

Optimización de MySQL

Tras analizar en nuestro primer artículo la optimización del modelo de datos, continuamos viendo cómo obtener el mejor rendimiento de nuestro servidor de Base de Datos MySQL.

Nos centramos ahora en la optimización de las consultas SQL.

Optimización de MySQL II – Las Consultas

Introducción

Las consultas a la base de datos son el típico punto de degradación de una aplicación tras varios meses o años de funcionamiento. Salvo que nuestra aplicación realice operaciones masivas de inserción de registros, debemos preocuparnos sólo de las operaciones de SELECT. Es en este tipo de operaciones donde notaremos como, poco a poco, se va degradando nuestra aplicación.

La degradación de una consulta que, inicialmente funcionaba bien, se produce por dos motivos. En primer lugar, porque crece el número de registros de nuestra base de datos. Lo habitual es que, al poner la aplicación en marcha tengamos pocos registros (clientes, pedidos, expedientes...) y que, según transcurre el tiempo, las tablas tengan cada vez más registros.

El segundo motivo de degradación es el incremento del número de usuarios, la concurrencia. Al desarrollar la aplicación posiblemente las consultas se probaron con un único usuario. A lo sumo con alguna simulación de carga (pero esto no es muy habitual). Incluso tras la puesta en marcha suele haber un periodo de aceptación en el que los usuarios van incorporándose poco a poco a la aplicación, usándola cada día un poco más.

Esta combinación de mayor cantidad de datos y mayor concurrencia puede poner en evidencia un diseño poco profesional de las consultas.

Naturalmente nuestras consultas estarán embebidas en un aplicación, no es habitual que los usuarios lancen consultas SQL directamente a la base de datos. Así que, los primeros síntomas de que algo empieza a fallar serán “esporádicos”. Tendremos comentarios como “La búsqueda de clientes va más lenta, ¿habéis tocado algo?” o incluso “Hoy la red va muy mal, alguien estará bajando películas”. Comentarios que no constituyen quejas formales y que, fácilmente pueden ser achacados a otras causas.

Pero la degradación de las consultas, aunque lenta, es progresiva. Y poco a poco (o quizás no tan poco a poco) nuestra aplicación se irá degradando hasta la sublevación de los usuarios. Antes de que ocurra deberemos “remangarnos” para resolver el problema.

Detección

Lo primero es detectar el problema, saber qué está pasando. Puede que en un principio nada nos haga sospechar de la base de datos. Simplemente, algunas operaciones de usuario (búsquedas, listados, etc.) serán lentas “esporádicamente”.

Iniciaremos nuestra operación de búsqueda descartando factores... la red, el servidor web, el sistema operativo... hasta llegar a la base de datos. Pero... ¿cómo podemos determinar si la base de datos es

el cuello de botella?

Afortunadamente MySQL tiene un log específico para detectar este tipo de situaciones, el “Slow Query Log”. Nos bastará con activar este log durante unos días y analizarlo.

Para ello, tendremos que modificar la sección [mysqld] del fichero de configuración (my.cnf):

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

Con la primera directiva estamos indicando en qué fichero se guardará el log. Conviene asegurarse de que existe un proceso de rotación para ese log (logrotate). Con la segunda directiva establecemos el tiempo (en segundos). En este caso, cualquier consulta que tarde más de 2 segundos en ejecutarse se considerará lenta y será registrada en el log. Y en la tercera directiva hará que también se registren en el log las consultas que no utilicen ningún índice para su resolución.

Revisando este log, tras varias horas o días de recogida de información, podremos identificar las consultas lentas de nuestra aplicación. Hay que tener en cuenta que el hecho de que una consulta este registrada una vez en el log sólo significa que esa vez se ejecutó lentamente. Puede ser que en esa ocasión ocurriera algo en el servidor (un proceso de copia de seguridad, un cliente lento...). Debemos concentrarnos en las consultas que más se repitan y peores tiempos tengan.

Podemos comprobar qué consultas coinciden con los procesos que los usuarios consideran lentos y, ahora sí, centrarnos en mejorarlas... de una en una.

