MySQL - Otimização e Desempemho

Prof. José Augusto Cintra

http://www.josecintra.com/blog

Junho/2020

Apresentação

O objetivo deste trabalho é apresentar alguns conceitos e técnicas de **otimização** com vistas a incrementar o **desempenho** de bancos de dados **MySQL**. No entanto, a maioria dos tópicos aqui discutidos valem para todos os SGBDs relacionais ou podem ser adaptados sem muitas dificuldades.

Não temos a pretensão de esgotar o assunto, pois ele extrapola as fronteiras do SGBD. Para um melhor aproveitamento dos conteúdo apresentado, espera-se que o leitor possua conhecimento prévio de **Modelagem de dados** e **SQL**.

Configurando Charset & Collation

Charset & Collation

Os campos tipo texto do MySQL (e outros SGBDs) possuem uma propriedade denominada de **CHARSET** que define o conjunto de caracteres válidos para serem armazenados. Esses conjuntos variam de acordo com o idioma escolhido.

Associado ao *charset*, existe o **COLLATION** que são conjuntos de regras sobre como efetuar comparações entre os caracteres do conjunto.

A configuração dessas propriedades pode ser feita na criação da tabela/BD ou definidas como padrão no arquivo de configuração do MySQL.

Dependendo da escolha desses conjuntos, a consulta aos campos texto podem ficar mais lentas ou não. E isso depende da versão do MySQL

Charset & Collation

Exemplo:

O MySQL 5 supera o MySQL 8 em performance usando o charset “latin1”. Já o MySQL 8 supera o MySQL 5 por uma ampla margem se usarmos o charset “utf8mb4”

Esteja ciente de que “utf8mb4” agora é o padrão MySQL 8, enquanto o MySQL 5 possui “latin1” por padrão.

Portanto, ao criar/configurar suas tabelas, verifique o conjunto de caracteres que você está usando, pois isso pode afetar a performance das suas consultas com campos texto.

Charset & Collation

Para exibir os conjuntos de caracteres disponível no servidor, execute o comando: **SHOW Charset**

Para exibir os COLLATIONs de um determinado Charset, use:
SHOW COLLATION

Exemplo: **SHOW COLLATION WHERE Charset = 'latin1'**

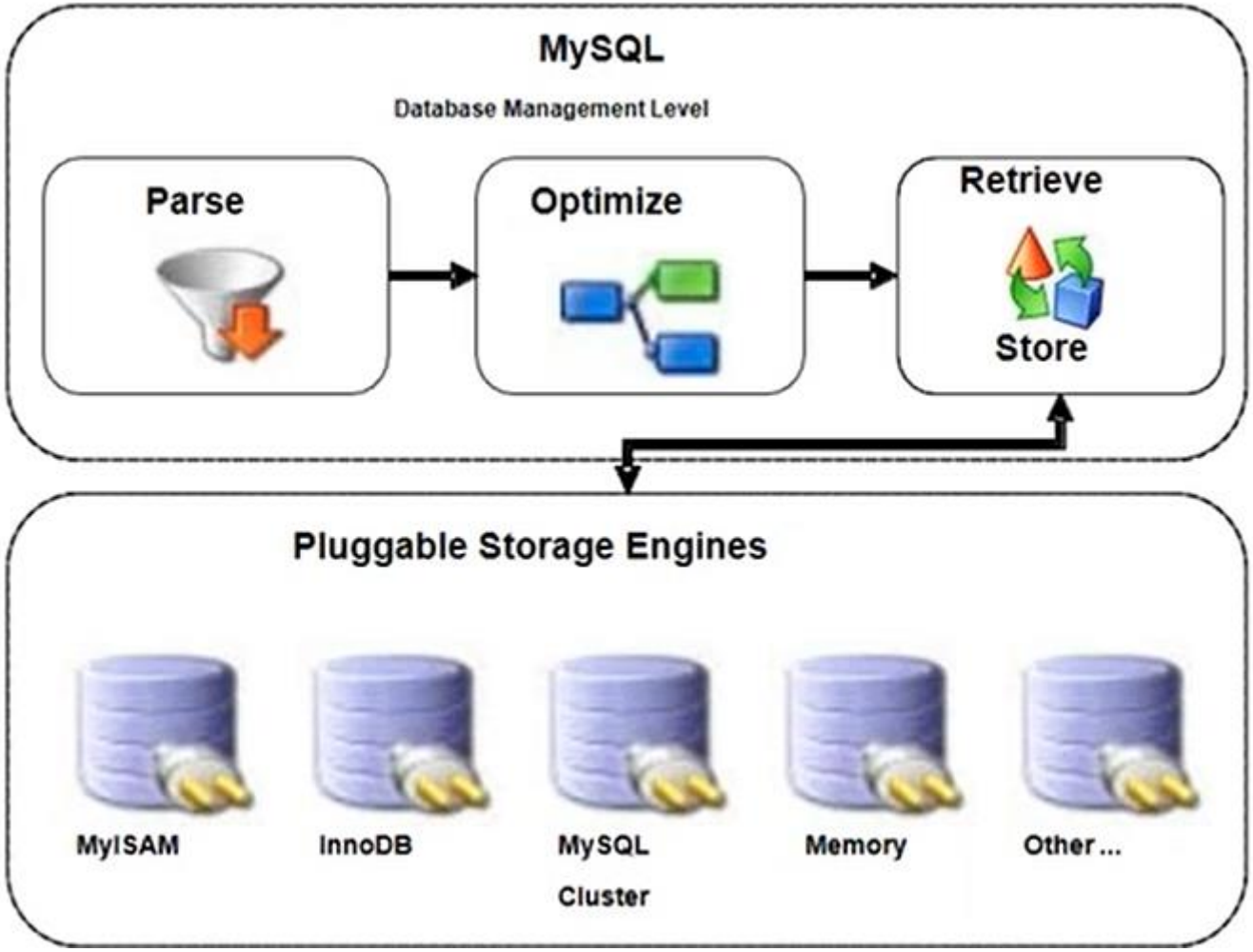
Para setar essas propriedades para todo o BD:

