Introducción: Historia y revisiones. Versiones y Forks. Métodos de instalación. Layouts de instalación.

MySQL

Caracteristicas

Motor BBDD
-relacional
-open source
-rápido y confiable
-escalable y fácil de usar
-cliente/servidor o embebido

Lo mas popular en el mundo
-Multiplataforma
-drivers y APIs

DDL (create, alter)
lenguaje proporcionado por el sistema de gestión de base de datos de definición de estructuras que almacenarán los datos y los procedimientos o funciones.

DML (Insert, update, delete)
lenguaje proporcionado por los sistemas gestores de bases de datos para llevar a cabo las tareas de consulta o modificación de los datos

Versiones y Forks. Métodos de instalación. Layouts de instalación.

Releases:

Community edition
-Free

MySQL Standard, MySQL Enterprise (pago) y MySQL cluster carrier grade
-Soporte comercial
-Herramientas empresariales (enterprise)
-Más niveles de soporte

Ventajas MySQL

Motivos económicos
-Menor TCO (Costo total de adquisición)

Motivos técnicos
-Robustez, confiable, buen rendimiento
-Escalable, versiones de clustering
-Amplio soporte y documentación

Distribuciones y versiones de MySQL
-Versiones antiguas EOL
Actualmente sin soporte
-Production release (GA) (General availability)
Última versión estable
-Production quality release
Versión estable anterior (mas estable que la ultima estable)
-Pre-production release
En desarrollo, inestable

Forks: percona, mariadb, drizzle

MariaDB
Fork de MySQL con licencia GPL. Tiene una alta compatibilidad con MySQL.

LAMP: acrónimo usado para describir un sistema de infraestructura de internet

Instalación: por comandos o por binario ejecutable.

Layouts: sistema de distribución de los elementos del diseño

Arquitectura: Programas cliente y servidor. Uso básico de la consola (cliente). Modo de conexión: cliente nativo, ODBC/JDBC. Storage Engines. InnoDB. Herramientas disponibles.

Arqitectura de Mysql:
La arquitectura de MySQL tiene como característica más notable el separar el motor de almacenamiento (que se encarga de los detalles de entrada-salida y representación de la información en memoria secundaria) del resto de los componentes de la arquitectura. Es decir, el diseño del gestor está preparado para que se pueda cambiar el gestor de almacenamiento. Esto permite incluso crear nuevos motores de almacenamiento especializados para ciertas tareas o tipos de aplicaciones.

Los conectores
Los conectores son bibliotecas en diferentes lenguajes de programación que permiten la conexión (remota o local) con servidores MySQL y la ejecución de consultas. Por ejemplo, el conector Connector/J permite conectarse a MySQL desde cualquier aplicación programada en lenguaje Java, y utilizando el Java Database Connectivity (JDBC) API.
El gestor de conexiones

La gestión de conexiones es responsable de mantener las múltiples conexiones de los clientes. Un gestor de conexiones inexistente o laxo simplemente crearía una conexión por cada cliente conectado. No obstante, las conexiones consumen recursos de máquina, y crearlas y destruirlas son también procesos costosos. Por eso, el gestor de conexiones de MySQL puede configurarse para limitar el número de conexiones concurrentes, y también implementa un pool de conexiones.
La idea es que muchas aplicaciones abren una conexión y la mantienen abierta y ociosa durante mucho tiempo (por ejemplo, durante toda la sesión de un usuario, que de vez en cuando se levanta para diferentes tareas mundanas como tomar café), y solo “de vez en cuando” se utiliza un hilo de ejecución para ejecutar una consulta, que además, típicamente tarda como mucho unos milisegundos. No tiene sentido mantener una conexión ociosa para cada usuario. De aquí proviene la idea de los pools de conexiones: hay un número de conexiones disponibles, y cada vez que una aplicación hace una solicitud, se le asigna una conexión del pool que no esté ocupada. Lógicamente, la aplicación cliente no es consciente de este mecanismo. En términos por ejemplo, de las interfaces JDBC, esto quiere decir que cada vez que enviamos una sentencia con llamadas como Statement.executeQuery() realmente puede que se cree una nueva colección, o quizá se tome una del pool. Es decir, las llamadas a Driver.getConnection() y a Connection.close() no siempre determinan el tiempo de vida de una conexión real en MySQL.
Además de la reducción en el tiempo de establecimiento de conexión (si se reusa una conexión ya creada del pool), la técnica permite limitar el número de conexiones simultáneas. Dado que las conexiones consumen recursos, es mejor limitar este número que llevar a una carga excesiva en el servidor, que podría acabar en una caída del sistema o un comportamiento impredecible. Nótese que gracias a los pools de conexiones, puede darse servicio a muchas conexiones concurrentes con un número limitado de conexiones que se reutilizan.
El gestor de conexiones también se ocupa de la autentificación de los usuarios. La autentificación por defecto se basa en el nombre de usuario, la máquina desde la que se conecta y la password.

El procesamiento y optimización de consultas

Cada vez que una consulta llega al gestor de MySQL, se analiza sintácticamente y se produce una representación intermedia de la misma. A partir de esa representación, MySQL toma una serie de decisiones, que pueden incluir el determinar el orden de lectura de las tablas, el uso de ciertos índices, o la re-escritura de la consulta en una forma más eficiente.
Existe la posibilidad de utilizar ciertas cláusulas en las consultas para ayudar al optimizador en su tarea, o bien podemos pedirle al servidor ciertas “explicaciones” sobre cómo ha planificado nuestras consultas, para entender mejor su funcionamiento.
Dado que la optimización de las consultas depende de las capacidades del gestor de almacenamiento que se esté utilizando, el optimizador “pregunta” al gestor si soporta ciertas características, y de este modo, puede decidir el tipo de optimización más adecuado.

La caché de consultas

MySQL implementa un caché de consultas, donde guarda consultas y sus resultados enteros. De este modo, el procesador de consultas, antes ni siquiera de plantear la optimización, busca la consulta en la caché, para evitarse realizar el trabajo en el caso de que tenga suerte y encuentre la consulta en la caché.

Cuando una tabla cambia, cualquier consulta guardada en la cache relacionada a la tabla se elimina.

