipo JSON ofrece mejores resultados que una columna `TEXT` para las lecturas. Sin embargo, en una tabla que contenga millones de documentos JSON, las lecturas requerirán un recorrido completo de la tabla si no se encuentra ningún índice y los rendimientos serán mediocres.

Utilizando columnas virtuales, podemos declarar una columna que solo contendrá el valor de un único campo de los documentos. Y ya que podemos indexar columnas virtuales, podemos crear un índice sobre uno o varios campos de nuestros documentos. Por supuesto, esta es la mejor manera de proceder para obtener buenos rendimientos en tablas de gran volumen.

Retomando el ejemplo anterior, si queremos añadir un índice sobre el atributo `nombre`, ejecutaremos los siguientes comandos:

```
mysql> ALTER TABLE personas ADD nom VARCHAR(50) GENERATED ALWAYS AS  
(details->"$.nombre") VIRTUAL;  
mysql> ALTER TABLE personas ADD INDEX idx_nombre (nombre);
```

Introducción

Cuando una aplicación pasa a producción, una nueva tarea atañe al administrador: la de supervisar que la base de datos funciona de forma correcta. Esta responsabilidad implica ser capaz de detectar lo antes posible que se produce un error, para asegurarse de que el sistema esté siempre disponible, y también comprobar que el aumento de la carga de la aplicación es tolerado por la base de datos. En caso de que la aplicación funcione de forma satisfactoria y aumente de forma progresiva en tamaño, es preferible que el administrador sea capaz de saber si la base de datos puede soportar la carga y cuánto tiempo podrá la configuración actual satisfacer las demandas de la aplicación.

La supervisión de la base de datos se articula, por lo tanto, en torno a dos dominios: un sistema de alerta que se activa tan pronto como la base de datos deje de funcionar según lo previsto o deje de funcionar del todo y un sistema de gráfico que permita ver la evolución a lo largo del tiempo de indicadores relacionados con la salud de la base de datos.

Veremos en este capítulo los datos que MySQL pone a nuestra disposición para realizar estas tareas, así como algunas herramientas existentes que nos permitirán ahorrar tiempo.

Acceso a los metadatos

1. Comandos específicos de MySQL

a. Comandos SHOW

Podemos tener acceso a los metadatos, es decir, a la información sobre la forma en que se estructuran en particular las bases de datos, las tablas o columnas, con una serie de comandos específicos que usan la palabra clave SHOW. Entre los comandos SHOW más útiles, encontraremos:

- SHOW SCHEMAS / SHOW DATABASES: lista las bases de datos de la instancia.
- SHOW TABLES: lista las tablas de una base de datos.
- SHOW COLUMNS: proporciona información sobre la estructura de una tabla.
- SHOW KEYS (o SHOW INDEX): proporciona los índices de una tabla.

Para SHOW COLUMNS y SHOW KEYS, deberemos especificar la tabla de la cual deseamos obtener información mediante una cláusula FROM:

```
mysql> USE sakila
Database changed
```

```
mysql> SHOW COLUMNS FROM film_text;
```

Field	Type	Null	Key	Default	Extra
film_id	smallint(6)	NO	PRI	NULL	
title	varchar(255)	NO	MUL	NULL	

description	text	YES	MUL	NULL	
-----	-----	-----	-----	-----	-----

SHOW FULL COLUMNS es una variante de SHOW COLUMNS que indica, además del juego de caracteres, la interclasificación (ver el capítulo Optimización para una explicación de estos conceptos) y los privilegios posibles de cada columna.

```
mysql> SHOW COLUMNS FROM City LIKE 'C%'\G
***** 1. row *****

Field: CountryCode
Type: char(3)
Null: NO
Key:
Default:
Extra:
```

```
mysql> SHOW FULL COLUMNS FROM City LIKE 'C%'\G
***** 1. row *****

Field: CountryCode
Type: char(3)
Collation: latin1_swedish_ci
Null: NO
Key:
Default:
Extra:
Privileges: select,insert,update,references
Comment:
```

Para SHOW TABLES, la cláusula FROM es opcional; si no se especifica, obtendrá la lista de tablas de la base de datos seleccionada.

Ejemplo de utilización de SHOW TABLES:

```
mysql> USE world
```

Database changed

```
mysql> SHOW TABLES;
```

```
+-----+  
| Tables_in_world |  
+-----+  
| City             |  
| Country          |  
| CountryLanguage |  
+-----+
```

```
mysql> SHOW TABLES FROM sakila;
```

```
+-----+  
| Tables_in_sakila |  
+-----+  
| actor             |  
| actor_info        |  
| address            |  
| category           |  
| city              |  
| country            |  
| ...               |  
+-----+
```

También puede usar con estos comandos una cláusula `LIKE` para limitar el número de filas de resultados.

Ejemplo con `SHOW TABLES`:

```
mysql> SHOW TABLES LIKE 'film%';
```

```
+-----+  
| Tables_in_sakila (film%) |  
+-----+  
| film                      |  
| film_actor                |  
| film_category             |  
+-----+
```

```
| film_list |
| film_text |
+-----+
```

- El cliente `mysqlshow` recoge las funcionalidades de los comandos `SHOW`. Encontrará la descripción de su utilización en la documentación en línea.

Otros comandos muy útiles están disponibles con `SHOW`: se trata de `SHOW STATUS`, `SHOW VARIABLES` y `SHOW PROCESSLIST`. Estos comandos se detallarán más adelante en este capítulo.

b. Comando **DESCRIBE**

Este comando es el equivalente del comando `SHOW COLUMNS`. Se usa con mayor frecuencia que `SHOW COLUMNS` solo porque es más conocido. Existe un equivalente más corto: `DESC`. Todas estas variantes pueden utilizarse indistintamente.