CREATE DATABASE database_name

CHARSET character_set_name

COLLATE collation_name;

Storage Engines



Storage Engines

Storage Engines ou **Engenharias de Armazenamento** são componentes de software plugáveis do MySQL que gerenciam a forma como os dados são tratados e armazenados tanto física como logicamente em operações CRUD.

Para facilitar o entendimento, podemos compará-las aos sistemas de arquivos dos sistemas operacionais (FAT, NTFS, etc)

O MySQL disponibiliza vários sistemas de armazenamento (engines), cada um com suas vantagens e desvantagens, sendo que podemos criar tabelas gerenciadas por engines diferentes dentro do mesmo banco de dados.

Storage Engines

Os recursos que o SGBD disponibiliza para as tarefas de armazenamento e consulta de dados, tais como Transações, Integridade Referencial, Concorrência e etc., dependem da Storage Engine adotada.

Portanto, se te perguntaram se o MySQL suporta transações, você vai dizer que sim, mas depende da Engenharia de armazenamento escolhida.

Quanto mais recursos de Bancos de Dados uma SE oferece, mais recursos do sistema ela consumirá e, provavelmente, menor desempenho apresentará.

A seguir, veremos as principais SEs do MySQL e suas características...

Storage Engines

Para relacionar a engines disponíveis no seu servidor, utilizamos os comando **SHOW ENGINES:**

Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	DEFAULT	Supports transactions, row-level locking, and fo...	YES	YES	YES
CSV	YES	CSV storage engine	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine (anything you write to ...	NO	NO	NO
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
MEMORY	YES	Hash based, stored in memory, useful for temp...	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO

Para especificar o engine na criação da tabela, usamos o comando:

CREATE TABLE t (i INT) ENGINE = X

Onde X é a *storage engine* desejada

Storage Engines - MyISAM

MyISAM foi uma das primeiras engines adotadas pelo MySQL, sendo que sua principal vantagem é o bom desempenho nas consultas. Pode ser usada em cargas batch e em tabelas temporárias.

Característica	Suporte
Integridade Referencial	Não
Transações	Não
Lock	Por Tabela
Indices B-tree	Sim
Índices Hash	Não

Storage Engines - InnoDB

InnoDB atualmente é a engine padrão do MySQL e sua principal vantagem é a alta confiabilidade e flexibilidade. É uma engine de uso geral, ideal para aplicações CRUD.

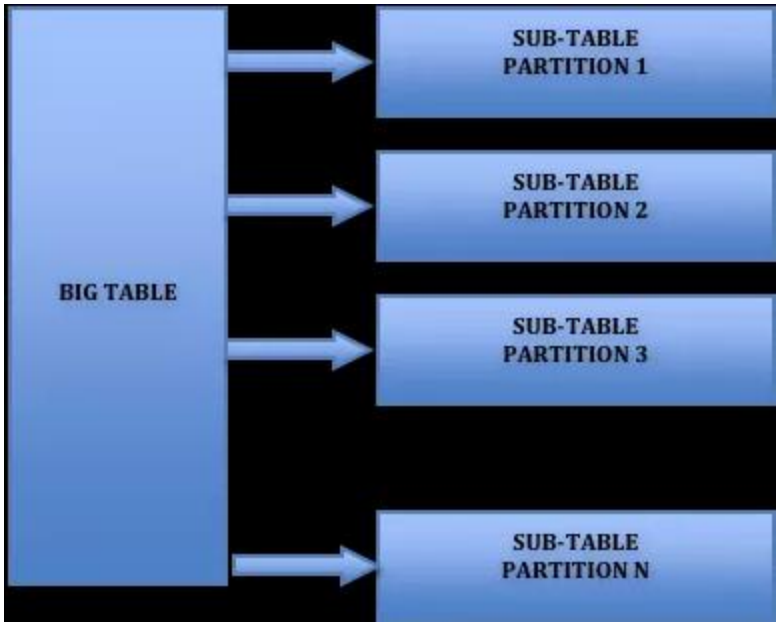
Característica	Suporte
Integridade Referencial	Sim
Transações	Sim
Mecanismo de Bloqueio (LOCK)	Por Registro
Indices B-tree	Sim
Índices Hash	Não

