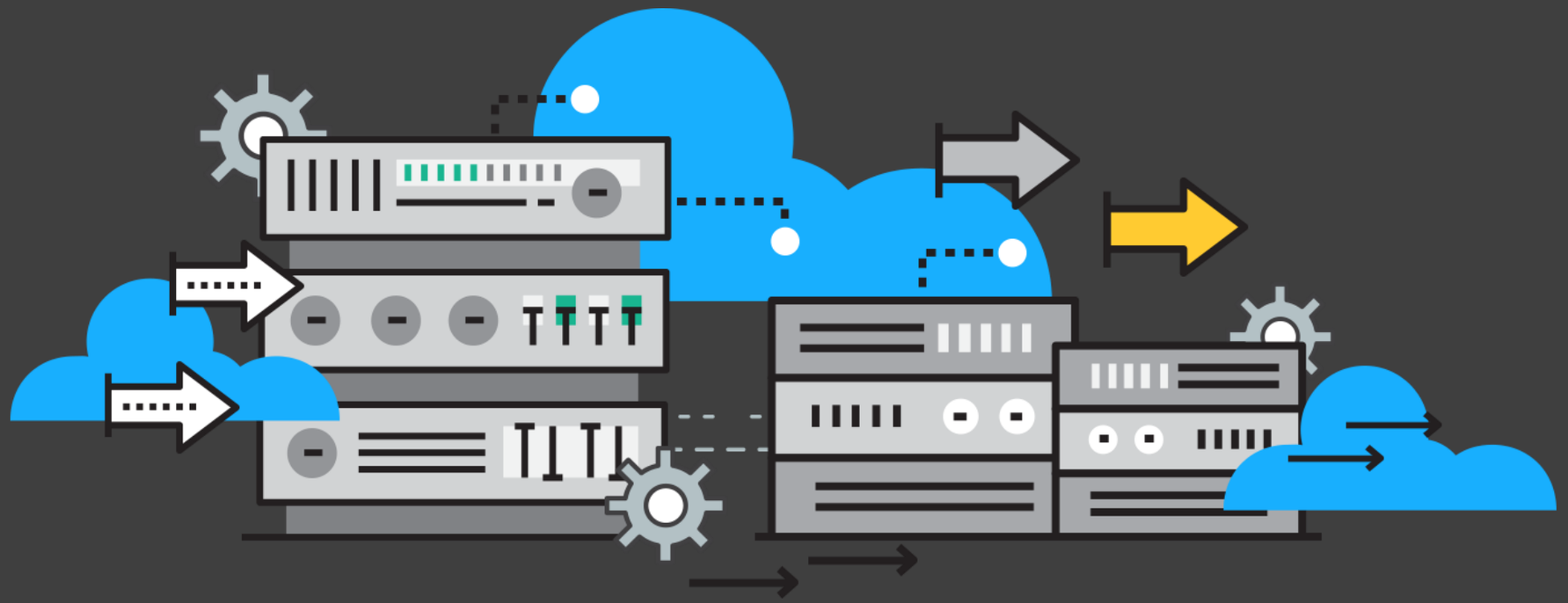
targettrust
treinamento e tecnologia

Be code

KINGHOST



Performance e Otimização de Banco de Dados MySQL

Índice

Antes de ler...

Algumas dicas são extremamente importantes para que você aproveite o melhor que esse ebook tem para oferecer:

1. Todas as técnicas apresentadas nesse ebook são válidas ou possuem foco para o SGBD MySQL. Isso não quer dizer que outros SGBDs não possuam recursos similares. Entretanto, a forma de utilizar esses recursos pode ser levemente ou totalmente diferente. Sendo assim, em alguns casos, é possível que você consiga aplicar os aprendizados desse ebook em outros SGBDs, sem maiores dificuldades; Já, em outros casos, não.

2. Esse ebook é um conteúdo de nível intermediário. Portanto, se você não possui esse nível de fluência com Banco de Dados Relacionais e MySQL. É necessário que você busque, em primeiro lugar, aprender os conceitos iniciais sobre estas tecnologias. Na Becode, temos um curso focado em introduzir o profissional no ramo do Banco de Dados Relacionais, utilizando o MySQL. Dê uma olhada!

Introdução

Neste e-book vamos mostrar as melhores práticas para a configuração de um servidor MySQL, visando o desenvolvimento de aplicações que consumam menos dados, de forma mais efetiva e otimizada.

Este ebook está dividido em quatro módulos principais:

- **Indexação de Tabelas**
- **Particionamento de Tabelas**
- **Otimização de Consultas**
- **Infraestrutura**

Esperamos que você goste do material e, principalmente, aprenda muito com ele! Boa leitura!

Indexação de tabelas

Como a indexação favorece a velocidade de buscas.

Indexação de Tabelas

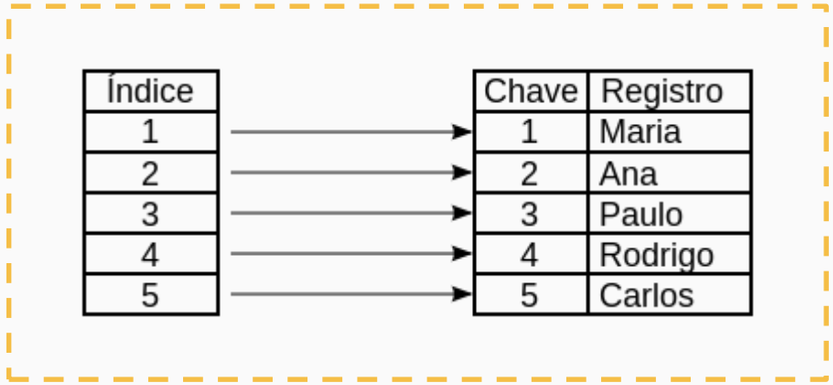
Este tópico aborda o funcionamento dos índices, sua importância para as questões de performance e relevância no consumo de dados, na utilização de um banco de dados MySQL. Para entender melhor os índices é preciso focarmos em quatro perguntas:

- **O que são?**
- **Quais tipos?**
- **Por que utilizar?**
- **Quando utilizar?**

O que são Índices?

