

UNIDAD 3: OPTIMIZACIÓN DEL RENDIMIENTO

Monitorización y optimización

Resultados de Aprendizaje a trabajar en esta Unidad:
RA5: Optimiza el rendimiento del sistema aplicando técnicas de monitorización y realizando adaptaciones.
Contenidos a trabajar en esta Unidad:
• Herramientas de monitorización disponibles en el sistema gestor. • Elementos y parámetros susceptibles de ser monitorizados. • Optimización. • Herramientas y sentencias para la gestión de índices. • Herramientas para la creación de alertas de rendimiento.

Contenidos de la Unidad:

1. INTRODUCCIÓN 2
2. HERRAMIENTAS DE MONITORIZACIÓN DE MySQL 4
3. ELEMENTOS Y PARÁMETROS SUSCEPTIBLES DE SER MONITORIZADOS 12
4. OPTIMIZACIÓN DEL RENDIMIENTO EN MySQL 20
5. ÍNDICES 25

ANEXO: SQL DIGEST 29

BIBLIOGRAFÍA 30

1. INTRODUCCIÓN

Como ya hemos visto en las anteriores de unidades, un sistema gestor de bases de datos (SGBD) es el software encargado de almacenar, organizar, consultar y modificar los datos de una base de datos. Sin embargo, no basta con que un SGBD funcione correctamente y devuelva los resultados esperados: también debe hacerlo con un rendimiento adecuado.

El **rendimiento de un SGBD** es su capacidad para procesar las operaciones solicitadas de forma eficiente, utilizando de manera razonable los recursos disponibles y manteniendo unos tiempos de respuesta adecuados.

En MySQL, el rendimiento depende de varios factores: cómo están diseñadas las tablas, qué consultas SQL se ejecutan, qué índices existen, cuántos usuarios trabajan simultáneamente, cómo está configurado el servidor y qué recursos físicos tiene el equipo donde se ejecuta, etc.

Por ejemplo, imaginemos una aplicación web de una tienda en línea que utiliza MySQL para almacenar productos, clientes y pedidos. Cuando un cliente busca un producto, la aplicación ejecuta una consulta SQL para recuperar la información correspondiente. Si la consulta tarda unas décimas de segundo, la experiencia puede ser satisfactoria. Si tarda varios segundos, la aplicación puede parecer lenta. Si muchos clientes realizan búsquedas simultáneamente, el problema puede agravarse y afectar al conjunto del servicio.

Por tanto, optimizar el rendimiento no consiste únicamente en conseguir que una consulta individual sea rápida, sino en **lograr que el sistema responda adecuadamente en las condiciones de uso previstas.**

Para analizar el rendimiento de un SGBD se utilizan distintas métricas. A continuación, se enumeran algunos ejemplos:

- Tiempo de respuesta: el tiempo que transcurre desde que se solicita una operación hasta que se obtiene su resultado.
- Rendimiento de procesamiento: cantidad de operaciones que el sistema puede completar por unidad de tiempo (generalmente 1 segundo).
- Consumo de CPU: cantidad de capacidad de procesamiento que utiliza el servidor.
- Consumo de memoria: memoria utilizada por el servidor y sus operaciones, datos, índices...
- Actividad de disco: lecturas y escrituras realizadas sobre el almacenamiento.

- Concurrencia y esperas: comportamiento del sistema cuando varias operaciones se ejecutan simultáneamente y deben esperar por recursos o bloqueos.

Estos indicadores pueden estar relacionados, pero no siempre son equivalentes. Además, la interpretación de las métricas debe tener en cuenta el contexto. Un valor aislado no suele ser suficiente para determinar si existe un problema de rendimiento. Es necesario analizar la evolución de las métricas, la carga de trabajo y los objetivos del sistema.

Por ejemplo, cuando un servidor presenta problemas de rendimiento, el primer paso que deberemos realizar es observar detenidamente el comportamiento del sistema, identificando posibles causas, y posteriormente, aplicar las medidas correctoras correspondientes. En este proceso nos encontramos con **tres actividades principales**: monitorización, diagnóstico y optimización.

La **monitorización** consiste en observar y recopilar información sobre el estado y el funcionamiento del SGBD a lo largo del tiempo. Su objetivo es **conocer cómo se comporta el sistema, detectar anomalías y disponer de datos que permitan investigar posibles problemas.**

En MySQL podemos monitorizar cosas como el número de conexiones abiertas, la cantidad de consultas que se ejecutan y el tiempo que tardan en ejecutar, el consumo de memoria y CPU, los errores registrados por el servidor...

El **diagnóstico** consiste en analizar la información recopilada **para identificar el origen de un problema de rendimiento**. Una vez detectada una anomalía, hay que investigar qué la está provocando. Para ello pueden utilizarse registros, estadísticas del servidor, planes de ejecución y herramientas específicas de MySQL.

Por último, la **optimización** consiste en **aplicar cambios destinados a mejorar el rendimiento del sistema**, una vez identificadas las causas que justifican esos cambios.

Algunas posibles medidas de optimización son la simplificación de una consulta SQL para reducir el trabajo que realiza, crear un índice que facilite la búsqueda de determinados datos, ajustar parámetros del servidor, ampliar los recursos disponibles del hardware si este es insuficiente...

Un problema de rendimiento puede manifestarse de distintas formas. Algunos **síntomas** habituales son:

- Consultas SQL lentas: puede ser causado por la lectura de un volumen elevado de filas, por la ejecución de operaciones de ordenación o agrupación costosas, por la combinación ineficiente de varias tablas...

- Saturación de recursos: CPU, memoria, disco
- Exceso de conexiones: si MySQL recibe más conexiones simultáneas de las que puede gestionar adecuadamente, o alcanza el límite configurado, puede provocar errores o saturar los recursos del servidor.
- Bloqueos y esperas: las transacciones concurrentes pueden necesitar acceder a los mismos datos, produciendo bloqueos y/o esperas para mantener la integridad de los datos.

Además, hay diferentes **factores que influyen en el rendimiento** de un sistema gestor de base de datos, siendo algunos de ellos los que se enumeran a continuación:

- Diseño de la base de datos: una estructura inadecuada puede obligar a ejecutar consultas complejas, almacenar información redundante o procesar más datos de los necesarios.
- Diseño de las consultas SQL: la forma en la que se escriben las consultas influye directamente en el trabajo que va a tener que realizar el sistema gestor. Por ejemplo, si recuperamos todas las columnas de una tabla, pero únicamente necesitamos la información de algunas de ellas.
- Uso de índices: los índices permiten localizar determinados registros sin tener que examinar necesariamente todas las filas de una tabla, pudiendo reducir el trabajo necesario en las consultas.
- Configuración del servidor: una configuración inadecuada puede limitar el rendimiento. Parte de esta configuración puede establecerse desde el archivo my.ini.
- Recursos físicos: el hardware que se esté utilizando también va a establecer límites sobre la capacidad del sistema.

2. HERRAMIENTAS DE MONITORIZACIÓN DE MySQL

MySQL dispone de varias herramientas para realizar las tareas de monitorización. Algunas muestran información general del servidor, otras registran las consultas ejecutadas y otras permiten analizar con más detalle el funcionamiento interno del SGBD.

En este apartado estudiaremos las principales herramientas disponibles en MySQL 8.0/8.4, trabajando en Windows con MySQL Workbench y la consola SQL.

