

Desenvolvimento de sistemas

Desempenho de banco de dados: monitoramento e gerenciamento, detecção de gargalos de desempenho, *tuning* e otimização de consultas, particionamento, escalabilidade do banco de dados

Garantir o desempenho suave e eficaz do banco de dados é um dos principais aspectos para qualquer negócio, não importa quão grande ou pequeno ele seja. Assim, obter uma visão abrangente do tempo de atividade ou da carga do banco de dados é fundamental em algumas situações, e este monitoramento deve ser realizado regularmente.

Por esse motivo, no mercado, existem muitas ferramentas de monitoramento para rastrear, analisar e medir o desempenho dos bancos de dados e investigar seus possíveis problemas. Essas ferramentas ajudam não apenas a otimizar as consultas, mas também a visualizar as métricas do banco de dados, coletar estatísticas e mostrar o *status* do desempenho para entender o impacto no desempenho geral e solucionar possíveis problemas.

A otimização do desempenho do banco de dados pode resultar nos seguintes benefícios:

- ◆ Acelerar a recuperação de dados
- ◆ Aumentar significativamente o desempenho da consulta
- ◆ Identificar o desempenho lento
- ◆ Prevenir possível tempo de inatividade e consumo excessivo de recursos
- ◆ Definir o impacto das alterações do banco de dados

Métricas de desempenho do banco de dados MySQL

O MySQL fornece várias métricas úteis as quais você deve monitorar para detectar gargalos e analisar quais consultas devem ser otimizadas.

As métricas de monitoramento de desempenho de banco de dados mais importantes abrangem o seguinte:

Clique ou toque para visualizar o conteúdo.

System

Representa os recursos do sistema no servidor. Eles incluem o uso de CPU (unidade central de processamento), de memória e de disco, a largura de banda de rede, as solicitações de leitura/gravação, as estatísticas de espera de entrada/saída, o tempo médio de leitura/gravação etc.

Taxa de transferência da consulta

Consiste em uma carga de trabalho do banco de dados que pode ser medida com o número de consultas por segundo. Com essa métrica, você pode analisar como seu servidor executa e processa consultas ao longo do tempo e também como acompanhar o desempenho de instruções específicas (**SELECT, INSERT, UPDATE e DELETE**).

Conexões

Representam o número de conexões abertas em execução simultânea, que podem sobrecarregar o servidor e diminuir o desempenho.

Uptime

É o tempo de inicialização e desligamento do banco de dados em uma instância MySQL.

Threads

Representam o número de clientes conectados simultaneamente.

Tempo de resposta

Consiste no tempo médio de resposta por consulta ou para todas as consultas no servidor de banco de dados ao qual você está conectado.

Latência

Corresponde à duração das consultas ou das operações.

Erros

Consistem no número de erros de código que ocorrem com mais frequência e causam falhas.

Consultas

Representam o número de instruções executadas com mais frequência pelo servidor, incluindo procedimentos armazenados.

Perguntas

Equivalem ao número de declarações enviadas pelos clientes.

Uso do *buffer pool*

Significa o uso de memória no servidor. Como regra, é utilizado para investigar problemas de desempenho.

Coletando e monitorando as métricas de desempenho do MySQL

Agora que você conheceu quais tipos de métricas de desempenho monitorar em seu banco de dados MySQL, saiba que as métricas e estatísticas de desempenho do banco de dados MySQL podem ser divididas em três modos: variáveis de *status* do servidor, esquema de desempenho e esquema sys. É importante observar que, considerando a localização do terminal de acesso do cliente (que está consultando o banco de dados) e até do servidor (no qual está instalado o serviço de banco de dados), as informações métricas e de desempenho coletadas podem alterar ou sofrer algumas modificações em função da localização dos dispositivos.

Confira agora várias maneiras de coletar os dados de que se precisa e também algumas ferramentas que auxiliam nessa tarefa:

Variáveis de *status* do servidor

O MySQL tem a capacidade de obter várias informações sobre o funcionamento do SGBD (Sistema de Gerenciamento de Banco de Dados), por isso ele mantém o controle de “contadores” chamados de variáveis de *status* do servidor. As variáveis de *status* do servidor fornecem informações sobre as operações realizadas no MySQL, as quais incluem, por exemplo, o número de inclusões, alterações, exclusões e consultas realizadas, entre outros. O número total de variáveis de *status* do servidor varia conforme a versão do MySQL Server que está sendo utilizada.

Essas variáveis podem ser acessadas com o seguinte comando:

```
SHOW [GLOBAL | SESSION] STATUS
```

Com **GLOBAL**, a instrução retorna valores agregados em todas as conexões, enquanto **SESSION** limita os valores apenas à conexão atual.

Você pode executar os comandos dos exemplos deste texto utilizando o MySQL Workbench. Experimente realizar várias conexões simultâneas e consultas em paralelo para verificar diferenças nos valores obtidos.

Para isso, considere o comando a seguir:

```
SHOW GLOBAL STATUS
```

variable_name	value
Aborted_connects	0
Com_alter_table	0
Com_clone	0
Com_commit	0
Com_create_db	1
Com_create_db	1
Com_create_function	0
Com_create_procedure	0
Com_create_table	34
Com_create_trigger	0
Com_create_view	0
Com_delete	0
Com_drop_table	0
Com_drop_trigger	0
Com_insert	30
Com_insert_select	0
Com_select	32
Com_update	20
Queries	140
Questions	86
Slow_queries	0
Threads_cached	0
Threads_connected	2

Threads_created	2
Threads_running	2

Você também pode visualizar uma única variável de *status* do servidor.