La cache ignora los siguientes tipos de consultas:

- Toda consulta que tiene subconsulta.
- Toda consulta que requiere una variable.
- Toda consulta generada por un; procedimiento almacenado, disparador (trigger) o evento.
- Funciones con fecha, hora, random (contenido variable)
- Toda consulta que usa tablas temporales.

El Control de Concurrencia

Es el mecanismo que se utiliza para evitar que lecturas o escrituras simultáneas a la misma porción de datos terminen en inconsistencias o efectos no deseados. El mecanismo que se utiliza para controlar este acceso es el de los bloqueos (locks). Cada vez que una aplicación quiere acceder a una porción de los datos, se le proporciona un bloqueo sobre los mismos. Varias aplicaciones que quieran leer simultáneamente no tienen ningún problema en hacerlo, para la lectura se proporcionan bloqueos compartidos (shared locks). Sin embargo, varios escritores o un escritor simultáneo con lectores puede producir problemas. Por eso, para la escritura se proporcionan bloqueos exclusivos (exclusive locks).

Aunque la idea parece simple, hay que tener en cuenta que la obtención y liberación de los bloqueos se realiza continuamente, y esto produce una sobrecarga en el procesamiento dentro del servidor. Además, hay diferentes políticas de bloqueo, por ejemplo, ¿es mejor bloquear cada tabla completa afectada o solo las filas de la tabla a las que quiere acceder una consulta?. Dada la existencia de diferentes técnicas, el control de concurrencia en MySQL se divide entre el servidor y cada gestor de almacenamiento.

La gestión de transacciones y recuperación

La gestión de transacciones permite dotar de semántica “todo o nada” a una consulta o a un conjunto de consultas que se declaran como una sola transacción. Es decir, si hay algún problema y parte de la consulta o algunas de las consultas no consiguen llevarse a cabo, el servidor anulará el efecto parcial de la parte que ya haya sido ejecutada. La recuperación permite “volver hacia atrás” (rollback) partes de una transacción.

Para complicarlo aún más, puede que una transacción implique más de una base de datos, y en ocasiones, a más de un servidor (transacciones distribuidas). MySQL proporciona soporte para todos estos tipos de transacciones, siempre que los gestores de almacenamiento utilizados en las bases de datos implicadas también lo soporten

Storage Engines. InnoDB. Herramientas disponibles.

Motores de almacenamiento

El elemento más notable de la arquitectura de MySQL es la denominada arquitectura de motores de almacenamiento reemplazables (pluggable storage engine architecture). La idea de esa arquitectura es hacer una interfaz abstracta con funciones comunes de gestión de datos en el nivel físico. De ese modo, el gestor de almacenamiento puede intercambiarse, e incluso un mismo servidor MySQL puede utilizar diferentes motores de almacenamiento para diferentes bases de datos o para diferentes tablas en la misma base de datos. Esto permite utilizar el motor de almacenamiento más adecuado para cada necesidad concreta. También permite que terceros puedan implementar motores de

almacenamiento nuevos para necesidades específicas, o adaptar el código de los existentes a ciertos requisitos de almacenamiento. Así, las interfaces definidas por MySQL aíslan el resto de los componentes de la arquitectura de las complejidades de la gestión física de datos, facilitando el mantenimiento de los motores de almacenamiento. También esto permite que ciertos motores de almacenamiento no implementen parte de los servicios, lo cual les hace inapropiados para algunas aplicaciones pero más eficientes para otros. Por ejemplo, un motor de almacenamiento que no implementa bloqueos en la base de datos no debe utilizarse en aplicaciones multi-usuario, pero la ausencia de sobrecarga de procesamiento en la gestión de los bloqueos para el acceso concurrente lo hará mucho más eficiente para una aplicación monousuario.
En consecuencia, una primera tarea de diseño físico en MySQL es la de decidir el motor de almacenamiento más apropiado
Los elementos que puede implementar un motor de almacenamiento son los siguientes:
• Concurrencia. Es responsabilidad del motor implementar una política de bloqueos (o no implementar ninguna). Una estrategia de bloqueos por fila permite una mayor concurrencia, pero también consume más tiempo de procesamiento en aplicaciones en las que la concurrencia no es realmente grande.
• Soporte de transacciones. No todas las aplicaciones necesitan soporte de transacciones.
• Comprobación de la integridad referencial, declarada como restricciones en el DDL de SQL.
• Almacenamiento físico, incluyendo todos los detalles de la representación en disco de la información.
• Soporte de índices. Dado que la forma y tipo de los índices depende mucho de los detalles del almacenamiento físico, cada motor de almacenamiento proporciona sus propios métodos de indexación (aunque algunos como los árboles B casi siempre se utilizan).
• Cachés de memoria. La eficiencia de los cachés de datos en memoria depende mucho de cómo procesan los datos las aplicaciones. MySQL implementa cachés comunes en el gestor de conexiones y la caché de consultas, pero algunos motores de almacenamiento pueden implementar cachés adicionales.
• Otros elementos para ayudar al rendimiento, como puede ser el uso de múltiples hilos para operaciones paralelas o mejoras de rendimiento para la inserción masiva.

