Optimización en MySQL



Bases de datos II

3º - Grado en Ingeniería Informática

Índice de Contenidos

[Introducción 3](#_Toc5994655)

[Optimizar a nivel de base de datos 3](#_Toc5994656)

[Optimizar a nivel de hardware 3](#_Toc5994657)

[Procesos de optimización implementados por MySQL 3](#_Toc5994658)

[Sentencia EXPLAIN 3](#_Toc5994659)

[Motores de almacenamiento MyISAM e InnoDB 5](#_Toc5994660)

[Índices 7](#_Toc5994661)

[Sentencia OPTIMIZE TABLE 9](#_Toc5994662)

[Sentencia INSERT DELAYED 9](#_Toc5994663)

[Slow Query Log 9](#_Toc5994664)

[Variables de sistema: long-query-time 10](#_Toc5994665)

[Show ProcessList 10](#_Toc5994666)

[Conclusiones 10](#_Toc5994667)

[Bibliografía 11](#_Toc5994668)

# Introducción

El rendimiento de una base de datos se basas en diversos factores, como las tablas, sentencias y la configuración que utiliza, siendo estos factores a nivel de la propia base de datos. Además, se requiere de una optimización a nivel de hardware, minimizando y haciendo lo más eficiente posible las operaciones de la CPU y las de entrada y salida.

## Optimizar a nivel de base de datos

Las tablas deben estar estructuradas correctamente: que las columnas tengan el tipo de dato adecuado y que cada tabla tenga un número apropiado de columnas.

Uso correcto del indexado, lo que se abordará posteriormente en detalle.

Para favorecer el rendimiento y la escalabilidad se debería usar el motor de almacenamiento más adecuado para cada tabla, siendo en ocasiones mejor el uso de motores transaccionales y, en otros casos, los motores no transaccionales.

## Optimizar a nivel de hardware

A medida que la base de datos se usa con mayor asiduidad, esta podría alcanzar su límite hardware, siendo necesario modificar la aplicación de la base de datos o reconfigurar el servidor de llegar a este punto. Los cuellos de botella de los sistemas suelen ser: las búsquedas en disco, la lectura y escritura en disco, los ciclos del procesador o, aunque en menor medida, el ancho de banda de la memoria.

# Procesos de optimización implementados por MySQL

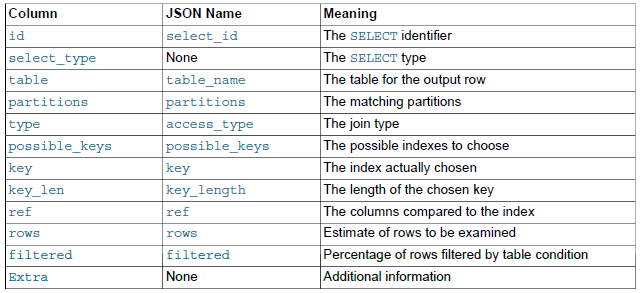
## Sentencia EXPLAIN

Muchas técnicas son consideradas por MySQL para la ejecución eficiente de una consulta SQL, la elección de estas depende de diversos factores como los detalles de las tablas, las columnas, los índices y las condiciones de la cláusula WHERE. No por realizar una consulta en una tabla gigante se han de leer todas las filas de esta para llegar a la solución, un join que atañe a muchas tablas puede ejecutarse sin comparar todas las filas de estas, etc. Estas estrategias que MySQL decide tomar se llaman “El plan de ejecución de la consulta” o el plan EXPLAIN.

La sentencia EXPLAIN proporciona información sobre como MySQL ejecuta sentencias, en concreto las sentencias SELECT, DELETE, INSERT, REPLACE y UPDATE. Cuando se utiliza EXPLAIN con una sentencia explicable, MySQL muestra información desde el optimizador explicando cómo se procesaría la sentencia, incluyendo información sobre las cómo se unen las tablas y en qué orden.

Con la ayuda de EXPLAIN se puede ver donde se deberían añadir índices en las tablas de forma que la ejecución de la consulta se vea acelerada al utilizar índices para encontrar filas. También es útil para comprobar si el optimizador está uniendo tablas en el orden óptimo.

EXPLAIN retorna una fila de información por cada tabla utilizada en la sentencia SELECT, estas filas están ordenadas de la misma forma en que MySQL las leería mientras procesa la sentencia.

A continuación, podemos ver las columnas que la sentencia EXPLAIN produce como salida.

