Acelerando suas queries com o particionamento do MySQL

24/05/2015

6 minute read

Quando você tem uma tabela com muitos registros no MySQL, as buscas podem se tornar extremamente lentas. Uma forma de otimizar a velocidade de suas buscas pode ser particionar a tabela.

Quando uma tabela é particionada, é como se você tivesse várias tabelinhas menores que, juntas, compõem a tabela completa.

Esta divisão é feita de acordo com algum critério lógico, de forma que quando você realize uma busca, o MySQL busque em apenas uma tabelinha, e não no universo todo de dados. Você não tem mais um grande universo de dados, mas vários "universinhos". Como seu universo de busca se torna menor, a busca fica mais rápida.

O critério lógico de particionamento sempre é definido com base nos valores de uma ou mais colunas predeterminadas. Dependendo dos valores que essas colunas assumirem, uma row é guardada em uma ou em outra partição. Assim, quando formos buscar um conjunto de dados via SELECT, dependendo dos valores especificados para aquelas colunas na cláusula WHERE, sabemos exatamente em quais partições os dados se encontram, e assim evitamos a busca nas demais partições.

Há diversos tipos de particionamento no MySQL. Cada um atende a um tipo de critério lógico de particionamento.

##Particionamento por RANGE

No particionamento por RANGE, o MySQL decide quais rows vão em uma partição de acordo com um intervalo de valores de uma coluna (ou expressão baseada em colunas). Na seguinte tabela:

```
CREATE TABLE `Funcionarios` (
    `cpf` VARCHAR(14) NOT NULL,
    `nome` VARCHAR(255) NOT NULL,
    `admissao` DATE NOT NULL
    PRIMARY KEY (`cpf`)
)
PARTITION BY RANGE(MONTH(admissao)) (
    PARTITION primeiro_trimestre VALUES LESS THAN (4),
    PARTITION segundo_trimestre VALUES LESS THAN (7),
```

```
    PARTITION terceito_trimestre VALUES LESS THAN (10),
    PARTITION quarto_trimestre VALUES LESS THAN MAXVALUE
);
```

Definimos 4 partições baseadas no mês da asmissão:

A partição primeiro_trimestre vai conter todos os registros de funcionários cuja admissão ocorreu nos meses 1, 2 ou 3 (janeiro, fevereiro ou março), ou seja, segundo a definição, todos os meses menores que 4. A partição segundo_trimestre conterá todos os registros de funcionários cujo mês de admissão é maior ou igual a 4 e menor que 7, ou seja, os meses de abril, maio e junho. A partição terceiro_trimestre conterá os registros de funcionários admitidos nos meses 7, 8 e 9, ou seja, julho, agosto e setembro. A partição quarto_trimestre conterá os registros dos demais funcionários. Assim, caso o seguinte select fosse feito:

```
SELECT * FROM Funcionarios WHERE admissao = '2015-03-15';
```

o MySQL saberia automaticamente que teria que procurar apenas na partição primeiro_trimestre, e não consideraria as outras partições na busca. O universo de busca fica muito menor, e a query mais rápida.

É importante notar que na expressão PARTITION BY (expr), expr deve ser uma expressão que retorne necessariamente um valor inteiro.

##Particionamento por LIST

O particionamento por LIST é bem parecido com o RANGE. Todavia, em vez de especificarmos um intervalo de valores, vamos especificar um conjunto discreto de valores, fixos, predeterminados.

```
CREATE TABLE `Funcionarios` (
    `cpf` VARCHAR(14) NOT NULL,
    `nome` VARCHAR(255) NOT NULL,
    `filial` INT NOT NULL
    PRIMARY KEY (`cpf`)
)
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),
);
```

Neste exemplo, se buscássemos todos os funcionários da filial número 5, o MySQL buscaria apenas na partição regiao_leste, que contém os funcionários das filiais 4, 5 e 6.

