

## **Optimización de MySQL**

Iniciamos una serie de tres artículos en los que analizaremos cómo obtener el mejor rendimiento de nuestro servidor de Base de Datos MySQL.

En este primer artículo, analizaremos como optimizar los modelos de datos de nuestras aplicaciones. Después veremos como optimizar las consultas a la base de datos mediante el análisis del plan de ejecución. Y, por último, veremos como optimizar la propia configuración de nuestro servidor.

## **Optimización de MySQL I – El Modelo de Datos**

### ***Introducción***

MySQL es, sin lugar a dudas, el servidor de base de datos más utilizado. No sólo predomina en el ámbito de Internet o del Software Libre, también se ha extendido a sistemas propietarios e incluso a aplicaciones de negocio. Según los datos de JoinVision de 2006, presenta la mayor cuota de mercado, por encima de otros gestores de base de datos relacionales como Oracle, SQL Server o DB2. A ello han contribuido tanto su distribución bajo licencia GPL como su rapidez y su facilidad de instalación, manejo y configuración.

Pero esta rapidez y sencillez han llevado a muchos programadores de aplicaciones a despreocuparse de la base de datos en sus sistemas. El rendimiento de MySQL pudo ser, en un principio, más que suficiente para sus propósitos y centraron sus esfuerzos en otros aspectos del sistema (programación, comunicaciones, etc.).

Sin embargo, no es raro que con el uso, el rendimiento de una aplicación decaiga. Y suele ser debido a la base de datos. Tanto el número de usuarios simultáneos como la cantidad de información a gestionar tienen un impacto importante en el rendimiento del servidor de base de datos.

Y el problema no es que MySQL no pueda gestionar eficazmente un gran número de usuarios concurrentes ni grandes bases de datos (que sí puede). El problema está en que las aplicaciones no fueron inicialmente diseñadas para la carga que han de soportar tras varios años de funcionamiento y, sólo entonces, aparecen los problemas de un diseño inicial poco ambicioso o poco previsor.

MySQL puede gestionar, de forma eficaz, un gran número de usuarios concurrentes y grandes bases de datos. Pero es necesario hacer un esfuerzo para diseñar adecuadamente tanto nuestro modelo de datos como las consultas de nuestra aplicación e incluso la configuración del sistema operativo. De estos tres factores depende que nuestro servidor sea capaz de responder en un futuro en las mejores condiciones de trabajo.

Conviene por tanto conocer las características que influyen en el rendimiento de nuestro servidor. Empecemos por el Modelo de Datos

### ***Factores de Optimización del Modelo de Datos***

El modelo de datos es uno de los factores críticos de optimización de MySQL. Es decir, las características, número y relaciones de las tablas de nuestra base de datos influyen sustancialmente

en el rendimiento de esta. El modelo de datos es además difícilmente modificable una vez desarrollada la aplicación y puesta en producción. No podremos entonces realizar cambios de importancia en el diseño de nuestras tablas porque esto nos obligaría no solo a complejos procesos de migración de datos, sino también a modificar el código de nuestra aplicación. Incluso es posible que sobre el mismo modelo de datos hayamos desarrollado nuevas aplicaciones que también deberían ser cambiadas. Por lo tanto, debemos diseñar cuidadosamente el modelo de datos, teniendo en cuenta que los errores que cometamos nos acompañarnos en el futuro y limitarán el rendimiento de nuestra aplicación.

Para optimizar el modelo de datos debemos trabajar sobre dos aspectos: la elección de los motores de almacenamiento y la normalización.

### ***Motores de Almacenamiento***

Entendemos por “motor de almacenamiento” el software responsable de la lógica del almacenamiento físico y el acceso a los datos. En versiones anteriores de MySQL se indicaba el motor de almacenamiento de cada tabla mediante la palabra clave TYPE al final de la sentencia de creación.

```
CREATE TABLE tabla (campos...) TYPE = MyISAM;
```

Hoy, aunque sigue admitiéndose temporalmente esta sintaxis, se ha impuesto el uso de la palabra clave ENGINE.

```
CREATE TABLE tabla (campos...) ENGINE = MyISAM;
```

MySQL soporta una extensa variedad de motores de almacenamiento. Y, aunque el motor por defecto MyISAM tiene grandes ventajas, conviene conocer las características de los demás para realizar la elección más acertada.