Ejemplo:

```
mysql> DESCRIBE City;
```

```
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| ID         | int(11)   | NO   | PRI | NULL    | auto_increment |
| Name       | char(35)  | NO   | MUL |          |                |
| CountryCode | char(3)   | NO   |     |          |                |
| District   | char(20)  | NO   |     |          |                |
| Population | int(11)   | NO   |     | 0        |                |
+-----+-----+-----+-----+-----+-----+
```

2. BBDD information_schema

Se trata de una base de datos virtual, en el sentido de que las tablas y los datos de `information_schema` no se almacenan en el disco, sino en la memoria. No encontrará ningún directorio `information_schema/` en su directorio de datos, ni archivos `.frm` para las tablas.

Entre las tablas utilizadas con frecuencia, encontrará `SCHEMATA` (información sobre las bases de datos de la instancia), `TABLES`, `COLUMNS` o `VIEWS`. Para cada nueva versión, están disponibles nuevas tablas: es importante conocer la versión del servidor para saber a qué información tiene acceso.

La base de datos `information_schema` recoge el mismo tipo de información que los comandos `SHOW`, pero ofrece más detalles y presenta sobre todo la ventaja de poder ser consultada por los comandos `SELECT` habituales.

Por ejemplo, para conocer la lista de tablas en la base `world`, podemos escribir el siguiente comando:

```
mysql> SELECT TABLE_NAME FROM information_schema.TABLES WHERE  
TABLE_SCHEMA = 'world';
```

```
+-----+  
| TABLE_NAME |  
+-----+  
| City        |  
| Country     |  
| CountryLanguage |  
+-----+
```

Como puede observarse, las peticiones sobre la base `information_schema` suelen ser mucho más largas de escribir que su equivalente con `SHOW`. Por eso los comandos `SHOW` son muy utilizados, en especial cuando se está conectado a la consola de un servidor.

Sin embargo, el hecho de poder aprovechar todas las posibilidades de los `SELECT` constituye una ventaja innegable en relación con los comandos `SHOW` cuando queremos

realizar operaciones complejas en los metadatos.

Por ejemplo, si desea conocer el número de tablas de sus bases de datos, no podrá tener el resultado directamente con los comandos `SHOW`. Sin embargo, con la base de datos `information_schema`, las peticiones son muy simples de escribir.

```
mysql> SELECT COUNT(*) AS Total, TABLE_SCHEMA FROM
information_schema.tables GROUP BY TABLE_SCHEMA;
```

```
+-----+-----+
| Total | TABLE_SCHEMA |
+-----+-----+
|    28 | information_schema |
|    23 | mysql |
|    23 | sakila |
|     5 | world |
+-----+-----+
```

La base de datos `information_schema` es visible por todos los usuarios, pero los derechos de acceso a las tablas de la base serán administrados de forma automática por el servidor en función de los derechos del usuario. Esto significa, por ejemplo, que un usuario que no tiene el derecho de acceso a una base de datos no podrá pasar por `information_schema` para determinar cuáles son las tablas de esta base.

- ☐ La base de datos `information_schema` solo está accesible en lectura. Cualquier intento de escritura será rechazado.
- ☐ Algunas peticiones sobre la base `information_schema` pueden ser extremadamente largas. En general, esto no plantea problemas cuando estas peticiones están destinadas a un administrador de la base de datos. Sin embargo, se desaconseja el uso de `information_schema` en una aplicación transaccional, donde se requieren tiempos de respuesta cortos.

Herramientas básicas para la supervisión

1. SHOW PROCESSLIST

`SHOW PROCESSLIST` permite que cualquier usuario que tenga el derecho `SUPER` vea la actividad de todos los clientes conectados al servidor. Este comando ofrece mucha información sobre el tráfico que recibe el servidor. Veremos cuáles son los clientes conectados en el momento de la ejecución de `SHOW PROCESSLIST`, las peticiones en curso de ejecución y las peticiones bloqueadas. También encontraremos el número de identificación de cada conexión (columna `ID`), lo que nos permitirá utilizar el comando `KILL` si necesitamos detener con urgencia la ejecución de una petición.

La visualización se realiza en columnas, como muestra el siguiente ejemplo:

```
mysql> SHOW PROCESSLIST;
```

Id	User	Host	db	Command	Time	State	Info
56	root	localhost	world	Sleep	19		NULL
57	root	localhost	world	Query	12	Locked	select count(*) from City
58	root	localhost	NULL	Query	0	NULL	SHOW PROCESSLIST

- ☐ Si no tiene el derecho SUPER, SHOW PROCESSLIST no le mostrará la actividad de los demás clientes.

Podemos usar la opción SHOW PROCESSLIST FULL, cuya única diferencia es que no se trunca la pantalla si las líneas son demasiado largas.

También podemos usar la tabla PROCESSLIST de la BBDD information_schema en lugar de SHOW PROCESSLIST:

```
mysql> SELECT * FROM information_schema.PROCESSLIST;
```

ID	USER	HOST	DB	COMMAND	TIME	STATE	INFO
58	root	localhost	NULL	Query	0	executing	SELECT * FROM information_schema.PROCESSLIST
57	root	localhost	world	Query	50	Locked	select count(*) from City
56	root	localhost	world	Sleep	57		NULL

- ☐ Veremos en la continuación de este capítulo que podemos obtener una visión similar, pero más detallada, con performance_schema.

2. SHOW GLOBAL STATUS

Para determinar si el servidor está bien configurado, necesitaremos por un lado conocer los recursos asignados al servidor y, por otro, saber qué parte de estos recursos utiliza el servidor. Este examen se basa en los dos comandos: `SHOW GLOBAL VARIABLES` (descrito en el capítulo Configuración del servidor) y `SHOW GLOBAL STATUS`, del cual vamos a hablar en la siguiente sección.

Examinaremos a menudo la visualización de estos dos comandos para un recurso específico (por ejemplo, la caché de peticiones); `SHOW GLOBAL STATUS` nos dará indicaciones sobre la forma en que el servidor se comporta y `SHOW GLOBAL VARIABLES` permitirá determinar en qué factor influir si el servidor no se comporta como esperamos.

El comando `SHOW GLOBAL STATUS` proporciona información sobre las variables de estado del servidor. El funcionamiento es idéntico al de `SHOW GLOBAL VARIABLES` (ver sección Configuración del servidor).