Storage Engines
"Pluggable architecture"
-Performance
-Robustez
-Escalabilidad

Los mas populares
-MyISAM
-InnoDB

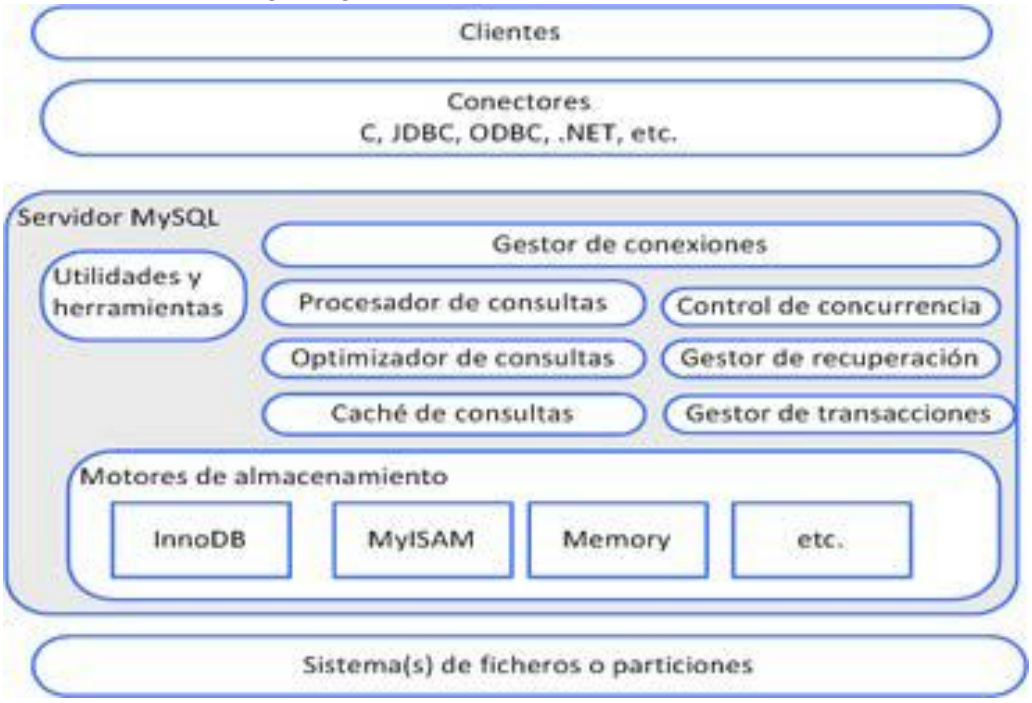
Diferencias
-InnoDB se recupera de un problema volviendo a ejecutar sus logs, mientras que MyISAM necesita repasar todos los indices y tablas que se hayan actualizado y reconstruido
-InnoDB almacena fisicamente los registros en el orden de la clave primaria, mientras que MyISAM lo hace en el orden en el que fueron añadidos

-InnoDB soporta transacciones, MyISAM no

MyISAM
-No soporta transacciones
-No ACID
-Es mas rapida para recuperar registros
-Es mas lenta para insertar y modificar
-Permite busquedas full text

InnoDB storage engine
-Por defecto en MySQL
-ACID transaction
-bloqueo de registros
-Optimiza consultas de PK
-Soporta FK en DDL
-Buffer pool
-Proceso de auto recuperacion

If you try to use a storage engine that is not compiled in or that is compiled in but deactivated, MySQL instead creates a table using the default storage engine. For example, in a replication setup, perhaps your master server uses InnoDB tables for maximum safety, but the slave servers use other storage engines for speed at the expense of durability or concurrency.
By default, a warning is generated whenever CREATE TABLE or ALTER TABLE cannot use the default storage engine.



Programas cliente y servidor

- El servidor MYSQL y los scripts de inicio del servidor:
    - mysqld es el servidor MySQL
    - mysqld_safe, mysql.server, y mysqld_multi son scripts de inicio del servidor
    - mysql_install_db inicializa el directorio "data" y las bases de datos que MySQL instala por defecto.
- Programas cliente que acceden al servidor:
    - mysql (consola) proporciona una interfaz de línea de comandos para ejecutar sentencias SQL en modo interactivo o por lotes.
    - mysqladmin es un cliente para administración.
    - mysqlcheck ejecuta operaciones de mantenimiento de tablas.
    - mysqldump, mysqlbackup y mysqlhotcopy son utilidades para copia de respaldo.
    - mysqlimport importación de ficheros de datos.
    - mysqlshow información relativa a tablas y bases de datos.
    - mysqlslap para testear estres

Se debe controlar el programa mysqld (se debe reiniciar luego de cada modificación en la configuración)
Linux sudo service mysqld (accion) {stop/restart/start}
Windows: Service Control Manager (SCM) o por consola.

Uso basico de la consola:

mysql -u root -p
SHOW TABLES; muestra las tablas de una base de datos
DESCRIBE autos; -> muestra la estructura de la tabla
USE DATABASE ejemplo; -> cambiar la base con la que se esta operando

Cuando usar la consola
-Recursos limitados
-Servidor sin GUI
-Redes lentas, puertos bloqueados, etc

Condiciones especiales
-Tareas de mantenimiento
-Disaster recovery
-Scripting de tareas

Modo de conexión: cliente nativo, ODBC/JDBC
Modos de conexion

Local
-Name pipes (servicio)
-Socket (daemon)

En red
-TCP/IP socket (port 3306)

Interfaces

-Nativa
-Conectores ODBC/JDBC

Modo de conexion
ODBC: Open database conectivity es un estandar de acceso a la base de datos.
JDBC: Java database conectivity es un derivado inspirado en ODBC, que permite la ejecucion de operaciones sobre bases de datos desde el lenguaje JAVA sin importar el SO o la base de datos a la que se accede

Bases de datos y tablas
-Schema
-Datos
-Estructura
-Permisos

Tabla
-Donde se guardan los datos
-Cero o más filas
-Se divide en columnas

Diseño de la Base de Datos: Tablas, Índices, Relaciones. Normas de estilo. Overview de tipos de datos: Numéricos, texto, fecha, otros.

Indices
El índice de una base de datos es una estructura de datos que mejora la velocidad de las operaciones, por medio de un identificador único de cada fila de una tabla

-Se definen dentro de una tabla
-Sobre una o más columnas
-Depende de las queries posteriores

Elementos importantes
-Mejora el rendimiento de las consultas
-Causa frecuente de problemas

Tipos de índices
-Primary
-Define clave primaria
-Unique
-Valores únicos para un valor
-Se permite definir múltiples UNIQUES por tabla
-Foreign key
-Establecen relaciones con otras tablas

Relaciones
-Aseguran consistencia de datos

Tipos

- 1 - 1
- 1 - N
- N - M

Tipos de datos

Numéricos
-Enteros
-Punto fijo
-Punto flotante

tinyint 1
smallint 2
mediumint 3
integer/int 4
bigint 4
float 4
double 8
precision real 8
decimal
numeric

Fecha/hora
-Completa
-Parcial

date (1001/9999)
datetime (fecha/hora 1001/9999)
timestamp fecha/hora 1970/2037
time: HH:MM:SS
year: 1901/2155

Texto
-Corto
-Largo
-Para datos binarios
-De lista

char (longitud fija)
varchar (longitud variable)
enum (campo que puede tener un unico valor de una lista especifica)
set (campo con uno, varios o ningun valor de una lista)

Diferencias, caracteristicas y limitaciones
-Decimal vs Float
-Char vs Varchar / Enum vs SET
-Datetime vs Timestamp

Funcionalidad avanzada: Views, Stored Procedures, Triggers. Ventajas y desventajas.