No contexto da estrutura de dados: trata-se de uma referência associada a uma chave. É utilizada para fins de otimização de consultas, pois permite a localização mais rápida de um registro.

No contexto de banco de dados: trata-se de uma estrutura (ou arquivo) auxiliar associado a uma tabela (ou coleção de dados). Possui a função de acelerar o tempo de acesso às linhas de uma tabela.



Tipos de Índices?

Existem quatro tipos básico de índices:

- **Compostos** - Mais de uma coluna da tabela sendo utilizada;
- **Simples** - Apenas uma coluna da tabela sendo utilizada;
- **Internos** - Dentro da estrutura de dados da tabela;
- **Externos** - Faz referência fora da tabela

Por que utilizar?

A utilização de índices é indicada para que haja um menor consumo e retorno mais rápido do servidor para a aplicação em questão. Resumindo, eles servem para proporcionar:

- Agilidade e performance em buscas;
- Menor consumo de recursos (Input e Output de disco), pois consome menos recurso do servidor, processamento e memória, além de trazer retorno para a aplicação de forma mais rápida.

Input e Output (leitura e gravação em disco)? Para cada busca realizada no servidor de banco de dados, caso aquele dado esteja em memória (cache), é necessário realizar uma busca diretamente no disco. Essas buscas tem um alto nível de processamento.

Quando utilizar?

Os índices devem ser utilizados sempre que houver um grande número de pesquisas sobre determinado item, afim de otimizar as buscas e melhorar a performance.

Como criar um índice simples?

Nos exemplos a seguir, vamos utilizar o banco de dados “perf”, criado com quatro tabelas: fotos, noticias, noticias2 e noticias3. Vamos começar selecionando nosso banco de dados. Abra o terminal e selecione o banco de dados “perf” com o seguinte comando:

```
USE perf;
```

Agora vamos aprender a criar um índice simples. Para isso, criamos uma tabela (“produtos”), adicionando um índice para o campo nome, através do comando **INDEX idx_nome(Nome):**

```
CREATE TABLE produtos (
        Codigo INT,
        Nome VARCHAR(50),
        key Codigo(Codigo),
        INDEX idx_nome(Nome)
);
```

O índice (idx_nome) referencia o campo nome, utilizando o campo completo como índice. Para visualizar os índices de uma tabela, você pode utilizar o comando **“SHOW INDEXES”.** A partir da tabela recém-criada, vamos para o prompt e escrevemos o comando:

```
SHOW INDEXES FROM produtos;
```

Agora você já sabe como criar um índice no momento da criação da tabela. Portanto, vamos excluir a tabela recém criada e aplicar um exemplo mais usual. Para isso, utilize o comando “DROP TABLE” para excluir a tabela recém criada.

```
DROP TABLE produtos;
```

Adicionando índices a tabelas existentes

É importante que o banco de dados seja planejado para a otimização de certas buscas. Caso contrário, no momento em que as aplicações necessitarem consumir mais dados do banco, poderemos ter problemas na performance. Em uma situação comum, não teríamos índices na estrutura inicial da tabela. Por isso, vamos criar novamente a tabela “produtos” sem adicionar índices.

```
CREATE TABLE produtos (
            Codigo INT,
            Nome VARCHAR(50),
            key Codigo(Codigo)
      );
```

Quando percebemos que a aplicação está perdendo performance, pois são feitas muitas requisições de busca pelo nome dos produtos, podemos resolver o problema através de um índice para o campo “nome”. Para adicionar um índice em uma tabela já criada, vamos utilizar o seguinte comando:

```
CREATE INDEX idx_nome ON produtos(Nome);
```

Tutorial completo sobre índices

Neste webinar realizado pela equipe da KingHost, detalhes práticos quanto a utilização de índices, benefícios trazidos, como fazer e entre outros pontos são elucidados de uma forma mais clara, onde você pode ver profissionais realizando na prática, técnicas de otimização de consultas através da criação de índices.

Inclusive, no Webinar, você consegue ver a diferença existente, em termos de performance, entre tabelas que utilizam índices e as que não utilizam. Para ver o vídeo, acesse o **seguinte link.**

(Esta explicação encontra-se entre os minutos 16:00 e 27:30)

Assista aqui o nosso vídeo tutorial e aprenda a usar índices em SQL

Particionamento de tabela

Divisão de tabelas, o caminho para mais desempenho.

Particionamento de Tabela

Os índices auxiliam em consultas e subconsultas, porém quando a tabela possui um grande volume de dados, o desempenho do banco de dados continua prejudicado.

Para resolver este problema, o particionamento de tabelas, ou seja, a divisão de uma tabela inteira em sub-tabelas menores, torna-se a solução ideal. A grosso modo, estamos quebrando tabelas em sub tabelas, ou seja, particionando.

Uso padrão das tabelas:
Dados e índices são armazenados em um arquivo único (dependente da engine de armazenamento).

Particionamento:
Dados e índices são armazenados em múltiplos arquivos, 1 por partição. Número máximo de partições: 1024 (isto é definição do MSQL).

Quais as vantagens?

A aplicação permanece igual!

Na aplicação, nada muda. A tabela continuará sendo vista como uma tabela única.