Esta tabla nos proporciona gran cantidad de información de cada una de las sentencias que ha de ejecutar MySQL. Todas las columnas son de gran utilidad en ciertas situaciones, pero hay ciertas columnas que tienen un interés constante de cara al interesado en optimizar manualmente una consulta, estas columnas son “type” y “rows”.

Type: Esta columna nos indica el tipo de join que se esta llevando a cabo entre las tablas de una consulta, por ejemplo, si una fila tiene el valor ALL en esta columna, significa que se esta llevando a cabo un producto cartesiano entre las tablas de la consulta y todas las demás filas de EXPLAIN que representan al resto de tablas también contarán con dicho valor en esta columna. Ya sabemos como de ineficientes son los productos cartesianos por lo que el programador puede reaccionar activamente a este output y modificar debidamente la consulta de forma que este campo tome un valor más razonable como por ejemplo “range” que limita las filas involucradas a un rango.

Rows: Esta columna va de la mano de la anteriormente mencionada type, aquí podremos encontrar el número de filas involucradas en la ejecución de la sentencia, es una muy buena representación visual de como de eficiente es una consulta y un buen dato a usar como comparación a la hora de modificar una consulta buscando la optimización de esta.

EXPLAIN además de toda la información mencionada anteriormente puede ofrecernos algo más de información extra si se ejecuta una sentencia SHOW WARNINGS justo después de la ejecución de una sentencia EXPLAIN, esta información extra está disponible para sentencias SELECT, DELETE, INSERT, REPLACE y UPDATE (A partir de la versión 8.0.12, previamente esta sentencia solo estaba disponible para los SELECT). Esta sentencia nos proporcionará notas y consejos sobre como califica el optimizador los nombres de nuestras tablas y columnas en la sentencia SELECT, también nos mostrará que apariencia tendría dicho select después de una reescritura y aplicación de los criterios de optimización, dicha sentencia de salida no tiene por que tener un formato ejecutable, si no mas bien informativo ya que contendrá mensajes y anotaciones “no-SQL”.

## Motores de almacenamiento MyISAM e InnoDB

InnoDB es el motor de almacenamiento que usan habitualmente los usuarios de MySQL, enfocándose en la fiabilidad y concurrencia de las bases de datos. Existen ciertas consideraciones a tener en cuenta a la hora de optimizar una base de datos que usa este motor de almacenamiento:

En cuanto a la optimización de tablas:

* Si detectamos que una tabla está creciendo de forma considerable, se debería considerar la posibilidad de usar la sentencia OPTIMIZE TABLE, que trataremos en detalle posteriormente. Así reorganizaremos la tabla y compactaremos cualquier espacio usado. Se trata de una medida muy directa y que mejorará el rendimiento de forma considerable.
* Tener una clave primaria larga, tanto en una sola columna como componiendo varias columnas, malgasta mucho espacio. En su lugar se recomienda utilizar claves primarias numéricas autoincrementales.
* Para almacenar cadenas de longitud variable mejor usar varchar en vez de char, ya que char va a tener un tamaño fijo incluso cuando la cadena es más pequeña o nula.
* Para tablas grandes con mucho texto y datos numéricos repetitivos se recomienda usar el formato de tupla COMPRESSED.

En cuanto a la optimización del gestor de transacciones hay que encontrar el equilibrio entre el rendimiento de las características transaccionales y la carga de trabajo de tu servidor.

* Con una base de datos sobrecargada quizás interese establecer autocommit a 0 para poder fusionar varias operaciones de modificación relacionadas en una sola transacción y después hacer commit.
* Se debería evitar realizar un rollback después de insertar, modificar o eliminar una gran cantidad de tuplas.
* Cuando una o varias columnas son eliminadas o modificadas, las propias columnas y sus archivos de recuperación asociados no son físicamente eliminados, si no que se guardan hasta que las transacciones que empezaron antes y todavía no han acabado, finalicen.

En cuanto a transacciones de solo lectura, las transacciones de escritura o las de bloqueo de escritura tienen un id, por lo tanto, eliminar los id de transacción restantes reducirá el tamaño interno de las estructuras consultadas. InnoDB detecta las transacciones de solo lectura cuando empiezan por START TRANSACTION READ ONLY, cuando autocommit está activo o cuando la transacción no empieza por READ ONLY, pero no realiza modificaciones ni bloqueos de lectura.