Storage Engines - Memory

A engine **Memory** armazena todos dados em memória e, por isso, oferece alta velocidade no acesso aos dados, sendo ideal para ser usada com tabelas temporárias e cache

Característica	Suporte
Integridade Referencial	Não
Transações	Não
Mecanismo de Bloqueio (LOCK)	Por Tabela
Indices B-tree	Sim
Índices Hash	Sim

Particionamento de tabelas



Particionamento de tabelas

Outro fator que afeta as consultas ao BD é o número de registros: Quanto maior o número de registros, mais lentas são as consultas.

Obviamente, o número de registros de uma tabela não pode ser reduzido, mas podemos dividi-los em conjuntos menores através do particionamento.
Dividindo uma tabela grande em partes menores (partições), as consultas a esses dados serão feitas nesses conjuntos menores de dados, o que agiliza o tempo de resposta.

Essa operação é feita de modo transparente, não afetando as querys, pois a consulta é feita na "tabela mãe" e o BD cuida de direcionar para as partições.

Particionamento de tabelas

Podemos definir os particionamentos de uma tabela já na sua criação. Para isso, é necessário estabelecer um **critério**, normalmente um campo, que servirá para definir os intervalos em que os registros serão particionados.

No MySQL, podemos particionar as tabelas através dos seguintes critérios:

RANGE → De acordo com um intervalo de valores de uma coluna.

LIST → Um intervalo de valores discretos, fixos.

HASH → Basta especificar o número de partições desejadas e o campo

KEY → É feito o particionamento automático de acordo com a chave primária ou chave única

Particionamento de tabelas

Exemplo 1: Supondo uma tabela de vendas com o campo "data da venda" (dt_venda), podemos particionar a tabela usando RANGE de acordo com trimestre:

```
PARTITION BY RANGE(MONTH(dt_venda)) (
    PARTITION primeiro_trimestre VALUES LESS THAN (4),
    PARTITION segundo_trimestre VALUES LESS THAN (7),
    PARTITION terceiro_trimestre VALUES LESS THAN (10),
    PARTITION quarto_trimestre VALUES LESS THAN MAXVALUE
)
```

Particionamento de tabelas

Exemplo 2: Supondo uma tabela de munícipios com a chave primária id_cidade, podemos particioná-la por região usando LIST:

```
PARTITION BY LIST(id_cidade) (
    PARTITION regiao1 VALUES IN (1, 2, 7),
    PARTITION regiao2 VALUES IN (3, 9),
    PARTITION regiao3 VALUES IN (4, 5, 6),
    PARTITION regiao4 VALUES IN (8),
)
```

Particionamento de tabelas

Exemplo 3: O particionamento por HASH é um dos mais simples, e de uso mais comum. Neste tipo de particionamento, você deve especificar a expressão de particionamento e o número de partições a utilizar.

Supondo a tabela de municípios do exemplo anterior:

```
PARTITION BY HASH (id_cidade)
PARTITIONS 10;
```

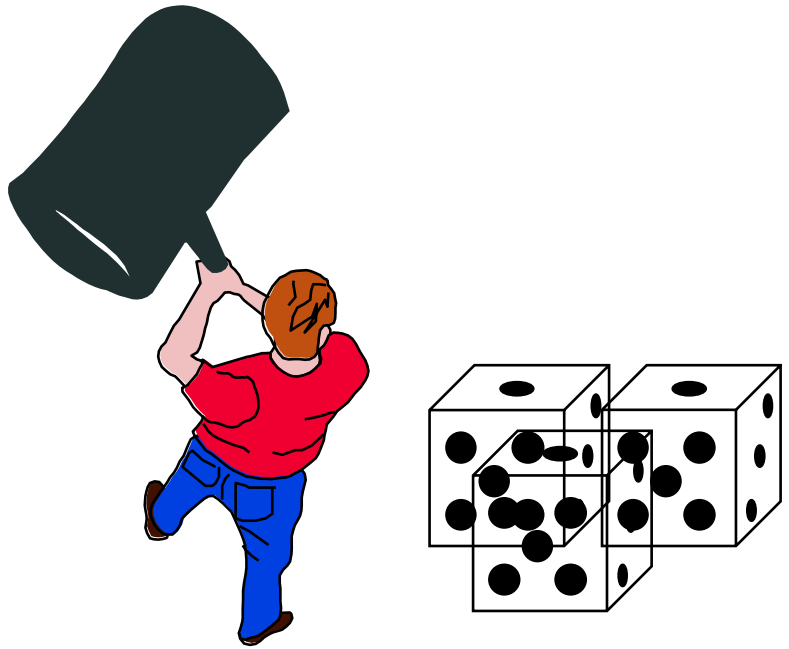
Nesse caso, seriam criadas 10 partições com tamanhos praticamente iguais

Particionamento de tabelas

Observações:

1. Para saber se o BD suporta o recurso de particionamento, pode-se usar o comando **SHOW PLUGIN**.
2. O comandos SELECT realizados em cima de tabelas particionadas devem incluir na cláusula WHERE o campo usado como critério de particionamento. Caso contrário, a pesquisa será realizada em todas as partições e o particionamento perderá sua finalidade.