2.1. MySQL Workbench. Monitorización mediante herramientas gráficas

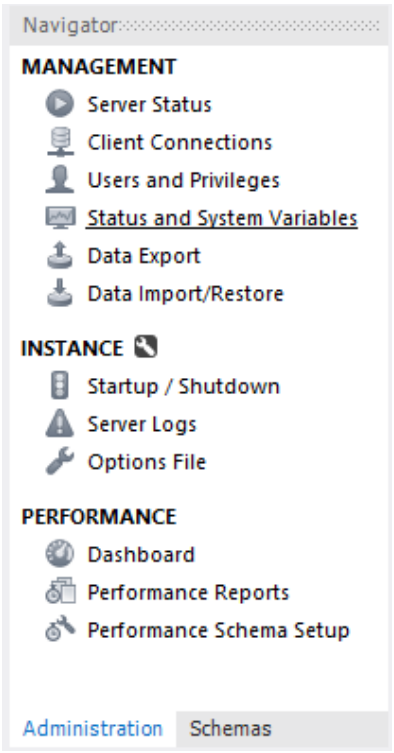
Al conectarnos a un servidor MySQL desde Workbench, podemos acceder a distintas opciones de administración y monitorización. Las opciones disponibles pueden variar

según la versión de Workbench, el servidor al que nos conectemos y los permisos de la cuenta utilizada.

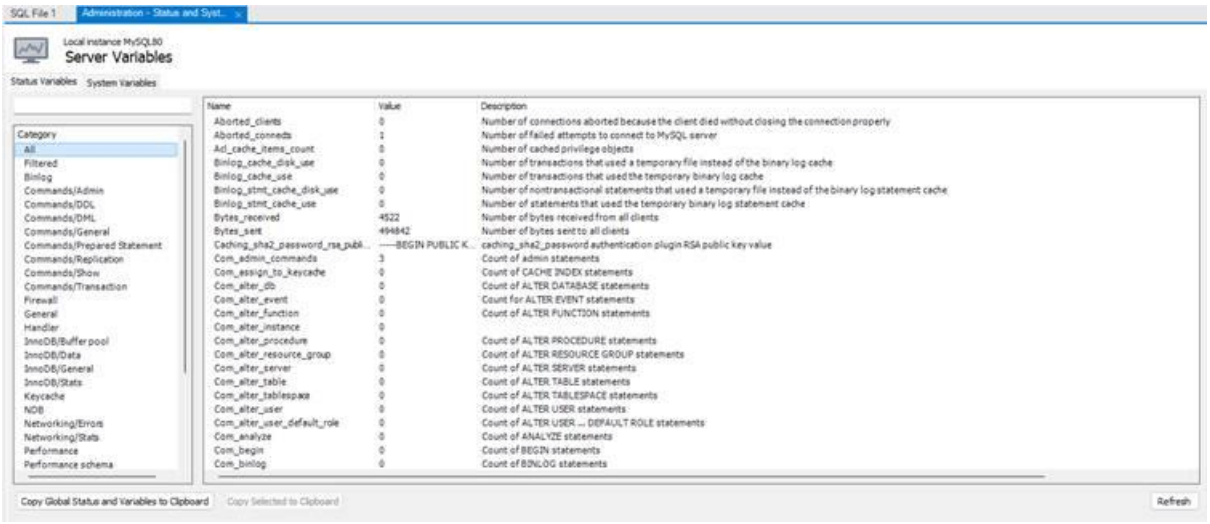
Entre la información que podemos consultar se encuentra:

- Estado del servidor: conexiones, actividad y determinadas estadísticas de funcionamiento.
- Variables de estado: contadores que muestran la actividad acumulada del servidor.
- Variables de configuración: parámetros que determinan cómo funciona MySQL.
- Conexiones y sesiones: información sobre las conexiones que mantienen los clientes.
- Rendimiento de consultas: información que puede ayudar a identificar operaciones costosas, dependiendo de las herramientas y vistas disponibles.

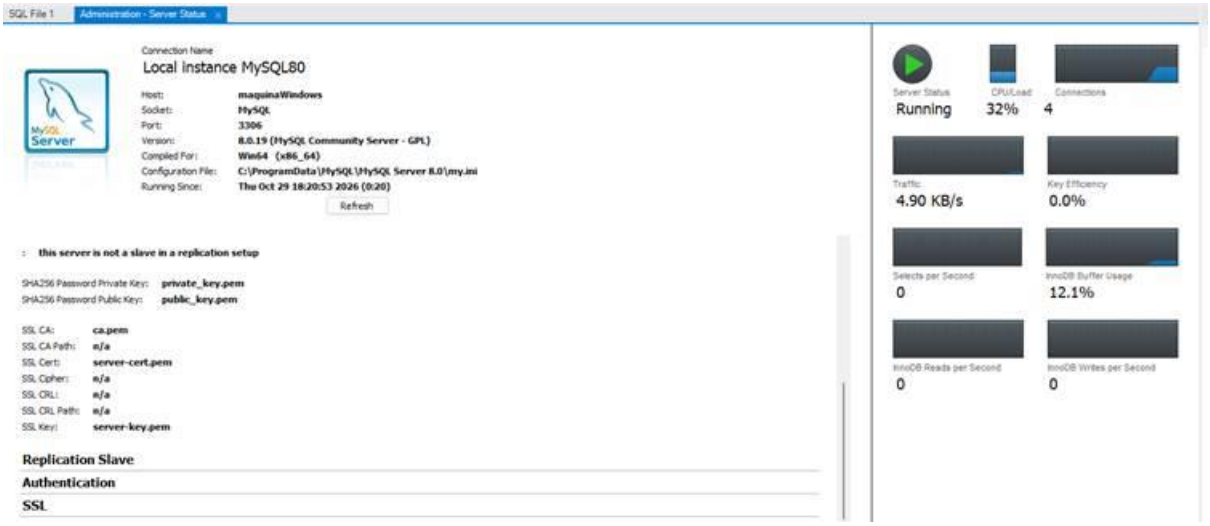
Para acceder a estas opciones, una vez creada una conexión, en el panel lateral izquierdo, tenemos la opción de "Status and System Variables".



Si la seleccionamos, tendremos un panel como el que se ve en la imagen, en el que podemos ver todas las variables de estado y sistema junto con sus valores.



Además, desde la opción "Server status" podemos ver diferente información sobre el estado del servidor, con un panel similar al que se ve en la siguiente imagen:



Workbench es especialmente útil para una primera aproximación visual, pero no sustituye al análisis detallado de las consultas, los registros y las estadísticas internas. Para investigar problemas complejos, suele ser necesario combinar varias herramientas.

2.2. Monitorización mediante consola SQL

Aunque las herramientas gráficas facilitan la interpretación de la información, también podemos monitorizar MySQL desde el cliente de consola.

Esta forma de monitorización es especialmente interesante para el perfil de administrador de SGBD ya que permite consultar información de forma precisa, automatizar comprobaciones y realizar posibles diagnósticos sin depender de interfaz gráfica.

Como ya hemos ido viendo en las anteriores unidades, MySQL proporciona diferentes sentencias SHOW para consultar información sobre su funcionamiento.

Podemos mostrar las **variables de estado globales del servidor**, donde encontraremos datos como contadores de conexiones, consultas, operaciones, actividad del sistema...

```
SHOW GLOBAL STATUS;
```

También podemos mostrar una variable concreta si conocemos su nombre. Por ejemplo, el máximo de conexiones simultaneas que han estado abiertas desde el inicio del servidor:

```
SHOW GLOBAL STATUS LIKE 'Max_used_connections';
```

O el número de conexiones que están abiertas actualmente en el servidor:

```
SHOW GLOBAL STATUS LIKE 'Threads_connected';
```

Estos valores ayudan a investigar si el servidor se aproxima a sus límites de conexiones, por lo que deben interpretarse junto con la configuración del servidor y la evolución de la carga de trabajo.

Otras variables útiles son:

- `Connections`: Número de intentos de conexión realizados al servidor desde que se inició.
- `Threads_running`: Número de hilos que no están simplemente esperando una nueva petición del cliente.
- `Slow_queries`: Número de consultas que han superado el umbral de lentitud establecido.

Además, como ya hemos visto en anteriores unidades, podemos consultar todas las variables que contengan unos caracteres concretos, por ejemplo, todas las variables cuyo nombre empiece por “threads”:

```
SHOW GLOBAL STATUS LIKE 'Threads%';
```

En todos los ejemplos se ha utilizado la palabra GLOBAL, que indica que estamos consultando el estado global del servidor. Dependiendo del contexto que queramos analizar, puede ser interesante consultar **valores asociados a la conexión actual**, consultando los valores de ámbito `SESSION`.