### **MyISAM**

Este es el motor por defecto de MySQL y la opción válida para la mayoría de situaciones.

MyISAM es un motor muy rápido, pero a costa de no disponer de soporte transaccional ni de integridad referencial. Es decir, no es posible definir claves externas sobre tablas MyISAM. Y, en el caso de que una tabla MyISAM participe en una transacción, no podrá efectuar la operación de ROLLBACK. Las tablas MyISAM funcionan siempre en modo AUTOCOMMIT. Eso no implica que el resto de tablas involucradas en una transacción, suponiendo que estén implementadas sobre motores transaccionales (InnoDB o BDB) no puedan realizar el ROLLBACK, todo lo contrario.

Para garantizar la concurrencia entre usuarios, el motor MyISAM implementa semáforos por tabla. Es decir, que si un usuario realiza una operación de lectura sobre la tabla, toda la tabla quedará bloqueada y se impedirán otras operaciones de escritura sobre ella mientras dure la operación inicial de lectura. Sí se permitirán otras operaciones de lectura. Si en su lugar se realiza una operación de escritura, toda la tabla quedará bloqueada tanto para escritura como para lectura.

Podría parecer que los semáforos a nivel de tabla penalizan la concurrencia. Pero debemos tener en cuenta que una granularidad menor, por ejemplo con semáforos a nivel de fila (como InnoDB) o incluso de registro, conlleva más trabajo de gestión del propio semáforo, con lo que el rendimiento se podrá ver mermado y con él la concurrencia. La estrategia del motor MyISAM es resolver la

consulta de la forma más rápida posible y esto supone hacerlo de la forma más sencilla posible.

MyISAM soporta tres tipos de índices: B-Tree, R-Tree (para datos espaciales) y Textuales. Los índices B-Tree son estructuras en árbol que combinan la flexibilidad con un alto rendimiento y un moderado consumo de espacio. Hablaremos más de índices en el próximo artículo (Optimización de Consultas).

Los índices R-Tree son los únicos utilizables para datos espaciales y nos permiten saber rápidamente si un conjunto de puntos están contenidos en una estructura geométrica determinada.

El tercer tipo de índice soportado por el motor MyISAM es el textual. Los índices textuales indexan palabras en campos de texto (VARCHAR, TEXT...) y son los más convenientes para implementar buscadores y otras aplicaciones que realicen búsquedas en textos.

El motor MyISAM tiene una característica especial, siempre sabe cuantas filas tiene su tabla. Así que las consultas del tipo COUNT(\*) no requerirán acceder a los datos y se ejecutarán rápidamente.

Por último, debemos también tener en cuenta que las tablas MyISAM se almacenan físicamente en ficheros individuales. En concreto, se utiliza un fichero con extensión .MYD para almacenar los datos de la tabla, otro con extensión .MYI para los índices definidos sobre la tabla y otro con extensión .frm para guardar su definición. Todos ellos con el mismo nombre que la tabla.

El modo de almacenamiento es importante para diseñar correctamente el proceso de copia de seguridad de nuestra base de datos. En el caso de MyISAM, y siempre que podamos detener el servidor de base de datos, bastará con realizar copias de estos ficheros para tener una copia de seguridad completa.

### **InnoDB**

InnoDB es el motor más utilizado en MySQL cuando se requiere soporte transaccional. No es tan rápido como MyISAM pero a cambio permite realizar operaciones de ROLLBACK transaccional y definir claves externas. Naturalmente, las claves externas deben hacer referencia a otras tablas InnoDB.

InnoDB implementa semáforos a nivel de fila. Pero utiliza un sistema de control de concurrencia denominado MVCC (Multi-Version Concurrency Control) que reduce el tiempo de gestión.

InnoDB utiliza exclusivamente índices B-Tree, pero con la característica de que son “incrustados” (clustered). Esto quiere decir que el índice se almacena junto a los datos. Al añadir un registro a una tabla MyISAM, éste se incorpora al final del fichero de almacenamiento. Será el índice, si está definido, quien señale la posición exacta de almacenamiento del registro en el fichero de datos. Sin embargo, en una tabla InnoDB, los registros se ordenan físicamente por su clave primaria. Esto hace que las búsquedas por clave primaria sean muy rápidas, porque al leer el índice, ya se está accediendo a los datos.