VIEWS
-Queries guardadas, devuelven result sets
-Una vista actúa como una tabla virtual.
-Vistas: Abstraen sentencias SQL
CREATE VIEW <nombre> AS <consulta>

STORED PROCEDURES
-Procesos de validacion / modificacion / en lote
-Se ejecutan mediante CALL
-No devuelven resultados

• Las definiciones de programas almacenados incluyen un cuerpo que se pueden utilizar sentencias compuestas, bucles, condicionales, y las variables declaradas.

Triggers: Un Trigger en MySQL es un programa almacenado(stored program), creado para ejecutarse automaticamente cuando ocurra un evento en nuestra base de datos. Dichos eventos son generados por los comandos INSERT, Update y DELETE, los cuales hacen parte del DML(Data Modeling Languaje) de SQL. Esto significa que invocaremos nuestros Triggers para ejecutar un bloque de instrucciones que proteja, restrinja o preparen la información de nuestras tablas, al momento de manipular nuestra información. Para crear triggers en MySQL necesitas los privilegios SUPER Y TRIGGER.
Diferencias entre Stored Procedure y Triggers - Los triggers son procedimientos que se ejecutan cuando se produce un cierto evento (insertar, borrar o actualizar una tabla) de forma automatica. - Los procedimientos almacenados son trozos de codigo sql que se ejecutan a traves de una orden del usuario (Exec).

STORED FUNCTION
-Devuelven un valor, se pueden usar dentro de una query

TRIGGERS
-Codigo ejecutable al hacer INSERT/UPDATE/DELETE (antes o despues)

EVENT
-Tareas que se ejecutan en cierta fecha/hora con repetición

Delete vs Truncate
-DELETE FROM <tabla> WHERE <condicion>
-TRUNCATE TABLE <table> => DDL
- En truncate (ddl) no se puede hacer rollback, ademas no muestra la cantidad de registros eliminados
- delete (dml) permite borrado selectivo, truncate borra todo
- truncate no se puede ejecutar si hay relacion a otra tabla.
- se puede activar un trigger en delete, mientras que truncate no

Modificar datos
Update vs SET

-UPDATE <tabla>
-SET campo1 = valor1 campo2 = valor2 WHERE <condicion>

Begin vs Commit
-BEGIN TRANSACTION
-COMMIT (o ROLL-BACK)

A: propiedad que asegura que la operacion sea realizada o no.
C: asegura que se ejecuten aquellas operaciones que no van a romper la reglas de integridad de la BBDD
I: asegura que la realizacion de multiples transacciones sobre un mismo dato sea independientes y no generen errores.
D: asegura que una vez realizada la operacion, esta persistira y no se podra deshacer aunque falle el sistema.
---
archivos de log
por defecto: FILES en directorio data/, puede ser TABLE
Error log
-Errores al iniciar, detener, etc. mysqld, y errores críticos durante la ejecución
-(por defecto)
General query log
-Conexiones y queries hechas por clientes (!)
-general_log
Binary log
-"Eventos" que modifican estructura y/o datos, para recovery/replicaction
-log-bin, binlog-do-db, binlog-ignore-db
Slow log
-Queries que tardan más de lo especificado en long_query_time (seg)
-slow_query_log
Por defecto error.log es el único que viene habilitado.

OPTIMIZACION
es otro de los aspectos para mantener la base de datos, es el mantenimiento preventivo.
Diseño de la BD
-columnas, datatypes, etc
-indices
-storage engines
Configuracion del Servidor
-variables de sistema
Queries
-joins, indices, etc.

-> Concepto de monitoreo. Monitoreo Proactivo y reactivo. Performance tuning. Buenas prácticas. Estado de los procesos. Query Execution Plan (QEP). Uso de EXPLAIN.

2.2 - Cuentas de usuario, roles, privilegios, diseño del esquema de seguridad. Checking

2.2 - Los privilegios pueden ser globales (a todas las BD) o por base de datos o por tabla.
Los privilegios se componen en 3 categorías: datos, estructura, y administración.
Datos: select, insert, update, delete, file.
Estructura: create, alter, drop, show view, triggers, etc.
Administración: grant, super, process, etc.

grant: Un usuario con este privilegio, puede asignar privilegios a otros usuarios.

super: Puede matar procesos, puede conectarse a la base de datos en cualquier circunstancia. (ejemplo mientras esta realiza un backup, o algo por estilo).

process: puede ver los procesos que están realizando todos los usuarios.

Regla general: ala aplicación se le da los privilegios de tipo "Datos".

Sistema de privilegios
puede o no detectar el host de origen
user@localhost / user@10.0.0.4
% como wildcard: user@% /user@192.168.0.%
No permite:
-denegar acceso explícitamente. (usa una lista blanca autorizando a un usuario o host determinado, pero no puede negar a un usuario o host determinado, no acepta lista negra).
-**CREATE/DROP por tabla => DB** (estos permisos se dan para toda la base de datos, no para una tabla específica).
-**contraseñas "por objeto", solo globales** (son contraseñas para todo, no se puede dar contraseñas para una tabla específica).

Tabla mysql.user contiene las cuentas de usuario del motor
-Solo legible por admins!
-Hashes: PASSWORD();
Nunca permisos globales, siempre especificos por BD
-Rol BD <> Usuarios Sistema
-Protip: un rol de BD por App
No dar permisos de GRANT, SUPER, PROCESS, FILE, etc. a menos que sea estrictamente necesario
Privilegios Mysql:
En MySQL existen cinco niveles distintos de privilegios:
Globales: se aplican al conjunto de todas las bases de datos en un servidor. Es el nivel más alto de privilegio, en el sentido de que su ámbito es el más general.
De base de datos: se refieren a bases de datos individuales, y por extensión, a todos los objetos que contiene cada base de datos.
De tabla: se aplican a tablas individuales, y por lo tanto, a todas las columnas de esas tabla.
De columna: se aplican a una columna en una tabla concreta.
De rutina: se aplican a los procedimientos almacenados.

