

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS DE CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

VITOR ANTONIO APOLINÁRIO
WILLIAN BORDIGNON GENERO

OTIMIZAÇÃO DE CONSULTAS

DISCIPLINA DE BANCO DE DADOS II

CHAPECÓ

2019

Sumário

Observação	3
Métodos utilizados - Slide 5	3
PostgreSQL	5
Otimizadas	5
Primeira consulta (indexada) - Slide 6	5
Segunda consulta (não indexada) - Slide 7	6
Terceira consulta (Subquery) - Slide 8	7
Não otimizadas	8
Primeira consulta (indexada) - Slide 9	8
Segunda consulta (não indexada) - Slide 10	8
Terceira consulta (Subquery) - Slide 11	9
MariaDB	10
Otimizadas	10
Primeira consulta - Slide 13	10
Segunda consulta - Slide 14	10
Terceira consulta - Slide 15	11
Não otimizadas	12
Primeira consulta - Slide 16	12
Segunda consulta - Slide 17	13
Terceira consulta - Slide 18	13
Resultados	14

Observação

Para compreender melhor as consultas leia juntamente com os slides que fornece informações que não foram aqui adicionadas para não estender excessivamente o tamanho do arquivo além do permitido. Em cada consulta de cada SGBD será identificado o slide correspondente.

Métodos utilizados - Slide 5

Merge Join: considerando duas tabelas (A e B) que tenham um identificador compartilhado entre ambas, pode ser utilizado o merge join da seguinte maneira, as duas tabelas são ordenadas, depois é lido uma tupla na tabela A e todas as tuplas que tenham identificador menor ou igual na tabela B. Supondo que seja lido em A uma tupla identificada por 2 em B será lido até encontrar um ID maior que 2. Portanto se B tiver tupla com ID 1 essa não terá junção com ninguém em A, se tiver 3 tuplas com ID 2 todas serão juntas com o ID 2 de A. Após ler todas tuplas iguais em B será lido a próxima tupla de A.

Hash Join: considerando duas tabelas (A e B) que tenham um identificador compartilhado entre ambas, pode ser utilizado o hash join da seguinte maneira, será realizado uma função hash (pode ser mod) sobre o ID da tabela A e o resultado salvo em uma partição. Lembrando que podem ter diferentes IDs em um mesmo bloco do bucket. Montada a partição da tabela A será lido a tabela B e feito o hash sobre cada tupla, esse resultado será verificado no bloco correspondente do bucket da tabela A. Os valores iguais serão salvos no resultado do join.

Aggregate: é o agrupamento com base em algum atributo. Portanto, se em nossa tabela diversos repetições de algum atributo podemos juntar todos os repetidos em um só e apresentar quantas repetições cada um deles tem, por exemplo.

Sort: é a ordenação dos dados de uma tabela que segue algum critério. Considerando que temos uma tabela de alunos com matrícula e nome, se realizarmos o sort utilizando matrícula, será possível ler todas as tuplas com matrícula em ordem crescente.

Materialize: salva os dados na memória como são lidos, para utilizá-los nos passos seguintes.

Gather: é um momento do plano onde é possível executar operações paralelas, consultas e até mesmo um plano “filho”, com o objetivo de otimizar a consulta.

PostgreSQL

Otimizadas

Primeira consulta (indexada) - Slide 6

```
1  -- films that have more than two genres
2  select
3      m.id,
4      m.name,
5      count(g.genre)
6  from
7      movies m
8  join movies_genres g on
9      g.movie_id = m.id
10 group by
11     m.id
12 having
13     count(g.genre) > 2;
```

A primeira consulta consulta busca encontrar os filmes que possuem mais de um gênero e isso é feito simplesmente usando um **merge join** entre a primary key dos filmes e o índice da tabela de gêneros. Após é realizado **aggregation** para mostrar quantos gêneros cada filme tem.

Segunda consulta (não indexada) - Slide 7

```
1  -- films that have more than two genres
2  select
3      m.id,
4      m.name,
5      count(g.genre)
6  from
7      movies m
8  join movies_genres g on
9      g.movie_id = m.id
10 group by
11     m.id
12 having
13     count(g.genre) > 2;
```

Como foram removidos os índices de “movies_genres” que facilitavam o nosso join, o Postgresql teve que encontrar uma maneira melhor de encontrar os “matchs”, escaneando essa tabela, e usando o método **sort** seguido do **materialize**, para tornar possível a utilização do merge inner join, e depois um **aggregate**.