Las tablas InnoDB realizan el almacenamiento físico en un “tablespace”, un espacio de disco que gestionan directamente. Si no se modifica con la opción `innodb_file_per_table` el tablespace InnoDB será común para todas las tablas InnoDB del servidor y se almacenará en un fichero de nombre `ibdata` en el directorio de datos de MySQL. Esto puede complicar enormemente la realización y recuperación de copias de seguridad.

### **Memory (Heap)**

Es una tabla MyISAM pero que en lugar de almacenarse en disco, se guarda exclusivamente en memoria.

Esto hace que sea muy rápida, pero no es persistente. Es decir, no cumple con el requerimiento de “Durabilidad” del test ACID. Pero resulta muy útil para tablas que se cargan al inicio de una aplicación y deben ser consultadas rápida y frecuentemente. Su tamaño está limitado por la memoria disponible, indicada en la variable `max_heap_table_size`.

Las tablas Memory admiten dos tipos de índices B-Tree y Hash.

### **Archive**

Es una tabla comprimida de sólo lectura.

### **Merge**

Consiste en una colección de tablas MyISAM idénticas que se tratan como una sola. Deben ser idénticas en todo, pero algunas pueden estar comprimidas.

### **Federated**

Se trata de una tabla que, en realidad, reside en otro servidor MySQL.

### **CSV**

Este motor almacena los datos en un fichero con formato CSV (Comma Separated Values). Esto hace posible acceder a ellos directamente con aplicaciones de ofimática.

### **BDB**

Se trata de otro motor transaccional con semáforos a nivel de página. Con la aparición de InnoDB se utiliza cada vez menos.

### **Blackhole**

Este motor no te servirá para almacenar información. Acepta entradas de datos pero, en lugar de almacenarlos, desaparecen. Cualquier consulta sobre él nos devolverá un conjunto vacío.

## ***Elección del Motor***

Debemos tener en cuenta que MySQL permite el uso de varios motores de almacenamiento en una misma base de datos. Incluso es posible combinar motores de almacenamiento distintos en una misma transacción SQL y en una misma consulta. Así que no tenemos que restringirnos al uso de uno sólo de ellos en nuestro modelo de datos. Debemos elegir la combinación más adecuada eligiendo un motor de almacenamiento para cada una de las tablas de nuestro modelo.

Para elegir el motor de almacenamiento más adecuado debemos tener en cuenta las consultas que ejecutaremos sobre él.

El motor MyISAM es el más adecuado cuando la velocidad de respuesta es el factor crítico. Así mismo, serán la mejor opción cuando tengamos que realizar búsquedas textuales o consultas que deban conocer rápidamente el número total de registros de la tabla.

Sin embargo, el motor InnoDB será la elección correcta cuando requiramos soporte para transacciones o integridad referencial.

El resto de motores se utilizan en circunstancias especiales. Pero pueden mejorar notablemente el rendimiento de nuestra base de datos.

Por ejemplo, el motor Archive resulta muy útil y rápido para datos estáticos o aplicaciones que tengamos que distribuir en cdrom.

El motor Merge es útil, entre otras cosas, para datos historiadados. Puedes, por ejemplo, tener una tabla de log para cada año o para cada mes. Según pasa el tiempo las comprimes y creas una nueva. Pero quieres que tu aplicación añada o consulte registros sin preocuparse de la fecha. Eso puede hacerlo en la tabla agrupada (Merge). El motor de almacenamiento se encargará de acceder a los datos distribuidos entre las tablas.

Mediante Federated tendremos acceso a tablas remotas, evitando complejos procesos de replicación entre servidores.

La utilidad de CSV resulta obvia; nos permitirá compartir datos directamente con hojas de cálculo.

Y, respecto a Blackhole, aunque pueda parecer imposible encontrarle alguna utilidad, lo cierto es que se utiliza para determinadas configuraciones de replicación en las que parte de los datos no necesitan ser enviados al esclavo. También es útil para medir tiempos de planificación de consultas.

Por lo tanto, debemos tener en cuenta las distintas opciones de almacenamiento que MySQL pone a nuestra disposición, así como las características de cada una para elegir la que mejor se adapte a las necesidades de nuestro modelo.