El comando SHOW también nos permite **consultar las variables de configuración** del servidor. Por ejemplo, podemos consultar todas las variables globales:

```
SHOW GLOBAL VARIABLES;
```

Para la consulta de variables, también podremos filtrar por nombre, con la misma sintaxis que en los ejemplos anteriores.

También podemos obtener **información sobre las conexiones** y sentencias que se están ejecutando:

```
SHOW FULL PROCESSLIST;
```

El resultado de este comando nos dará información como el identificador de la conexión, el usuario, el equipo de origen, la base de datos en la que se está trabajando, el estado de la conexión, y en caso de que corresponda, la sentencia SQL que se está ejecutando.

Esta información permite detectar consultas que llevan mucho tiempo ejecutándose o conexiones que puedan estar bloqueadas.

2.3. Performance Schema

Performance Schema es un componente de MySQL diseñado para recopilar información detallada sobre la ejecución interna del servidor.

Mientras que las sentencias `SHOW` permiten consultar indicadores generales, Performance Schema proporciona información más detallada sobre eventos, consultas, esperas, transacciones y otros aspectos del funcionamiento de MySQL.

La información recopilada se almacena en tablas del esquema `performance_schema`, que pueden consultarse mediante sentencias SQL normales. No es necesario conocer todas las tablas de este esquema. Para comenzar, interesa comprender qué información ofrece y aprender a consultar algunas de sus tablas más útiles. No obstante, la documentación oficial de MySQL tiene toda la información sobre este componente de MySQL.

Entre sus **características principales** destacan:

- Permite recopilar información sobre actividades internas del servidor.
- Puede registrar cuánto tiempo tardan determinadas operaciones.
- Permite la consulta de la información mediante consultas SQL.
- Es posible configurar qué instrumentos y eventos se recopilan.
- La información recopilada en sus tablas no constituye un historial persistente después de reiniciar el servidor.

Para comprobar si nuestro servidor tiene el Performance Schema habilitado, podemos consultar la variable de configuración correspondiente:

```
SHOW GLOBAL VARIABLES LIKE 'performance_schema';
```

Si está habilitado, podemos consultar las tablas disponibles:

```
SHOW TABLES FROM performance_schema;
```

Si la cuenta con la que estamos accediendo tiene los permisos necesarios, podemos acceder a todas ellas mediante consultas `SELECT`.

Una de las tareas más importantes de la monitorización es identificar qué grupos de consultas consumen más recursos. Para ello, podemos utilizar la tabla `performance_schema.events_statements_summary_by_digest` en la que se agrupan sentencias por una representación normalizada de su estructura, denominada Digest (más información en Anexo).

Algunas de las columnas más importantes de esta tabla son:

- `DIGEST_TEXT`: representación normalizada de la estructura de la sentencia SQL.
- `COUNT_STAR`: número de veces que se ha contabilizado ese grupo de sentencias.
- `SUM_TIMER_WAIT`: tiempo total acumulado de ejecución, expresado internamente en picosegundos.
- `AVG_TIMER_WAIT`: tiempo medio por ejecución, también expresado internamente en picosegundos.

Nota: un picosegundo equivale a la billonésima parte de un segundo, es decir, 10^{-12} segundos o 0,000000000001 segundos.

Aunque Performance Schema es una herramienta muy potente, hay que conocer sus **límites**:

- No sustituye a los registros persistentes del servidor.
- No explica automáticamente la causa de todos los problemas, únicamente nos da información que deberemos interpretar, con la que puede que encontremos alguna solución.
- Los datos deben interpretarse en relación con la carga de trabajo y el contexto.

2.4. Esquema sys

El esquema `sys` proporciona vistas y otros objetos que facilitan la interpretación de los datos recopilados por Performance Schema. Presenta parte de la información recopilada de una forma más sencilla de consultar e interpretar.

El esquema `sys` está incluido en las instalaciones habituales de MySQL a partir de la versión 5. Las vistas disponibles y sus columnas pueden variar ligeramente entre versiones de MySQL.

Una de las vistas más útiles es `sys.statement_analysis`, que presenta estadísticas agrupadas sobre las sentencias SQL analizadas. Podemos consultar las diez primeras filas ordenadas por tiempo total de ejecución:

```
SELECT * FROM sys.statement_analysis ORDER BY total_latency DESC LIMIT 10;
```

Algunas de las columnas más interesantes son:

- `query`: representación normalizada de la consulta.
- `exec_count`: número de ejecuciones contabilizadas.
- `total_latency`: tiempo total acumulado.
- `avg_latency`: tiempo medio por ejecución.
- `rows_examined`: número de filas examinadas.
- `rows_sent`: número de filas devueltas al cliente.

También podemos utilizar vistas del esquema sys para investigar las operaciones de entrada/salida.

Por ejemplo:

```
SELECT * FROM sys.io_global_by_file_by_latency ORDER BY total_latency DESC
LIMIT 10;
```

Esta vista resume la actividad de entrada/salida por archivo, agrupando información sobre el tiempo de espera asociado a las operaciones. Puede ayudar a identificar archivos que acumulan una cantidad importante de tiempo de espera, aunque los resultados deben interpretarse teniendo en cuenta el volumen de operaciones y el tipo de archivo.

2.5. Registros del servidor (logs)

Como ya hemos ido viendo, los registros son archivos donde MySQL almacena información sobre determinados acontecimientos, errores y operaciones. Los registros conservan la información de forma persistente, siempre que estén habilitados y configurados para ello.

MySQL dispone de varios tipos de registros o logs: registro de errores, registro general, registro de consultas lentas y registro binario. Para más información sobre ellos, consultar los apuntes de unidades anteriores.

2.6. Análisis de consultas

Cuando se detecta una consulta lenta, es necesario comprender cómo la está ejecutando MySQL, y analizar que operaciones está realizando.

Para ello, disponemos de diferentes herramientas que proporcionan información sobre el plan de ejecución y el funcionamiento del optimizador.

EXPLAIN permite consultar el plan de ejecución que MySQL prevé utilizar para una sentencia.

Por ejemplo:

```
EXPLAIN SELECT * FROM pedidos WHERE id_cliente = 125;
```

El resultado proporciona información sobre cómo se accederá a la tabla, qué índices se consideran y cuántas filas estima MySQL que tendrá que examinar. Entre las columnas más importantes del formato tradicional de EXPLAIN se encuentran:

- `Table`: tabla a la que hace referencia la operación
- `Type`: tipo de acceso previsto a la tabla
- `Possible_keys`: índices que podrían utilizarse
- `Key`: Índice que MySQL prevé utilizar, si corresponde.
- `Rows`: Estimación del número de filas que se examinarán.
- `Filtered:` Porcentaje estimado de filas que superarán determinadas condiciones de filtrado.
- `Extra:` Información adicional sobre la ejecución prevista.

Es importante interpretar las columnas conjuntamente. Por ejemplo, si key aparece como NULL, significa que no se ha seleccionado un índice para ese acceso a la tabla. No demuestra automáticamente que la consulta esté mal optimizada: en ocasiones, examinar muchas filas puede ser más eficiente que utilizar un índice. Del mismo modo, rows contiene una estimación, no un recuento exacto de las filas que se examinarán durante la ejecución real.

`EXPLAIN ANALYZE` permite ejecutar una consulta y mostrar información sobre el plan de ejecución real.

Por ejemplo:

```
EXPLAIN ANALYZE SELECT * FROM pedidos WHERE id_cliente = 125;
```

Con este comando, además de mostrar las operaciones realizadas, proporciona información sobre el tiempo empleado en las operaciones, el número real de filas procesadas, el número de veces que se ha ejecutado cada operación... Esta información es muy útil para investigar consultas costosas y comprobar si las estimaciones de MySQL se aproximan al comportamiento real.