El Proceso de Consulta de MySQL

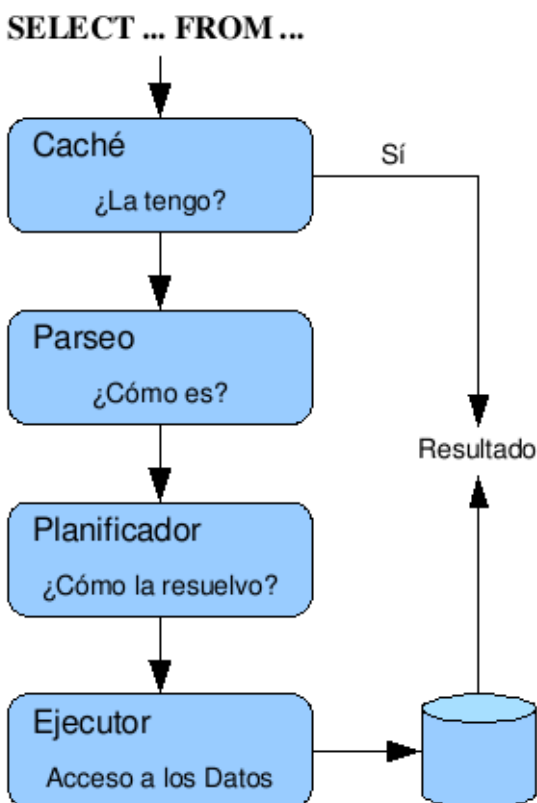
Para poder mejorar el rendimiento de una consulta debemos conocer antes cómo las resuelve MySQL. Es lo que se conoce como “proceso de consulta”.

Cuando MySQL recibe una consulta, procede de la siguiente forma:

En primer lugar, busca la consulta en la caché. Si hay suerte, ya tendrá los resultados guardados de una consulta anterior y no tendrá que resolverla.

En segundo lugar, si no ha podido utilizar la caché, analiza la consulta, comprueba su sintaxis e identifica su tipo; es la fase de “parseo”.

Con la información obtenida del análisis, llega al siguiente paso, la planificación. Este es el punto crítico y del que dependerá el tiempo de resolución. El planificador de MySQL decide cual será el proceso para



resolver la consulta, qué parte se resolverá primero, qué índices se utilizarán y cómo se obtendrán los datos.

Por último, en la fase de ejecución solo resta ejecutar el plan trazado y entregar los resultados al cliente.

Vamos a ver este proceso paso a paso:

La Caché de Consultas

MySQL registra las consultas de tipo SELECT y su resultado. Como lo normal es que se acceda a la base de datos a través de una aplicación, las consultas repetidas son muy frecuentes (listas de poblaciones, de códigos, de nombres...).

Si MySQL recibe una consulta que tiene en la caché, simplemente entrega al cliente el mismo conjunto de resultados que produjo en su ejecución anterior. Naturalmente, las consultas de modificación de datos (INSERT, DELETE, UPDATE...) invalidan las consultas afectadas de la caché y provocan la eliminación de estas de la caché.

Podemos utilizar la caché para mejorar el rendimiento de nuestra base de datos. La variable `mysql_cache_type` (en `my.cnf`) establece el tipo de caché que utilizará MySQL. Un valor a 1 hará que todas las consultas de tipo SELECT sean cacheadas, salvo que expresamente indiquen lo contrario mediante el modificador `SQL_NO_CACHE` en la sentencia SELECT. Un valor de `query_cache_type` igual a 2 hará lo contrario. Las consultas no serán cacheadas salvo que expresamente lo soliciten con `SQL_CACHE`.

De esta forma podremos controlar y mejorar la calidad de la caché, haciendo que las consultas repetidas sean cacheadas y evitando que las consultas que no se repiten (buscadores, datos de un cliente...) consuman espacio de almacenamiento.

El tamaño de la cache se establece en la variable `query_cache_size`. Y el límite de almacenamiento por consulta en la variable `query_cache_limit`. La estrategia concreta de uso de estas variables dependerá de las características del servidor y de la aplicación.