Hay que optimizar también los logs de recuperación, siguiendo algunas pautas:

* Hacer los ficheros de logs de recuperación grandes, al menos tanto como el búfer compartido. De no ser así se realizarían demasiadas escrituras en disco innecesarias.
* Aumentar el tamaño del búfer de log. Con ello se podrán ejecutar transacciones mayores sin la necesidad de escribir en el log antes de que la transacción realice el commit.

Para mejorar las entradas y salidas de disco:

* Aumentar el tamaño del búfer compartido.
* Ajustar el método de flush.
* Configurar un tamaño umbral para escritura en el búfer.

El motor de almacenamiento MyISAM tiene un mejor rendimiento con operaciones de leer la mayoría de los datos o aquellas con un grado de concurrencia bajo, porque las tablas tienen un bloqueo de que limita la capacidad de realizar varias modificaciones simultáneas.

Consejos para optimizar tablas MyISAM para mejorar el rendimiento de las consultas:

* Usar la sentencia ANALYZE TABLE después de cargar datos a la tabla.
* Evitar sentencias SELECT en tablas que son modificadas con frecuencia para evitar problemas de bloqueos de tabla.
* Realizar inserciones múltiples y simultáneas de tuplas y evitar eliminar tuplas, o usar OPTIMIZE TABLE después de borrar una tupla.
* En tablas que son modificadas habitualmente hay que evitar tener columnas con un tipo de datos de longitud variable.
* No suele ser útil dividir una tabla en varias por el hecho de que las tuplas se están volviendo grandes.
* Si se necesita hacer cálculos habituales con una tabla con muchas tuplas, podría ser preferible introducir una nueva tabla que se encargue exclusivamente de ello.
* Usar periódicamente OPTIMIZE TABLE para evitar la fragmentación en las tablas con formato dinámico.

En cuanto a inserciones de datos:

* A una tabla MyISAM se le puede realizar inserciones concurrentes para añadir múltiples tuplas al mismo tiempo que se están ejecutando sentencias select.
* Es posible conseguir que LOAD DATA vaya más rápido cuando tiene muchos índices: realizando un flush a la tabla, eliminando todo uso de índices, insertar los datos con LOAD DATA, si solo pretendes leer de la tabla en el futuro convendría comprimirla, volver a crear los índices y realizar otro flush.
* Para incrementar la velocidad de ejecución de las inserciones que se realizan con múltiples sentencias para tablas no transaccionales hay que bloquear estas.
* Para mejorar el rendimiento de INSERT y de LOAD DATA conviene aumentar el tamaño de la caché de claves.

REPAIR TABLE en MyISAM es similar a usar myisamchk con el fin de reparar operaciones.

## Índices

Los índices en MySQL son utilizados para encontrar filas con valores de columnas específicos rápidamente, sin ellos, MySQL tendría que empezar por la primera fila e ir leyendo secuencialmente la tabla completa hasta encontrar las filas necesarias. Naturalmente, a mayor tamaño tenga la tabla en cuestión, mayor es el coste de este procedimiento, por lo que el uso de índices para facilitar esta búsqueda se vuelve muy importante.

La creación de índices en una o varias de las columnas objetivo de una consulta es la mejor manera de mejorar el rendimiento de las operaciones SELECT. Estos índices actúan como puntero a las filas de la tabla, permitiendo a la consulta determinar rápidamente que filas cumplen cierta condición en la cláusula WHERE y la recuperación del resto de información de dichas filas. Tampoco es bueno crear índices sin control en todas las columnas de una consulta ya hacen a MySQL gastar espacio y tiempo en determinar que índices se utilizarán.

MySQL utiliza los índices para operaciones muy diversas:

* Para encontrar filas que cumplan la cláusula WHERE rápidamente.
* Para evitar considerar ciertas filas, MySQL procura utilizar el índice más selectivo.
* Si la tabla tiene índices multi-columna
* Los índices pueden ser utilizados por MySQL para devolver filas de otras tablas cuando se hacen joins. En este caso son más eficientes si están declarados del mismo tipo y tamaño.
* Para encontrar el MAX() y MIN() de una columna indexada específica.
* En algunos casos una consulta se puede optimizar de manera que devuelva valores sin ni siquiera consultar las filas de datos (Un índice que provee todos los resultados necesarios para una consulta se llama un “covering index”).

Los índices se encargan de optimizaciones de todo tipo en una base de datos MySQL, a continuación, comentaremos algunas de ellas.