Privilegios en MySQL

Privilege	Column	Context
CREATE	Create_priv	databases, tables, or indexes
DROP	Drop_priv	databases, tables, or views
GRANT OPTION	Grant_priv	databases, tables, or stored routines
LOCK TABLES	Lock_tables_priv	databases
REFERENCES	References_priv	databases or tables
EVENT	Event_priv	databases
ALTER	Alter_priv	tables
DELETE	Delete_priv	tables
INDEX	Index_priv	tables
INSERT	Insert_priv	tables or columns
SELECT	Select_priv	tables or columns
UPDATE	Update_priv	tables or columns
CREATE TEMPORARY TABLES	Create_tmp_table_priv	tables
TRIGGER	Trigger_priv	tables
CREATE VIEW	Create_view_priv	views
SHOW VIEW	Show_view_priv	views
ALTER ROUTINE	Alter_routine_priv	stored routines
CREATE ROUTINE	Create_routine_priv	stored routines
EXECUTE	Execute_priv	stored routines
FILE	File_priv	file access on server host
CREATE TABLESPACE	Create_tablespace_priv	server administration

2.3 – Configuración del servidor. Variables de sistema y de estado. SQL Mode, ajustes de configuraciones más importantes.

CONFIGURACION DEL SERVIDOR:

VARIABLES DE SISTEMA DEL SERVIDOR:

Son variables indican cómo está configurado. Todas ellas tienen valores por defecto pero pueden setearse al arrancar el servidor usando opciones en la línea de comandos o en ficheros de opciones. La mayoría de ellos pueden cambiarse en tiempo de ejecución usando el comando SET.

El servidor mysqld mantiene dos clases de variables. Las variables globales afectan las operaciones globales del servidor. Las variables de sesión afectan las operaciones para conexiones individuales de clientes.

VARIABLES DE ESTADO DEL SERVIDOR:

Son de solo lectura y el servidor las mantiene porque proveen de información sobre sus operaciones. Estas variables y sus valores pueden ser vistas utilizando la sentencia SHOW STATUS. Basicamente contienen contadores y estadísticas sobre el servidor y son útiles para moitorear el rendimiento del mismo.

SQL MODE:

MySQL server puede operar en distintos modos SQL, y puede aplicar estos modos de forma distinta a diferentes clientes. Esto permite que cada aplicación ajuste el modo de operación del servidor a sus propios requerimientos, por lo tanto se considera que por

defecto MySQL resulta demasiado permisivo, por lo que conviene, mediante el comando SET [SESSION|GLOBAL] sql_mode='modes' donde los ‘modes’ mas importantes son:
• ANSI: Cambia el comportamiento y la sintaxis para cumplir mejor el SQL.
• STRICT_TRANS_TABLES: Si un valor no puede insertarse tal y como se da en una tabla transaccional, se aborta el comando. Para tablas no transaccionales, aborta el comando si el valor se encuentra en un comando que implique un sólo registro o el primer registro de un comando de varios registros.
• TRADITIONAL: Hace que MySQL se comporte como un sistema de bases de datos SQL ``tradicional''. Una simple descripción de este modo es `` **da un error en lugar de una alerta''** cuando se inserta un valor incorrecto en la columna. Puede que no sea lo que quiera si está usando un motor de almacenamiento no transaccional, ya que los cambios en los datos anteriores al error no se deshacen, resultando en una actualización ``parcial'.

2.4 Archivos logs

Error log:
- Errores al inicial, detener, etc. mysqld
- Errores críticos durante la ejecución.
- var/log/syslog

General Querylog.
- Conexiones y queries hechas por clientes
- general_log

Binary log:
- Eventos que modifican datos, para recovery/replication
- log_bin, binlog_do_db, binlog_ignore_db

Slow log:
- Queries que tarda más de lo especificado en long_query_log (seg).
- slow_query_log

Por defecto error.log es el único que viene habilitado.

BACKUPS:clasificacion
Metodo
*FISICO vs LOGICO
Disponibilidad
*ONLINE(hot) vs OFFLINE(cold) VS WARM
Ubicacion
*LOCALES VS REMOTOS
Cobertura
*FULL vs INCREMENTALES
EL BACKUP SIEMPRE TIENE QUE VENIR CON UNA RESTAURACION O UNA PRUEBA PARA SABER QUE EL BACKUP FUNCIONE.

Concepto de backup y restore

Es importante realizar una copia de seguridad de sus bases de datos para que pueda recuperar sus datos y estar en funcionamiento de nuevo en caso de problemas se producen, tales como caídas del sistema, fallos de hardware, o los usuarios borrar datos por error. Las copias de seguridad son también esenciales como medida de seguridad antes de actualizar una instalación de MySQL, y que se puede utilizar para transferir una instalación de MySQL a otro sistema.
METODO DE BACKUP: fisico (raw)
se toma el directorio de datos y se comprime

al copiar:
*se hacen copias exactas del directorio data/ de mysql
*puede incluir logs y configuracion
*solo cuando el servidor no esta iniciado
*granularidad: depende del storage engine
*archivos mas pequeños
al restaurar:
*no requieren conversion
*el hardware de destino debe ser identico o similiar
*proceso mas rapido
METODOS DE BACKUP: logico(sql)
al copiar:
*se hacen queries al servidor para obtener la estructura y/o datos, se guardan archivos como .sql
*no incluyen logs ni configuracion
*el servidor puede estar corriendo
*granularidad controlable
*archivos mas grandes
al restaurar:
*requieren conversion
*backups 100% portables entre arquitecturas
*mas lento
DISPONIBILIDAD
Online:
*el backup se realiza con el servidor en funcionamiento
*menos intrusivo para los clientes
*comprobar integridad de datos
offline:
requiere tener el servidor detenido. Por esta razón se puede realizar el backup de una réplica del servidor principal, para no perjudicar a los usuarios.
Warm:
*online + locks
se aplica tanto a backups como a recoveries

Ubicación

Local: la copia de seguridad se inicia en el mismo host donde está corriendo el servidor Mysql, aunque esto se puede hacer logueado localmente de forma remota.