Análisis

En el proceso de análisis (“parseo”), MySQL determina el tipo de consulta, las tablas relacionadas, las características de la clausula WHERE... pero todo esto apenas tiene incidencia en el rendimiento. Lo único que puedes tener en cuenta es, que cuanto más largas sean las consultas, más tiempo de análisis necesitarán. Así que, “lo bueno si breve, dos veces bueno”.

Planificación

El planificador es el punto crítico en la resolución de una consulta. Del plan de ejecución que elabore dependerá que la consulta tarde unas pocas décimas de segundo o varios minutos. Pero el trabajo del Planificador no es fácil, para entenderlo debemos tener en cuenta dos factores:

El planificador trabaja para cualquier consulta. No es viable programar un planificador específico para nuestra aplicación, así que nos tenemos que conformar con el planificador de MySQL. Este está optimizado para “cualquier aplicación”. Y, aunque acumula los años de experiencia de los programadores de MySQL y utiliza estadísticas de ejecución para decidir cuál será el mejor plan, no

siempre acertará.

En segundo lugar, el planificador no tiene tiempo de encontrar la solución óptima. Sería absurdo que utilizara tres segundos en planificar una consulta que, en el peor de los casos, sólo requerirá dos segundos para resolverse. El planificador debe encontrar el mejor plan posible, en el menor tiempo posible.

Por eso necesitamos conocer cómo trabaja, qué decisiones toma y, sobre todo, como influir en sus decisiones cuando nos convenga.

Centrándonos en lo importante, el planificador debe decidir dos aspectos principales en el plan de ejecución: qué índice se va a utilizar para resolver la consulta y en qué orden se realizarán los joins de las tablas.

Vamos a revisar primero los tipos de índices que hay en MySQL para después analizar cómo toma sus decisiones el Planificador.

Índices

En los distintos motores de almacenamiento de MySQL podemos encontrar los siguientes tipos de índices:

B-Tree: Índice con estructura de árbol, muy utilizado por su flexibilidad, rendimiento y ahorro de espacio. Además de agilizar las consultas directas por los campos indexados, también agilizan las consultas por rango (BETWEEN) y por comparadores (<, >, >=, <=).

MySQL utiliza índices B-Tree en los motores MyISAM, HEAP e InnoDB. En el caso de InnoDB, el índice está “incrustado”, es decir, que se almacena junto a los datos (y no en un fichero separado como en MyISAM).

Hash: Índice basado en algoritmos de hash que garantiza (casi siempre) la dispersión de valores de la clave. Así se evita que en un rango de valores de índice se acumulen la mayoría de los resultados, como puede suceder en los B-Tree (por ejemplo, que la mayoría de teléfonos se acumulen en las entradas de índice que empiezan por 91 y 93). El algoritmo de hash hace que valores del índice muy similares se indexen separados.

R-Tree: Son índices utilizados para datos espaciales y multidimensionales. Permiten localizar rápidamente puntos contenidos en diferentes tipos de áreas.

Textuales: Los índices textuales permiten localizar rápidamente palabras completas en campos de texto (varchar, text, etc.). Son los utilizados habitualmente en la implementación de buscadores. MySQL dispone de este tipo de índices para tablas MyISAM. Estos índices se implementan como una estructura de árboles B-Tree de dos niveles. Dicho de forma sencilla e imprecisa, el primer árbol B-Tree almacena cada palabra indexada, el número de filas en las que aparece y un puntero a la raíz de su segundo árbol asociado. El segundo árbol B-Tree almacena por cada fila el peso de la palabra en la fila (“la importancia”) y un puntero a los datos (rowid).

Afortunadamente no es necesario entender la estructura ni el funcionamiento de los índices para poder utilizarlos adecuadamente.

La Elección de Índice

Para elegir el índice que se utilizará en la consulta, el Planificador de MySQL busca los índices

aplicables, consulta sus estadísticas y elige el índice que, en su opinión, implique consultar un menor número de registros.

Resulta evidente la importancia que adquieren las estadísticas de los índices que almacena MySQL. Si no están actualizadas, el planificador puede elegir utilizar un índice que, según sus estadísticas, devolverá 10 resultados y encontrarse con que realmente devuelve 300.000.