La más remarcable seguramente sea la optimización de clave primaria. La clave primaria de una tabla representa la columna o columnas que vas a utilizar en las consultas más vitales, estas columnas tienen un índice asociado para mejorar el rendimiento de consulta, esta mejora de rendimiento también se beneficia en parte por la condición de NOT NULL que tiene la clave primaria. Con el motor de almacenamiento InnoDB en concreto, las tablas de datos están físicamente organizadas para hacer consultas y ordenaciones ultrarrápidas basándose en la clave primaria de estas tablas.

Los índices también son responsables de la optimización con índices espaciales, esto es, permite la creación de índices de tipo espacial (SPATIAL) en columnas geométrico-valuadas no nulas. El optimizador comprueba el atributo SRID para determinar que sistema de referencia espacial utilizar para las comparaciones, estas comparaciones solo pueden llevarse a cabo entre columnas con índices espaciales y SRID-restringidas, lo que quiere decir que deben de tener un SRID explícito, además el valor de su SRID debe coincidir. El optimizador ignora los índices de las columnas no SRID-restringidas, pero aun así MySQL los conserva.

La optimización de clave foránea también se aprovecha del uso de índices. Si se diese el caso en el cual se ha de trabajar con tablas con un gran número de columnas y se quieren incluir gran parte de estas en una consulta, a veces es mejor proceder a la separación de los datos menos usados en subtablas replicando el atributo ID de la tabla padre, disponiendo así de una clave primaría en cada tabla pequeña acelerando las consultas y así podemos consultar solo las columnas necesarias utilizando una operación join.

Los índices de una sola columna son el tipo de índice más común, el almacenado de una copia de los valores de dicha columna en una estructura de datos permitirá hacer búsquedas rápidas de las filas que contengan dichos valores. La estructura de datos B-tree permite al índice encontrar rápidamente un valor, un set de valores o un rango correspondientes a los operadores tales como “<, >=, =, BETWEEN, IN y más en una cláusula WHERE.

MySQL también puede crear índices compuestos, es decir, índices multi-columna los cuales pueden llegar a consistir de hasta 16 columnas. Estos índices pueden usarse para consultas individuales de una sola de las columnas a las que afectan o varias de forma que un solo índice compuesto puede acelerar muchas clases de consultas, estos índices pueden considerarse como arrays ordenados cuyas filas contienen valores creados mediante la concatenación de los valores de las diferentes columnas.

Por último, echaremos un vistazo a los dos principales tipos de estructuras de datos para el almacenamiento y gestión de los índices, el uso de estos depende directamente del motor de almacenamiento que estemos utilizando, InnoDB utiliza B-trees y MEMORY utiliza índices hasheados (aunque puede cambiar entre ambos).

Los B-trees o árboles-B como comentamos anteriormente pueden utilizarse para la comparación de columnas con diversos operadores, a esos se suma la sentencia LIKE en casos en los que el argumento a comparar sea una cadena que no comience por un carácter comodín. A destacar diremos que en este tipo de árboles MySQL no utilizará índices que no afecten a todos los grupos AND que se encuentren dentro de la cláusula WHERE.

Los índices hasheados son algo diferentes de los discutidos anteriormente, solo pueden utilizarse para comparaciones de igualdad que utilicen los operadores = o <=>, siendo estas extremadamente rápidas.

## Sentencia OPTIMIZE TABLE

La sentencia OPTIMIZE TABLE reorganiza el almacenamiento físico de una tabla de datos y los datos indexados asociados para reducir el espacio de almacenamiento y mejorar la eficiencia entrada/salida cuando se accede a la tabla. Los cambios exactos que se realizan sobre la tabla dependen directamente del motor de almacenamiento usado.

Cuando el tamaño de la cantidad de datos en las tablas se estabiliza o cuando estos crecen en decenas o cientos de megabytes es interesante considerar la ejecución de esta sentencia para compactar, optimizar y ahorrar espacio. Esta simple técnica puede mejorar todas las operaciones que envuelvan a esta tabla en ellas y funciona en todas las ocasiones a diferencia de la optimización de índices o el ajuste de consultas manual, técnicas que, aunque útiles hay situaciones en las que pueden presentarse no tan prácticas.

## Sentencia INSERT DELAYED

A la hora de optimizar sentencias INSERT se deberían fusionar múltiples operaciones pequeñas en una sola; realizar el menor número de conexiones con la tabla donde se van a insertar las nuevas tuplas y mandar el mayor de estas de una sola vez, retrasando la actualización de índices y la comprobación de consistencia.