Mais dados em uma única tabela

Dependendo do sistema de arquivos utilizado pelo servidor MySQL, há um limite máximo de tamanho de arquivo, o que limita a quantidade de dados que podem ser inseridos em uma tabela. Utilizando o particionamento, partimos a tabela em múltiplos arquivos, aumentando assim o limite máximo de dados que podem ser inseridos nessa tabela. Em sistemas de arquivo modernos, esse limite é muito alto, mas ainda assim, é uma vantagem.

Remoção de dados absoletos

Dependendo do particionamento utilizado, a remoção de dados obsoletos é facilitada. Esta vantagem é mais fácil de explicar utilizando o exemplo abaixo.

Pense em uma escola, onde todas as matrículas encontram-se em uma única tabela. Neste caso, poderíamos particionar esta tabela única pelo ano de matricula, gerando diversas outras sub-tabelas. Portanto, caso não houvesse mais o interesse por um determinado ano de matrícula, a remoção poderia ser aplicada especificamente naquele ano de interesse. Desta forma, removendo os dados absoletos.

Desempenho de buscas otimizado

O desempenho de buscas é otimizado, pois diminui o universo no qual elas são realizadas.

Antes a busca era realizada baseando-se em um universo enorme de dados de uma tabela, agora a busca baseia-se em apenas uma fração deste universo, bem menor e subdividido.

IMPORTANTE: É essencial que, na cláusula WHERE da busca, seja especificado o critério de particionamento para que o MySQL possa determinar em quais partições deve realizar a busca. Caso contrário, um 'full-scan' será realizado, ou seja, uma busca em todas as partições; isso será mais lento que buscar 'full-scan' em uma tabela não-particionada.

Tipos de particionamento disponíveis

- RANGE [COLUMNS]
- LIST [COLUMNS]
- HASH
- [Linear] Key

Range Columns e List Columns só estão disponíveis a partir da versão 5.5 do MySQL.
Os servidores da **KINGHOST** já possuem essa variante.

RANGE

Em Range, o particionamento é baseado em intervalo de valores. Como exemplo, podemos usar uma tabela de matrículas de uma escola. A “partição 1” guarda as matrículas que vão do ano de 1990 até 1999 e a “partição 2” irá do ano 2000 até 2010.

Neste intervalo de valores os registros estarão na partição delimitada. Com isso o MySQL saberá automaticamente em qual partição, ou seja, em qual arquivo ele irá armazenar e qual arquivo ele irá buscar.

IMPORTANTE: O Range pode ser particionado por uma coluna do tipo INT, podendo ser smallint e suas variantes, mas precisa ser um campo inteiro. Exemplo:

```
PARTITION BY RANGE (YEAR(matricula) )
```

EXEMPLO DE RANGE

Em uma tabela de admissão de funcionários, vamos particionar pelo mês de admissão. Para isso, criamos uma partição para cada trimestre, sem nos preocuparmos com o ano e nestas partições estarão os registros do primeiro, segundo, terceiro e quarto trimestres de todos os anos. Ao executar uma **QUERY** em que a solicitação seja para buscar todos os funcionários admitidos em 20 de maio de 2015, teremos o seguinte:

Select *From Funcionarios WHERE admissão = '2015-05-20'

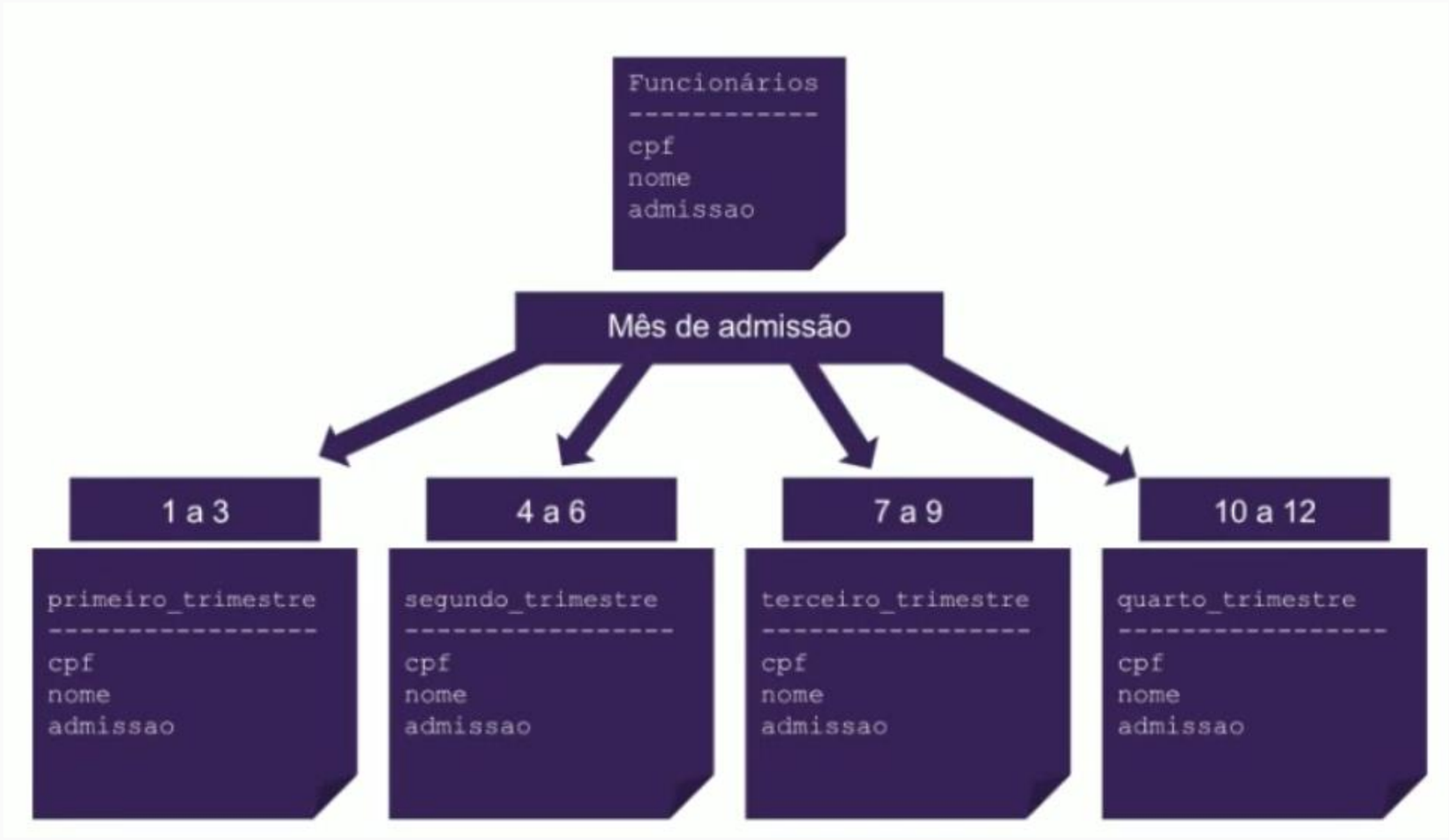
O MySQL irá entender que maio é o mês cinco e irá até o particionamento do segundo trimestre executando a lógica de busca a seguir:

As partições são avaliadas de cima para baixo. Ou seja, quando a busca é efetuada para achar uma data no mês de maio, o sistema irá se perguntar "partition primeiro trimestre" e verá que cinco não é menor que quatro, mas cinco é menor do que sete, então ele irá para a segunda partição, onde está o mês de maio e a informação desejada.

RANGE

```
CREATE TABLE `Funcionarios` (
      `cpf` VARCHAR(14) NOT NULL,
      `nome` VARCHAR(255) NOT NULL,
      `admissao` DATE NOT NULL
)
PARTITION BY RANGE(MONTH(admissao)) (
      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
);
```

Modelo Conceitual



RANGE COLUMNS

RANGE COLUMNS é uma variante de RANGE. Neste caso, é possível utilizar mais de uma coluna para realizar o particionamento. Podemos então usar diversas colunas e tipos que não sejam campos inteiros.

PARTITION BY RANGE COLUMNS (YEAR(matricula), id_cidade)

RANGE COLUMNS

```
CREATE TABLE `xyz` (
    `a` INT NOT NULL,
    `b` INT NOT NULL,
    `c` DATETIME NOT NULL
)
PARTITION BY RANGE COLUMNS (a, MONTH(c)) (
    PARTITION p0 VALUES LESS THAN (3, 7),
    PARTITION p1 VALUES LESS THAN (4, 9),
    PARTITION p2 VALUES LESS THAN (4, 11),
    PARTITION p3 VALUES LESS THAN (MAXVALUE, MAXVALUE)
);
```

LIST

LIST é muito parecido com RANGE, porém, a especificação deve ser feita por valores específicos.

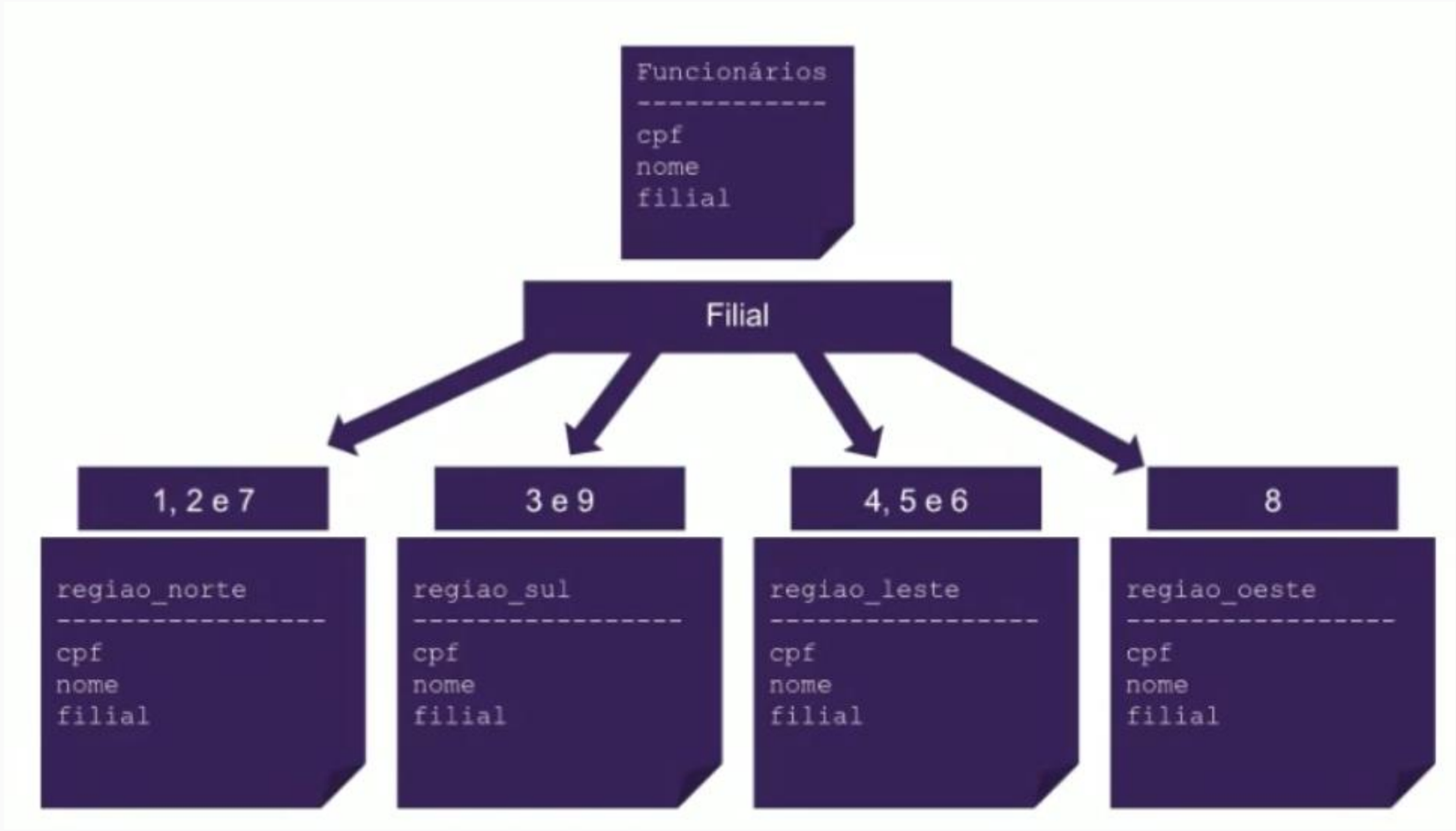
PARTITION BY LIST (YEAR(matricula))