Una diferencia importante entre `EXPLAIN` y `EXPLAIN ANALYZE` es que este último ejecuta la sentencia, por lo que debe utilizarse con cuidado, especialmente si la sentencia modifica datos.

2.7. Herramientas externas de monitorización

En una instalación local de MySQL, Workbench y las herramientas incorporadas al SGBD pueden ser suficientes para realizar numerosas tareas de administración. Sin embargo, en una organización con varios servidores, aplicaciones y servicios, puede resultar necesario disponer de un sistema centralizado de monitorización.

Estas soluciones permiten recopilar métricas de diferentes equipos, representar su evolución, establecer umbrales y generar notificaciones cuando se detectan anomalías. Algunas soluciones de este tipo son Prometheus, Grafana y MySQL Enterprise Monitor.

Las herramientas externas complementan las capacidades de MySQL permitiendo, entre otras cosas, centralizar la información de varios servidores, consultar la evolución de las métricas a lo largo del tiempo, visualizar tendencias mediante gráficos, establecer umbrales, generar alertas...

No sustituyen a las herramientas propias del SGBD, pero en sistemas grandes, pueden ser de mucha utilidad para centralizar la información y visualizarla de forma más sencilla.

3. ELEMENTOS Y PARÁMETROS SUSCEPTIBLES DE SER MONITORIZADOS

Para administrar correctamente un servidor MySQL no basta con saber utilizar las herramientas de monitorización: también es necesario conocer qué elementos debemos observar, qué información proporciona cada uno y cómo interpretar los valores obtenidos.

Es importante recordar que no existe un valor universal que determine cuándo un servidor funciona correctamente. Un consumo de CPU del 80 %, por ejemplo, puede ser normal en un servidor que está procesando una carga elevada, pero preocupante si aparece acompañado de tiempos de respuesta excesivos y un comportamiento anómalo. Por ello, las métricas deben interpretarse teniendo en cuenta las características del servidor, su carga de trabajo y la evolución de los valores a lo largo del tiempo.

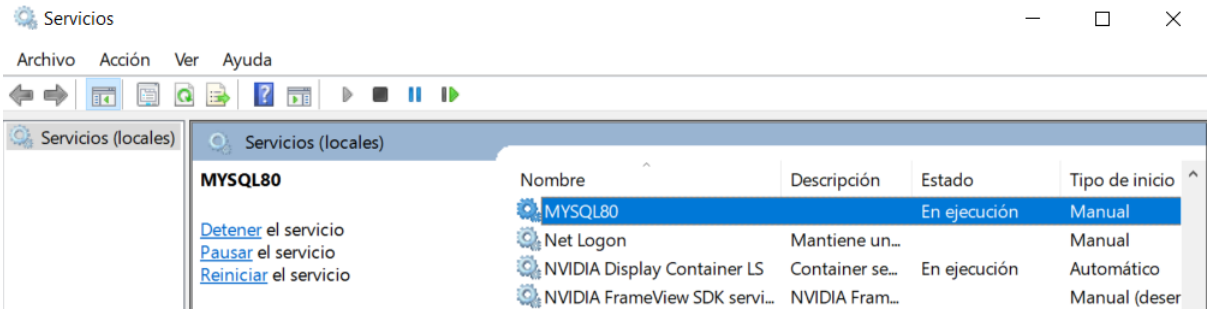
En este apartado estudiaremos los principales elementos que podemos monitorizar en MySQL, las variables y tablas que permiten obtener información sobre ellos y las herramientas que podemos utilizar para analizar los resultados.

3.1. Estado general del servidor y disponibilidad del servicio

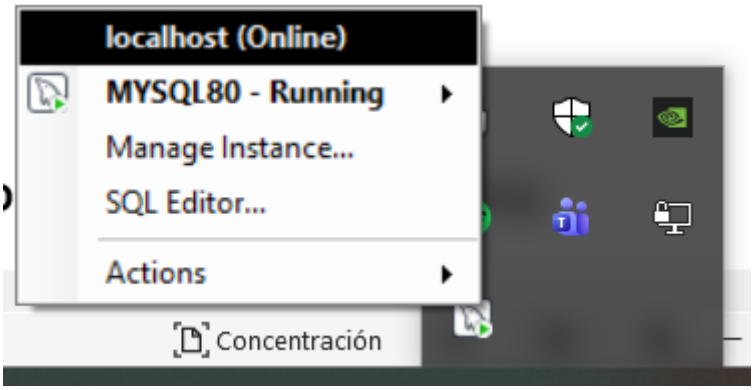
Uno de los primeros aspectos que deberemos monitorizar es el estado general del servidor MySQL. El objetivo es comprobar si el servicio está funcionando, cuánto tiempo lleva activo, si se está produciendo errores que puedan indicar problemas de disponibilidad o funcionamiento...

La **disponibilidad del servidor** indica si el servicio MySQL se encuentra operativo y puede atender las solicitudes de los clientes. Para comprobar el estado del servicio, existen diferentes formas.

En Windows, una forma es desde la aplicación Servicios: pulsamos Windows + R, ejecutamos services.msc, y localizamos el servicio correspondiente a MySQL, que puede aparecer como MySQL80 o similares. Si está activo, su estado debería ser “En ejecución”.



También en Windows, si tenemos instalado MySQL Notifier, podremos consultar fácilmente el estado del servidor desde la barra de tareas de Windows.



Desde MySQL podemos comprobar que el servidor responde ejecutando el comando:

```
SELECT VERSION();
```

Si obtenemos la versión instalada, significa que la conexión se ha establecido correctamente y que el servidor está atendiendo la solicitud.

Por otro lado, **el tiempo de actividad (uptime)** indica cuánto tiempo lleva funcionando el servidor desde su último inicio. Lo podemos consultar con el siguiente comando:

```
SHOW GLOBAL STATUS LIKE 'Uptime';
```

```
mysql> SHOW GLOBAL STATUS LIKE 'Uptime';
+---------------+---------+
| Variable_name | Value   |
+---------------+---------+
| Uptime        | 2433004 |
+---------------+---------+
1 row in set (0.21 sec)
```

Además, MySQL cuenta con diferentes contadores (variables de estado) que permiten detectar problemas durante el establecimiento y mantenimiento de las conexiones.

Algunos ejemplos son:

- `Aborted_connects`: intentos de conexión que no han podido completarse correctamente.
- `Aborted_clients`: conexiones interrumpidas por los clientes sin cerrarse correctamente.
- `Connection_errors_max_connections`: conexiones rechazadas por haberse alcanzado el límite máximo permitido.

Estos valores los podemos consultar mediante consultas del tipo:

```
SHOW GLOBAL STATUS LIKE 'Aborted%';
```

Si estos contadores aumentan de forma significativa, conviene investigar las causas. Por ejemplo, un incremento de `Connection_errors_max_connections` puede indicar que el servidor está recibiendo más conexiones simultáneas de las que permite su configuración.

3.2. Conexiones, sesiones y consultas activas

Las **conexiones** permiten que las aplicaciones y los usuarios se comuniquen con el servidor MySQL. Cada cliente que accede al servidor establece una conexión, que puede utilizar para ejecutar distintas sentencias SQL.

Monitorizar las conexiones es importante porque un número excesivo de conexiones simultáneas puede incrementar el consumo de recursos y afectar al rendimiento.

Como ya hemos ido trabajando en el módulo, MySQL proporciona varias variables de estado para conocer la actividad relacionada con las conexiones. Algunas de las más interesantes con las siguientes:

- `Threads_connected:` Número de conexiones abiertas actualmente.
- `Threads_running`: Número de hilos que no están en estado de espera de una nueva petición del cliente.
- `Max_used_connections`: Máximo número de conexiones simultáneas utilizadas desde el inicio del servidor.
- `Connections:` Número de intentos de conexión realizados desde el inicio, tanto satisfactorios como fallidos.
- `Threads_created`: Número de hilos creados para atender conexiones.
- `Threads_cached`: Número de hilos disponibles en la caché de hilos.