Terceira consulta (Subquery) - Slide 8

```
1  --how many percents a genre represents
2  select
3      gnr,
4      cnt * 100 / (
5          select
6              count(*)
7              from movies_genres g ) as perc
8  from
9      (select
10         g.genre as gnr,
11         count(*) as cnt
12      from
13         movies_genres g
14      group by
15         g.genre) z;
```

- (1) Para contar o total de registros na tabela movies_genres, é feito uma **agregação**, e iniciadas operações **paralelas**, que possivelmente ajudam no processo de contagem, após isso é gerado um novo **agrupamento**.
- (2) Para contar quantas vezes cada gênero é referenciado, é **escaneada** a tabela “movies_genres”, e **agrupados** os registros, iniciando operações paralelas que possivelmente vão ajudar a contabilizar os registros, que são posteriormente **ordenados** e novamente **agregados**.
- (3) Tendo o número de ocorrências de cada gênero (na subquery), e o total de ocorrências, o plano trata de juntar os dados através de um **subquery scan**.

Não otimizadas

Primeira consulta (indexada) - Slide 9

```
1 -- films that have more than two genres
2 select
3     m.id,
4     m.name,
5     count(g.genre)
6 from
7     movies m
8 join movies_genres g on
9     g.movie_id = m.id
10 group by
11     m.id
12 having
13     count(g.genre) > 2;
```

```
6 from
7     movies m
8 join movies_genres g on
9     g.movie_id = m.id
10 join (
11     select
12         distinct genre
13     from
14         movies_genres r) gd on
15     gd.genre = g.genre
```

O join adicionado à consulta não surte efeito no resultado da mesma. Em termos de desempenho, impacta bastante, porque:

- (1) Para fazer o join “desnecessário” (em vermelho), a tabela movie_genres interna (do subselect) tem que ser **escaneada** e **agregada**, e é vítima de um **hash**, para identificar valores **distintos** do atributo genre, e depois fazer um **hash inner join** com a tabela “movies_genres” externa para a junção, seguido de um **sort** e **materialize**, para permitir um merge inner join indexado com a tabela “movies” e depois um agrupamento final.

Segunda consulta (não indexada) - Slide 10

```
1 -- films that have more than two genres
2 select
3     m.id,
4     m.name,
5     count(g.genre)
6 from
7     movies m
8 join movies_genres g on
9     g.movie_id = m.id
10 group by
11     m.id
12 having
13     count(g.genre) > 2;
```


```
6 from
7     movies m
8 join movies_genres g on
9     g.movie_id = m.id
10 join (
11     select
12         distinct genre
13     from
14         movies_genres r) gd on
15     gd.genre = g.genre
```


Embora esta consulta seja igual à anterior porém com os campos indexados, o resultado do plano de execução foi exatamente o mesmo. Supomos que é porque o otimizador não consegue utilizar-se dos índices para tratar o **distinct**.

Terceira consulta (Subquery) - Slide 11

```
1  --how many percents a genre represents
2  select
3      gnr,
4      cnt * 100 / (
5          select
6              count(*)
7          from movies_genres g ) as perc
8  from
9      (select
10         g.genre as gnr,
11         count(*) as cnt
12      from
13         movies_genres g
14      group by
15         g.genre) z;
```

```
4      cnt * 100 / (
5          select
6              count(*)
7          from
8              movies_genres g join movies m on m.id = g.movie_id
9              ) as perc
10 from
```



Afim de piorar o desempenho da consulta sem alterar o resultado, foi adicionado o join em vermelho acima. O plano de execução gerado é semelhante ao da “subconsulta otimizada”:

- (1) Para contar o total de registros na tabela `movies_genres`, agora é necessário antes fazer um **hash inner join** com a tabela `movies`, o que torna essa consulta mais custosa do que a otimizada abordada previamente.
- (2) Para contar quantas vezes cada gênero é referenciado, é **escaneada** a tabela “`movies_genres`”, e **agrupados** os registros, iniciando operações paralelas que possivelmente vão ajudar a contabilizar os registros, que são posteriormente **ordenados** e novamente **agregados**.
- (3) Tendo o número de ocorrências de cada gênero (na subquery), e o total de ocorrências, o plano trata de juntar os dados através de um **subquery scan**.

MariaDB

Otimizadas

Primeira consulta - Slide 13

```
1  -- films that have more than two genres
2  select
3      m.id,
4      m.name,
5      count(g.genre)
6  from
7      movies m
8  join movies_genres g on
9      g.movie_id = m.id
10 group by
11     m.id
12 having
13     count(g.genre) > 2;
```

Inicialmente é realizado um select simples sobre a tabela de filmes e depois é feito outro select agora sobre a tabela de gêneros e realizado o join utilizando o **índice** dos filmes na tabela de gêneros. O SGBD MariaDB não fornece informações de qual método é usado para realizar o join.