Para mejorar la velocidad de las sentencias INSERT hay que usar sentencias INSERT con múltiples listas de VALUE cuando se desee insertar varias tuplas a la vez, usar la sentencia LOAD DATA cuando se cargue una tabla de un archivo de texto o insertar valores explícitamente solo cuando el valor a insertar es distinto del valor por defecto de la columna.

La opción DELAYED es una extensión de MySQL al estándar SQL, que se encuentra actualmente obsoleta desde MySQL 5.6. Anteriormente, cuando un cliente usaba esta sentencia sobre una tabla, la tupla se almacenaba en una cola para insertarse cuando esta tabla no estuviera en uso.

En MySQL 8.0 el servidor reconoce la opción DELAYED, pero la ignora y trata la sentencia como un simple INSERT. Se planea eliminar esta palabra reservada en futuras versiones.

## Slow Query Log

La variable slow query log, de ser activada, sirve para encontrar las sentencias que mayor tiempo de ejecución requieren y que son las principales candidatas para ser optimizadas. Para ello busca las sentencias que llevan más de una cantidad de segundos y que requieren de al menos un número concreto de tuplas para ser revisadas.

Estas sentencias lentas se guardarán para poder ser revisadas en un archivo de log que, de no ser especificada su ubicación, se situará en *nombre\_host*-slow.log en la carpeta de datos del servidor. Además, la variable de estado slow queries se incrementa por cada sentencia lenta.

En el log encontraremos la sentencia ejecutada, seguida de una # y en una línea el tiempo de ejecución de la sentencia, el tiempo de bloqueo, las tuplas examinadas y las que se han mandado al cliente.

Si activáramos la variable log slow extra nos aparecerían en el log muchos más datos relacionados con la sentencia ejecutada, como el número de error que produce, los bytes recibidos, o la hora en la que empezó y acabó, entre otros.

## Variables de sistema: long-query-time

Recientemente explicábamos la variable slow query log, relacionada íntimamente con long query time, debido a que esta última es el número mínimo de segundos que tardarán en ejecutarse las sentencias que se guardarán en el log.

El valor por defecto que recibe es de 10 segundos y el mínimo es 0. Por otro lado, se puede especificar en microsegundos si se requiere de ello.

Este valor se compara en tiempo de ejecución, no de procesado, por lo que una sentencia que está por debajo del umbral en un sistema con poca carga y que, por lo tanto, no entraría en el log, podría superar ese umbral en un sistema con una carga mayor.

## Show ProcessList

Esta sentencia muestra los hilos actualmente en ejecución. Se requiere de permisos para poder ver todos los hilos, de no ser así solo se pueden ver los relacionados con tu usuario.

Se trata de una sentencia muy útil para ver qué ocurre cuando se dan ciertos errores, como el de *demasiadas conexiones*.

Para cada hilo se muestran una serie de datos, como son el id del hilo, el nombre del usuario, que puede ser de un cliente o de un usuario del sistema; el nombre del host y el puerto del cliente que ejecuta una sentencia, la base de datos por defecto, el comando que está ejecutando el hilo, el tiempo en segundos que lleva el hilo en su actual estado, el estado en el que está y, por último, la sentencia que está llevando a cabo el hilo o, de no ejecutar ninguna, NULL.

# Conclusiones

Este trabajo ha destacado el gran trabajo que hay detrás de una simple sentencia SQL, un trabajo que a simple vista no es apreciable y podría parecer poco importante, pero según aumenta la escala de una base de datos, la optimización que todas estas medidas ayudan a implementar se convierte en indispensable. También destacar el descubrimiento de la sentencia EXPLAIN la cual seguro que nos es de gran ayuda en un futuro cuando nos veamos en la situación de enfrentarnos a una base de datos MySQL, especialmente en entornos de producción donde los recursos son limitados y afinar nuestras consultas al máximo para minimizar el desperdicio de recursos se convierta en un imperativo.

Además de aprender nuevas cosas, también hemos tenido oportunidad de repasar y profundizar en aspectos que ya conocíamos como son los índices de las tablas MySQL. Previa realización de este trabajo no teníamos un conocimiento muy amplio y no terminábamos de entender por qué estos elementos eran tan útiles, pero de ahora en adelante los intentaremos dar uso cuando la ocasión se presente.

# Bibliografía

manual de referencia de MySQL - <https://dev.mysql.com/doc/>