También podemos consultar el límite de conexiones configurado. Se encuentra en la variable del servidore de nombre `max_connections`, que establece el número máximo

de conexiones simultáneas permitidas. No se debe confundir con la variable de estado `max_used_connections`, que indica que máximo de conexiones simultáneas que realmente se ha alcanzado.

Además, también podemos realizar una inspección de las conexiones activas con el comando siguiente:

```
SHOW FULL PROCESSLIST;
```

Con él podemos encontrar una conexión cuyo estado indique que está ejecutando una consulta y que lleva varios segundos en esa situación. Esto puede servir para localizar operaciones que necesitan investigarse.

Algunos síntomas que pueden indicar problemas relacionados con las conexiones son:

- El número de conexiones se aproxima repetidamente a `max_connections.`
- Aumenta el contador `connection_errors_max_connections`.
- Se acumulan conexiones que permanecen inactivas durante mucho tiempo.
- Las aplicaciones reciben errores al intentar conectarse.

3.3. Rendimiento de las consultas SQL

Las consultas SQL son uno de los elementos más importantes que debemos monitorizar, ya que una consulta ineficiente puede consumir muchos recursos y afectar a otras operaciones del servidor.

Es importante recordar que no todas las consultas tienen el mismo coste computacional. Algunas recuperan unos pocos registros, mientras que otras examinan millones de filas, realizan agrupaciones, ordenaciones, combinan varias tablas...

Para poder valorar el rendimiento de las consultas, MySQL mantiene diferentes contadores (variables de estado) que permiten conocer cuántas sentencias se han ejecutado. Entre ellos destacan:

- `questions`: Número de sentencias enviadas al servidor por los clientes.
- `queries`: Número de sentencias ejecutadas por el servidor, incluyendo las ejecutadas dentro de programas almacenados.
- `slow_queries`: Número de consultas que han superado el umbral de duración configurado mediante `long_query_time`.

Es importante recordar que estos valores son acumulados desde el inicio del servidor o desde el último reinicio de los contadores correspondientes.

En cuanto al tiempo de ejecución de las consultas, es decir, el tiempo que necesita una consulta para completar su trabajo, es uno de los indicadores más importantes para

identificar las operaciones más lentas. Esta información la podemos obtener en la vista `sys.statement_analysis`.

Otro indicador importante es la cantidad de filas que MySQL necesita examinar para obtener el resultado de una consulta. Debemos distinguir entre las filas examinadas (los registros que MySQL necesita revisar durante el procesamiento) y las filas devueltas (los registros que finalmente se devuelven al cliente). Una diferencia muy elevada entre filas examinadas y filas devueltas puede indicar que una consulta realiza más trabajo del necesario.

Por tanto, para detectar consultas que pueden perjudicar al rendimiento, deberemos combinar distintos indicadores como son un tiempo de ejecución elevado, un tiempo total acumulado elevado, gran cantidad de filas examinadas, número excesivo de ejecuciones...

3.4. Uso de CPU, memoria, almacenamiento y actividad de entrada/salida

El rendimiento de MySQL también va a estar condicionado por los recursos físicos del equipo donde se ejecuta el servidor. Por eso, es necesario monitorizar el consumo de recursos que utiliza.

La CPU es el componente encargado de ejecutar las instrucciones de los procesos, y por tanto MySQL va a utilizar la CPU para procesar consultas SQL, comparar y filtrar registros, realizar ordenaciones y agrupaciones de datos, ejecutar operaciones de unión entre tablas, gestionar conexiones y transacciones, etc. Un consumo elevado de CPU puede indicar que el servidor está procesando un volumen importante de trabajo o que determinadas consultas requieren demasiadas operaciones.

En Windows, podemos monitorizar el uso de la CPU desde el listado de procesos de la máquina y mirando qué porcentaje de CPU utiliza proceso del servidor MySQL. Esto también se puede consultar desde MySQL Workbench.

La memoria RAM permite almacenar temporalmente información que el servidor necesita utilizar durante su funcionamiento. MySQL necesita memoria para tareas como mantener datos e índices en caché, gestionar las conexiones, ejecutar consultas, administrar las transacciones, etc.

En Windows podemos monitorizar el consumo de memoria del proceso del servidor MySQL desde el administrador de tareas.

Otro factor para tener en cuenta es que la velocidad del almacenamiento donde se guardan los archivos de datos, índices, etc. puede influir notablemente en el rendimiento, especialmente cuando las consultas necesitan recuperar muchos datos que no están disponibles en memoria.

3.5. InnoDB Buffer Pool

Como veíamos al principio del módulo, InnoDB es el motor de almacenamiento predeterminado de MySQL, y el más utilizado en la mayoría de las instalaciones actuales. Una de sus estructuras fundamentales para el rendimiento es el InnoDB Buffer Pool, una zona de memoria que almacena copias de páginas de datos e índices para reducir la necesidad de acceder continuamente al almacenamiento.

Cuando MySQL necesita consultar información almacenada en una tabla InnoDB, puede ocurrir que los datos ya estén disponibles en memoria o que sea necesario recuperarlos desde el disco. El buffer pool permite mantener en memoria las páginas utilizadas por InnoDB para acelerar las operaciones posteriores. Esto reduce las operaciones de entrada/salida y puede mejorar considerablemente el rendimiento.

El buffer pool no almacena necesariamente tablas completas: trabaja con páginas de datos, que son las unidades que InnoDB utiliza para organizar la información.

El tamaño del buffer pool se controla mediante la variable de configuración `innodb_buffer_pool_size`, por lo que podemos consultar su valor actual mediante:

```
SHOW GLOBAL VARIABLES LIKE 'innodb_buffer_pool_size';
```

El resultado se expresa en bytes.

Un buffer pool demasiado pequeño puede provocar que InnoDB tenga que recuperar frecuentemente páginas desde el almacenamiento. Por el contrario, asignar demasiada memoria puede dejar recursos insuficientes para Windows, MySQL y otros procesos. Por ello, su tamaño debe adaptarse a la memoria disponible y a las necesidades de la carga de trabajo.

MySQL proporciona variables de estado que permiten conocer cómo está funcionando el buffer pool. Algunas de las más importantes son:

- `Innodb_buffer_pool_read_requests:` Número de solicitudes lógicas de lectura realizadas al buffer pool.
- `Innodb_buffer_pool_reads`: Número de solicitudes de lectura que han requerido recuperar páginas desde el almacenamiento.
- `Innodb_buffer_pool_pages_total`: Número total de páginas del buffer pool.
- `Innodb_buffer_pool_pages_data:` Número de páginas que contienen datos.
- `Innodb_buffer_pool_pages_free`: Número de páginas libres.

Cuando una operación modifica datos, las páginas correspondientes pueden quedar modificadas en el buffer pool antes de que los cambios se escriban definitivamente en los archivos de datos. Estas páginas se denominan **dirty pages o páginas sucias**.

InnoDB utiliza mecanismos internos para escribirlas en el almacenamiento y mantener la coherencia de los datos.

Podemos consultar su número mediante:

```
SHOW GLOBAL STATUS LIKE 'Innodb_buffer_pool_pages_dirty';
```

Un número elevado no significa necesariamente que exista un problema. Lo importante es observar cómo evoluciona y si InnoDB puede escribir las páginas modificadas con suficiente rapidez.

3.6. Bloqueos, esperas, transacciones y concurrencia

Cuando varios usuarios trabajan simultáneamente con una base de datos, es posible que intenten acceder o modificar los mismos datos. Para mantener la integridad y evitar modificaciones incompatibles, MySQL utiliza mecanismos de bloqueo. Estos mecanismos son necesarios, pero pueden provocar esperas que afectan al rendimiento.

Un **bloqueo (lock)** es un mecanismo que controla el acceso simultáneo a determinados datos.