Modelagem de Dados & Desempenho



Modelagem de Dados & Desempenho

Desde a fase de planejamento e modelagem do Banco de Dados, a preocupação com o desempenho já deve ser levada em conta.

O tamanho das tabelas e campos, os tipos de dados e seus relacionamentos têm grande influência na velocidade das consultas e outras operações.

A seguir veremos os principais pontos que precisam ser estudados nesse aspecto.

Modelagem de Dados & Desempenho

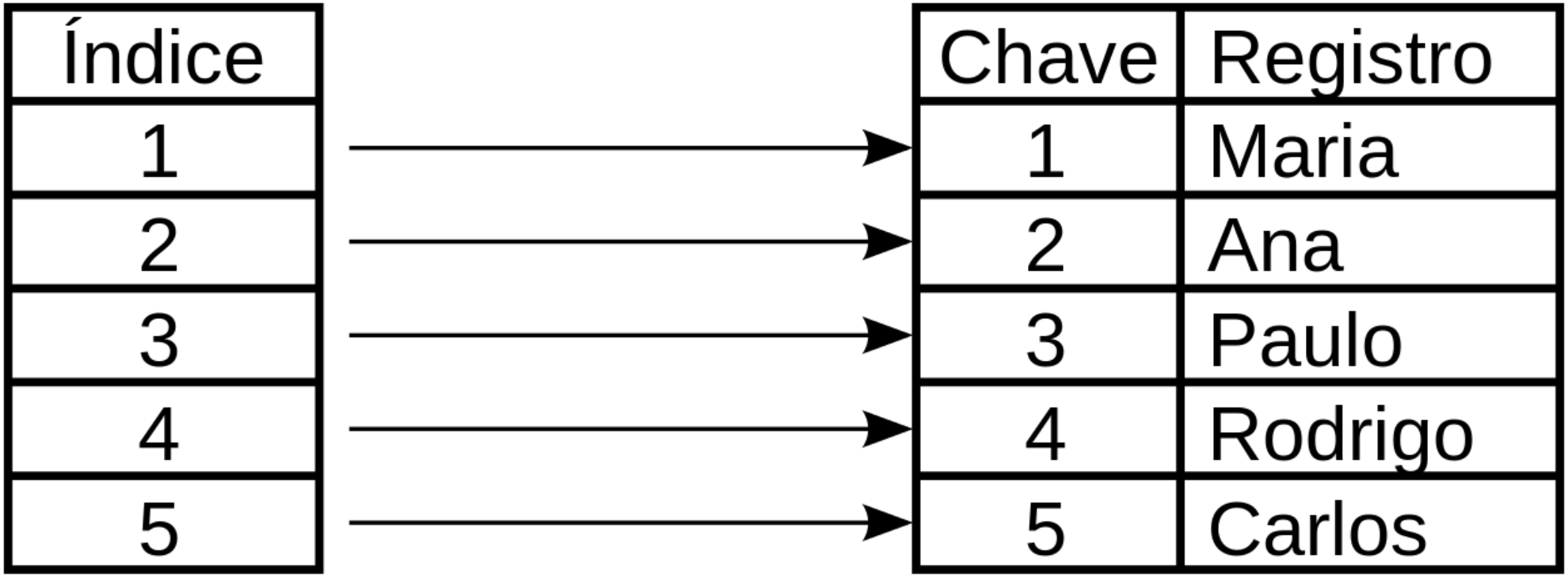
- Quanto menores as tabelas, mais rápidas são as consultas a seus dados. Por isso as tabelas devem ter o menor número de campos possível. Para manter as tabelas pequenas, aplique a normalização, mas não exagere, pois normalização excessiva retarda consultas que possuem muitos joins.
- Os campos também devem ter tamanhos os menores possíveis
- A chaves primárias e estrangeiras, de preferência, devem ser numéricas ou, no máximo, do tipo CHAR.
- Antes de definir os tipos de dados e tamanhos para cada campo, um estudo rigoroso deve ser feito.

Modelagem de Dados & Desempenho

Exemplos:

- Campos como CPF e CEP podem ser definidos como numéricos ou do tipo CHAR, sem a pontuação.
- A chave primária de tabelas que possuem poucos registros podem ser do tipo UNSIGNED TINYINT ao invés de INT;
- Campos como "data de nascimento" podem ser do tipo DATE (3 bytes) ao invés de DATETIME (8 bytes).
- Consultas em campos do tipo CHAR são mais rápidas que em tipos VARCHAR. Por isso, use CHAR para os campos textos que possuem tamanho fixo e VARCHAR para campos com tamanhos variáveis.
- Campos BLOB/TEXT devem ser criado em tabelas separadas ou armazenados em arquivos.