Remoto: se inicia la copia de seguridad desde otro host (logueado ahí, no donde está el servidor corriendo)

ALCANCE DE BACKUPS
tiene que ver con que datos y modificaciones incluyo
Full
-backups completos hasta una cierta fecha
Incremental backups
-desde un punto concreto hacia adelante (o hasta otro punto)
-se hacen habilitando el binary log
Incrremental recovery = "point-in-time"

Segun el alcance el backup e ser completo o incremental
una buena forma es hacer un backup completo con el metodo logico utilizando sql y seguido limpiar los logs binarios (flush Logs)

OPTIMIZACION
es otro de los aspectos para mantener la base de datos, es el mantenimiento preventivo.
Diseño de la BD
-columnas, datatypes, etc
-indices
-storage engines
Configuracion del Servidor
-variables de sistema
Queries
-joins, indices, etc.
BUENAS PRACTICAS

Conocer la configuracion inherente al storage engine
-variables de entorno
-ej innodb_buffer_pool_size

Administración de MySQL

INICIAR/DETENER MySQL

Esto se trada de controlar el servidor MySQL.

Iniciar:
LINUX: sudo service mysql start
Detener:
LINUX: sudo service mysql stop

Mantenimiento y Monitoreo de MySQL

Concepto de monitoreo.
Monitoreo proactivo y reactivo.
Performance tuning. Buenas prácticas.
Estado de los procesos.
Query Execution Plan (QEP). Uso de EXPLAIN.

OPTIMIZACION

DISEÑO DE DB
-columnas, data types (deben ser los mas adecuados), etc
-indices
-storage engines
CONFIG. DEL SERVIDOR
-variables del sistema
QUERIES
-joins, indices, etc
7.1. Panorámica sobre optimización
Para optimizar un sistema el factor mas importante es su diseño, y de acurdo a los procesos que cumple debemos tambien sus cuellos de botella. Estos cuellos de botella pueden ser:

• Búsqueda en Disco.
El disco necesita cierto tiempo para encontrar un paquete de datos. La manera de optimizar el tiempo de búsqueda es distribuir los datos dentro de más de un disco.
• Lectura y escritura en disco.
Cuando el disco se encuentra en la posición correcta, necesitamos leer los datos. Esto es fácil de optimizar, puesto que podemos leer en paralelo desde múltiples discos.
• Ciclos de CPU.
Cuando tenemos datos en la memoria principal, necesitamos procesarlos para obtener algún resultado. Tener tablas pequeñas en comparación con la cantidad de memoria es el factor más común de limitación. Pero con tablas pequeñas, la rapidez no es usualmente el problema.
• Ancho de banda de Memoria.
Cuando el CPU necesita más datos de los que puede almacenar en la cache de la CPU, el ancho principal de la memoria se convierte en un cuello de botella. Es poco común en la mayoría de casos, pero debe tenerse en cuenta.
Optimización del tamaño de los datos
MySQL soporta muchos motores de almacenamiento diferentes tipos de tabla) y formatos de fila. Para cada tabla, usted puede decidir qué almacenamiento y método de indexación usar. Elegir el formato de tabla adecuada para su aplicación puede dar una ganancia de gran rendimiento
DISEÑO DE BASES
Usted puede obtener un mejor rendimiento para una tabla y reducir al mínimo el espacio de almacenamiento mediante el uso de las técnicas que se enumeran aquí:
COLUMNAS DE TABLA
• Utilize los mas eficientes tipos de datos. Por ejemplo en un dato entero, si no es necesario usar un INT utilize MEDIUMINT, esto utilizaria un 25% menos de espacio.
• En lo posible declarar las columnas como **NOT NULL,** esto agiliza las operaciones SQL, y permite un mejor uso de los indices(solo si es necesario, utiliza **NULL**)
FORMATO DE FILA
• Las tablas InnoDB utilizan un formato de almacenamiento compacto.
• Para minimizar espacio aún más mediante el almacenamiento de datos de la tabla en forma comprimido especificar ROW_FORMAT = COMPRESSED cuando se crea tablas InnoDB, o ejecute el comando myisampack en una tabla MyISAM existente.
• Para tablas MyISAM, si no tienen alguna columna de longitud variable (VARCHAR, TEXT, o BLOB), se usa un formato de fila con tamaño fijo.
INDICES
• El índice principal de una tabla debe ser tan corto como sea posible. Esto hace que la identificación de cada fila fácil y eficiente.
• Cree sólo los índices que usted necesita para mejorar el rendimiento de la consulta. Indices son buenos para la recuperación, pero ralentiza la inserción y actualización de las operaciones.
INDICES-DEFINICION
La mejor manera de mejorar el rendimiento de las operaciones de SELECT es la creación de índices en una o varias de las columnas que se ponen a prueba en la consulta. Las entradas de índice actúan como punteros a las filas de la tabla, permitiendo la consulta para determinar rápidamente qué las filas coinciden con una condición en la cláusula WHERE,

Usted debe encontrar el equilibrio adecuado para lograr consultas rápidas que utilizan el conjunto óptimo de índices.
COMO UTILIZA MYSQL LOS INDICES
- Para buscar las filas que coincidan con una cláusula WHERE con rapidez.
- Para eliminar las filas de consideración. Si hay una elección entre varios índices, MySQL wen las columnas de manera más eficiente si se declaran como del mismo tipo y tamaño. En este contexto, VARCHAR y CHAR se consideran la misma si se declara como el mismo tamaño. Por ejemplo, VARCHAR (10) y CHAR (10) son del mismo tamaño, pero VARCHAR (10) y CHAR (15) no lo son.

EXPLAIN nombre_de_tabla
O:
EXPLAIN SELECT opciones_de_select
La sentencia EXPLAIN puede utilizarse como un sinónimo de DESCRIBE o también como una manera para obtener información
acerca de cómo MySQL ejecuta una sentencia SELECT:
- EXPLAIN nombre_de_tabla es sinónimo de DESCRIBE nombre_de_tabla o SHOW COLUMNS FROM nombre_de_tabla.
- Cuando se precede una sentencia SELECT con la palabra EXPLAIN, MySQL muestra información del optimizador sobre el plan de ejecución de la sentencia, proporcionando también información acerca de cómo y en qué orden están unidas (join) las tablas.