Si se está realizando una transacción y esta todavía no ha finalizado, los registros implicados se mantendrán bloqueados. Mientras se mantiene el bloqueo, otra transacción puede tener que esperar si está intentando realizar una operación incompatible sobre la misma fila.

Este comportamiento es normal y necesario para evitar inconsistencias. El problema aparece cuando las transacciones mantienen bloqueos durante demasiado tiempo o se producen numerosas esperas.

MySQL proporciona una sentencia que permite obtener información detallada sobre el estado del motor de almacenamiento:

```
SHOW ENGINE INNODB STATUS;
```

Esta sentencia devuelve un informe con información sobre diferentes aspectos internos de InnoDB, como transacciones, bloqueos y esperas, interbloqueos recientes, estado del buffer pool...

3.7. Estadísticas de tablas

Las tablas y los índices son elementos fundamentales de cualquier base de datos relacional. Su tamaño, estructura, y utilización pueden influir directamente en el rendimiento de las consultas.

Por ello, es importante monitorizar aspectos como el número de registros, el espacio ocupado, el tamaño de los índices, etc.

A medida que una base de datos almacena más información, aumenta el espacio que necesita para guardar sus datos e índices. Podemos consultar el tamaño de las tablas mediante la vista `information_schema.tables`.

Por ejemplo, para conocer el tamaño de las tablas de una base de datos, podemos ejecutar:

```
SELECT
    TABLE_NAME, TABLE_ROWS, ROUND(DATA_LENGTH / 1024 / 1024, 2) AS datos_MB,
    ROUND(INDEX_LENGTH / 1024 / 1024, 2) AS indices_MB,
    ROUND((DATA_LENGTH + INDEX_LENGTH) / 1024 / 1024, 2) AS total_MB
FROM information_schema.tables
WHERE TABLE_SCHEMA = 'nombreBaseDeDatos'
ORDER BY (DATA_LENGTH + INDEX_LENGTH) DESC;
```

En este ejemplo de consulta, se van a mostrar las columnas siguientes:

- `Table_name`: que muestra el nombre de la tabla.
- `Table_rows`: el número estimado de filas de la tabla.
- `Data_length`: el espacio ocupado en bytes.
- `Index_length`: el espacio ocupado por los índices, expresado en bytes.

Si necesitamos saber el número real de filas de una tabla concreta, se puede utilizar la siguiente sentencia:

```
SELECT COUNT(*) FROM nombreTabla;
```

Además, podemos consultar diferentes estadísticas de utilización de tablas, ya que MySQL proporciona información sobre las operaciones de lectura, escritura y acceso a tablas.

Para ello se puede consultar la siguiente tabla, que recoge estadísticas de operaciones de entrada/salida por tabla. Entre otros datos, permite conocer el número de operaciones de lectura, inserción, actualización y borrado, así como los tiempos asociados a estas operaciones.

```
SELECT * FROM performance_schema.table_io_waits_summary_by_table;
```

Para facilitar su interpretación, también se pueden utilizar vistas del esquema sys. Esta vista permite identificar las tablas que acumulan más tiempo en operaciones de entrada/salida, y puede resultar útil para detectar tablas que reciben una gran cantidad de actividad o que están asociadas a operaciones costosas.

```
SELECT * FROM sys.schema_table_statistics ORDER BY total_latency DESC LIMIT 10;
```

4. OPTIMIZACIÓN DEL RENDIMIENTO EN MySQL

La **optimización del rendimiento de un SGBD** consiste en aplicar medidas que permitan mejorar la velocidad de ejecución de las consultas, reducir el consumo de recursos y garantizar que el sistema pueda atender correctamente a varios usuarios de forma simultánea.

No existe una única configuración válida para todos los servidores. Las medidas que deben aplicarse dependen del tamaño de la base de datos, del número de usuarios, del tipo de consultas que se ejecutan y de los recursos disponibles.

En este apartado estudiaremos distintas técnicas para optimizar MySQL, tanto desde el punto de vista de la configuración del servidor como desde el diseño de las bases de datos y la ejecución de las consultas.

4.1. Optimización de la instancia MySQL

Una **instancia MySQL** está formada por el servidor en ejecución y los recursos que utiliza para gestionar las bases de datos, las conexiones y las consultas.

Optimizar la instancia significa ajustar su configuración para que aproveche adecuadamente los recursos disponibles y responda de forma eficiente a las peticiones de los clientes.

A continuación se enumeran algunos de los parámetros más importantes.

- **Número máximo de conexiones**

El parámetro `max_connections` establece el número máximo de conexiones simultáneas que puede aceptar el servidor.

Si el número de conexiones disponibles se agota, los nuevos clientes pueden recibir un error indicando que se ha superado el número máximo de conexiones.

Podemos consultar su valor mediante:

```
SHOW VARIABLES LIKE 'max_connections';
```

Podemos modificar su valor:

```
SET GLOBAL max_connections = 200;
```

Este cambio se aplica durante la ejecución actual del servidor, pero no necesariamente se conserva después de reiniciarlo. Para establecer un valor persistente, podemos modificar el archivo de configuración de MySQL:

```
[mysqld]

max_connections = 200
```

Tras modificar el archivo, será necesario reiniciar el servicio MySQL para aplicar el cambio.

Aumentar `max_connections` no siempre mejora el rendimiento. Cada conexión puede consumir memoria y otros recursos. Si se establece un valor demasiado elevado, el servidor podría quedarse sin memoria disponible.

- **Tamaño del buffer pool de InnoDB**

InnoDB es el motor de almacenamiento predeterminado de MySQL y utiliza una zona de memoria denominada buffer pool para almacenar en caché datos e índices.

Su objetivo es reducir los accesos al disco. Cuando MySQL necesita consultar un registro, intenta localizarlo primero en el buffer pool. Si no está disponible, tendrá que recuperarlo del almacenamiento.

El parámetro que controla su tamaño es:

```
SHOW VARIABLES LIKE 'innodb_buffer_pool_size';
```

En servidores dedicados a MySQL, el buffer pool suele ser uno de los parámetros más importantes para mejorar el rendimiento. Sin embargo, debe dimensionarse teniendo en cuenta la memoria RAM disponible y los recursos que necesitan el sistema operativo y el resto de aplicaciones.

- **Tamaño de las tablas temporales**

MySQL puede crear tablas temporales durante la ejecución de determinadas consultas, por ejemplo, cuando se realizan operaciones de agrupación u ordenación.

Los parámetros `tmp_table_size` y `max_heap_table_size` intervienen en el tamaño máximo de las tablas temporales que pueden mantenerse en memoria mediante el motor MEMORY.

Si una tabla temporal supera los límites establecidos, MySQL puede necesitar utilizar almacenamiento en disco, lo que puede incrementar el tiempo de ejecución. No obstante, aumentar estos parámetros indiscriminadamente puede incrementar el consumo de memoria, especialmente cuando existen muchas conexiones simultáneas.

4.2. Optimización de los recursos del servidor

El rendimiento de MySQL depende directamente de los recursos físicos y lógicos del equipo donde se ejecuta.

A continuación, se verán los principales recursos que se deben tener en cuenta.

- **CPU**

La CPU interviene en la ejecución de consultas, operaciones de ordenación, agrupaciones, cálculos y gestión de conexiones.

Un consumo elevado y continuado puede indicar que el servidor está ejecutando demasiadas operaciones o que algunas consultas requieren un procesamiento excesivo.

Algunas medidas para reducir la carga de CPU son: optimizar las consultas SQL, evitar recuperar columnas innecesarias, utilizar índices, reducir las operaciones de ordenación y agrupación innecesarias...

- **Memoria RAM**

MySQL utiliza memoria para mantener datos e índices en caché, gestionar conexiones y ejecutar consultas. Una memoria insuficiente puede provocar un mayor número de accesos al disco, reduciendo el rendimiento.