Índices


Índice
1
2
3
4
5
Chave
Registro
1
Maria
2
Ana
3
Paulo
4
Rodrigo
5
Carlos

Índices

Índices são objetos do banco de dados que ajudam a tornar a consulta aos dados muito mais rápida.

Um Índice recupera os registros através de uma chave de pesquisa qualquer que não precisa ser necessariamente a chave primária. **É uma estrutura auxiliar associada aos campos de uma tabela ou view. Sua função é acelerar o tempo de acesso, criando ponteiros para os dados armazenados em colunas específicas**.

O banco de dados usa o índice de forma semelhante ao sumário de um livro, onde você encontra a página correta de acordo com o assunto e não precisa ficar passando folha por folha até encontrar a desejada.

Índices

O processo de criação e utilização dos índices funciona, resumidamente, da seguinte maneira:

1. Seleciona-se uma tabela
2. Para essa tabela, escolhemos um ou mais campos para serem indexados
3. Cria-se um índice ou mais índices para esses campos
4. Quando uma consulta nessa tabela é realizada, o sistema verifica se existem índices criados para os campos pesquisados na consulta. Em caso positivo, essa consulta será feita através dos índices. Caso contrário, a pesquisa será realizada diretamente nos dados

Conclusão: As consultas são mais rápidas quando existem índices criados para os campos pesquisados.

Índices

Por que as consultas são mais rápidas com os índices?

1. Os índices são estruturas menores e organizadas de forma a agilizar as pesquisas. São como tabelas onde os registros estão ordenados, o que facilita as buscas.
2. Existem algoritmos especializados em realizar pesquisas em índices. Sem os índices, a busca é feita através da pesquisa sequencial (linear) diretamente nas tabelas que são estruturas maiores e mais complexas.

Categorias de Índices

Os índices podem ser classificados da seguinte maneira:

- **Clusterizados** → Esse tipo de índice é armazenado juntamente com os dados na própria tabela e impõe a ordem dos registros. Cada tabela somente pode possuir um único índice clusterizado.
- **Secundários** → São armazenados em estruturas fora da tabela. Podem ser criados vários índices secundários para a mesma tabela.

Obs: No MySQL, ao definir um campo como chave primária, é criado automaticamente um índice clusterizado (PRIMARY) para ela.

Tipos de Índices

O MySQL disponibiliza dois tipos de algoritmos para pesquisa em índices:

B-tree → Padrão. É o tipo mais utilizado na maioria dos SGBDs. Permite fazer pesquisas por comparação e igualdade

Hash → É mais rápido que o B-tree, no entanto só faz pesquisas por igualdade.

Índices – Outras características

- Para eliminar as duplicidade dos campos indexados, podemos definir o índice como UNIQUE
- Para efetuar buscas de "substrings" por similaridade dentro de campos texto de qualquer tamanho é possível criar índices do tipo FULLTEXT que apresentam maior precisão e rapidez nesse tipo de consulta.
- Para agilizar consultas de dados no padrão dos sistemas de informações Geográficas, podemos criar índices do tipo SPATIAL

Criando Índices

Para criarmos índices para uma determinada tabela, usamos o comando CREATE INDEX, passando as seguintes informações:

- O nome do índice
- O nome da tabela
- Os campos que vão ser indexados
- Informações adicionais sobre o índice

CREATE INDEX nome_indice ON nome_tabela (lista_colunas)

Observação: Para campos tipo texto, é possível criar índices por parte do valor (prefix) por motivos de performance já que, quanto menor o índice, mais rápida é a consulta

Criando Índices - Exemplos

Seja a tabela:

```
CREATE TABLE cliente(
id INT AUTO_INCREMENT PRIMARY KEY,
nome VARCHAR(80) NOT NULL,
cpf int(11)
nascimento date,
);
```