## ***Normalización***

El segundo factor de optimización del modelo de datos es la normalización. “Normalización” es una técnica de diseño de modelos de datos relacionales que estructura paso a paso las tablas para reducir el riesgo de inconsistencias. Existe la creencia de que un modelo de datos “es mejor” cuanto mayor sea su nivel de normalización. Así, un modelo en tercera forma normal (3FN) sería mejor que otro en segunda forma normal (2FN). Nada más lejos de la realidad.

La calidad de un modelo de datos no depende exclusivamente de su forma normal, sino de su capacidad para responder a las consultas que tendremos que realizar sobre él. El proceso de normalización del modelo de datos elimina las dependencias funcionales entre atributos. Pero, al mismo tiempo, complica el diseño de las consultas y pierde rendimiento. Pongamos un ejemplo:

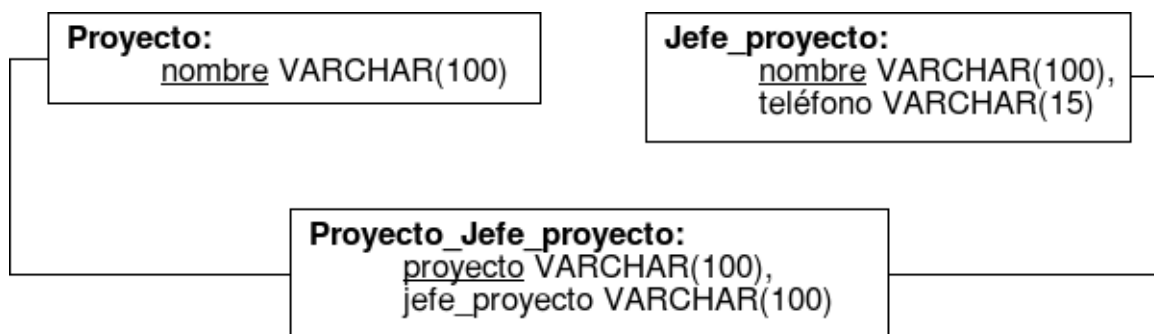
El siguiente modelo de datos recoge la información de proyectos y jefes de proyecto asignados a los mismos:

<b>Proyecto:</b> <code>nombre VARCHAR(100),</code> <code>jefe_proyecto VARCHAR(100),</code> <code>teléfono VARCHAR(15)</code>
--

Como puede verse, existe un atributo secundario (teléfono) que no depende de la clave (nombre) sino de otro atributo secundario (jefe\_proyecto). Está por lo tanto, en segunda forma normal. La consulta para obtener la lista de proyectos, jefes de proyecto asociados y teléfonos, es muy sencilla y rápida de ejecutar:

```
SELECT nombre, jefe_proyecto, teléfono FROM Proyecto;
```

Si llevamos este modelo a tercera forma normal, obtendremos lo siguiente:



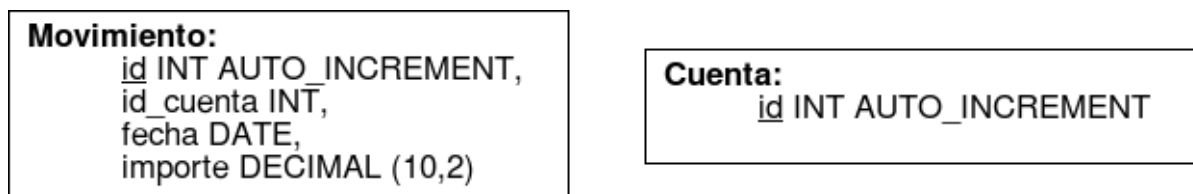
Y la consulta para obtener la información anterior requiere dos joins:

```
SELECT Proyecto.nombre, Jefe_proyecto.nombre,  
Jefe_proyecto.teléfono FROM Proyecto_Jefe_proyecto LEFT JOIN  
Proyecto ON Proyecto_Jefe_proyecto.proyecto = Proyecto.nombre  
LEFT JOIN Jefe_proyecto ON Jefe_proyecto.nombre =  
Proyecto_Jefe_proyecto.jefe_proyecto;
```

Estamos, por tanto, perdiendo rendimiento a cambio de normalización.