Algunas medidas que pueden mejorar su aprovechamiento son: dimensionar adecuadamente el buffer pool de InnoDB, ajustar los parámetros relacionados con tablas temporales, evitar excesivas conexiones simultaneas...

Es importante recordar que MySQL comparte la memoria RAM con el sistema operativo y con todos los programas que se encuentren activos.

- **Almacenamiento y operaciones de entrada/salida**

Las operaciones de lectura y escritura en disco pueden convertirse en un cuello de botella, especialmente cuando se trabaja con bases de datos grandes.

Algunas medidas para mejorar el rendimiento son evitar lecturas innecesarias aplicando índices, reducir las operaciones de escritura que no aporten valor, mantener espacio libre suficiente en las unidades de almacenamiento...

En un servidor MySQL también es importante supervisar el espacio ocupado por los archivos de datos, los registros de transacciones y los archivos de log.

4.3. Actualización de estadísticas del optimizador de consultas

MySQL incorpora un componente denominado optimizador de consultas, cuya función es determinar cómo ejecutar una sentencia SQL de la forma más eficiente posible.

Para tomar decisiones, el optimizador utiliza información estadística sobre las tablas y los índices. Por ejemplo, el optimizador puede necesitar estimar cuántas filas contiene una tabla, cuántos valores diferentes existen en una columna, que proporción de registros cumple una condición...

Estas estadísticas permiten al optimizador estimar el coste de las diferentes alternativas de ejecución. Si las estadísticas no representan adecuadamente la situación actual de la tabla, podría elegir un plan de ejecución poco eficiente.

Por eso, es importante actualizar las estadísticas cuando se producen modificaciones importantes en los datos.

MySQL permite actualizar las estadísticas de una tabla mediante:

```
ANALYZE TABLE clientes;
```

Esta operación recopila información estadística que puede utilizar el optimizador, ayudándole a tomar mejores decisiones de ejecución.

4.4. Particionamiento de tablas

El particionamiento consiste en dividir una tabla grande en varias partes más pequeñas, denominadas particiones, manteniendo la posibilidad de consultar los datos como si pertenecieran a una única tabla.

Esta técnica puede resultar útil cuando se trabaja con tablas que contienen grandes cantidades de registros. Por ejemplo, una tabla ventas podría contener millones de operaciones correspondientes a diferentes años. En lugar de almacenar todos los registros en una única partición, podríamos dividirlos según el año de venta.

MySQL dispone de diferentes métodos para dividir los datos:

- RANGE: distribuye los registros según intervalos de valores. Es habitual utilizarlo con fechas o cantidades numéricas.
- LIST: distribuye los registros según la lista de valores determinados.
- HASH: distribuye los registros aplicando una función hash sobre una columna.
- KEY: utiliza una función hash interna de MySQL para distribuir los registros.

Por ejemplo, imaginemos una tabla que almacena las ventas de una empresa y queremos dividir los registros por año. Podemos crear una tabla particionada de la siguiente forma:

```
CREATE TABLE ventas ( id INT NOT NULL,  fecha DATE NOT NULL, importe DECIMAL(10,2), PRIMARY KEY (id, fecha))

PARTITION BY RANGE (YEAR(fecha)) (

    PARTITION p2024 VALUES LESS THAN (2025),

    PARTITION p2025 VALUES LESS THAN (2026),

    PARTITION p2026 VALUES LESS THAN (2027),

    PARTITION pfuturo VALUES LESS THAN MAXVALUE);
```

En este ejemplo, los registros anteriores a 2025 se almacenan en p2024, los registros de 2025 en p2025, y así sucesivamente.

La tabla se consulta sin tener en consideración las particiones:

```
SELECT * FROM ventas WHERE fecha >= '2025-01-01'  AND fecha < '2026-01-01';
```

MySQL aplicará una técnica denominada **partition pruning** (eliminación de particiones), que consiste en identificar qué particiones contienen los datos que necesita una consulta, y evitar acceder a las demás.

Por tanto, las principales ventajas del particionamiento son:

- Facilita la gestión de tablas muy grandes.
- Puede reducir la cantidad de datos a examinar en determinadas consultas.
- Permite realizar operaciones de mantenimiento sobre particiones concretas, facilitando la eliminación de conjuntos de datos antiguos mediante operaciones sobre particiones.

Los principales inconvenientes son:

- Aumenta la complejidad del diseño y mantenimiento.
- No garantiza que todas las consultas sean más rápidas.
- Requiere planificar adecuadamente el criterio de particionamiento.

4.5. Paralelización de consultas

La paralelización consiste en dividir un trabajo en varias tareas que pueden ejecutarse simultáneamente, aprovechando diferentes núcleos de CPU. En algunos sistemas gestores de bases de datos, determinadas consultas pueden distribuirse en varios procesos o hilos para reducir el tiempo de ejecución.

MySQL no dispone de un mecanismo de paralelización de consultas SQL, aunque algunos SGBD como Oracle Database sí cuenta con ello.

4.6. Optimización de consultas SQL

Una de las formas más importantes de mejorar el rendimiento de MySQL consiste en optimizar las consultas SQL.

Algunas formas sencillas de optimizar las consultas de SQL son:

- **Evitar SELECT ***

Una práctica habitual consiste en recuperar todas las columnas de una tabla, aunque solo necesitemos algunas.

```
SELECT * FROM clientes;
```

Si únicamente necesitamos algunas columnas de la tabla, es preferible especificarlo en la consulta, reduciendo la cantidad de información que debe transferirse y procesarse.

- **Filtrar los registros correctamente**

La cláusula `WHERE` permite limitar los registros que devuelve una consulta.

Es importante que las condiciones de esta cláusula estén correctamente planteadas y que cuando sea necesario, existan también índices de búsqueda.

- **Limitar los resultados**

Cuando no necesitamos todos los registros que nos va a devolver una consulta, podemos utilizar LIMIT.

Por ejemplo:

```
SELECT * FROM productos ORDER BY precio DESC LIMIT 10;
```

Esta consulta devuelve únicamente los diez productos más caros.

`LIMIT` no garantiza que MySQL evite procesar todos los registros. Si debe ordenar una gran cantidad de filas antes de devolver las diez primeras, la operación puede seguir siendo costosa.

5. ÍNDICES

Los índices son estructuras de datos que permiten localizar registros de una tabla de forma más rápida, evitando tener que recorrer todos sus registros en determinadas consultas.

Podemos compararlos con el índice alfabético de un libro. Si queremos encontrar información sobre un concepto, no necesitamos leer todas las páginas: buscamos el término en el índice y consultamos la página indicada. En una base de datos sucede algo parecido. Un índice permite que MySQL encuentre los registros que cumplen una condición sin tener que examinar necesariamente toda la tabla.

Por ejemplo, vamos a partir de la siguiente tabla con datos de alumnos.

```
Alumnos(id, nombre, apellidos, fechaNacimiento, DNI, localidadResidencia)
```

Si queremos localizar todos los alumnos que viven en Oviedo:

```
SELECT * FROM alumnos WHERE ciudad = 'Oviedo';
```

Si no hay un índice creado sobre la columna `localidadResidencia`, MySQL va a tener que examinar todos los registros de la tabla para comprobar cuáles cumplen esa condición. Este procedimiento se denomina recorrido completo de tabla (full table scan). Registro a registro, comprobará si cada uno de los alumnos vive en Oviedo.

Sin embargo, si creamos un índice sobre la columna ciudad, MySQL dispondrá de una estructura adicional que le permitirá localizar más rápidamente los registros correspondientes a cada ciudad.

Conceptualmente, podemos imaginar que el índice contiene algo parecido a esto:

- Avilés: registros 3, 5, 7, 8, 9, 10
- Gijón: registros 2, 4, 6, 12
- Oviedo: registros 1, 13, 14, 15

Al consultar el índice, el SGBD localizará "Oviedo" en el índice, y podrá acceder directamente a los registros (filas de la BBDD) correspondientes, sin necesidad de consultar todas las filas de la tabla comprobado el valor de esa columna.