EXPLAIN es una ayuda para decidir qué índices agregar a las tablas, con el fin de que las sentencias SELECT encuentren registros más rápidamente. EXPLAIN puede utilizarse también para verificar si el optimizador une (join) las tablas en el orden óptimo. Si no fuera así, se puede forzar al optimizador a unir las tablas en el orden en el que se especifican en la sentencia SELECT empezando la sentencia con SELECT STRAIGHT_JOIN en vez de simplemente SELECT.
Si un índice no está siendo utilizado por las sentencias SELECT cuando debiera, debe ejecutarse el comando ANALYZE TABLE, a fin de actualizar las estadísticas de la tabla como la cardinalidad de sus claves, que pueden afectar a la decisiones que el optimizador toma
EXPLAIN muestra una línea de información para cada tabla utilizada en la sentencia SELECT. Las tablas se muestran en el mismo orden en el que MySQL las leería al procesar la consulta. MySQL resuelve todas las uniones (joins) usando un método de single-sweep multi-join.
EXPLAIN retorna una tabla; cada línea de esta tabla muestra información acerca de una tabla

Mantenimiento proactivo: se busca el por que de la falla además de arreglarse
Mantenimiento reactivo: se arregla cuando se encuentra una falla

SHOW PROCESS_LIST

Muestra las conexiones al motor de bases de datos. Cada fila es una conexión
Se listan: id de proceso, usuario, host y puerto, base de datos, comando y tiempo
Según los permisos se pueden matar los procesos

The MySQL Performance Schema is a feature for monitoring MySQL Server execution at a low level.
Guarda datos sobre cualquier evento que le lleve tiempo al servidor.

Particionado

Se usa para reducir la cantidad de lecturas físicas a la base de datos cuando ejecutamos consultas

permite distribuir porciones de tablas en un sistema de ficheros de acuerdo a reglas definidas por el usuario para ajustarse a sus necesidades.

Tipos de particionado
Estas reglas reciben el nombre de funciones de particionado y existen varios tipos de funciones distintas: particionado por rangos o listas de valores, funciones hash internas o lineales y por clave

Modalidades
Existen dos principales modalidades de particionado: horizontal y vertical

Horizontal: consiste en tener varias tablas con las mismas columnas en cada una de ellas y distribuir la cantidad de registros (generalmente por años, meses, etc)

Vertical: generalmente la aplicamos sin darnos cuenta, por ejemplo cuando tenemos una columna de tipo BLOB con una foto que no leemos frecuentemente y la ponemos en otra tabla referenciándola con la clave foránea.

Cuando al proceso de particionado vertical le sigue uno horizontal, es decir, se particionan horizontalmente los fragmentos verticales resultantes, se habla de la partición mixta HV. En el caso contrario, estaremos ante una partición VH.

El único requisito necesario para utilizar el particionado es que los ejecutables de MySQL estén compilados con soporte para particionado

Ventajas

optimización a la hora de acceder a los datos, ya que podemos unificar datos comunes en las mismas particiones. En otras palabras, al realizar una consulta a una tabla particionada, en lugar de buscar por toda la tabla, reduciremos la búsqueda a aquellas particiones en las que sepamos que hay datos que nos interesen

Otra ventaja del particionado es que nos aporta mucha flexibilidad si pensamos en escalar nuestra base de datos, ya que por ejemplo, podemos guardar cada partición en distintos discos físicos.

Limitaciones

Debemos utilizar el mismo motor de almacenamiento para todas las particiones de la misma tabla

El tamaño máximo de particiones (incluyendo subparticiones) para una tabla es de 1024.

Los índices *FULLTEXT* no están soportados en tablas particionadas.

Los tipos de datos *POINT* o *GEOMETRY* no se pueden utilizar en tablas particionadas.

El tipo de datos de una clave que se use para el particionado debe ser un entero o una expresión que devuelva un entero como resultado o bien *NULL*.

Ni las tablas temporales ni las tablas de *logs* se pueden particionar.

Una clave de particionado no puede ser una subconsulta.

Replicacion

La replicación es el proceso de copiar y mantener actualizados los datos en varios nodos de bases de datos ya sean estos persistentes o no.

La replicación permite que los datos de un servidor (máster) de bases de datos, sea replicado a uno o más servidores (slaves). La replicación es asíncrona por defecto, por lo tanto, los servidores esclavos no necesitan estar conectados permanentemente para recibir actualizaciones desde el servidor maestro. Esto significa que las actualizaciones pueden ocurrir sobre conexiones a larga distancia e incluso sobre conexiones temporales o intermitentes, como servicio de acceso teléfonico. Dependiendo de la configuración, se pueden replicar todas las bases de datos, sólo las bases de datos seleccionadas o incluso algunas tablas de una base de datos.

Ventajas

Repartir la carga entre varios servidores esclavos para mejorar el rendimiento. En este entorno, todas las escrituras y actualizaciones deben tomar lugar en el servidor maestro. Las lecturas, sin embargo, pueden tomar lugar en uno o más servidores esclavos. Este modelo puede mejorar el rendimiento de las escrituras (ya que el servidor maestro está dedicado a las actualizaciones), a su vez incrementando dramáticamente la velocidad de lectura a través del aumento de la cantidad de servidores esclavos.

Se pueden crear datos en vivo en el servidor maestro, mientras el análisis de la información puede tomar lugar en el esclavo sin afectar el rendimiento del servidor maestro

Si una sucursal quisiera trabajar con una copia de sus datos principales, puede utilizar replicación para crear una copia local de los datos para su uso sin necesidad de acceso permanente al maestro.

Contras

El tener que replicar el binary log a muchos esclavos puede reducir el rendimiento del servidor maestro.

Tipos de replicación

La replicación en MySQL por defecto es asíncrona. Con replicación asíncrona, si el maestro falla, las transacciones que ha cometido tal vez no se hayan transmitido a ningún esclavo.

Se puede usar replicación semi sincrónica como alternativa
Un esclavo indica si soporta replicación semi sincrónica cuando se conecta al maestro

Si la replicación semi sincrónica está habilitada en el servidor maestro y hay, aunque sea un esclavo semisíncrono, un hilo que comete una transacción en maestro se bloquea después de que el commit fue hecho, y espera hasta que al menos un esclavo semisíncrono reconoce que ha recibido todos los eventos para la transacción, o hasta que se supera un timeout.