EXEMPLO LIST

Ao invés de particionar por mês de admissão, temos funcionários por filial e há uma partição que é a região norte, que são os funcionários cuja a filial são um(1), dois(2) e Sete(7). Então, se buscarmos todos os funcionários que trabalham na filial 7, o MySQL irá entender que não é necessário ler as outras regiões e irá direto para a região norte. Observe que não há intervalo de valores neste exemplo. Lembrando que LIST precisa ter valores inteiros e para LIST COLUMNS não é necessário que haja valores inteiros.

```
CREATE TABLE `Funcionarios` (
    `cpf` VARCHAR(14) NOT NULL,
    `nome` VARCHAR(255) NOT NULL,
    `filial` INT NOT NULL
)
PARTITION BY LIST(filial) (
    PARTITION regiao_norte    VALUES IN (1, 2, 7),
    PARTITION regiao_sul      VALUES IN (3, 9),
    PARTITION regiao_leste    VALUES IN (4, 5, 6),
    PARTITION regiao_oeste    VALUES IN (8),
);
```

Modelo Conceitual



LIST COLUMNS

A diferença entre LIST e LIST COLUMNS é a mesma do RANGE e RANGE COLUMNS. LIST só aceita uma coluna para realizar o particionamento e List Columns várias colunas de vários tipos.

PARTITION BY LIST COLUMNS (YEAR(matricula), id_cidade)

LIST COLUMNS

```
CREATE TABLE `Funcionarios` (
    `cpf` VARCHAR(14) NOT NULL,
    `estado`VARCHAR(2) NOT NULL DEFAULT 'RS'
)
PARTITION BY LIST COLUMNS (estado) (
    PARTITION regiao_sul VALUES IN ('RS', 'SC, 'PR'),
    PARTITION regiao_sudeste VALUES IN ('SP', 'RJ', 'MG', 'ES'),
    PARTITION regiao_centro_oeste VALUES IN ('MT', 'MS', 'GO', 'DF'),
    PARTITION regiao_norte VALUES IN ('AC', 'AM', 'RO', 'RR', 'PA', 'AP', 'TO'),
    PARTITION regiao_nordeste VALUES IN ('MA', 'PI', 'CE', 'RN', 'PB', 'PE', 'AL', 'SE', 'BA')
);
```

HASH

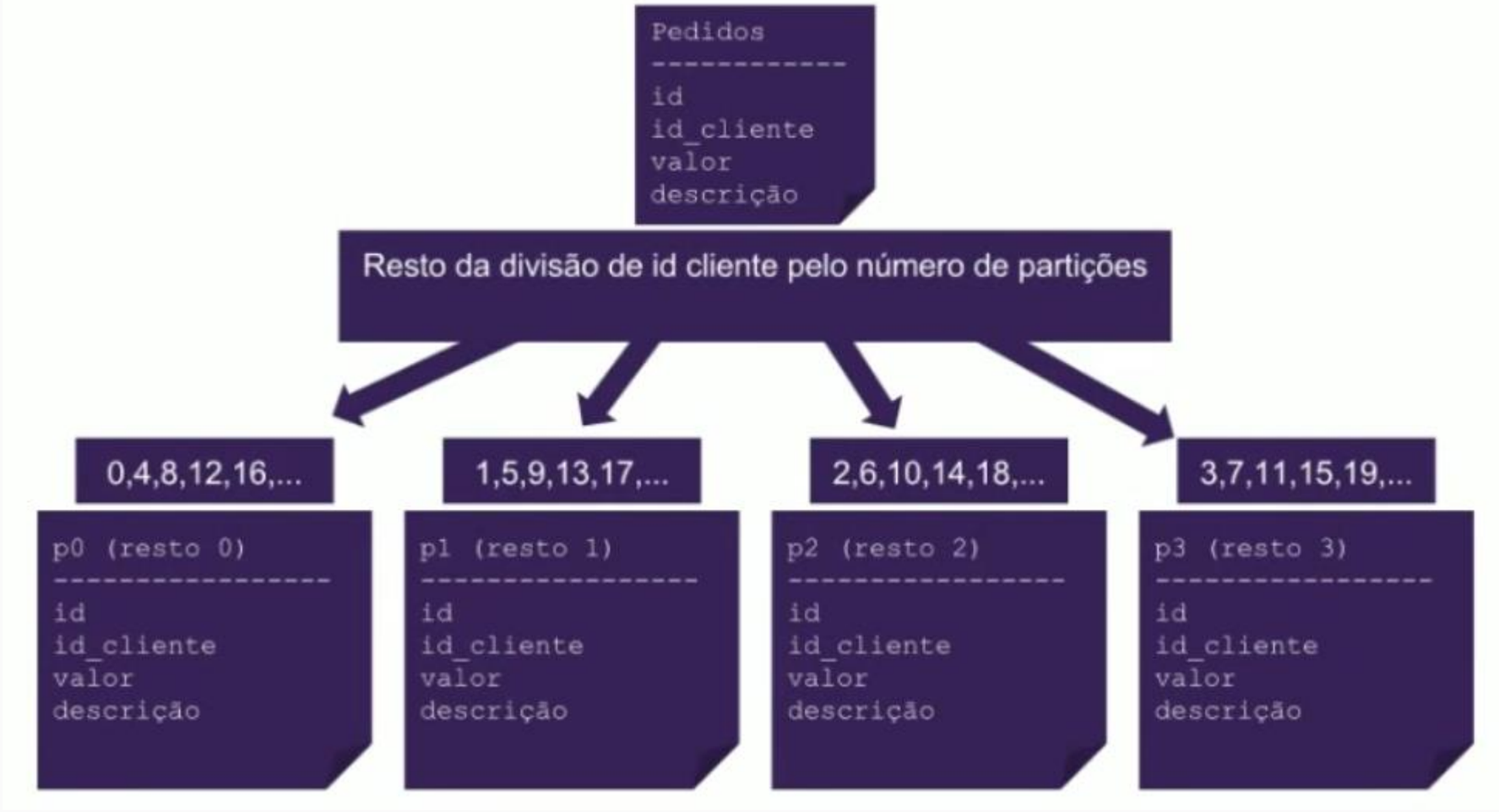
Um dos tipos de particionamentos mais comuns e utilizados. Particiona com base em um cálculo sem especificar o que vai em qual partição, sendo que o MySQL faz o cálculo e define. O HASH normalmente é utilizado para ter uma distribuição uniforme entre as partições, para assim, obter um desempenho melhor na busca.