Tenga en cuenta que las tablas correspondientes en `information_schema` se llaman `GLOBAL_STATUS` para las variables de carácter global y `SESSION_STATUS` para las variables del estado de las sesiones, y que los mismos datos están disponibles con `mysqladmin` utilizando `mysqladminextended-status`.

Las variables de estado se agrupan a menudo en algunos temas importantes. Entre los temas más frecuentemente consultados, encontramos:

- `Com_XXX`: estas variables indican el número de llamadas a un comando. Al igual que `com_select` representa el número de `SELECT` ejecutados por el servidor, `com_commit` el número de `COMMIT`, etc.
- `Handler_XXX`: estas variables indican las operaciones solicitadas por el servidor a los motores de almacenamiento y pueden utilizarse como complemento del comando `EXPLAIN` para tener una idea del número de operaciones necesarias para realizar una petición. Por ejemplo, el valor de la variable `Handler_read_rnd_next` es alto si sus peticiones efectúan muchos recorridos completos en las tablas.
- `Innodb_xxx`: estas variables dan información sobre `InnoDB` y coinciden con

las dadas por el comando `SHOW ENGINE INNODB STATUS`, descrito en la sección siguiente.

- `Key_XXX`: estas variables se refieren a la caché de índices MyISAM. Para más información al respecto, consulte el capítulo Optimización.
- `Open_XXX`: estas variables indican las operaciones sobre los descriptores de archivos. Son en especial interesantes si usamos MyISAM y pueden ayudarnos a dimensionar de forma correcta la caché de tablas (ver el capítulo Optimización).
- `Qcache_xxx`: estas variables informan sobre el estado de utilización de la caché de peticiones y pueden ayudar a determinar si la caché es útil para nuestra aplicación y, si es así, cómo configurarla (ver el capítulo Optimización para más información).
- `Select_xxx`: hemos visto en el capítulo Optimización que una petición `SELECT` puede ejecutarse de diversas maneras (recorrido completo de la tabla, recorrido de una parte de un índice...). Estas variables nos dan el número de `SELECT` de cada tipo. Una variable interesante para verificar de forma periódica es `Select_full_join`. Proporciona el número de productos cruzados, es decir, el número de joins incondicionales. Los productos cruzados son muy lentos en ejecución; esta variable debería permanecer siempre cerca de 0.
- `Sort_xxx`: cuando MySQL no puede clasificar utilizando un índice, las filas se remiten sin seleccionar por el motor de almacenamiento y el servidor se encarga de la clasificación. El valor de una de las variables `sort_xxx` se incrementará en función del tipo de clasificación elegido.
- `Threads_xxx`: estas variables ofrecen información sobre la actividad de los threads conectados al servidor, así como sobre el número de threads creados o en caché.

Para todas estas variables, así como aquellas que no están cubiertas aquí, la documentación oficial en línea será su fuente de información más precisa.

A menudo, es más interesante ver la variación de una variable en un intervalo de tiempo (por ejemplo, 1 segundo o 10 segundos) en lugar de solo conocer el valor desde el arranque del servidor. Por ejemplo, si estamos interesados en las tablas temporales creadas en disco, que provocan regularmente problemas de rendimiento, debemos considerar la variable `Created_tmp_disk_tables`. ¿Pero qué deducimos del valor siguiente, sabiendo

que el servidor se ejecuta desde hace nueve días?

```
mysql> SHOW GLOBAL STATUS LIKE 'Created_tmp_disk_tables';
+-----+-----+
| Variable_name          | Value      |
+-----+-----+
| Created_tmp_disk_tables | 1105626    |
+-----+-----+
```

Por desgracia, no mucho... El servidor ha creado muchas tablas en disco, pero es posible que las modificaciones hayan tenido lugar en las últimas 24 horas para corregir el problema.

Para conocer la variación de esta variable, podemos usar la opción `-r` de `mysqladmin`, que nos da la diferencia entre el valor actual de una variable y su valor anterior.

```
shell> mysqladmin extended-status -i10 -r -c3 | grep
'Created_tmp_disk_tables'
| Created_tmp_disk_tables          | 1105626    |
| Created_tmp_disk_tables          | 4          |
| Created_tmp_disk_tables          | 21         |
```

Aquí, podemos deducir que las peticiones necesitan tablas temporales en disco para su ejecución. Si deseamos tener una mejor idea del número de estas peticiones, habría que examinar más intervalos de 10 segundos, incrementando el valor de la opción `-c` que indica el número de veces que `mysqladmin` deberá ejecutarse (y debería hacerse este examen en las horas punta de la aplicación para estar seguro de obtener datos significativos).

Señalemos por último que `mysqladmin -r` es muy útil para examinar la variación de una variable, pero que resulta difícil leer la pantalla cuando consultamos 10, 30 o 50 variables de forma simultánea. En este caso, recomendamos utilizar `pt-mext`, de

Percona Toolkit. Esta herramienta es muy fácil de usar: basta con pasar como argumento el comando `mysqladmin` que queramos. Si deseamos ver las variaciones de una o varias variables, pasamos la opción `-r` a `pt-mext` en lugar de `mysqladmin` (la opción `-r` para `pt-mext` tiene el mismo significado que para `mysqladmin`).

```
shell> pt-mext -r -- mysqladmin extended-status -i10 -c3
Handler_prepare          329090816          3932          2416
Handler_read_first       732488            3            4
Handler_read_key         2147483647        1365688       768783
Handler_read_next        2147483647        2207017      1086766
Handler_read_prev        2147483647        26266         0
Handler_read_rnd         310851564         2489         4734
Handler_read_rnd_next    2147483647         772          2979
```

- ☐ Preste atención: `pt-mext` no funciona bajo Windows. Pero podemos registrar el resultado del comando `mysqladmin` en un archivo y luego ejecutar `pt-mext` sobre este archivo:

```
shell Windows> mysqladmin extended-status -i10 -c3 > status.out
(mover el archivo status.out a /tmp/status/out en un servidor Linux)
shell Linux> pt-mext -r -- cat /tmp/status.out | grep Handler
Handler_prepare          329090816          3932          2416
Handler_read_first       732488            3            4
Handler_read_key         2147483647        1365688       768783
Handler_read_next        2147483647        2207017      1086766
Handler_read_prev        2147483647        26266         0
Handler_read_rnd         310851564         2489         4734
Handler_read_rnd_next    2147483647         772          2979
```


3. SHOW ENGINE INNODB STATUS

Este comando es la principal fuente de información sobre el funcionamiento interno de InnoDB para su instancia. Su visualización es difícil de leer porque, en lugar de presentar sus resultados en columnas, como es el caso habitual, la salida se reduce a una larga cadena de caracteres.

