Optimización en MySQL

Bases de datos II

3º - Grado en Ingeniería Informática

Índice de Contenidos

[Introducción 3](#_Toc5840603)

[Optimizar a nivel de base de datos 3](#_Toc5840604)

[Optimizar a nivel de hardware 3](#_Toc5840605)

[Procesos de optimización implementados por MySQL 3](#_Toc5840606)

[Sentencia EXPLAIN 3](#_Toc5840607)

[Motores de almacenamiento MyISAM e InnoDB 3](#_Toc5840608)

[Índices 3](#_Toc5840609)

[Sentencia OPTIMIZE Table 4](#_Toc5840610)

[Sentencia INSERT DELAYED 4](#_Toc5840611)

[Slow Query Log 4](#_Toc5840612)

[Variables de sistema: long-query-time 4](#_Toc5840613)

[Show ProcessList 4](#_Toc5840614)

[Conclusiones 4](#_Toc5840615)

[Bibliografía 4](#_Toc5840616)

# 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

## 

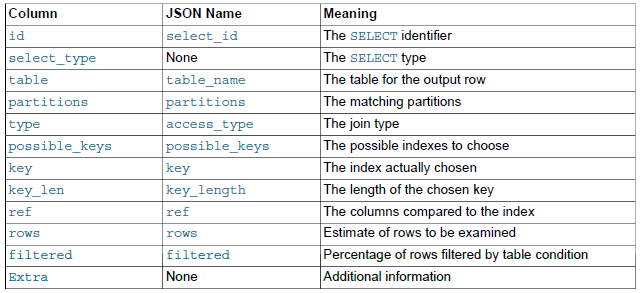
## Sentencia EXPLAIN

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

## 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 ultra-rápidas 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

Once your data reaches a stable size, or a growing table has increased by tens or some hundreds of megabytes, consider using the OPTIMIZE TABLE statement to reorganize the table and compact any wasted space. The reorganized tables require less disk I/O to perform full table scans. This is a straightforward technique that can improve performance when other techniques such as improving index usage or tuning application code are not practical.

OPTIMIZE TABLE copies the data part of the table and rebuilds the indexes. The benefits come from improved packing of data within indexes, and reduced fragmentation within the tablespaces and on disk. The benefits vary depending on the data in each table. You may find that there are significant gains for some and not for others, or that the gains decrease over time until you next optimize the table. This operation can be slow if the table is large or if the indexes being rebuilt do not fit into the buffer pool. The first run after adding a lot of data to a table is often much slower than later runs.

## Sentencia INSERT DELAYED

## Slow Query Log

## Variables de sistema: long-query-time

## Show ProcessList

# Conclusiones

# Bibliografía

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