PARTITION BY HASH (YEAR(matricula))

Como funciona o cálculo?

Especifica-se para a “PARTITION” uma coluna (inteira) e o MySQL irá dividir o valor da coluna pelo número de partições definidas para a tabela e o restante da divisão nos dirá em qual partição ele será armazenado.

Modelo Conceitual



EXEMPLO HASH

Em uma tabela de pedidos que foi particionada em 4. P0, P1, P2 e P3. O cálculo será feito pelo HASH de Id de cliente e pelo número de partições. Ou seja, se for necessário consultar os pedidos do cliente 0 ele vai ir para a P0. Caso seja necessário os dados do cliente 14, vai ir para a P2, conforme diagrama da página anterior.

Isso garante que sempre haverá uma distribuição uniforme ao longo das partições.

```
CREATE TABLE `Pedidos` (
    `id`NOT NULL AUTO_INCREMENT,
    `id_cliente` INT NOT NULL,
    `valor` DECIMAL(5, 2) NOT NULL,
    `descricao` VARCHAR(255) NOT NULL
)
PARTITION BY HASH (id_cliente)
PARTITIONS 4;
```

KEY

Semelhante ao Hash, mas é possível especificar uma coluna ou não. Possui uma série de regras que devem ser levadas em consideração para o uso correto, entre elas:

Se não é especificada a coluna, o MySQL tentará usar:

- Chave primária (se houver)
- Chave única (se houver)

OBS: se não houver nenhuma das duas, a aplicação apresentará erro.

Se forem especificadas colunas:

- Essas colunas precisam pertencer a chave primária ou a chave única.

OBS: se não houver nenhuma das duas, a aplicação apresentará erro.

KEY

```
CREATE TABLE k1 (
    id INT NOT NULL,
    name VARCHAR(20),
    PRIMARY KEY (id)
)
PARTITION BY KEY()
PARTITIONS 2;
```

[LINEAR] HASH e [LINEAR] KEY?

Tanto com o particionamento com o HASH, quanto com o particionamento por KEY, é possível utilizar o “LINEAR” antes de executar o comando (na frente de HASH ou KEY.)

Contudo, isto irá mudar o cálculo padrão para outro baseado em potências de 2. Um cálculo um pouco mais complexo, para a grande maioria dos casos, pode ser considerado desnecessário. Portanto, tanto na KingHost, quanto na Becode, é utilizado com mais frequência o comando com a ausência do LINEAR.

SUBPARTICIONAMENTO

O foco do subparticionamento é partir partições, sempre lembrando que só podem ser usadas 1024 subpartições.

Para utilizar o subparticionamento, é preciso seguir duas restrições.

1 - O "PARTITION" mestre precisa ser do tipo RANGE ou LIST;

2 - O subparticionamento deve ser do tipo HASH ou KEY.

Portanto, não se pode subparticionar uma tabela particionada, de forma mestre, por HASH ou KEY.

EXEMPLO SUBPARTICIONAMENTO

Tabela Correio

Particionamento por correspondências enviadas e subparticionado por ID do malote.