He aquí un ejemplo del principio de la pantalla del comando (observe aquí la importancia del \G para obtener una visualización legible).

Descargado en: www.detodoprogramacion.org

```
mysql> SHOW ENGINE INNODB STATUS\G
***** 1. row *****

Status:
=====
100317 10:41:23 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 9 seconds
-----
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 6, signal count 6
Mutex spin waits 4, rounds 40, OS waits 2
RW-shared spins 6, OS waits 3; RW-excl spins 1, OS waits 1
-----
TRANSACTIONS
-----
Trx id counter 0 1792
Purge done for trx's n:o < 0 0 undo n:o < 0 0
...
```

El resultado del comando está dividido en secciones (SEMAPHORES y TRANSACTIONS en el ejemplo anterior), que se centran cada una en una parte del funcionamiento del motor.

a. SEMAPHORES

Algunas partes del código de InnoDB funcionan con semáforos, que son estructuras de datos cuya finalidad es controlar el acceso a recursos compartidos. En el caso de que tengamos muchos clientes efectuando peticiones de forma simultánea, es posible que los semáforos se conviertan en puntos de contención y planteen problemas de rendimiento. Encontraremos información en esta sección que nos permitirá comprender mejor lo que está ocurriendo. Para otros casos, esta sección probablemente no interesará.

b. LAST FOREIGN KEY ERROR

Esta sección solo aparece cuando el servidor encuentra errores en las claves externas. Cuando las peticiones fallan a causa de problemas en las claves externas, los mensajes de error recuperados por el servidor suelen ser poco claros y no ayudan a determinar la fuente del problema. En cambio, mirando en esta sección, encontraremos sin duda el origen exacto del error, lo que facilitará la corrección.

c. LAST DETECTED DEADLOCK

Al igual que para lo anterior, esta sección solo aparece cuando surgen interbloqueos. Los datos mostrados aquí permiten conocer la causa exacta del interbloqueo. Observe que a veces es difícil leer toda la información, ya que la pantalla se ve contaminada por información que en realidad es útil para los desarrolladores InnoDB.

d. TRANSACTIONS

En esta sección tenemos un resumen sobre el estado del motor cara a cara con las transacciones, así como la lista de transacciones en curso en el momento en que el comando `SHOW ENGINE INNODB STATUS` fue ejecutado.

e. FILE I/O

Esta sección muestra la actividad de los threads internos de InnoDB para todo lo que se refiere a los accesos a disco. Mostrará siempre al menos cuatro threads: `Insert buffer thread`, `log thread`, `read thread` y `write thread`. Este número puede ajustarse con la variable `innodb_file_io_threads`, para obtener más threads de lecturas y escrituras.

También podemos observar el número de operaciones de sincronización de los discos

realizados en los registros. Si este número es muy alto, podemos decidir si es necesario aumentar el tamaño de los registros InnoDB con la variable `innodb_log_file_size` o, llegado el caso, si es preferible no utilizar el modo `AUTOCOMMIT`.

f. INSERT BUFFER AND ADAPTATIVE HASH INDEX

Esta sección da las estadísticas para la caché de inserción (insert buffer), como el tamaño o el espacio disponible, y para el índice adaptativo. Se trata de dos aspectos avanzados del funcionamiento de InnoDB.

Descargado en: www.detodoprogramacion.org

g. LOG

Encontraremos en esta sección estadísticas sobre los archivos de registro de InnoDB.

h. BUFFER POOL AND MEMORY

Esta sección proporciona indicadores sobre la actividad del buffer pool (recuérdese que se trata de una zona de memoria donde InnoDB almacena los datos e índices para reducir la carga del disco). Encontraremos en particular su tamaño, el espacio libre y la tasa de aciertos. Esta información nos puede ayudar para dimensionar de forma correcta el buffer pool: por ejemplo, si el espacio libre es bajo, al igual que la tasa de aciertos, el espacio asignado es con toda probabilidad demasiado justo.

i. ROW OPERATIONS

Esta sección nos indica de forma global el número de operaciones de lectura y escritura que han sido realizadas por el motor desde su arranque.

Performance Schema

1. Rol

Descargado en: www.detodoprogramacion.org

Performance Schema es un mecanismo de instrumentación propuesto desde MySQL 5.5, que permite recoger mucha información sobre el comportamiento del servidor durante su funcionamiento. Los datos se almacenan en una base dedicada llamada `performance_schema` y pueden ser consultados por comandos SQL clásicos.