Observe o comando:

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

Variable_name	Value
Com_select	32

Essas instruções também oferecem suporte à correspondência de padrões para consultar uma família de métricas relacionadas simultaneamente. Para verificar variáveis de erros de conexão, por exemplo, há o seguinte comando:

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

variable_name	value
Connection_erros_accept	0
Connection_erros_internal	0
Connection_erros_max_connection	15
Connection_erros_peer_address	0
Connection_erros_select	0
Connection_erros_tcpwrap	0

Com inúmeras variáveis disponíveis para monitorar seu banco de dados MySQL, em quais estatísticas você deve ter mais atenção? Embora possa variar de um caso de uso para outro, observe a seguir uma lista de algumas das métricas críticas a serem acompanhadas em cada uma das três principais categorias de desempenho do banco de dados:

Clique ou toque para visualizar o conteúdo.

- ◆ **Questions** – É um contador interno incrementado para todas as instruções enviadas pelos aplicativos clientes.
- ◆ **Queries** – Determina o número de instruções executadas pelo servidor (inclui instruções enviadas pelo cliente e instruções executadas em procedimentos armazenados).
- ◆ **Com_select** – Conta quantas instruções **SELECT** foram executadas e indica o nível de atividade somente leitura.
- ◆ **Com_insert, Com_update e Com_delete** – Indicam o nível de operações de gravação, geralmente resumidos em uma única variável
- ◆ **Slow_Queries** – Mensuram o número de consultas que excedem um limite de tempo predefinido no parâmetro **long_query_time**. Isso simplifica muito a tarefa de encontrar consultas ineficientes ou demoradas. Para encontrar o valor definido no parâmetro citado, consulte a seção “Monitorar o desempenho das consultas no MySQL”.
- ◆ **Aborted_connects** – Conta quantas tentativas de conectar ao servidor MySQL falharam.
- ◆ **Threads_connected** – Determina o número de conexões atualmente abertas.
- ◆ **Threads_running** – Conta quantos threads não estão “dormindo”.

Esquema de desempenho do MySQL

O esquema de desempenho do MySQL armazena estatísticas sobre eventos do servidor e execução de consultas em um nível baixo. A partir do MySQL versão 5.6.6, ele é habilitado por padrão.

O MySQL exibe o esquema de desempenho como o banco de dados chamado “performance_schema”, contendo tabelas que você pode consultar utilizando instruções SQL. Como resultado, você obtém informações sobre o desempenho que podem incluir eventos atuais, históricos e resumos de eventos, instâncias de objetos, dados de configuração, duração dos eventos do servidor, monitoramento de *status*, entre outros.

Para visualizar todas as tabelas disponíveis no banco de dados “performance_schema”, execute a seguinte instrução:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES  
WHERE TABLE_SCHEMA = 'performance_schema'
```

TABLE_NAME
accounts
cond_instances
data_lock_waits
data_locks
events_errors_summary_by_account_by_error
events_errors_summary_by_host_by_error
events_errors_summary_by_thread_by_error
events_statements_summary_by_digest
table_handles
table_io_waits_summary_by_index_usage
table_io_waits_summary_by_table
table_lock_waits_summary_by_table
threads
user_defined_functions
user_variables_by_thread
users
variables_by_thread
variables_info

As principais estatísticas de desempenho podem ser encontradas consultando várias tabelas no banco de dados “performance_schema”, especificamente a tabela “events_statements_summary_by_digest”, responsável por capturar informações sobre latência, erros e volume de consulta.

A instrução a seguir retorna a consulta com o maior tempo de execução:

```
SELECT digest_text, avg_timer_wait  
FROM performance_schema.events_statements_summary_by_digest  
ORDER BY avg_timer_wait DESC  
LIMIT 1;
```

digest_text	avg_timer_wait
SELECT * FROM Produtos A INNER JOIN Fornecedores ON (A.	154868700000

Esquema sys no MySQL

O esquema sys do MySQL inclui procedimentos armazenados, exibições e funções armazenadas para interpretar os dados coletados pelo esquema de desempenho e verificar como uma instância do MySQL está sendo executada no momento. Por padrão, esse esquema está habilitado no MySQL v5.7.7 e posterior.

O esquema sys fornece informações sobre o uso do banco de dados, incluindo conexões atuais, consultas que estão sendo executadas, tamanho do *buffer* e bloqueios, e resume a atividade da instrução, latência de E/S, uso de memória pelo *host* e usuários, estatísticas de espera, recursos consumidos pelos usuários etc.

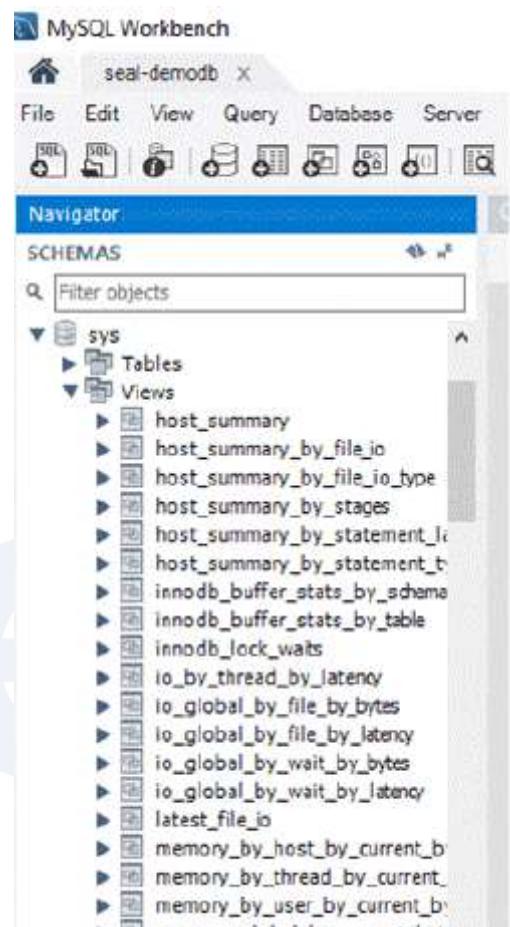


Figura 1 – Esquema sys no MySQL

A seguir, confira um exemplo em ação com a view **sys.schema_tables_with_full_table_scans**, usada para realizar uma verificação completa de tabelas, o que é um problema nos banco de dados, pois consome muitos recursos e reduz a velocidade. Essa consulta é útil para se identificar o número de linhas escaneadas assim como a latência.

Execute o comando a seguir:

```
SELECT * FROM sys.schema_tables_with_full_table_scans
```

Object_schema	Object_name	Rows_full_scanned	latency
Super	produtos	4062	37.95 ms

Monitorar o desempenho das consultas no MySQL

Para analisar o desempenho da consulta e detectar consultas de longa duração e que consomem recursos, os usuários podem monitorar diferentes métricas, incluindo consumo de CPU e memória, tempo de execução, atividade do disco, estatísticas de espera, ciclos de E/S etc.

No MySQL, você pode identificar e investigar *logs* de consultas lentas. Para isso, certifique-se de que os *logs* estejam habilitados, executando o seguinte comando:

```
SHOW GLOBAL VARIABLES LIKE 'slow%log%'
```

Variable_name	Value
slow_query_log	ON
slow_query_log_file	DESKTOP-C39PM6V-slow.log

Como pode ser visto, a variável **slow_query_log** está **ON**, ou seja, habilitada, e o nome do arquivo do *log* é “DESKTOP-C39PM6V-slow.log”. Entretanto, caso no seu ambiente esteja **OFF**, é necessário, para habilitar, esse recurso, usar a seguinte instrução:

```
SET GLOBAL slow_query_log = 'ON';
```

Por padrão, quando o *log* de consulta lenta está ativado, ele registra qualquer consulta que demore mais de dez segundos para ser executada. Para consultar o tempo no seu ambiente, pesquise a variável **long_query_time**.

```
SHOW GLOBAL VARIABLES LIKE 'long%time'
```

Variable_name	Value
Long_query_time	10.0000000

Todas essas variáveis podem ser utilizadas para pesquisar consultas que levam muito tempo para serem executadas e devem ser analisadas para otimizar o desempenho da consulta.

Entretanto, quando terminar a solução de encontrar problemas de consultas lentas no arquivo de *log*, é importante desativá-la, para evitar a gravação de informações desnecessárias. Para fazer isso, execute o seguinte comando:

```
SET GLOBAL slow_query_log = 'OFF';
```

Identificar e encerrar consultas com comandos no MySQL

Outro recurso que é largamente utilizado para detectar consultas lentas do MySQL é executar o seguinte comando:

```
SHOW PROCESSLIST
```

Id	User	Host	db	Command	Time	State	Info
168	root	localhost:57915	super	Query	158		SELECT * FROM Produto
169	root	localhost:57916	super	Query	0	Init	SHOW PROCESSLIST

Esse comando permite que você visualize consultas ativas e verifique o *status* de uma consulta. Isso se torna muito útil quando uma consulta foi executada há algum tempo e está presa, por lentidão. Assim, é possível encerrar a execução e, para isso, no exemplo a seguir, substitua <thread_id> pela Id da consulta que você deseja encerrar:

```
kill <thread_id>;
```

Os *logs* de consulta do MySQL são métricas úteis de monitoramento de consulta. No entanto, eles não fornecem uma imagem completa do desempenho da consulta. Nesse caso, as ferramentas de desempenho de consulta de terceiros do MySQL podem ajudar.

Ferramentas de monitoramento de desempenho do MySQL

O mercado de ferramentas de monitoramento de desempenho MySQL oferece muitas ferramentas para analisar e otimizar o desempenho de consultas, como, por exemplo, MySQL Enterprise Monitor e MySQL Workbench.

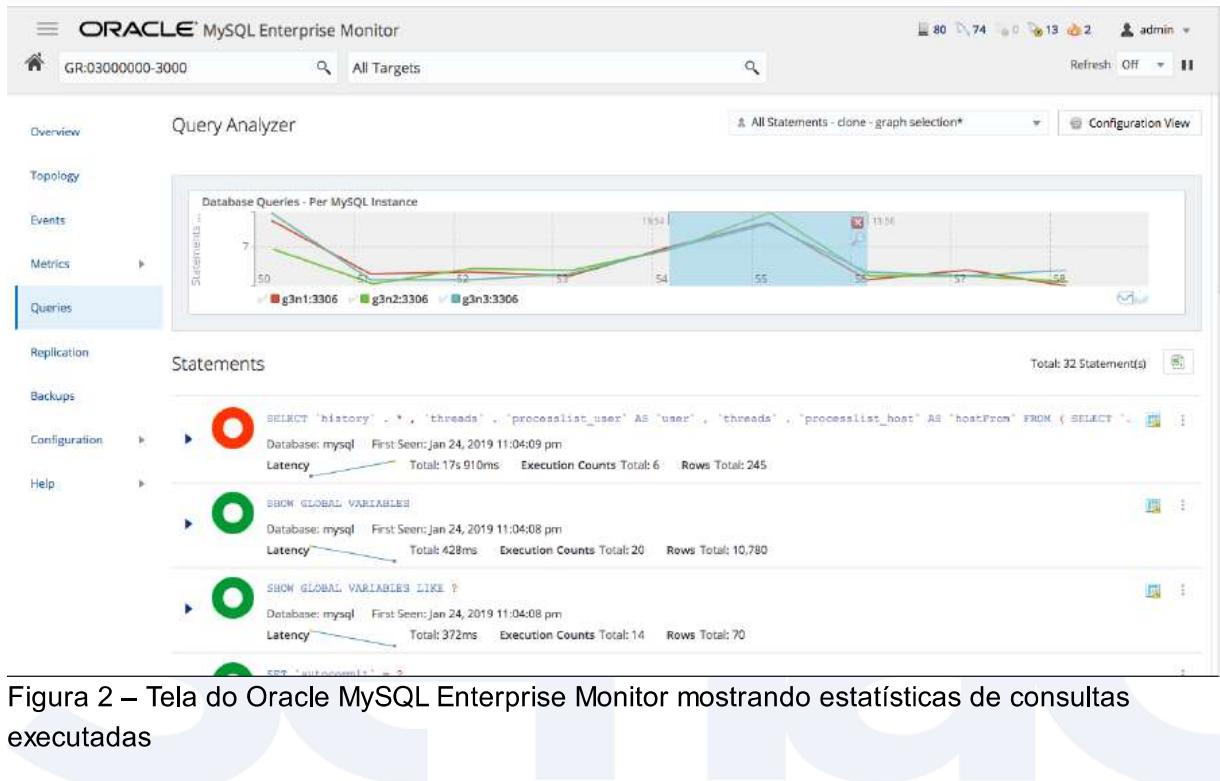


Figura 2 – Tela do Oracle MySQL Enterprise Monitor mostrando estatísticas de consultas executadas

Aprenda mais agora sobre essas ferramentas de análise e otimização do desempenho de consultas:

MySQL Enterprise Monitor

É uma ferramenta de monitoramento projetada para acompanhar instâncias e hosts MySQL em tempo real, alertar os usuários sobre possíveis problemas e notificá-los como eles podem ser resolvidos. Com a ferramenta, você pode otimizar o desempenho, ficar atento a lançamentos e correções de bugs, gerenciar e evitar problemas ou tempo de inatividade. O painel Enterprise disponível no MySQL Enterprise Monitor ajuda você a verificar estatísticas de execução, pesquisar as consultas mais caras, observar as métricas de ajuste de desempenho do InnoDB, identificar vulnerabilidades de segurança, analisar consultas visualmente e visualizar as métricas dos servidores MySQL que você está executando. Ainda assim, deve-se notar que o MySQL Enterprise Monitor depende do esquema de desempenho do MySQL. Desta forma, para um trabalho correto, ele deve ser ativado.

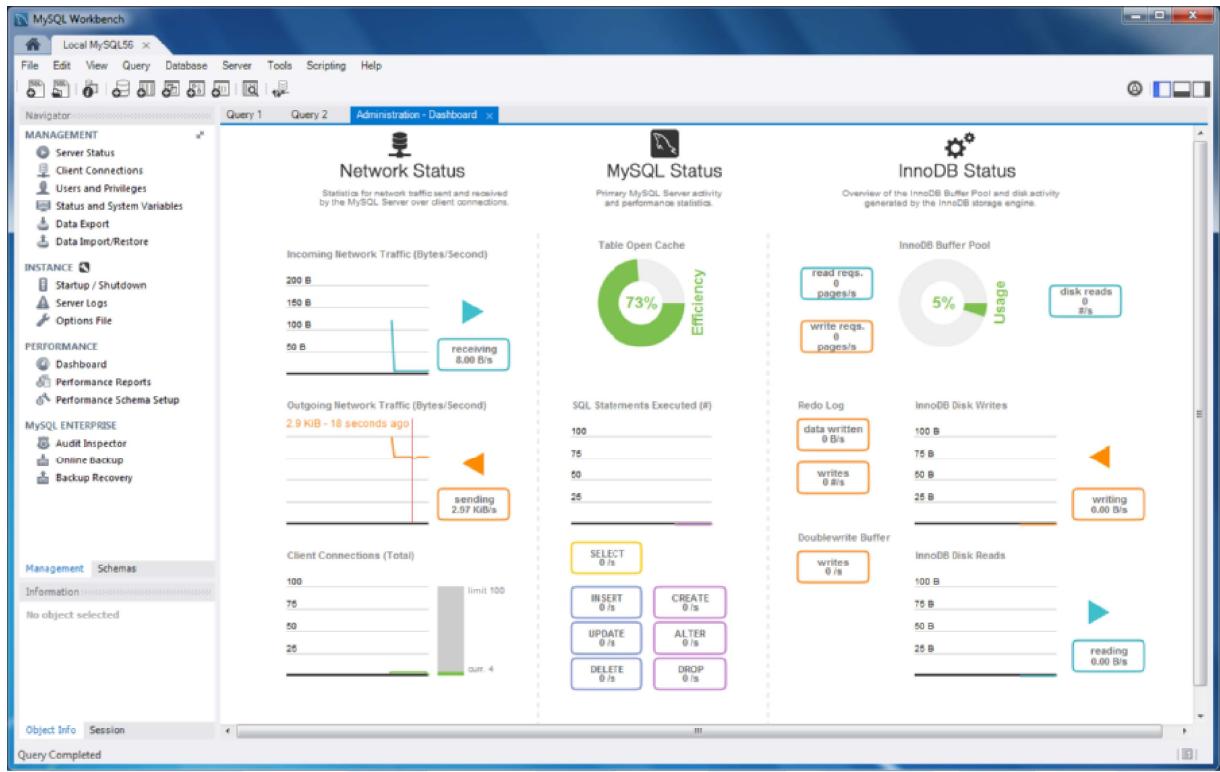


Figura 3 – Tela Dashboard do MySQL Workbench

MySQL Workbench

É uma ferramenta de modelagem visual para bancos de dados MySQL que, além de muitos recursos úteis e avançados, fornece um conjunto de ferramentas para visualizar e melhorar o desempenho do banco de dados. O painel de desempenho pode visualizar as principais métricas de desempenho, como tráfego de rede de entrada e saída, estatísticas de desempenho, instruções SQL executadas, *status* do InnoDB, incluindo atividade de disco, gravações e leituras. Além disso, com relatórios de desempenho, torna-se muito mais fácil analisar o desempenho do banco de dados MySQL. Para ajustar o desempenho da instrução SQL, verifique o plano de explicação.

Experimente acessar no MySQL Workbench na região lateral a área **Navigator**, a aba **Administration**, e clicar no item **Dashboard** para acessar as informações de estatística de desempenho. Teste realizando consultas diversas, conexões paralelas e outras operações.

Tuning e otimização de consultas

À medida que os volumes de dados crescem e a tecnologia fica cada vez mais complexa, torna-se mais importante otimizar os bancos de dados MySQL adequadamente para oferecer experiência ao usuário final e reduzir os custos de infraestrutura. Monitorar o desempenho do MySQL pode ajudar os profissionais de banco de dados a identificarem rapidamente gargalos, a direcionarem operações insuficientes por meio de uma revisão dos planos de execução de consultas e a eliminarem quaisquer jogos de adivinhação.

Com a complexidade adicional de volumes de dados crescentes e as cargas de trabalho em constante mudança, o ajuste de desempenho do banco de dados e a otimização de consultas MySQL são agora necessários para maximizar a utilização de recursos e o desempenho do sistema.

Benefícios do ajuste de desempenho do MySQL

Uma vez ajustado corretamente, o banco de dados oferece resultados de desempenho que valem a pena com ótimas funcionalidades. Ele não apenas reduz a carga de tarefas indesejadas, mas também optimiza o banco de dados MySQL para uma recuperação de dados mais rápida.

Neste sentido, são apresentadas a seguir sete boas práticas para otimizar a velocidade dos dados em consultas no seu banco de dados:

Clique ou toque para visualizar o conteúdo.

Indexação

Em primeiro lugar, garanta a indexação de todos os predicados nas cláusulas **WHERE**, **JOIN**, **ORDER BY** e **GROUP BY**, pois a indexação imprópria de consultas SQL pode causar varreduras de tabela, que eventualmente resultam em problemas de bloqueio, entre outros. Portanto, recomenda-se indexar todas as colunas de predicho para que o banco de dados possa experimentar a otimização de consulta do MySQL.

Evite utilizar funções em predicados

O banco de dados não usa um índice se houver alguma função predefinida na coluna, por exemplo:

```
SELECT * FROM TABLE1 WHERE UPPER(COL1) = 'ABC'
```

Por causa da função **UPPER()**, o banco de dados não utiliza o índice em **COL1**. Se não houver nenhuma maneira de evitar essa função no SQL, você terá que criar um novo índice baseado em função ou gerar colunas personalizadas no banco de dados para melhorar o desempenho.

Evite usar um curinga (%) no início de um predicado

O predicado **LIKE '%abc'** causa uma varredura completa da tabela, como, por exemplo:

```
SELECT * FROM TABLE1 WHERE COL1 LIKE '%ABC'
```

Na maioria dos casos, esse uso de curinga traz grandes limitações de desempenho.

Evite colunas desnecessárias na cláusula SELECT

Em vez de usar **SELECT ***, sempre especifique colunas na cláusula **SELECT** para melhorar o desempenho do MySQL, pois colunas desnecessárias causam carga adicional no banco de dados, diminuindo seu desempenho, bem como o de todo o processo sistemático.

Utilize INNER JOIN

Utilize a junção externa somente quando for necessário. Usá-la desnecessariamente não apenas limita o desempenho do banco de dados, mas também limita as opções de otimização de consulta MySQL, resultando em uma execução mais lenta de instruções SQL.

Utilize DISTINCT e UNION

Utilize-os somente se for necessário. O uso de operadores **UNION** e **DISTINCT** sem nenhum objetivo principal causa classificação indesejada e lentidão na execução do SQL. Em vez de **UNION**, usar **UNION ALL** traz mais eficiência no processo e melhora o desempenho do MySQL com mais precisão.

Cláusula ORDER BY

Essa cláusula é obrigatória no SQL se você espera obter um resultado classificado.

A expressão-chave **ORDER BY** classifica o conjunto de resultados em colunas de instrução predefinidas. Embora a instrução traga vantagem para os administradores de banco de dados para obter os dados classificados, ela também produz um pouco de impacto no desempenho na execução do SQL. Isso ocorre porque a consulta primeiramente precisa classificar os dados para depois produzir o conjunto de resultados final, causando uma operação um pouco complexa na execução do SQL.

Cenários práticos para otimização de consultas

Para que se possa aplicar as boas práticas de otimização de consultas apresentadas anteriormente, será utilizado para a demonstração o banco de dados “Sakila”, que é um banco para estudo/treinamento disponível no site oficial do MySQL, na seção **Example Databases**. Para praticar os cenários a seguir, tenha o banco de dados “Sakila” configurado em seu computador.

The screenshot shows the MySQL Documentation page. At the top, there's a search bar and navigation links for MySQL.COM, DOWNLOADS, DOCUMENTATION (which is highlighted in orange), and DEVELOPER ZONE. Below the header, there's a secondary navigation bar with links for MySQL Server, MySQL Enterprise, Workbench, InnoDB Cluster, MySQL NDB Cluster, Connectors, and More. On the left, there's a sidebar with a search bar, a "Search Current Docs" button, and links for Archives and About. The main content area is titled "Other MySQL Documentation". It includes a note about additional documentation and links to Archives and About. Below this, it lists "MySQL Server Doxygen Documentation" with a "Title" column and "HTML Online" links. Under "Expert Guides", there's a table with columns for Language, Title, Version, HTML, Online, and PDF. The "Sakila database" row in the "Example Databases" section is highlighted with a red box. This row has a "Title" of "sakila database", a "DB Download" link to GitHub, and "HTML Setup Guide" and "PDF Setup Guide" links for US Ltr | A4.

Figura 4 – Página da documentação do MySQL, em que o *script* do Sakila Database pode ser baixado

Depois de baixar o arquivo compactado (TGZ ou ZIP), descompacte-o e, no MySQL Workbench, execute primeiro o arquivo “sakila-schema.sql” e depois “sakila-data.sql”. Caso você queira analisar as estruturas desse banco de dados e seus relacionamentos etc., abra o arquivo “sakila.mwb” que traz os diagramas ER do banco. São várias tabelas, mas concentre-se em algumas especificamente. O mais importante é você ter uma base de dados composta de informações que lhe permitam analisar questões de desempenho.

Uma dessas situações é quando uma consulta retorna mais dados do que precisa, gerando demanda de trabalho extra para o servidor MySQL. Isso adiciona sobrecarga de rede e consome memória e recursos de CPU no servidor de aplicação.

A seguir, confira alguns erros típicos que precisam ser evitados:

Buscar mais registros do que o necessário

Um erro comum é assumir que o MySQL fornece resultados sob demanda em vez de calcular e retornar o conjunto de resultados completo. Muitas vezes, isso é visto em aplicativos projetados por pessoas familiarizadas com outros sistemas de banco de dados, habituados a técnicas como emitir uma instrução **SELECT** que retorna muitas linhas e, em seguida, buscar a primeira linha e fechar o conjunto de resultados.

Um exemplo disso é um profissional buscar os 100 títulos mais recentes para um site de filmes quando só precisam ser mostrados dez deles na primeira página. Esse profissional acredita que o MySQL fornecerá a ele essas dez linhas e interromperá a execução da consulta, mas o que o MySQL realmente faz é gerar o conjunto de resultados completo. A melhor solução seria adicionar uma cláusula **LIMIT** à consulta, como no exemplo a seguir:

```
SELECT film_id, title, description, release_year  
FROM sakila.film LIMIT 0,10
```

film_id	title	description	release_year
1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies	2006
2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China	2006
3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory	2006
4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank	2006
5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf of Mexico	2006
6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China	2006

7	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who must Discover a Butler in A Jet Boat	2006
8	AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Confront a Monkey in Ancient India	2006
9	ALABAMA DEVIL	A Thoughtful Panorama of a Database Administrator And a Mad Scientist who must Outgun a Mad Scientist in A Jet Boat	2006
10	ALADDIN CALENDAR	A Action-Packed Tale of a Man And a Lumberjack who must Reach a Feminist in Ancient China	2006

A sintaxe de **LIMIT** é a seguinte:

LIMIT [inicio,] quantidade_de_linhas

Ou seja, pode-se ou não informar um registro inicial a retornar (por exemplo, do 5º registro em diante), indicando um numeral no primeiro parâmetro e, obrigatoriamente, um número de registros que se quer retornar.

Buscar todas as colunas de uma junção de várias tabelas

Sempre desconfie quando vir **SELECT ***. Você realmente precisa de todas as colunas? Provavelmente não. Recuperar todas as colunas pode impedir otimizações (como cobrir índices), bem como adicionar E/S, memória e sobrecarga de CPU para o

servidor.

Alguns administradores de banco de dados banem universalmente a cláusula **SELECT *** por causa deste fato e para reduzir o risco de problemas quando alguém altera a lista de colunas da tabela.

Observe o exemplo a seguir.

Se você deseja recuperar todos os atores que aparecem em “Academy Dinosour”, não escreva desta forma:

```
SELECT * FROM sakila.actor  
INNER JOIN sakila.film_actor USING(actor_id)  
INNER JOIN sakila.film USING(film_id)  
WHERE sakila.film.title = 'Academy Dinosaur';
```

Isso retorna todas as colunas de todas as três tabelas. Em vez disso, escreva a consulta da seguinte forma:

```
SELECT sakila.actor.* FROM sakila.actor  
INNER JOIN sakila.film_actor USING(actor_id)  
INNER JOIN sakila.film USING(film_id)  
WHERE sakila.film.title = 'Academy Dinosaur';
```

actor_id	first_name	last_name	last_update
1	PENELOPE	GUINNESS	15/02/2006 04:34
10	CHRISTIAN	GABLE	15/02/2006 04:34
20	LUCILLE	TRACY	15/02/2006 04:34
30	SANDRA	PECK	15/02/2006 04:34
40	JOHNNY	CAGE	15/02/2006 04:34
53	MENA	TEMPLE	15/02/2006 04:34
108	WARREN	NOLTE	15/02/2006 04:34
162	OPRAH	KILMER	15/02/2006 04:34
188	ROCK	DUKAKIS	15/02/2006 04:34
198	MARY	KEITEL	15/02/2006 04:34

Como se percebe pelo exemplo, uma maneira prática de trazer todas as colunas de apenas uma tabela é usar **[nome_tabela].***. Depois disso, incluem-se outras colunas interessantes à consulta (por exemplo: **SELECT sakila.actor.***, **sakila.film.title**). Obviamente, ao invés disso, seria possível listar uma por uma as colunas desejadas (**SELECT actor_id, first_name etc.**).

Evidentemente, pedir mais dados do que você realmente precisa nem sempre é ruim. Em muitos casos investigados, as pessoas relatam que a abordagem de desperdício simplifica o desenvolvimento, pois permite que o desenvolvedor use o mesmo código em mais de um lugar. Essa é uma consideração razoável, desde que você saiba quanto custa em termos de desempenho. Também pode ser útil recuperar mais dados do que você realmente precisa se você utilizar algum tipo de *cache* em seu aplicativo ou se tiver considerando outro benefício. Buscar e armazenar em *cache* objetos completos pode ser preferível a executar muitas consultas separadas que recuperam apenas partes do objeto.

Otimizar consultas com uso de índices

Se você não estiver obtendo um bom resultado na consulta, uma boa alternativa para tentar resolver o problema é adicionar um índice apropriado. A indexação foi abordada detalhadamente no conteúdo **Índices (index)** desta unidade. Saiba que os índices são muito importantes para a otimização de consultas, pois permitem que o MySQL encontre linhas com um tipo de acesso mais eficiente, que examina menos dados.

Observe, como exemplo, uma consulta simples no banco de dados “Sakila”:

```
SELECT * FROM sakila.film_actor WHERE film_id = 1;
```

actor_id	film_id	last_update
1	1	15/02/2006 05:05
10	1	15/02/2006 05:05
20	1	15/02/2006 05:05
30	1	15/02/2006 05:05
40	1	15/02/2006 05:05
53	1	15/02/2006 05:05
108	1	15/02/2006 05:05
162	1	15/02/2006 05:05
188	1	15/02/2006 05:05
198	1	15/02/2006 05:05

Essa consulta retornará dez linhas e, quando você utilizar o comando **EXPLAIN**, será mostrado o tipo de acesso **ref** no índice **idx_fk_film_id** para executar a consulta:

```
EXPLAIN SELECT * FROM sakila.film_actor WHERE film_id = 1
```

id	select_type	table	type	possible_keys	key	key_len	ref
1	SIMPLE	film_actor	ref	idx_fk_film_id	idx_fk_film_id	2	c

EXPLAIN mostra que o MySQL estimou que precisava acessar apenas dez linhas. Em outras palavras, o otimizador de consulta sabia que o tipo de acesso escolhido poderia satisfazer a consulta com eficiência. O que aconteceria se não houvesse um índice adequado para a consulta? O MySQL teria que utilizar um tipo de acesso menos ideal, como se pode notar se for descartado o índice e executada a consulta novamente:

```
ALTER TABLE sakila.film_actor DROP KEY idx_fk_film_id;  
EXPLAIN SELECT * FROM sakila.film_actor WHERE film_id = 1
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows
1	SIMPLE	film_actor	ALL	NULL	NULL	NULL	NULL	5073

Como visto, o MySQL agora estima que terá que examinar 5.073 linhas para satisfazer a consulta. O “Using Where” mostra que o servidor MySQL está usando a cláusula **WHERE** para descartar linhas depois que o mecanismo de armazenamento as lê.

Esse exemplo ilustra como é importante ter bons índices, pois eles ajudam as consultas a obterem um bom tipo de acesso e examina apenas as linhas que precisam.

Particionamento

O particionamento é uma técnica que melhora o desempenho e a capacidade de gerenciamento, simplifica a manutenção e reduz o custo de armazenamento de grandes quantidades de dados. Essa técnica também é conhecida por alguns autores como “fragmentação”.

No MySQL, a partir da versão 5.7, o particionamento tornou-se nativo do mecanismo de armazenamento e preteriu o método antigo, em que o próprio MySQL tinha que lidar com as partições. Isso significa que as partições InnoDB (e uma quantidade maior de partições) são uma escolha melhor do que no passado.

O particionamento pode ser obtido sem dividir tabelas, porém colocando-as fisicamente em unidades de disco individuais. Outra possibilidade é que os dados podem ser divididos em servidores geograficamente separados, diminuindo custos e aumentando a *performance* de acesso aos dados.

O particionamento permite que tabelas, índices e tabelas organizadas por índice sejam subdivididos em partes menores, portanto, as consultas que acessam apenas uma fração dos dados podem ser executadas mais rapidamente porque há menos dados para verificar.

Tipos de particionamento

Existem duas formas principais de particionamento:

Clique ou toque para visualizar o conteúdo.

Particionamento horizontal

Divide as linhas da tabela em várias partições (com base em uma lógica). Todas as colunas definidas para uma tabela são encontradas em cada partição, portanto, nenhum atributo de tabela real está ausente. Toda a partição pode ser endereçada individualmente ou coletivamente.

Como exemplo, considere uma tabela que contém transações de venda de um ano inteiro sendo particionada horizontalmente em doze partições distintas, em que cada partição contém os dados de um mês.

idVenda	cliente	produto	quantidade	valor	dataVenda
1000001	David Alexander	Refrigerante	2	5.50	15/01/2022
1000225	Jarred Carlson	Arroz 5KG	3	8.50	18/01/2022
1025555	Sue Charles	Feijão	5	6.50	02/02/2022
1055555	Simon Mitchel	Detergente	2	2.10	15/02/2022
1022552	Richard Zeng	Leite	6	2.70	18/06/2022
1000254	Evandro Silva	Bolacha	4	1.50	20/06/2022
1000256	David Milar	Refrigerante	2	5.50	01/12/2022
1000257	Jarred Charles	Feijão	6	6.50	03/12/2022

The diagram illustrates the horizontal partitioning of the table. Four vertical lines divide the table into five segments. The first segment (from the left) contains the first two rows (idVendas 1000001 and 1000225). The second segment contains the next two rows (1025555 and 1055555). The third segment contains the next two rows (1022552 and 1000254). The fourth segment contains the last two rows (1000256 and 1000257). Below the table, four server icons are shown, each associated with a partition label: 'A' under the first segment, 'C' under the second, 'M' under the third, and 'Z' under the fourth. This visualizes how the logical rows are physically distributed across different servers.

Figura 5 – Particionamento horizontal é sobre divisão de linhas

Particionamento vertical

Divide uma tabela em várias tabelas que contêm menos colunas. Assim como no particionamento horizontal, no particionamento vertical, uma consulta verifica menos dados, o que aumenta o desempenho da consulta.