Para evitar la degradación de rendimiento resulta vital analizar y optimizar las tablas periódicamente, sobre todo si el número de registros varía frecuentemente. Para ello debemos utilizar los comandos `ANALYZE TABLE` y `OPTIMIZE TABLE`. Esto hará que el Planificador tenga estadísticas actualizadas al elegir el índice.

Pero también hay otra conclusión poco evidente: ¡cuidado con los procedimientos almacenados! Existe la creencia de que los procedimientos almacenados son más rápidos porque su plan de ejecución ya está compilado... Pero probablemente este plan se compiló durante el desarrollo de la aplicación, con tablas pobladas con escasos datos de pruebas e incluso sin alguno de los índices que pudieron crearse posteriormente. Es decir, si no tenemos cuidado, los procedimientos almacenados se estarán ejecutando con planes anticuados. Y entonces pueden llegar a ser mucho (pero mucho) más lentos que una consulta cuyo plan de ejecución se compile con información actualizada.

Veamos la importancia de las estadísticas con un ejemplo, ¿cómo se puede resolver la siguiente consulta?

```
SELECT nombre, apellidos FROM persona WHERE codigo_postal LIKE '06%' AND fecha_nacimiento < '1968';
```

Suponiendo que la tabla `persona` tenga dos índices definidos, uno sobre `codigo_postal` y otro sobre `fecha_nacimiento`, la decisión del índice a utilizar es crítica. MySQL puede buscar primero todas las filas con el código postal de Badajoz y luego filtrar los resultados con fecha de nacimiento anterior a 1968. O viceversa.

Si resulta que en la tabla hay 200 personas de Badajoz y 10 millones de personas nacidas antes de 1968, la primera decisión será la correcta, pero si las estadísticas son a la inversa... MySQL no puede conocer la población de Badajoz, sólo puede guiarse en esta decisión por las estadísticas de sus índices. Y si estas estadísticas están obsoletas, sus decisiones no pueden ser buenas.

La Elección del Orden de Join

El otro punto crítico del Planificador es decidir el orden en que se harán los Joins. Analicemos la siguiente consulta:

```
SELECT persona.apellidos, pedido.fecha, pais.nombre
FROM persona, pedido, pais
WHERE pedido.id_persona = persona.id AND
      persona.id_pais = pais.id AND
      pedido.fecha < '2008-02-13' AND
      persona.apellidos LIKE 'García%';
```

Vemos que de nuevo, el número de filas de cada tabla y sobre todo, la calidad de los índices que hubiéramos definido sobre `pedido.fecha` y `persona.apellidos` influirán notablemente en

el tiempo de resolución de la consulta.

Pero aquí influye algo más que la actualización de los índices. Si imaginamos una consulta con 7 tablas y un índice por tabla (algo no muy infrecuente), vemos que MySQL tiene que analizar más de 5.000 combinaciones posibles para el orden en el que efectuará los joins (exactamente $7! = 5040$). De hecho, puede tardar varios segundos en encontrar un plan de ejecución para una consulta que se ejecutará en décimas de segundo.

En estas situaciones, lo mejor es utilizar STRAIGHT JOIN para indicarle a MySQL el orden en el que queremos que haga el Join de las tablas y ahorrándole así el trabajo.

El Plan de Ejecución

Pero, ¿cómo podemos saber lo que MySQL planea hacer? Para ello tenemos la sentencia EXPLAIN, que nos mostrará el plan de ejecución que MySQL ha compilado para una consulta.

Por ejemplo, este es el plan de ejecución para una consulta sencilla:

```
EXPLAIN SELECT * FROM persona WHERE nif='007'\G
***** 1. row *****
  id: 1
  select_type: SIMPLE
  table: persona
  type: const
  possible_keys: PRIMARY
  key: PRIMARY
  key_len: 12
  ref: const
  rows: 1
  Extra:
1 row in set (0.00 sec)
```

La información que nos ofrece el comando EXPLAIN es la siguiente:

- **id:** El identificador de la tabla en la consulta. MySQL asignará un identificador diferente a cada tabla involucrada en la consulta.
- **select_type:** La función que desempeña esta tabla en la consulta. Puede ser SIMPLE, PRIMARY, UNION, DEPENDENT UNION, UNION RESULT, SUBQUERY, DEPENDENT SUBQUERY, DERIVED y UNCACHEABLE SUBQUERY.
- **table:** Nombre de la tabla.
- **type:** El tipo de Join que realizará MySQL según la relación entre datos de ambas tablas. Puede ser const, system, eq_ref, ref, range, index o ALL.
- **possible_keys:** La lista de índices candidatos que MySQL ha encontrado para la consulta.
- **key:** El índice elegido para resolver la consulta.
- **key_len:** Tamaño del índice, en bytes.
- **ref:** Las columnas o valores con los que se filtrará el índice.

- **rows:** El número estimado de filas que MySQL cree que tendrá que examinar para resolver la consulta.
- **Extra:** Información adicional

En este ejemplo, la tabla persona tiene como clave primaria el campo nif. El planificador nos está diciendo:

- **select_type:** SIMPLE. La consulta es un simple select, sin uniones ni selects anidados.
- **type:** const. El valor const indica que la tabla tiene, como máximo, un resultado para la consulta. Y por lo tanto, el planificador puede considerar la columna como una constante para el resto de su evaluación. Esto es típico en búsquedas por índices únicos (entre ellos las claves primarias) y dan lugar a consultas muy rápidas.
- **possible_keys:** PRIMARY. El único índice posible es la clave primaria.
- **key:** PRIMARY. El índice elegido es la clave primaria.
- **key_len:** 12. El tamaño de cada entrada del índice elegido.
- **ref:** const. En la comparación con el índice, estamos utilizando la constante '007'.
- **rows:** 1. Al tratarse de la clave primaria, MySQL está seguro de que el resultado será de 1 única fila.

Veamos ahora un ejemplo algo más complicado:

```
EXPLAIN SELECT * FROM pedido WHERE id = (SELECT MAX(id) FROM pedido)\G
```

```
***** 1. row *****
    id: 1
  select_type: PRIMARY
    table: pedido
    type: const
 possible_keys: PRIMARY
    key: PRIMARY
  key_len: 4
    ref: const
    rows: 1
  Extra:
***** 2. row *****
    id: 2
  select_type: SUBQUERY
    table: NULL
    type: NULL
 possible_keys: NULL
    key: NULL
```

```
key_len: NULL
ref: NULL
rows: NULL
Extra: Select tables optimized away
```

Este plan de ejecución está dividido en dos consultas. La primaria (PRIMARY) con id=1, es igual al caso anterior. Pero la segunda es una subconsulta (id=2) con cosas diferentes. ¿Porqué están todos los valores a NULL?

El planificador nos está diciendo que no tiene que utilizar ninguna tabla ni ningún índice para resolver la segunda consulta. Las tablas de tipo MyISAM son especialmente rápidas en consultas de COUNT, MAX y MIN si estas operaciones se realizan sobre índices únicos (como lo es la clave primaria). MySQL tiene la información que necesita para la consulta almacenada en memoria. Ese es el significado del valor de “Extra”.

Otras consultas serán más complicadas, y necesitaremos tiempo y paciencia para interpretar el plan de ejecución de MySQL y decidir si es el más adecuado. Utilizando el comando EXPLAIN podremos saber cómo planea resolver MySQL una consulta. Y por lo tanto podremos detectar planes de ejecución poco eficientes que hagan lentas nuestras consultas. Ahora la clave es: ¿cómo puedo influir en el plan de ejecución?

Tenemos varias acciones posibles:

Crear índices

La primera opción para mejorar el rendimiento de una consulta es la creación de índices. Siempre podemos evitar que MySQL tenga que escanear toda una tabla comparando el valor de una columna si definimos el índice adecuado sobre ella.

¡Pero cuidado! Como dicen los americanos “There is no free lunch”, y tendremos que pagar un precio por la creación de índices.

Los índices penalizan las consultas de inserción y modificación. En muchas aplicaciones esto no será un problema, y será rentable crear índices que mejoren las consultas de SELECT a costa de ralentizar los INSERT y UPDATE. Pero en algunas aplicaciones con entradas masivas y simultáneas de datos, esto puede ser un problema.