##Particionamento por RANGE COLUMNS

É igual ao particionamento com RANGE, mas a expressão de particionamento pode ser de outros tipos que não um número inteiro, a saber: DATE, DATETIME, CHAR, VARCHAR, BINARY e VARBINARY. Além disso, pode-se usar tuplas de colunas em vez de uma só coluna.

```
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)
);
```

Neste exemplo, se buscássemos por uma row em que a = 4 e c = '2015-10-01', o MySQL buscaria na partição p2. Já uma row em que a = 4 e c = '2015-11-20' estaria na partição p3.

##Particionamento por LIST COLUMNS

Assim como o RANGE COLUMNS é uma extensão do RANGE, o LIST COLUMNS é uma extensão do LIST. Ele também permite que a expressão de particionamento seja de outros tipos, e também permite tuplas.

```
CREATE TABLE `Funcionarios` (
    `cpf` VARCHAR(14) NOT NULL,
    `estado`VARCHAR(2) NOT NULL DEFAULT 'RS',
    PRIMARY KEY (`cpf`)
)
PARTITION BY LIST COLUMNS (estado) (
    PARTITION regiao_sul VALUES IN ('RS', 'SC, 'PR'),
    PARTITION regiao_sudeste VALUES IN ('SP', 'RG', 'MG', 'ES'),
    PARTITION regiao_centro_oeste VALUES IN ('MT', 'MS', 'GO', '
    PARTITION regiao_norte VALUES IN ('AC', 'AM', 'RO', 'RR', 'P
    PARTITION regiao_nordeste VALUES IN ('MA', 'PI', 'CE', 'RN',
);
```

##Particionamento por HASH ou por LINEAR HASH

O particionamento por HASH é um dos mais simples, e de uso mais comum. A sua expressão de particionamento deve ser um valor inteiro. Neste tipo de particionamento, você deve especificar não só a expressão de particionamento, mas também o número de partições a utilizar. O MySQL vai nomear essas partições automaticamente, e vai escolher a melhor partição para sua row de acordo com o módulo da expressão de particionamento pelo número de partições. Esse é o tipo de particionamento recomendado se você quer ter uma distribuição semelhante de rows entre as 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 10;
```

Neste exemplo, criamos 10 partições por id de cliente. Isso significa, por exemplo, que todos os pedidos do cliente com id 7 vão residir na partição 7, pois 7 % 10 = 7 (sete módulo dez é igual a sete). Já os pedidos do cliente 22 estarão todos na partição 2, pois 22 % 10 = 2.

O número máximo de partições por tabela que o MySQL permite é 1024.

O particionamento por LINEAR HASH é muito semelhante ao por HASH. Todavia, em vez do módulo, o MySQL usa outra fórmula baseada em potências de 2.

##Particionamento por KEY ou por LINEAR Key

Muito parecida com a partição por HASH. Todavia, no particionamento por KEY, o MySQL server é que vai escolher o melhor algoritmo para o cálculo da partição resultante.

A expressão de particionamento pode ser zero ou mais colunas, conforme segue:

Se não é especificada nenhuma coluna, o MySQL vai usar a chave primária se houver; se não houver chave primária, vai usar uma chave única que houver. Se houver colunas especificadas, elas precisam fazer parte da chave primária ou da chave única. LINEAR KEY é semelhante à KEY, usando cálculo de potências de 2 (assim como LINEAR HASH).

Um cuidado deve ser tomado: se você efetuar uma busca (SELECT) e não especificar na cláusula WHERE o valor da coluna de particionamento, o MySQL não vai saber em qual partição buscar, e vai acabar varrendo todas as partições para efetuar sua busca. Isso será mais lento que buscar em uma tabela não-particionada. Assim sendo, o particionamento é uma poderosa ferramenta, mas suas queries terão de ser adaptadas para aproveitar esse particionamento.

Quer saber mais sobre particionamento no MySQL? Consulte a documentação oficial do MySQL sobre particionamento.