Considere, como exemplo, uma tabela que contém várias colunas de texto, ou BLOB (*binary large object*), muito amplas e que não são endereçadas. Muitas vezes, essa tabela é dividida em duas tabelas, que têm as colunas mais referenciadas em uma tabela e os dados de texto, ou BLOB, em outra.



Figura 6 – Particionamento vertical é sobre dividir colunas

Vantagens e desvantagens

O particionamento permite que você tenha mais controle sobre como os dados são gerenciados dentro do banco de dados. Por exemplo, você pode descartar partições específicas em uma tabela particionada em que os dados perdem sua utilidade. O processo de adição de novos dados, em alguns casos, pode ser bastante facilitado pela adição de uma ou mais novas partições para armazenamento desses dados, utilizando o comando **ALTER TABLE**.

No particionamento, é possível armazenar mais dados em uma tabela do que podem ser mantidos em um único disco ou partição do sistema de arquivos. A seleção de partição também suporta as instruções de modificação de dados **DELETE**, **INSERT**, **REPLACE**, **UPDATE**, **LOAD DATA** e **LOAD XML**.

No entanto, o particionamento é desvantajoso quando não utilizado corretamente e em um ambiente adequado. As principais razões ocorrem na administração, quando uma tabela tiver partições em vários discos de um computador ou em vários sistemas em diferentes computadores.

Escalabilidade do banco de dados

À medida que o *hardware* continua a inovar em ritmo acelerado (dispositivos de armazenamento em massa, CPU e redes), também é aumentada a capacidade do *software* de banco de dados, permitindo o crescimento do número de transações por segundo. Porém, ainda há a necessidade de adicionar capacidade de tempos em tempos.

Existem duas abordagens para “escalar” um banco de dados, ambas com seus próprios prós e contras: vertical (*scale up*) e horizontal (*scale out*).

Vertical

Essa abordagem envolve adicionar mais recursos físicos ou virtuais ao servidor que hospeda o banco de dados – mais CPU, mais memória ou mais armazenamento. Essa é a abordagem tradicional e praticamente todos os bancos de dados podem ser ampliados.

Vantagens

- ◆ Do ponto de vista do desenvolvimento, não há necessidade de mudar nada. O mesmo código será conectado sem problemas ao banco de dados existente.
- ◆ Facilidade em implementar e administrar, pois há um único servidor para o gerenciamento. A configuração também é simples.
- ◆ Os custos do *data center* (centro de processamento de dados), relacionados a espaço, refrigeração e energia, são menores.

Desvantagens

- ◆ Embora os custos iniciais do *software* sejam baixos, se o licenciamento for baseado no número de núcleos, você terá que pagar mais toda vez que aumentar a escala.
- ◆ Os custos de *hardware* também podem ser altos, pois você precisa comprar servidores *high-end*.
- ◆ Há escopo limitado para atualizações. O servidor pode ser incrementado até certo ponto. Se a máquina, em sua otimização máxima, ainda assim não conseguir comportar o banco de dados, outras abordagens precisam ser aplicadas.

Horizontal

Essa abordagem envolve a adição de mais instâncias/nós do banco de dados para lidar com o aumento da carga de trabalho. Quando precisar de mais capacidade, basta adicionar mais servidores ao *cluster*. Além disso, o *hardware* utilizado tende a compor-se de servidores menores e mais baratos.

A maioria dos produtos de banco de dados não será dimensionada dessa maneira e, dependendo de como esse dimensionamento for implementado, os aplicativos precisarão ser reescritos para funcionar com o banco de dados. Existem duas técnicas diferentes que podem ser utilizadas neste caso:

Replicação de dados: para cargas de trabalho de leitura intensa, você pode ter uma cópia primária que aceita alterações de dados e várias réplicas somente leitura desses dados. A desvantagem disso é que, para gravações de dados, a cópia primária torna-se um gargalo.

Banco de dados federado: envolve a distribuição de leituras e gravações em vários nós. Isso é obtido particionando (fragmentando) os dados em vários servidores de banco de dados. Há um elemento de replicação de banco de dados em que alguns dados são mantidos em todos ou em vários nós.

Clique ou toque para visualizar o conteúdo.

Vantagens

- ◆ Custos muito mais baratos, pois os nós individuais são menores.
- ◆ As atualizações são mais fáceis, tudo o que você precisa fazer é adicionar um nó adicional. Em teoria, você pode adicionar um número infinito de nós.
- ◆ A resiliência e a tolerância a falhas geralmente são mais fáceis de alcançar e gerenciar devido aos vários nós.

Desvantagens

- ◆ O aumento maciço na complexidade leva a uma série de problemas, tais como:
 - ◆ • Maiores incidentes de *bugs de software*, pois o código é mais complexo.
 - ◆ • O gerenciamento da infraestrutura torna-se mais complexo com o aumento do número de nós.
- ◆ As taxas de licenciamento podem ser mais altas, pois você tem mais nós para licenciar.
- ◆ Os custos do *data center*, em termos de espaço, refrigeração e energia, são maiores.
- ◆ A complexidade e os custos da rede são maiores, pois a latência levará a dados inconsistentes.

Encerramento

Como estudado neste material, as métricas de desempenho ajudam na análise e no melhoramento do desempenho de consultas no MySQL. Se você estiver desenvolvendo seu monitoramento do MySQL, capturar as métricas descritas poderá ajudá-lo a entender os padrões de uso do banco de dados e as possíveis restrições. Essas métricas também o auxiliarão a identificar quando é necessário expandir ou mover suas instâncias de banco de dados para *hosts* mais poderosos, com o objetivo de manter um bom desempenho do seu banco de dados.

Neste material, você teve uma visão geral do MySQL Enterprise Monitor e do MySQL Workbench como ferramentas para monitoramento. Além disso, com relação à otimização de consultas, foram apresentadas sete boas práticas, bem como uma visão conceitual de particionamento e escalabilidade de banco de dados.