Performance Schema recibe novedades importantes en cada nueva versión de MySQL; por tanto, es importante tener las siguientes consideraciones en mente:

- Con MySQL 5.5, desactivábamos en general Performance Schema (lo que era la opción por defecto), ya que la implementación era pobre, pero sobre todo tenía una significativa disminución de rendimiento.
 - A partir de MySQL 5.6, Performance Schema se activa por defecto. La disminución de rendimiento es mucho más limitada, y la instrumentación, mucho más rica. Las posibilidades son tan grandes que instalar el esquema `sys` (véase más adelante en esta sección) se recomienda para explotar con mayor facilidad la información proporcionada. Sin embargo, si no pensamos utilizar las estadísticas de Performance Schema, es recomendable desactivar la característica para recuperar un poco de memoria y un poco de rendimiento.
 - A partir de MySQL 5.7, el esquema `sys` se instala por defecto, lo que hace mucho más simple la explotación de los datos de Performance Schema. Otra de las novedades destacadas es la aparición de tablas para supervisar la replicación, ya que el comando tradicional `SHOW SLAVE STATUS` se adapta cada vez menos a las nuevas posibilidades (replicación paralela, replicación multifuente).
-

2. Configuración

La configuración de Performance Schema comienza por la verificación de la activación o no de la funcionalidad. Solo debemos verificar el valor de la variable `performance_schema`. Por ejemplo:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'performance_schema';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| performance_schema | ON    |
+-----+-----+
```

La configuración se realiza a continuación, seleccionando los instrumentos y los consumidores que se activarán. Los instrumentos representan los puntos de instrumentación: cuantos más active, más detalles obtendrá sobre lo que pasa dentro del servidor, pero con una mayor disminución del rendimiento. Los consumidores son los destinos de los puntos de instrumentación. De esta forma es posible configurar un punto de instrumentación para conocer el valor instantáneo, al igual que guardar un histórico.

La elección de los instrumentos y consumidores se hace actualizando las tablas `setup_instruments` y `setup_consumers` en la base `performance_schema`.

Por ejemplo, en un servidor MySQL 5.7, podemos ver que más de 1 000 puntos de instrumentación están disponibles:

```
mysql> select count(*) from setup_instruments;
+-----+
| count(*) |
+-----+
|      1012 |
+-----+
```

☐ A título comparativo, MySQL 5.5 dispone de 222 instrumentos.

Pero de estos 1012 instrumentos solo un tercio se activan:

```
mysql> select count(*) from setup_instruments where enabled='YES';
+-----+
| count(*) |
+-----+
|      327 |
+-----+
```

Si deseamos activar un punto de instrumentación, debemos ejecutar una petición del tipo siguiente:

```
mysql> UPDATE performance_schema.setup_instruments SET ENABLED =
'YES', TIMED = 'YES' WHERE NAME = 'transaction';
```

Si deseamos activar los consumidores correspondientes al punto de instrumentación anterior, ejecutaremos los siguientes comandos:

```
mysql> UPDATE performance_schema.setup_consumers SET ENABLED =
'YES' WHERE NAME = 'events_transactions_current';
```

```
mysql> UPDATE performance_schema.setup_consumers SET ENABLED =
'YES' WHERE NAME = 'events_transactions_history';
```

```
mysql> UPDATE performance_schema.setup_consumers SET ENABLED =
'YES' WHERE NAME = 'events_transactions_history_long';
```

O más simple:

```
mysql> UPDATE performance_schema.setup_consumers SET ENABLED =  
'YES' WHERE NAME LIKE 'events_transactions%';
```

3. Esquema sys

Performance Schema ofrece una instrumentación rica cuya información está accesible a través de peticiones SQL. Sin embargo, en realidad, es muy posible perderse en el laberinto de las tablas (más de 80 con MySQL 5.7). Si no somos capaces de acceder de forma rápida a la información que nos interesa, es casi seguro que no usaremos Performance Schema.

Por esta razón se desarrolló el esquema `sys`. Contiene muchas vistas basadas en las tablas de Performance Schema, dando así acceso a la información más frecuente.

Si usamos MySQL 5.7, `sys` se instala por defecto. Si utilizamos MySQL 5.6, debemos primero realizar algunas operaciones:

- Visite el sitio del proyecto (<https://github.com/mysql/mysql-sys>) y descargue los archivos.
- Importe el archivo SQL:

```
# mysql -uroot < sys_56.sql
```

- `sys` no está disponible para MySQL 5.5. Esto no es realmente una limitación, ya que, para poder utilizar Performance Schema, es más razonable limitarse a las versiones 5.6 y superiores.

Observando el nombre de las tablas en el esquema `sys`, verá que las tablas se organizan en pares, por ejemplo `host_summary` y `x$host_summary`. ¡Esto no es casualidad!

La idea es ofrecer un formato de datos fácilmente utilizable por los scripts (tablas

comenzando por x\$) al mismo tiempo que un formateo de datos fácilmente legible por un humano (otras tablas).

Observemos el siguiente ejemplo. Para la tabla `host_summary`, veremos que se menciona la unidad de tiempo, mientras que en la tabla `x$host_summary` encontraremos una cifra bruta:

```
mysql> select * from x$host_summary_by_file_io;
```

host	ios	io_latency
background	2013	245583609648
localhost	776	27492347428

```
mysql> select * from host_summary_by_file_io;
```

host	ios	io_latency
background	2013	245.58 ms
localhost	776	27.49 ms

Por tomar otro ejemplo, una de las vistas se llama `processlist`, y es equivalente a `SHOW PROCESSLIST`, pero con mucha más información. Encontraremos, por ejemplo, una columna `progress` que puede ser interesante para estimar el tiempo de ejecución pendiente si se procesa un comando `ALTER TABLE` en una tabla de gran tamaño.

Una descripción de todas las vistas del esquema `sys` está disponible en la siguiente dirección: <https://dev.mysql.com/doc/refman/5.7/en/sys-schema-reference.html>

Identificación de los problemas de las peticiones

1. Peticiones lentas

Como era de esperar, la identificación de las peticiones lentas se efectúa en principio gracias al registro de peticiones lentas, en general capturando todas las peticiones durante cierto tiempo (una hora, por ejemplo) con la opción `long_query_time = 0`. La dificultad es poder interpretar luego la información de los datos capturados. La mejor herramienta para esta tarea es `pt-query-digest`, cuyas principales características veremos más adelante en esta sección.

Pero si está usando MySQL 5.6 o 5.7, existe una solución mucho más rápida: `performance_schema` o en concreto varias vistas de la base `sys`. Todas las vistas siguientes pueden ayudarnos a encontrar las peticiones mal optimizadas:

```
mysql> show tables like 'statement%';
+-----+
| Tables_in_sys (statement%) |
+-----+
| statement_analysis         |
| statements_with_errors_or_warnings |
| statements_with_full_table_scans |
| statements_with_runtimes_in_95th_percentile |
| statements_with_sorting    |
| statements_with_temp_tables |
+-----+
```

El nombre de cada vista es explícito. Si por ejemplo intenta identificar las peticiones de creación de tablas temporales, mirará en la vista `statements_with_temp_tables`. Si buscamos las peticiones que recorren las tablas completas, las vemos del lado de la vista `statements_with_full_table_scans`. También será útil mirar `statement_analysis`, para que podamos ordenar las peticiones con la columna `total_latency`, lo que dará una lista de las peticiones más pesadas.

- ☐ Verifique siempre que los consumidores siguientes están activados (lo están por defecto):

```
mysql> select * from performance_schema.setup_consumers where name
like 'events%statement%' and enabled = 'yes';
+-----+-----+
| NAME                                | ENABLED |
+-----+-----+
| events_statements_current           | YES     |
| events_statements_history           | YES     |
+-----+-----+
```

`performance_schema` no es tan preciso como el diario de peticiones lentas porque la ventana de captura es más limitada (depende de la memoria disponible para `performance_schema` y el número de peticiones realizadas). Para el registro de peticiones lentas, a partir del momento en que tenga suficiente espacio en disco, podrá capturar todas las peticiones que desee.

Si `performance_schema` no está disponible o si deseamos analizar las peticiones durante un largo período de tiempo, `pt-query-digest` nos será de gran ayuda. La forma más sencilla de invocar la herramienta es ejecutar el script con solo el parámetro del nombre del registro de peticiones lentas:

```
$ pt-query-digest mysql-slow.log
```

- ☐ Preste atención: `pt-query-digest` consumirá el 100 % del procesador durante todo el tiempo de análisis. ¡No debe usarse en servidores de producción!

Obtendremos una lista de peticiones, clasificadas de la más pesada a la menos pesada en términos de tiempo total de ejecución:

```
# Profile
# Rank Query ID           Response time    Calls R/Call    Item
# ==== =====
# 1 0x67A347A2812914DF  4082.0000 23.8%    339 12.0413 SELECT
SYNC_HISTO
# 2 0x488E7BAC017768E8  2801.0000 16.4%     80 35.0125 SELECT
SLIDE_? PRODUCTO_?
# 3 0x5ECCFCA304C3072E  2236.0000 13.1%    347  6.4438 SELECT
SLIDE_?
# 4 0xFCBC9AC2F4997DCC  1945.0000 11.4%     80 24.3125 SELECT
CONTENIDO? PRODUCTO_?
# 5 0x5F139E3C468E7A22  1522.0000  8.9%    200  7.6100 SELECT
information_schema.tables
```

Luego, para cada petición, obtendremos muchas estadísticas:

```
# Query 1: 0.00 QPS, 0.00x concurrency, ID 0x67A347A2812914DF at byte
429889
# This item is included in the report because it matches --limit.
#           pct    total    min    max    avg    95%  stddev
median
# Count           17      339
```

```

# Exec time      23    4082s      3s    136s      12s      25s      13s 9s
# Lock time      0      0      0      0      0      0      0 0
# Rows sent      84 556.94M 428.09k 19.43M 1.64M 2.88M 1.79M
1.53M
# Rows exam      32 556.94M 428.09k 19.43M 1.64M 2.88M 1.79M
1.53M
# Users          1    root
# Hosts          1 localhost
# Databases      4 ms (266), ms_histo... (67), ms_c (3)... 1
more
# Time range 2009-07-20 11:09:26 to 2009-10-13 16:21:25
# bytes          5 17.40k      51      59 52.56 51.63 1.14
51.63
# Query_time distribution
# 1us
# 10us
# 100us
# 1ms
# 10ms
# 100ms
# 1s #####
# 10s+ #####
# Tables
# SHOW TABLE STATUS FROM `ms_historico` LIKE 'SYNC_HISTO'\G
# SHOW CREATE TABLE `ms_historico`.`SYNC_HISTO`\G
SELECT /*!40001 SQL_NO_CACHE */ * FROM `SYNC_HISTO`\G
# Converted for EXPLAIN
# EXPLAIN
SELECT /*!40001 SQL_NO_CACHE */ * FROM `SYNC_HISTO`\G

```

Después de haber mirado el plan de ejecución de la petición y las estructuras de las tablas, estas estadísticas nos ayudarán a comprender lo que puede mejorarse en la petición.

`pt-query-digest` es capaz de analizar las peticiones procedentes de varias fuentes:

- La fuente más común es un registro de peticiones lentas.
- También podemos proporcionar la salida de `tcpdump` capturada de la siguiente

manera:

```
$ tcpdump -s 65535 -x -nn -q -tttt -i any port 3306 > tcpdump.txt  
$ pt-query-digest --type tcpdump tcpdump.txt
```

- También podemos solicitar a la herramienta que capture de forma directa las peticiones empleando el comando `SHOW PROCESSLIST`:

```
$ pt-query-digest --processlist h=localhost,u=root,p=mi_cont
```

Este último método no se recomienda por lo general: `SHOW PROCESSLIST` se ejecuta a intervalos regulares (por ejemplo, cada 10 milisegundos); por lo tanto, no se capturarán las peticiones que se ejecutan entre dos intervalos. Además, `SHOW PROCESSLIST` bloquea el servidor: ejecutar el comando de vez en cuando no produce penalización, pero hacerlo 100 veces por segundo provoca una degradación de rendimiento. Reserve esta técnica solo para los servidores en los que no puede funcionar ninguno de los demás métodos.

2. Deadlocks

Muchos administradores de bases de datos se encuentran desamparados frente a los deadlocks, ya que la instrumentación disponible no es excepcional. Por defecto, la única vista que se puede obtener se encuentra en la salida de `SHOW ENGINE INNODB STATUS`. Por desgracia, esta vista no es práctica del todo para visualizar y le falta información para poder identificar el problema de forma correcta.

Pero antes de comenzar a resolver los problemas de deadlocks, la primera cuestión que hay que plantearse es: ¿con qué frecuencia se producen? En efecto, si usamos InnoDB, los bloqueos se plantearán para cada transacción que contenga escrituras y, cuantas más conexiones realicen escrituras de forma simultánea, mayor será la probabilidad de deadlocks. Debemos saber, pues, que es imposible eliminar los deadlocks por completo. Es

inútil extendernos en interminables sesiones de análisis si el sistema genera un deadlock cada tres días.

Para obtener información sobre la frecuencia de los deadlocks, podemos consultar la salida de `SHOW ENGINE INNODB STATUS` de forma regular. Una solución más sencilla es utilizar `pt-deadlock-logger` del Percona Toolkit, que se encargará de filtrar la salida de `SHOW ENGINE INNODB STATUS` para imprimir un resultado (con la fecha y hora) cada vez que ocurre un deadlock.

Por ejemplo, para pedir al script que pruebe cada 30 segundos si ocurre un deadlock, usamos un comando similar a:

```
$ pt-deadlock-logger -h=127.0.0.1,u=root --interval=30s
```

La herramienta tiene muchas opciones para, por ejemplo, almacenar los resultados en una tabla, ejecutar en segundo plano o detenerse de forma automática después de un período de tiempo indicado.

Si después de este primer análisis apreciamos que los deadlocks son frecuentes, por ejemplo cada diez minutos, el siguiente paso es comprender la razón de estos deadlocks. Se trata a menudo de la etapa más compleja, ya que, por defecto, la información proporcionada por `SHOW ENGINE INNODB STATUS` es incompleta.

Consideremos el siguiente ejemplo. Si estamos ejecutando las peticiones siguientes en dos sesiones diferentes, obtendremos un deadlock:

```
mysql1> start transaction;
mysql2> start transaction;
mysql1> update t1 set a='test' where id=1;
mysql2> update t2 set v='test' where id=3;
mysql1> update t2 set v='test2' where id=3;
```

Esta petición será bloqueada porque la otra sesión plantea un bloqueo en la fila para la cual `id=3`, impidiendo cualquier modificación simultánea.

```
mysql2> update t1 set a='test2' where id=1; # Deadlock para
una fila de las sesiones
```

Pero si miramos lo que dice SHOW ENGINE INNODB STATUS, solo encontraremos la última petición de cada transacción.

```
-----
LATEST DETECTED DEADLOCK
-----
2016-06-27 14:50:25 0x7f7248f69700
*** (1) TRANSACTION:
TRANSACTION 3354, ACTIVE 25 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 4 lock struct(s), heap size 1136, 2 row lock(s), undo log
entries 1
MySQL thread id 5, OS thread handle 140128826853120, query id 29
localhost
root updating
update t1 set a='test2' where id=1
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 27 page no 3 n bits 72 index PRIMARY of table
`test`.
`t1` trx id 3354 lock_mode X locks rec but not gap waiting
*** (2) TRANSACTION:
TRANSACTION 3353, ACTIVE 65 sec starting index read
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1136, 2 row lock(s), undo log entries 1
MySQL thread id 4, OS thread handle 140128827119360, query id 30
localhost root updating
update t2 set v='test2' where id=3
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 27 page no 3 n bits 72 index PRIMARY of table
`test`.
`t1` trx id 3353 lock_mode X locks rec but not gap
```

```
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 28 page no 3 n bits 72 index PRIMARY of table
`test`.
`t2` trx id 3353 lock_mode X locks rec but not gap waiting
*** WE ROLL BACK TRANSACTION (2)
```

Por supuesto, todo esto no tiene ningún sentido: ¿cómo puede una petición sobre t_1 obtener un bloqueo sobre la tabla t_2 y viceversa? El desconocimiento de las peticiones anteriores a esta transacción impide comprender la razón del deadlock.

Si utilizamos MariaDB o Percona Server, existe un método para obtener todas las peticiones de las transacciones que intervienen en un deadlock: basta con activar el registro de peticiones lentas con los parámetros `log_slow_verbosity = innodb` y `long_query_time = 0`. De esta forma, obtendremos todas las peticiones realizadas en el servidor con su identificador de transacción. Ya que `SHOW ENGINE INNODB STATUS` proporciona la identificación de las transacciones que han creado un deadlock, podemos reconstruir la secuencia de las peticiones para cada una de las transacciones.

Herramientas de supervisión del sistema

1. Cacti

Cacti es una herramienta de supervisión disponible para UNIX/Linux y Windows, que permite obtener gráficos sobre diversos parámetros del servidor MySQL a través de una interfaz web. Cacti es muy simple de instalar y configurar, lo que es muy importante si necesitamos construir de forma rápida una solución gráfica que muestra la evolución en el tiempo del rendimiento del servidor.



Cacti funciona con un sistema de plantillas (templates) y podremos encontrar con facilidad en Internet las plantillas para MySQL. Una buena opción es descargar las plantillas facilitadas por Percona en la siguiente dirección: <https://www.percona.com/software/mysql-tools/percona-monitoring-plugins>

2. Grafana

Cacti es un proyecto antiguo; lo notaremos en particular en la interfaz de visualización de gráficos, que es totalmente obsoleta. Grafana (<http://www.grafana.org>) es mucho más moderna y permite visualizar cualquier tipo de series de datos. También podemos recoger gráficos en páginas especiales para, por ejemplo, visualizar directamente el consumo de CPU de todos los servidores.

3. Nagios

Nagios es un sistema de supervisión muy completo y usado con frecuencia. Permite supervisar múltiples parámetros en los servidores y puede desencadenar alertas sobre la interfaz, por SMS o correo electrónico, si un parámetro escapa de una gama de valores predeterminados. Una interfaz web permite consultar las pantallas de monitorización.

El principal escollo de la utilización de Nagios es su configuración, que desanima a más de un administrador de sistema. A pesar de ello, existen comunidades en torno a este software, lo que puede ser útil si tenemos dificultad para desplegar esta herramienta.

Observe que Nagios es modular y acepta el uso de plug-ins para enriquecer sus funcionalidades. Nagios Exchange es el sitio centralizando todos estos plugins (<http://exchange.nagios.org>).

4. Identificación de los problemas de sistema con Linux

a. `vmstat`

`vmstat`, instalada por defecto en la mayoría de los sistemas UNIX/Linux, proporciona información sobre la actividad de los procesadores, memoria RAM, discos y los procesos que se ejecutan.

Ejecutando de forma periódica este comando o indicando un intervalo de forma directa (por ejemplo, `vmstat 10` para ejecutar el comando de forma automática cada 10 segundos), podemos ver la evolución de los diferentes subsistemas de su servidor.

El informe presentado por `vmstat` está dividido en grupos de columnas:

- `procs` representa la información sobre los procesos.
- `memory` ofrece información acerca de la memoria.
- `swap` informa sobre la actividad de swap.
- `io` permite conocer la actividad del disco.
- `system` hace saber cómo el sistema administra los procesos concurrentes.
- `cpu` informa sobre la actividad de los procesadores.

Consulte la página `man` de `vmstat` para obtener información precisa sobre cada una de las columnas y conocer las opciones que nos permite filtrar la visualización de lo que nos

interesa.

El siguiente informe proporciona un ejemplo del resultado de `vmstat` para un servidor MySQL que sufre un pico de actividad de CPU:

```
shell> vmstat 5
procs -----memory----- ---swap-- -----io----- -system--
-----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo    in    cs
us sy id wa
 0  0       0 2761508  56828 394688    0    0     0    15  438  438
28  1 70  0
 1  0       0 2761516  56828 394696    0    0     0     0  566  387
61  1 38  0
 2  0       0 2758416  56836 394696    0    0     0     4  702  371
98  2  0  0
 1  0       0 2761268  56844 394696    0    0     0     9  707  352
100 0  0  0
 1  0       0 2761128  56872 394744    0    0    14     5  697  372
99  1  0  0
 2  0       0 2761004  56880 394744    0    0     0    30  713  361
100 0  0  0
 2  0       0 2761004  56880 394744    0    0     0     0  703  346
99  1  0  0
 0  0       0 2761004  56888 394744    0    0     0     6  362  379
17  1 82  0
```

Observe que, en la primera línea, la columna `us` (tiempo usuario) no es muy alta, y luego llega al 100 %, mostrando un período de saturación, y finalmente vuelve a descender. La columna `id` (tiempo sin actividad) sigue una evolución inversa.

b. iostat

`iostat` es otro comando de sistema que proporciona el mismo tipo de información que `vmstat`, pero limitado a los procesadores y los discos. La ventaja de este comando es que muestra más detalles que `vmstat`, en concreto en lo que respecta a la actividad del disco.

Al igual que para `vmstat`, podemos especificar un intervalo para ejecutar el comando de manera periódica.

`iostat` no se instala siempre por defecto en los equipos; puede ser necesario instalar el paquete `sysstat`.

Ejemplo de informe generado con este comando:

```
shell> iostat -x 5

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           50,00    0,00   3,20    0,40    0,00   46,40

Device:            rrqm/s   wrqm/s     r/s     w/s  rsec/s  wsec/s
avgrq-sz avgqu-sz   await  svctm  %util
sda              0,00    1,80    0,40    1,60   128,00   27,20
77,60    0,02    8,00    8,00    1,60

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           94,40    0,00   5,60    0,00    0,00    0,00

Device:            rrqm/s   wrqm/s     r/s     w/s  rsec/s  wsec/s
avgrq-sz avgqu-sz   await  svctm  %util
sda              0,00    2,20    0,20    5,80    4,80   64,00
11,47    0,01    1,87    1,33    0,80

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           83,17    0,00   8,22    0,00    0,00   8,62

Device:            rrqm/s   wrqm/s     r/s     w/s  rsec/s  wsec/s
avgrq-sz avgqu-sz   await  svctm  %util
sda              0,00    0,00    0,20    0,20    6,41    1,60
20,00    0,00    8,00    8,00    0,32
```

□ Una alternativa a `iostat` es `pt-diskstats`, del Percona Toolkit, que utilizaremos, por ejemplo, de la siguiente manera para obtener estadísticas

sobre los accesos a disco cada segundo durante 60 segundos:

```
# pt-diskstats --group-by=all --iterations=60
```

c. mpstat

`mpstat` es un comando interesante para conocer con precisión el uso de cada uno de los núcleos del sistema. En efecto, una cifra del uso global de la CPU no es siempre muy significativa. Por ejemplo, si tenemos un servidor con 8 núcleos y la utilización de CPU es del 12,5 %, esto quiere decir que cada núcleo se utiliza el 12,5 % (todo va bien) o que un núcleo se utiliza al 100 %, mientras que los otros siete están inactivos (sin duda, un problema de rendimiento).

Ejecute `mpstat` de la siguiente manera para obtener estadísticas cada segundo durante 60 segundos:

```
# mpstat -P ALL 1 60
```

Y obtendrá la siguiente salida:

16:31:58	CPU	%usr	%nice	%sys	%iowait	%irq	%soft
%steal	%guest	%gnice	%idle				
16:31:59	all	3,02	0,00	1,26	0,00	0,00	0,00
0,00	0,00	0,00	95,72				
16:31:59	0	2,02	0,00	0,00	0,00	0,00	0,00
0,00	0,00	0,00	97,98				
16:31:59	1	1,01	0,00	1,01	0,00	0,00	0,00
0,00	0,00	0,00	97,98				
16:31:59	2	1,00	0,00	2,00	0,00	0,00	0,00
0,00	0,00	0,00	97,00				
16:31:59	3	8,08	0,00	1,01	0,00	0,00	0,00
0,00	0,00	0,00	90,91				

☐ Si `mpstat` no está instalado por defecto, forma parte del paquete `sysstat`.

La página `man` del comando nos dará toda la información necesaria para una correcta utilización.



<https://dogramcode.com/bloglibros>



Dogram