- O particionamento por LIST para verificar se está enviado ou não:
  Se 0, **“cartas a enviar“**; se 1, **"cartas enviadas".**
- Então as partições foram subparticionadas em mais 512 subpartições e chegando no total de 1024 subpartições. Como subparticionamos pelo ID do malote, e estamos buscando todas as cartas a enviar do malote 1, o MySQL já sabe que ele terá que buscar o **“cartas a enviar 1”.**

IMPORTANTE: A dificuldade de aplicação utilizando Range Columns e Range, acontece seguidamente com a definição de Max Value. Para evitar isso, é comum não definir o **“Max Value”** e sim o **“Values Lass Than”.**

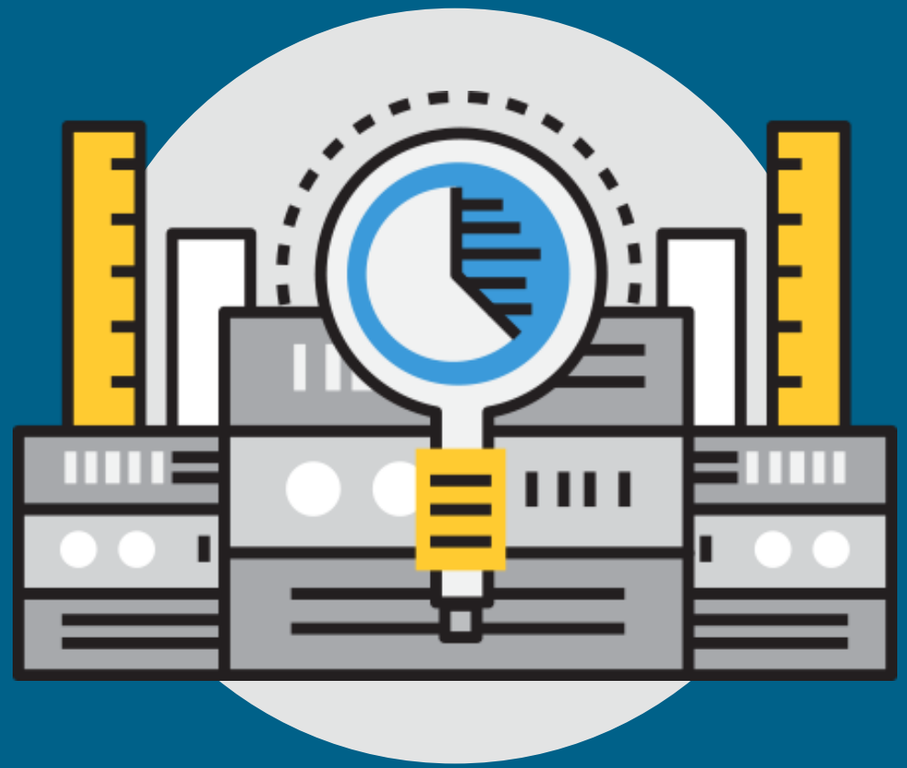
Subparticionamento

```
CREATE TABLE `Cartas` (
     `id` int(11) NOT NULL AUTO_INCREMENT,
     `id_malote` int(11) NOT NULL,
     `endereco` varchar(100) NOT NULL,
     `enviado` TINYINY(1) NOT NULL DEFAULT 0,
     PRIMARY KEY (`id`,`enviado`,`id malote`),
) ENGINE=InnoDB

PARTITION BY LIST (enviado)
SUBPARTITION BY HASH (id_malote)
SUBPARTITIONS 512
(
     PARTITION Cartas_a_enviar VALUES IN (0),
     PARTITION Cartas_enviadas VALUES IN (1)
);
```

Otimização de consultas

O que fazer para sua consulta ser mais rápida?



Índices x Consultas

Os índices devem corresponder a maioria das consultas executadas em um sistema e vice-versa.

Se um select é executado uma vez por dia, por data de matrícula, talvez não seja um candidato interessante para ser indexado. Mas, se todos os usuários que entram precisam ser identificados pelo "id do usuário", então é melhor que você tenha um índice para este id.

O que deve ser indexado?

Campos utilizados frequentemente em buscas, pois devemos criar índices para o que usamos com mais frequência. Veja alguns exemplos na próxima página.

Cláusula WHERE

SELECT * FROM Clientes WHERE **cidade** = 'Porto Alegre'

Colunas junção em JOINS

SELECT p.numero _pedido, c.nome FROM Pedidos AS.p

JOIN Cliente AS.c

ON p.**cpf_cliente**= c.**cpf**

WHERE p.estado = 'fechado'

Colunas usadas em valores mín ou máx

SELECT MAX(**valor**) FROM Dividas

Colunas utilizadas para GROUP BY ou ORDER BY

SELECT MAX (valor) FROM Dividas

GROUP BY **id_cliente**

ORDER BY **data_criação**

Algumas dicas extras

- Procure sempre usar o menor tipo de dados possível. Dando preferências àqueles dados que utilizem menos espaço para uma coluna. O objetivo é gastar menos espaço em disco e memória, gerando índices menores. Conseguimos assim carregar mais dados para a tabela (em cache) e gerar mais performance.

- O comando **SELECT * FROM 'tabela' PROCEDURE ANALYSE** analisa a tabela em busca de campos que estão subaproveitados e podem ser enxugados, otimizando o espaço e ganhando em performance.

- Crie índices parciais para texto, utilizando no índice um número de caracteres necessários a criação de uma cardinalidade razoavelmente alta. Em outras palavras, se o campo possuir 150 caracteres, indexar o campo por completo será muito custoso para a performance da sua base de dados.

- **Índices compostos**
  I. CPF
  II. data_nascimento
  III. valor-divida

  Vale a pena indexar para consultas que comecem pelo primeiro campo e busque os outros campos, em ordem. Entretanto, não é proveitoso para índices que começam pelo meio.

- Na dúvida sobre se vale a pena indexar ou não, pense no seu índice como um sumário de um livro. Onde apenas as páginas mais procuradas (capítulos) são apresentadas. Em outras palavras, em um livro você não irá indexar cada palavra, certo? Caso contrário, o sumário irá perder o seu sentido.

Não é bom indexar

Colunas só de exibição.

Exemplo: coluna nome de funcionários.

Quando você não costuma realizar buscas pelo nome de funcionário, então você não deve criar índices para nome de funcionários, simples.

Colunas com pouca variação de valor

Exemplo: Indexação por gênero.