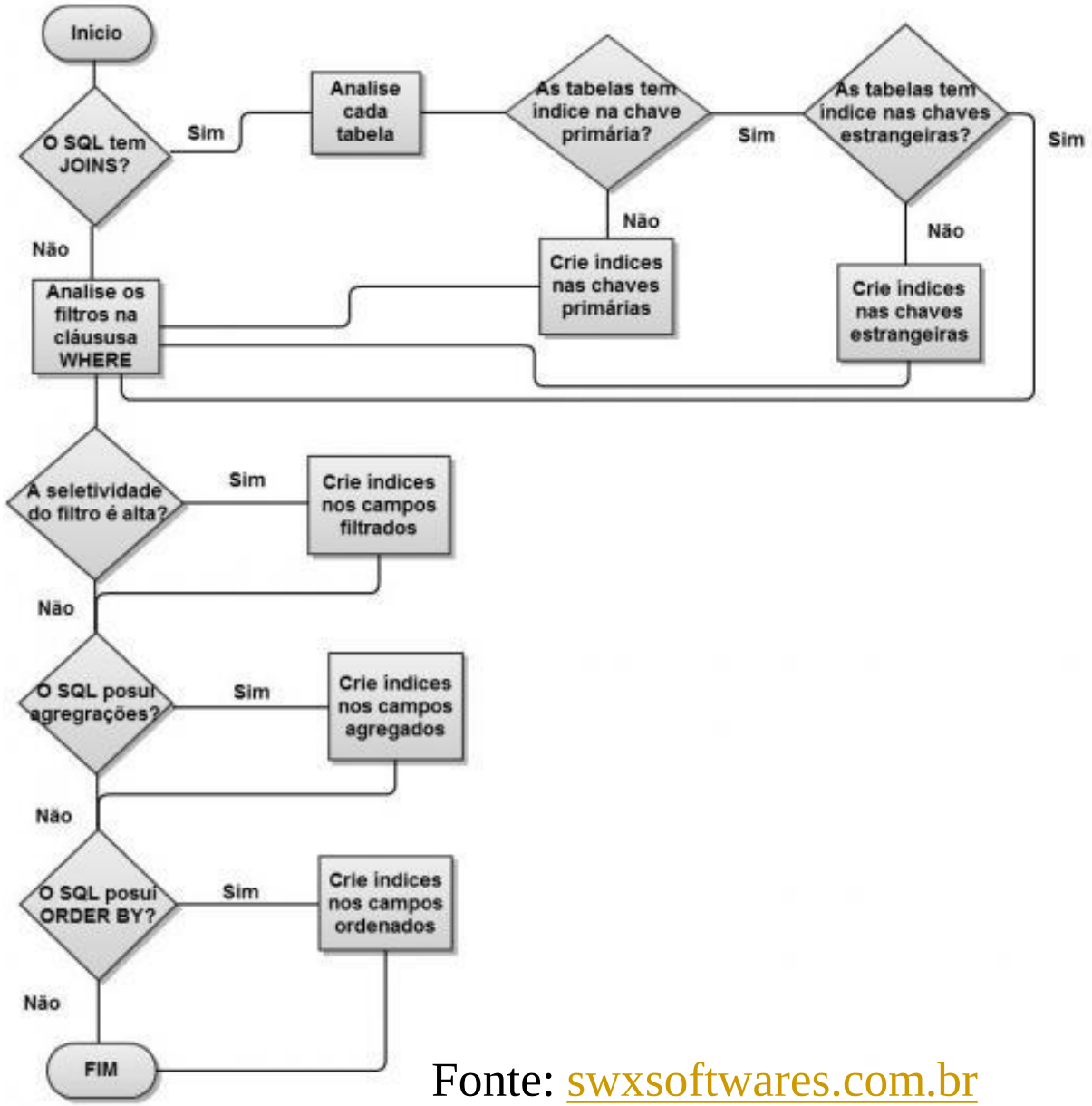
- Criando índice por data de nascimento:
  CREATE INDEX idx_nasc ON cliente (nascimento)
- Criando índice para os primeiros 30 caracteres do nome:
  CREATE INDEX idx_parte_nome ON cliente (nome(30))
- Criando um índice único para o cpf do tipo B-tree (opcional):
  CREATE UNIQUE INDEX idx_cpf ON cliente (cpf) USING BTREE

Quando NÃO usar índices

Os índices agilizam as consultas aos dados, mas retardam as demais operações de manutenção como INSERTS e UPDATEs. Isso se dá porque, quando alteramos os dados, os índices devem ser atualizados também. Além disso, índices também ocupam espaço precioso no servidor, o que pode ser prejudicial ao desempenho do SGBD. Por essas razões, a criação de índices deve ser feita de forma criteriosa.

A seguir, segue um fluxo que pode auxiliar o planejamento na criação de índices...

Fluxo para Criação de Índices



Fonte: swxsoftwares.com.br

Analisando o Plano de Execução

Analisando o Plano de Execução

Fonte: devmedia

Quando um comando SQL é executado no MySQL, ele passa antes pelo otimizador que usa as estatísticas do banco de dados para criar um plano de execução que vai retornar os dados da melhor maneira possível.

Entender o plano de execução é fundamental para resolver e evitar problemas de performance das consultas.

Analisando o Plano de Execução

O otimizador de queries do MySQL cria o plano de execução do comando com base nos índices, da seguinte forma:

1. Obtém os filtros da consulta (as cláusulas WHERE)
2. Verifica para cada filtro, se existem índices para os campos pesquisados e quais são os critérios
3. Caso existam índices e os índices são úteis, então a pesquisa será feita a partir dos índices
4. Caso contrário, a pesquisa será feita diretamente nos dados de forma sequencial

Podemos visualizar o plano de execução do otimizador através do comando **EXPLAIN**

O comando EXPLAIN

O comando **EXPLAIN** espera como parâmetro o próprio comando **SELECT** a ser analisado e retorna um relatório com os seguintes campos para cada tabela envolvida:

Coluna	Significado
id	Número sequencial
select_type	Tipo do select
table	Tabela analisada
partitions	Partição atingida
type	Tipo da junção
possible_keys	Possíveis índices a serem usados
key	O index escolhido
key_len	Tamanho da chave
ref	As colunas pesquisadas do índice
rows	Número de linhas retornado
filtered	Porcentagem e linhas em relação ao total
Extra	Informações adicionais

O comando EXPLAIN

Os campos do relatório do EXPLAIN que merecem maior atenção, são:

type → Resultados como **ALL** ou **INDEX** neste campo indica que a query apresenta problemas de lentidão pois está fazendo uma leitura sequencial. Já resultados como **CONST** ou **EQ_REF** são indicativos de que os índices foram bem utilizados

possible_keys e **key** → Ajudam a identificar se os índices disponíveis e o índice escolhido estão de acordo com as expectativas

rows → Um número excessivo de linhas retornadas pode indicar que a query necessita de mais restrições

extra → Resultados como "Using temporary" ou "Using filesort" indicam consultas com problemas de execução

Forçando o uso de Índices

Pode-se forçar ou ignorar a utilização de índices:

- *USE INDEX* define quais índices poderão ser utilizados para a consulta. Caso o MySQL ache que nenhum dos índices é performático, nenhum índice é utilizado.
- *FORCE INDEX* obriga a utilizar um dos índices listados mesmo que o índice não seja performático. E só não será utilizado se for impossível de se aplicar.
- *IGNORE INDEX* desconsidera os índices listados para a realização da consulta.

Exemplo:

```
SELECT * FROM tabela [USE INDEX|FORCE INDEX|IGNORE INDEX] (indice1,indice2) WHERE condicoes
```

Manutenção de tabelas & Índices

Manutenção de Tabelas e Índices

Quando uma tabela recebe um número muito alto de operações CRUD, o espaço ocupado pode aumentar muito e ficar fragmentado.

Isso pode resultar em muito espaço sem uso, bem como tornar os índices desorganizados, influenciando negativamente no desempenho do banco de dados.

Desta forma, se torna necessário realizar periodicamente operações de análise e manutenção de tabelas que veremos a seguir...

Manutenção de Tabelas e Índices

Para determinar quais tabelas no Banco de Dados estão com problemas de fragmentação, podemos realizar o seguinte comando:

```
select
   table_name,
   round(data_length/1024/1024) as 'Usado',
   round(data_free/1024/1024)as 'Sem uso'
from information_schema.tables
   where round(data_free/1024/1024) > 500
order by
   data_free_mb;
```

Isso exibirá a lista de todas as tabelas que possuem no mínimo 500 MB de espaço não utilizado e são candidatas a uma desfragmentação.

Comando OPTIMIZE TABLE

O comando **OPTIMIZE TABLE** é usado para reorganizar as tabelas e seus índices. Sua sintaxe é simples:

```
OPTIMIZE TABLE nome_da_tabela
```

Este comando realizará as seguintes operações:

1. Cria uma tabela temporária
2. Deleta a tabela original depois da otimização
3. Renomeia a tabela temporária para o nome original e finaliza

Isso vai fazer com que todo o espaço seja desfragmentado e os índices atualizados. “Tudo bunitinho”.

Essa pode ser uma operação demorada, dependendo do tamanho da tabela.

Comando ANALIZE TABLE

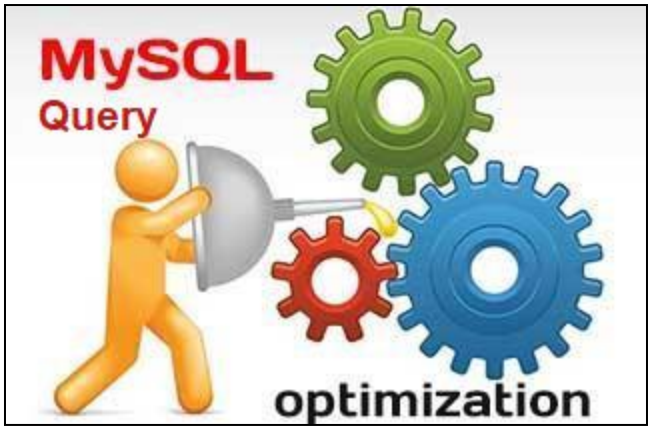
O MySQL armazena informações sobre os índices em um recurso do sistema chamado **key distribution analysis**. Esse recurso é utilizado pelo otimizador de consultas na preparação do plano de execução para, entre coisas, decidir quais índices usar em uma consulta, como vimos no comando **EXPLAIN**.

Para verificar e atualizar o KDA, podemos executar o comando **ANALIZETABLE** da seguinte forma:

```
ANALIZE TABLE nome_da_tabela
```

Esse comando limpa as estatísticas da tabela de forma que elas serão coletadas novamente na próxima vez que a tabela for acessada.