El caso anterior es, para casi cualquier sistema, un caso en el que la pérdida de rendimiento se ve totalmente superada por las ventajas de la prevención de inconsistencias y redundancias. Pero hay otros casos en lo que es totalmente al contrario.

Supongamos que creamos un modelo de datos para la información de cuentas bancarias y sus movimientos asociados. Podríamos pensar inicialmente, en crear un modelo en 3FN de la siguiente forma:



Podríamos desarrollar toda nuestra aplicación basada en este modelo... ¡hasta que llegase el momento de calcular el saldo de una cuenta! Una operación tan sencilla y habitual, con este modelo se convierte en:

```
SELECT SUM(importe) FROM Movimiento WHERE id_cuenta = id_cuenta;
```

Esta operación funcionará bien los primeros meses. Pero cuando la base de datos incremente su volumen con varios miles de usuarios y varios miles de movimientos por cuenta, el tiempo consumido en realizar esta operación se incrementará. Básicamente, MySQL debe leer todos los movimientos de una cuenta desde su creación para poder calcular el saldo. Y, por cierto, esta

operación además del propio tiempo que requiere, también bloqueará la tabla impidiendo mientras dure cualquier operación de escritura sobre ella y nos obligará a guardar permanentemente todos los movimientos de todas las cuentas que permanezcan abiertas. Lo que se dice, “una joya de consulta”.

Sin duda, en un caso como este es imprescindible “desnormalizar” el modelo para garantizar que mantiene el rendimiento ante el incremento de volumen. Por ejemplo, con el siguiente esquema:

**Movimiento:**

```
id INT AUTO_INCREMENT,  
id_cuenta INT,  
fecha DATE,  
importe DECIMAL (10,2),  
saldo DECIMAL (20,2)
```

**Cuenta:**

```
id INT AUTO_INCREMENT
```

Con este esquema, la obtención del saldo de una cuenta es una operación sencilla:

```
SELECT saldo FROM Movimiento WHERE id IN (SELECT MAX(id) FROM  
Movimiento WHERE id_cuenta = id_cuenta);
```

Si además hubiéramos tenido la previsión de crear un índice compuesto por los tres campos (id\_cuenta, id, saldo) la resolución de esta consulta ni siquiera tendría que consultar la tabla, toda su información estaría en el índice. Y, con un poco de suerte, la mayoría de las páginas del índice estarían en la caché de disco. Más rápido imposible.

Pero tendremos que pagar un precio a cambio de la pérdida de normalización. Todas las operaciones que anoten movimientos deberán también grabar correctamente el saldo. Se trata por tanto de un caso de equilibrio, en el que tendremos que ponderar las ventajas e inconvenientes de cada opción.

### ***¿Qué mejoras se pueden esperar?***

Hemos visto que tanto la elección correcta de los motores de almacenamiento como el nivel de normalización de cada entidad del modelo de datos influirán en el rendimiento de nuestra base de datos. Nos queda responder a la pregunta de cuánto pueden llegar a influir.

Las decisiones sobre el motor de almacenamiento aportarán variaciones del rendimiento de, a lo sumo, un orden de magnitud.

Así, en una operación de inserción masiva, una tabla InnoDB será entre 3 y 8 veces más lenta que una tabla MyISAM. Y esta a su vez será un 30% más lenta que una tabla Memory. Aunque algunas operaciones pueden verse más penalizadas e incluso ser imposibles de realizar. Este es el caso de las búsquedas textuales, que sólo serán viables sobre tablas MyISAM y de las consultas que requieran gran rapidez en determinar el número de registros de una tabla (COUNT(\*)). Pero, por lo general, es la transaccionalidad y no el rendimiento quien determina el motor de almacenamiento de cada tabla. Otros factores como la realización de copias de seguridad también influirán en la decisión.

Sin embargo, el nivel de normalización tendrá una gran influencia sobre el rendimiento final de la aplicación. La diferencia entre realizar una consulta directa o recurrir a varios joins para obtener el

mismo resultado puede ser de tres órdenes de magnitud o incluso más.

Además, se da la circunstancia que cambiar de motor de almacenamiento, salvo para tablas realmente grandes, es viable y sencillo. Bastará con volcar la tabla, recrearla cambiando su definición y volver a cargarla. Sin embargo, como ya hemos apuntado, modificar la normalización de un modelo de datos en producción será, normalmente, inviable.

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