Segunda consulta - Slide 14

```
1  -- films that have more than two genres
2  select
3      m.id,
4      m.name,
5      count(g.genre)
6  from
7      movies m
8  join movies_genres g on
9      g.movie_id = m.id
10 group by
11     m.id
12 having
13     count(g.genre) > 2;
```

Não tendo índice para realizar a consulta é necessário realizar um select sobre simples sobre a tabela de gêneros e usar **filesort**, o resultado é salvo temporariamente e após é feito outro select simples, dessa vez sobre a tabela de filmes, tudo isso executa paralelamente enquanto é realizado o join entre as duas tabelas. O SGBD MariaDB não fornece informações de qual método é usado para realizar o join.

Terceira consulta - Slide 15

```

1  --how many percents a genre represents
2  select
3      gnr,
4      cnt * 100 / (
5          select
6              count(*)
7          from movies_genres g ) as perc
8  from
9      (select
10         g.genre as gnr,
11         count(*) as cnt
12      from
13         movies_genres g
14      group by
15         g.genre) z;

```

Primeiramente é realizado um select do tipo **subquery** na tabela gênero utilizando índices para acessar os dados e encontrar quantos filmes possui gênero. Também é realizado um select tipo **derivado** que irá contar quantos filmes cada gênero possui, é realizado um **filesort** para ordenar as tuplas e salva temporariamente essa nova tabela, que será usada por um select do tipo **primário** que irá realizar o cálculo da porcentagem.

Não otimizadas

Primeira consulta - Slide 16

```

1  -- films that have more than two genres
2  select
3      m.id,
4      m.name,
5      count(g.genre)
6  from
7      movies m
8  join movies_genres g on
9      g.movie_id = m.id
10 group by
11     m.id
12 having
13     count(g.genre) > 2;

```

```

6  from
7      movies m
8  join movies_genres g on
9      g.movie_id = m.id
10 join (
11     select
12         distinct genre
13     from
14         movies_genres r) gd on
15     gd.genre = g.genre

```

É realizado um select tipo **derivado** para encontrar os gêneros distintos usando índices e salvando temporariamente. Depois é realizado um select primário sobre a tabela temporária que juntamente faz o join entre eles. Outro select primário é feito o outro join entre filmes e gêneros usando índices novamente e por fim o select que mostra quais filmes tem mais de 2 gêneros.

Segunda consulta - Slide 17

```
1  -- films that have more than two genres
2  select
3      m.id,
4      m.name,
5      count(g.genre)
6  from
7      movies m
8  join movies_genres g on
9      g.movie_id = m.id
10 group by
11     m.id
12 having
13     count(g.genre) > 2;
```

```
6  from
7      movies m
8  join movies_genres g on
9      g.movie_id = m.id
10 join (
11     select
12         distinct genre
13     from
14         movies_genres r) gd on
15     gd.genre = g.genre
```

O funcionamento dessa consulta e da anterior são semelhantes porém não tem índices para ser usados, será realizado o select para encontrar os gêneros distintos e salvo temporariamente. É feito o join entre a tabela de gêneros e a tabela temporária por meio de um select primário. Após será realizado o join entre a tabela de filmes e a de gêneros também por meio do select primário. E após outro select que fornece os filmes com mais de 2 gêneros e utiliza o filesort para ordenar a tabela.


Terceira consulta - Slide 18

```

1  --how many percents a genre represents
2  select
3      gnr,
4      cnt * 100 / (
5          select
6              count(*)
7          from movies_genres g ) as perc
8  from
9      (select
10         g.genre as gnr,
11         count(*) as cnt
12      from
13         movies_genres g
14      group by
15         g.genre) z;

4      cnt * 100 / (
5      select
6          count(*)
7      from
8          movies_genres g join movies m on m.id = g.movie_id
9          ) as perc
10 from

```



Realizado um select subquery dentro de um select primário para obter a quantidade de filmes ligados com algum gênero e também um join entre as tabelas de filmes e gêneros, ambas usando índices.

Uma consulta primária é realizada sobre uma tabela temporária que surgiu de um select derivado para contar quantos filmes cada gênero tem, essa tabela usou índice e filesort para ordenar os dados e conseguir agrupá-los.

Essa tabela primária irá fornecer quanto cada gênero representa.

Resultados

	Postgres		Mariadb	
	Otimizada	N Otimizada	Otimizada	N Otimizada
Indexed join	303	751	1076	1495
Simple join	613	760	1428	2130
Subselect	79	258	427	1211

Legenda: o valor é em milisegundos.