Dicas de
Otimização de Consultas SQL



Dicas de Otimização de consultas SQL

Muitas vezes, o problema na execução da consulta está na forma como o comando SQL foi construído. Vamos relacionar agora algumas dicas para nortear a criação de comandos SELECT com foco no desempenho:

- Evite o uso de asterisco (*) nas consultas. Isso pode causar uma sobrecarga desnecessária. Prefira selecionar apenas os campos que vão ser efetivamente utilizados.
- Sempre restrinja a quantidade de linhas e colunas retornadas em suas consultas. Ou seja, sempre tente filtrar com o número maior de cláusulas WHERE, sem desviar do objetivo da consulta.
- Muita atenção com os JOINS. Verifique se existem índices criados para as chaves da junção e se a junção é realmente necessária

Dicas de Otimização de consultas SQL

- A ordem decrescente de performance dos operadores em consultas é:
  - =
  - >, >=, <, <=
  - LIKE
  - <>
- Evite executar cálculos matemáticos na cláusula WHERE. Se possível, envie esses cálculos já prontos. Por exemplo, a condição **taxa * 2 < 100** poderia ser escrita **taxa < 50**
- Avalie bem se as VIEWs, são realmente úteis. Caso não sejam, prefira consultar diretamente as tabelas.
- Evite funções definidas pelo usuário na cláusula WHERE
- Evite usar IN com muitos itens

Dicas de Otimização de consultas SQL

- Consultas com cláusulas WHERE conectadas por operadores AND ou OR são avaliadas da esquerda para a direita. Logo, é melhor colocar as colunas com **maior probabilidade** de serem **falsas** no começo da consultas com AND. Com o OR é o contrário, coloque as possíveis **verdadeiras** antes.
- O LIKE é potencialmente lento. Quando utilizá-lo é recomendável não utilizar o "%" no começo do valor.
- Procure reduzir o número de colunas em GROUP BY
- Em consultas muito frequentes, procure criar uma STORED PROCEDURE. Isso se deve à otimizações no plano de execução dos procedimentos armazenados.

Configurações do Servidor

Configurações do Servidor

Algumas configurações de performance podem ser feitas no arquivo de configuração do MYSQL (my.cnf ou my.ini). Basta editar esse arquivo na seção [mysqld].

- **Monitoramento de consultas lentas**

```
long_query_time = 60
log_slow_queries = /var/log/mysql-slow.log
```

De acordo com a configuração acima, o MySQL vai monitorar as consultas que demoram mais de 60 segundos para serem executadas e vai gravar o log de resultados no arquivo mysql-slow.log

Configurações do Servidor

- **Número de conexões simultâneas**

```
max_connections = 200
```

O parâmetro acima determina um número máximo de 200 conexões. Aumentar esse parâmetro influencia a quantidade de Memória disponível.

Configurações do Servidor

- **Cache de tabelas**

Esse parâmetro define o número de tabelas abertas no cache. Um valor maior melhora a performance, mas diminui a memória. Está diretamente relacionado com o parâmetro max_connections.

Multiplique max_connections pelo número de tabelas em cache:

```
table_cache = 2000
```

Neste exemplo configuramos 10 tabelas para cada conexão.

Configurações do Servidor

- Quantidade de memória alocada para o resultado das queries:

```
query_cache_size = 128M
```

Aumenta a performance das consultas. Os valores para esse parâmetro devem ser múltiplos de 1024. Cuidado para não setar para um valor muito alto.

Configurações de Hardware & Software

Atualização do Software

A Oracle divulgou no lançamento do MySQL 8 que essa versão era duas vezes mais rápida que as anteriores, devido aos inúmeros aperfeiçoamentos realizados no seu desenvolvimento.

Por isso, a **atualização da versão** do servidor deve ser a primeira providência que o administrador deve realizar, não só com vistas à melhoria do **desempenho** do banco, mas também por questões de **segurança e integridade**.

É claro que nem sempre isso é possível de se fazer com frequência em ambientes de produção, mas essa é uma atividade que deve estar no checklist do DBA.

Isso vale também para o Sistema Operacional, Servidor WEB e demais softwares envolvidos.

Hardware & Software

Algumas recomendações:

- CPUs mais rápidas são melhores do que mais CPUs
- 64 bits com SOs de 64 bits
- I/O: discos mais rápidos e com mais espaço
- Mais memória (óbvio)
- Separar o servidor de banco de dados do servidor de aplicação.
- Rede dedicada

Receita para Otimizar Consultas Lentas

Receita para Otimizar Consultas Lentas

Nessa receita, só é necessário executar o próximo item se não obteve sucesso no anterior:

1. Descobrir qual a query está lenta, usando o **log_slow_queries**
2. Otimizar as tabelas envolvidas com o comando OPTIMIZE
3. Executar o **EXPLAIN.** Com base nele, pode ser necessário:
    1. Ajustar/criar índices;
    2. Restringir os resultados da consulta
    3. Ajustar os joins
    4. Particionar
    5. Aplicar as recomendações/dicas sobre SELECTs e modelagem de dados
4. Ajustar configurações do Servidor/Hardware/SO/Rede
5. ?

FIM

e Obrigado!

Referências

Links:

- mysql.com
- devmedia.com
- king.host
- imasters.com.br

Livros:

- Alto Desempenho em MySQL – Schwartz e outros – Alta Books