Por tanto, hemos visto que la función principal de un índice es reducir la cantidad de trabajo necesaria para localizar determinados registros. No obstante, debemos tener en cuenta que un índice no acelera automáticamente todas las consultas. MySQL decide si resulta conveniente utilizarlo en función de la consulta y de los datos que contiene la tabla.

5.1. Almacenamiento de índices

Un índice es una estructura adicional que el SGBD mantiene para facilitar el acceso a los datos, pero no sustituye a una tabla, ni modifica su contenido.

Cuando creamos un índice la tabla sigue almacenando sus registros, MySQL crea una estructura adicional con los valores de las columnas indexadas, y el índice ocupará un espacio de almacenamiento adicional.

En el InnoDB hay una particularidad importante. Las tablas se organizan físicamente en función de su clave primaria. Esta organización se denomina índice agrupado (clustered index), y va a ser el índice principal de todas las tablas. Los demás índices serán índices secundarios, que almacenarán el valor de la columna indexada junto con la clave primaria de los registros correspondientes.

5.2. Tipos de índices en MySQL

- **Índice PRIMARY KEY**

Cuando definimos una clave primaria, MySQL crea automáticamente el índice correspondiente. Esto no ocurre con todos los SGBD.

- **Índice UNIQUE**

Un índice UNIQUE garantiza que no existan valores duplicados en una columna o conjunto de columnas indexadas.

Por ejemplo, si queremos evitar que dos alumnos tengan el mismo correo electrónico:

```
CREATE TABLE alumnos (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100),
    email VARCHAR(150) UNIQUE
);
```

También podemos crear el índice después de haber creado la tabla:

```
CREATE UNIQUE INDEX idx_email ON alumnos(email);
```

- **Índice normal o secundario**

Es un índice que permite acelerar las búsquedas, pero no impide que existan valores repetidos. Volviendo al ejemplo de los alumnos que viven en Oviedo, podríamos crear el siguiente índice:

```
CREATE INDEX idx_ciudad ON alumnos(localidadResidencia);
```

Ahora MySQL puede optar por realizar la consulta utilizando esta nueva estructura creada.

- **Índices compuestos**

Un índice compuesto es aquel que está formado por dos o más columnas.

Por ejemplo:

```
CREATE INDEX idx_cliente_fecha ON pedidos(id_cliente, fecha);
```

- **Índices FULLTEXT**

Los índices FULLTEXT están diseñados para realizar búsquedas de texto en columnas de tipo textual (TEXT).

De esta forma, podremos buscar determinados términos en los registros.

A diferencia de una búsqueda convencional con LIKE, los índices FULLTEXT están preparados para localizar términos en textos y ofrecen diferentes modalidades de búsqueda.

5.3. Trabajar con índices en SQL

Podemos crear índices de dos maneras principales: en la creación o modificación de una tabla, o en una sentencia separada.

Podemos incluir un índice en la definición de una tabla, por ejemplo:

```
CREATE TABLE productos (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100),
    precio DECIMAL(10,2),
    categoria VARCHAR(50),
    INDEX idx_categoria (categoria)
);
```

Para **crear un índice** sobre una tabla ya existente, la sintaxis es la siguiente:

```
CREATE [UNIQUE] INDEX nombreIndice ON nombreTabla (columna);
```

Por ejemplo:

```
CREATE INDEX idx_residencia ON alumnos(localidadResidencia);
```

Para crear un índice sobre una tabla ya existente, también podemos utilizar `ALTER TABLE`:

```
ALTER TABLE productos ADD INDEX idx_categoria (categoria);
```

Cuando ya tenemos una base de datos con índices creados, es importante conocer cuáles existen sobre cada tabla y qué columnas incluyen. Esto lo podemos hacer con el siguiente comando:

```
SHOW INDEX FROM nombreTabla;
```

El resultado incluye información sobre los índices existentes: si admite valores duplicados, nombre del índice, nombre de la columna indexada...

Si queremos modificar un índice, no existe una sentencia que nos permita directamente modificar las columnas de un índice ya creado. Deberemos eliminar el índice creado, y crear uno nuevo.

Para **eliminar un índice**, la sintaxis es la siguiente:

```
DROP INDEX nombreIndice ON nombreTabla;
```

Al eliminar un índice no se eliminan los registros de la tabla. Únicamente desaparece la estructura auxiliar. Si únicamente queremos **modificar el nombre del índice**, sí lo podemos hacer sin necesidad de eliminarlo. La sintaxis es:

```
ALTER TABLE nombreTabla
RENAME INDEX nombreIndice_viejo TO nombreIndice_nuevo;
```

5.4. ¿Cómo saber si las consultas están usando los índices creados?

Crear un índice no garantiza que el SGBD vaya a utilizarlo. Para comprobar si efectivamente se está utilizando al realizar consultas, podemos utilizar EXPLAIN para que nos muestre el plan de ejecución elegido por el optimizador.

Por ejemplo, volviendo al ejemplo de los alumnos residentes en Oviedo:

```
EXPLAIN SELECT * FROM alumnos WHERE ciudad = 'Oviedo';
```

En el resultado tendremos que prestar atención a los campos:

- `Type`: tipo de acceso utilizado para recuperar los registros.
- `Possible_keys`: índices que podrían utilizarse.
- `Key`: índice seleccionado por MySQL.
- `Rows`: número estimado de registros que se examinarán.

ANEXO: SQL DIGEST

Se puede definir el SQL Digest como una representación estandarizada o normalizada de una sentencia SQL, que se utiliza para agrupar consultas similares.

El analizador del motor de la base de datos sustituye los valores específicos (parámetros, literales, cadenas de texto...) por comodines, pudiendo así agrupar diferentes sentencias con una misma estructura, sin tener en cuenta sus valores concretos.

Esto permite a diferentes herramientas, como MySQL Performance Schema rastrear la frecuencia de ejecución, el tiempo acumulado, etc. de estas sentencias, detectando patrones de consultas lentas sin importar los datos exactos que estén buscando los usuarios.

Por ejemplo:

- Consulta A: `SELECT * FROM empleados WHERE apellido1 = 'Sanchez'`
- Consulta B: `SELECT * FROM empleados WHERE apellido1 = 'Lopez'`
- Digest: `SELECT * FROM empleados WHERE apellido1 = ?`

Otro ejemplo:

- Consulta A: `SELECT id, nombre FROM usuarios WHERE edad > 18 AND ciudad = 'Madrid';`
- Consulta B: `SELECT id, nombre FROM usuarios WHERE edad > 30 AND ciudad = 'Barcelona';`

- Digest: `SELECT id , nombre FROM usuarios WHERE edad > ? AND ciudad = ?;`

Un último ejemplo:

- Consulta A: `INSERT INTO productos (nombre, precio) VALUES ('Teclado Mecánico', 85.50);`
- Consulta B: `INSERT INTO productos (nombre, precio) VALUES ('Ratón Gamer', 45.00);`
- Digest: `INSERT INTO productos ( nombre , precio ) VALUES ( ? , ? );`

Con las consultas normalizadas Digest calcula un hash, que será el identificador de esa estructura de sentencia. Todas las sentencias con la misma estructura, es decir, con una misma normalización Digest, compartirán un mismo hash.

En la tabla de performance_schema.events_statements_summary_by_digest de MySQL, junto al hash de cada sentencia aparecerá la estructura Digest, el número de veces que se ha ejecutado, y el tiempo total que han tardado dichas ejecuciones.

De esta forma el administrador de un SGBD puede consular qué sentencias se ejecutan más o menudo, o consumen un gran tiempo de ejecución, y puede decidir optimizarlas.

BIBLIOGRAFÍA

Documentación MySQL. Performance Schema. Disponible en: https://dev.mysql.com/doc/refman/8.4/en/performance-schema.html

Documentación MySQL. Sys Schema. Disponible en: https://dev.mysql.com/doc/refman/8.4/en/sys-schema.html