Usar Correctamente los Índices

En ocasiones, puede que el índice adecuado ya exista, pero que MySQL decida no utilizarlo. Ya hemos dicho que el planificador no es perfecto, no tiene tiempo de serlo si quiere resolver la consulta en un tiempo prudente. Así que, en ocasiones, veremos que el plan de ejecución no utiliza el índice adecuado.

Podemos proponer a MySQL una lista de índices a considerar, descartando otros, con el modificador USE INDEX. También podemos obligarle a utilizar un índice concreto con FORCE INDEX, e incluso podemos indicarle que ignore una lista de índices con IGNORE INDEX.

Pero normalmente los problemas con los índices no son achacables al planificador: o bien no se han

actualizado las estadísticas o bien no se utilizan correctamente los índices.

Ya hemos insistido en la importancia de actualizar las estadísticas con ANALIZE TABLE y OPTIMIZE TABLE.

El segundo problema se produce con los índices Full-Text. Estos índices no se utilizan igual que el resto. Así, la siguiente consulta:

```
SELECT * FROM persona WHERE apellidos LIKE '%García%';
```

no utilizará un índice de tipo Full-Text que tuviéramos definido sobre la columna apellidos. En su lugar, debemos utilizar MATCH AGAINST

```
SELECT * FROM persona (apellidos) MATCH AGAINST ('García')
```

Reescribir la Consulta

En ocasiones puede ser útil reescribir la consulta para que el planificador cambie su interpretación. Por ejemplo, podemos utilizar variables de servidor para dividir una consulta con subselects en dos consultas. O podemos utilizar uniones de resultados (UNION) en lugar de restricciones OR.

Estos cambios influirán en el proceso de planificación de MySQL y, entendiendo adecuadamente los criterios de selección, podemos mejorar nuestras consultas.

Ejecución

Y nos queda la última fase del Proceso de Consulta. Durante la ejecución MySQL ejecuta el plan trazado, por lo que apenas tiene relevancia en el rendimiento de la consulta, salvo que la consulta esté devolviendo grandes conjuntos de resultados.

Problemas con los Resultados Grandes

Al resolver una consulta, MySQL bloqueará la tabla contra escrituras para garantizar la integridad de los datos. Esto puede ser un problema si el resultado es grande y cliente es lento en procesarlo (conexión lenta, máquina de pequeña capacidad...). En estos casos la tabla quedará bloqueada impidiendo el acceso de escritura a otros usuarios. Podemos solucionarlo indicando a MySQL que almacene el resultado en una tabla temporal utilizando SQL_BUFFER_RESULT y liberar así la tabla.

Además, si ya sabemos que el resultado de una consulta va a ser grande, podemos avisar al planificador con la opción SQL_BIG_RESULT. De esta forma, podrá tomar decisiones más agresivas y buscar un plan de ejecución mejor.

¿Qué mejoras se pueden esperar?

Si como vimos en el artículo anterior, el diseño del modelo de datos es crítico para el rendimiento futuro de nuestra aplicación, la optimización de consultas es la técnica que necesitamos dominar para optimizar el rendimiento actual.

Cuando el rendimiento de nuestra aplicación se degrada, y detectamos que se debe a la lentitud de algunas consultas, no será fácil optimizar el modelo de datos, pero sí podremos optimizar las consultas. ¿Hasta dónde podemos llegar?

Si la consulta está bien diseñada, las estadísticas están actualizadas y se utilizan los índices correctos; apenas habrá nada que hacer. Las mejoras serán de un 10-30%. Pero si alguna de estas cosas falla, podremos mejorar el rendimiento radicalmente. La diferencia entre una consulta que debe escanear 200.000 registros y luego ordenarlos frente a la misma consulta debidamente optimizada para evitar el escaneo y obtener directamente la ordenación del índice utilizado puede de ser de hasta tres órdenes de magnitud o incluso superior. Y eso, en una aplicación lenta, traza la diferencia entre la desesperación de los usuarios y su satisfacción.

Miguel Jaque Barbero
Ilke Benson - Dtor. Proyectos
mjaque@ilkebenson.com