Não é um bom candidato a indexação, pois há poucos dados e é preciso o contrário. Quanto mais valores possíveis, maior a sua cardinalidade e, portanto, maior o aproveitamento da indexação.

Cardinalidade

Trata-se do número de valores possíveis que esse índice pode indexar, em relação ao número de linhas desta tabela. Sendo o índice de cardinalidade mais alto, a chave primária o qual diz exatamente onde está uma linha.

Particionamento x Consultas

- Ao utilizar particionamento, é importante que todas as consultas especifiquem o valor da coluna utilizada na expressão de particionamento.

- Caso não haja especificação, o MySQL terá que abrir todas as partições, realizando um “full scan” e deixando o desempenho sobrecarregado e lento.

JOINS

Execute somente se precisar e evite JOINS desnecessários, pois trata-se de um produto cartesiano muito custoso. Entretanto, caso o Joins seja realmente necessário, crie índices para facilitar a junção, mas não exagere!

IMPORTANTE: se possível, evite indexar buscas cujo o comando EXPLAIN mostre **"using temporary"** e/ou **"using filesort"**. Essas são operações muito custosas e muito demoradas.

EXPLAIN

Um ótimo comando do MySQL que permite explicar como será determinada QUERY. Uma de suas funcionalidades é exibir os índices possíveis e quais já são utilizados

EXPLAIN SELECT * FROM Clientes WHERE cidade = Porto Alegre

EXPLAIN PARTITIONS

Para otimização do seu banco de dados, utilize o comando EXPLAIN PARTITIONS para saber se suas consultas utilizam apenas as partições necessárias.

EXPLAIN PARTITIONS SELECT * FROM Clientes WHERE cidade = 'Porto Alegre'

Se a busca envolve muitas partições, revise o seu SELECT para que especifique melhor os valores possíveis das colunas de particionamento.

targettrust
treinamento e tecnologia

Be code

KINGHOST

Infraestrutura

Boas práticas que deram certo na Becode e KINGHOST

Servidor BD x Servidor Aplicação

Boa prática: separar o servidor de banco de dados, do servidor de aplicação.

Há muitos painéis onde são instalados e-mail, banco de dados, a aplicação e php rodando tudo no mesmo servidor. Isso é altamente não recomendado. Separe os servidores.

O que utilizar nesse servidor de dados?

SSD

- Processo sendo efetuado desde 2010 na KINGHOST, para substituir os discos.

- Atualmente 100% dos servidores de banco de dados estão usando discos SSD.

Essa substituição facilitou backup, aumentou SLA uptime dos servidores e ganho de performance. Em 2010, a KINGHOST levava cerca de 18 minutos para subir um banco de dados, hoje com os discos SSD leva cerca de 1 a 2 minutos e o servidor volta para produção mais rápido e com maior agilidade.

Rede Dedicada

É recomendado que você separe os servidores, mas você terá que ligar os dois servidores (banco de dados e aplicação)

- Dedicada para conexão com banco de dados
- Resolução de DNS para rede interna

Monitore suas consultas lentas

Monitores o tempo médio que levou para executar as consultas, os registros analisados, quantidade de execuções e as consultas em si.

Controle queries com execução acima de 2G e sem índice, pois você poderá decidir se elas merecem ou não ter índices criados.

Perguntas frequentes

1: No caso do comando select, quando defino uma quantidade dentro do índice isso irá valer para todo o texto do campo ou pegará apenas os primeiros caracteres?

Sim. Apenas os primeiros caracteres, mas se você buscar o restante ele irá encontrar o campo inteiro.

2: Se eu já tenho uma tabela com 100 mil registros e quero particionar ela por data de admissão, supomos que eu divida. o MySQL irá reestruturar a tabela ou o particionamento só será usado para novos inserts?

Ele irá reestruturar a tabela. Ele fará um "alter table" e criará uma nova tabela particionada, distribuindo os registros existentes. Clone a tabela e faça uma cópia, não edite dados em produção, assim você previne perdas.

Pronto para otimizar o seu Banco de Dados MySQL!?

Esperamos que através deste ebook você ganhe uma melhor visão das melhores práticas para o desenvolvimento de aplicações que consomem dados de um servidor MySQL. Como você viu, existe uma série de ações que podem otimizar o desempenho geral dos seu banco de dados.

Para melhor compreensão e aproveitamento do conteúdo apresentado neste e-book é importante assistir os vídeos apresentados no decorrer do texto que expõe exemplos de otimizações na prática. Caso queira, você também pode acompanhar **o webinar completo sobre performance e otimização de banco de dados mysql.**

Autores

Jeronimo Fagundes

Software Engineer
na KingHost

Nemora Dornelles

Instrutora na Becode e
TargetTrust

targettrust
treinamento e tecnologia

Reconhecida como a melhor escola de TI do sul do Brasil, a TargetTrust acumula mais de 20 anos de experiência em treinamentos para profissionais e empresas do mercado de TI e negócios. Veja mais sobre a TargetTrust e nossos treinamentos presenciais e In Company, acesse o nosso site.

www.targettrust.com.br

A Becode nasceu para ser a sua fonte segura de informações e aprendizado em TI e negócios na Web. E, para atingir esse objetivo, adotamos uma metodologia de ensino única em nossos treinamentos. Desta forma, proporcionando a mesma qualidade e garantia de ensino presente em cursos presenciais, agora também no ambiente online! Conheça mais sobre a Becode e nossa metodologia.

www.becode.com.br

KINGHOST

Segurança, suporte técnico especializado e servidores 100% brasileiros. Essas são as três principais características pelas quais a KingHost é conhecida e recomendada pelos clientes dos mais de 300 mil sites que hospedamos. Conheça mais sobre a nossa história e serviços aqui.

www.kinghost.com.br