El esclavo acusa recibo de los acontecimientos de una transacción sólo después de los eventos se han escrito a su relay log y volcado a disco.
Si se produce un tiempo de espera sin que ningún esclavo haya reconocido la transacción, el maestro vuelve a la replicación asincrónica. Cuando al menos un esclavo semisincrónico se pone al día, el servidor maestro vuelve a la replicación semisincrónico.

La replicación semisincrónica debe estar habilitado en ambos lados maestro y esclavo. Si la replicación semisincrónica está desactivada en el maestro, o habilitado en el maestro, pero en ningún esclavo, el maestro utiliza la replicación asincrónica.

Si se produce un tiempo de espera sin que ningún esclavo haya reconocido la transacción, el maestro vuelve a la replicación asincrónica. Cuando al menos un esclavo semisincrónico se pone al día, el servidor maestro vuelve a la replicación semisincrónico.

Formatos de replicación
Statement-based replication
se corresponde con el estándar de formato de binary logging basado en sentencias.

Row-based replication
En este formato de replicación, el servidor maestro escribe eventos al binary log que indican cómo han cambiado filas individuales de una tabla.

Mixed-based replication
Cuando el formato mixto está en efecto, los registros basados en sentencias son usados por defecto, pero automáticamente cambiados a registros basados en filas en casos particulares.

metadata tribbles
objetivo mejorar el rendimiento de acceso a tablas de mucho tamaño
ANTIPATRON
*SEPARAR LA TABLA EN VARIAS CON LA MISMA ESTRUCTURA
*SE DEBE CREAR UNA NUEVA TABLA POR CADA NUEVO VALOR
EJ:

```
CREATE TABLE bugs_2005(..)
CREATE TABLE bugs_2006(..)
CREATE TABLE bugs_2007(..)
CREATE TABLE bugs_2008(..)
```

no se puede aplicar id's autoincrementales, sino que se debe incrementar manualmente.

El mismo dato esta en varias tablas
Dificulta las consultas, ya que en lugar de hacerla en una sola tabla hay que hacerlas en varias tablas.
Las estructuras no se mantienen sincronizadas.
SOLUCION #1
*Particionado horizontal, se deja al motor de base de datos que realice particiones mediante criterios definidos en la creacion de tablas.
SOLUCION #2
* particionado Vertical se mueven las columnas pesadas o las menos usadas a otra tabla con una relacion 1:1
Las columnas tambien pueden ser tribbles

```
CREATE TABLE xxxx(
        id_1
        ID_2
        ID_3
);
```

ANTIPATRON ENTIDAD ATRIBUTO VALOR
OBJETIVO utilizar filas de una base de datos para representar una base de datos, hacer una table con un conjunto variable de atributos.
Guargar todos los atributos en una segunda tabla, uno por fila.
*DIFICULTAD para confiar en los nombres de atributos
*DIFICULTAD en la integridad de tipos de datos
*DIFICULTAD para diferenciar entre los atributos que son obligatorios y no obligatorios los contraints sql se aplican a todas las filas no a filas seleccionadas dependiendo del valor en nombre de atributo
*DIFICULTAD para recontruir una fila de atributos, hacer la inversa, para poder generar la tabla. Se necesita realizar un join.
SOLUCION
* usar los metadatos para metadatos
* definir atributos en columnas
SOLUCION #1 Herencia de tabla simple
*una tabla con muchas columnas
SOLUCION #2 Herencia de tabla concreta
definir tablas similares para tipos similares
duplicar columnas en comun en cada tabla
utilizar union para buscar en las dos tablas
SOLUCION #3 Herencia de clases
columnas comunes en la tabla base
columnas de Subtipos especificos en tablas de subtipo
es facil buscar en columnas en comun
USOS APROPIADOS
si los atributos tienen que ser flexibles y dinamicos
realizar las constraints en la aplicacion
ASOCIACIONES POLIMORFICAS
OBJETIVO: referenciar a multiples padres
una foreing key no puede referenciar a dos tablas
se puede definir la tabla sin identidad referencial
no se puede usar una tabla diferente para cada fila

no se puede hacer un join a una sola tabla
SOLUCION #1
Arcos exclusivos
se generan dos ids, uno para cada tabla padre que puedan ser nulables y por codigo se debe controlar que si uno es nulo el otro no lo puede ser. No pueden ser nulos los dos Ids.
SOLUCION #2
Invertir la relacion
generar una tabla intermedia que relacione la tabla polimorfica con cada uno de sus padres
SOLUCION #3
Usar una tabla padre con los campos en comun y sobre esa tabla padre asociamos la polimorfica
Se puede aplicar integridad referencial y las queries som mas simples
Naive Trees - ANTIPATRONES
Lista adyacente
es un patron que lo que hace es poner un ID de la tabla padre en una relacion circular
cada entrada en el arbol conoce a su padre inmediato
* es facil insertar un registro nuevo,
* es facil mover un subarbol a un nivel superior o inferior
* es facil pedir los hijos inmediatos de un nodo
* es facil conocer el padre del nodo
* es dificil en una sola query obtener todos los descendientes
SOLUCION #1
*path enumeration - guardar el camino en un string
- almacenta guarda la cadena de ansestros como un string en cada nodo
- es bueno para "breadcrumbs" y ordenamiento por jerarquias
- es facil para pedir todos los ancestros
- es facil obtener los descendientes
- es facil agregar hijos, se necesitan de tres querys, (guardar el registro, ubicar el id del padre,actualizar el registro)
- no hay integridad referencial por lo que eliminar un nodo con sus hijos require de una query especial
* nested sets - guardar un valor izquierdo y un valor derecho
- cada nodo identifica a sus descendientes con dos numeros
- el numero derecho es mayor que todos los numeros del lado izquierdo
- es facil preguntar por un ancestro
- es facil pedir los descendientes
- una insercion requiere una actualizacion completa del arbol
- es dificil obtener el padre inmediato
Closure tables
- guarda todas las rutas desde los ancestros a los descendientes
- guarda una tabla con nodos y en otra tabla gurda las relaciones entre nodos
- facil para pedir los descendientes
- facil para pedir los ascendientes
- para insertar se necesitan tres querys
- para borrar un hijo tambien es facil y permite integridad referencial