UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

CAMPUS CHAPECÓ

BANCO DE DADOS II

CLEISSON VIEIRA RAIMUNDI

NICOLAS KOLLING RIBAS

PLANO DE CONSULTA COM O FUNCIONAMENTO DOS ALGORITMOS

CHAPECÓ

2019

Plano de Consultas

PostgresSQL - Join sem índice (otimizado):

```
SELECT first_name, last_name FROM actors JOIN roles on id = actor_id WHERE role = 'Han Solo';
```

```
Gather  (cost=1000.42..40070.88 rows=63 width=14) (actual time=34.167..137.523 rows=9 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Nested Loop  (cost=0.42..39064.58 rows=26 width=14) (actual time=26.149..132.962 rows=3 loops=3)
        ->  Parallel Seq Scan on roles  (cost=0.00..38845.07 rows=26 width=4) (actual time=26.115..132.920 rows=3 loops=3)
              Filter: ((role)::text = 'Han Solo'::text)
              Rows Removed by Filter: 1143986
        ->  Index Scan using actors_pkey on actors  (cost=0.42..8.44 rows=1 width=18) (actual time=0.012..0.012 rows=1 loops=9)
              Index Cond: (id = roles.actor_id)
Planning time: 0.303 ms
Execution time: 137.559 ms
(11 registros)
```

Index Scan: em uma varredura de índice (index scan), o método de acesso ao índice é responsável por regurgitar os IDs de todas as tuplas que foram informadas que correspondem às chaves de varredura (id = actor_id). O método de acesso não está envolvido na busca das tuplas da tabela pai do índice nem na determinação de passar o teste de qualificação de tempo da verificação ou outras condições.

Parallel Seq Scan: os blocos da tabela serão divididos entre os processos cooperantes. Blocos são entregues um de cada vez, para que o acesso à tabela permaneça sequencial. São selecionados as tuplas que atendem ao filtro role = ‘Han Solo da tabela roles’.

Nested Loop: loops aninhados, vai iterar dados de uma tabela sobre outra tabela de forma simples. Nesse caso existe uma “tabela” com dados criados a partir do Parallel Seq Scan e a tabela Actors, neste caso o condição de join está indexada, ou seja, será um index nested loop

Gather: quando o otimizador determina que a consulta paralela é a estratégia de execução mais rápida para uma consulta específica, ela criará um plano de consulta que inclui um nó Gather. Esse nó terá exatamente um plano filho, que é a parte do plano que será executada em paralelo. Se o nó Gather estiver no topo da árvore do plano, toda a consulta será executada em paralelo. Se estiver em outro lugar na árvore do plano, somente a parte do plano abaixo será executada em paralelo.

PostgresSQL - Join sem índice (não otimizado):

Query: SELECT first_name, last_name FROM actors join roles on id = actor_id where role = (select distinct role from roles where role = 'Han Solo');

```
Gather  (cost=40771.66..79888.80 rows=99 width=14) (actual time=149.439..236.560 rows=9 loops=1)
  Workers Planned: 2
  Params Evaluated: $1
  Workers Launched: 2
  InitPlan 1 (returns $1)
    ->  Unique  (cost=1000.00..39771.23 rows=64 width=9) (actual time=18.151..118.869 rows=1 loops=1)
          ->  Gather  (cost=1000.00..39771.23 rows=64 width=9) (actual time=18.149..118.920 rows=9 loops=1)
                Workers Planned: 2
                Workers Launched: 2
                ->  Parallel Seq Scan on roles roles_1  (cost=0.00..38764.83 rows=27 width=9) (actual time=24.031..115.928 rows=3 loops=3)
                      Filter: ((role)::text = 'Han Solo'::text)
                      Rows Removed by Filter: 1143986
  ->  Nested Loop  (cost=0.42..39107.66 rows=41 width=14) (actual time=51.069..112.148 rows=3 loops=3)
        ->  Parallel Seq Scan on roles  (cost=0.00..38764.83 rows=41 width=4) (actual time=51.058..112.125 rows=3 loops=3)
              Filter: ((role)::text = ($1)::text)
              Rows Removed by Filter: 1143986
        ->  Index Scan using actors_pkey on actors  (cost=0.42..8.36 rows=1 width=18) (actual time=0.006..0.006 rows=1 loops=9)
              Index Cond: (id = roles.actor_id)
Planning Time: 0.389 ms
Execution Time: 236.661 ms
```

Unique: vai realizar uma busca sequencial paralela na tabela roles onde o filtra vai ser role = ‘Han Solo’, retornando todas as tuplas que atendem a essa condição. Então o Unique vai eliminar as repetidas e retornar ‘Han Solo’

Index Scan: em uma varredura de índice (index scan), o método de acesso ao índice é responsável por regurgitar os IDs de todas as tuplas que foram informadas que correspondem às chaves de varredura (actors.id = roles.actor_id).

Parallel Seq Scan: realiza uma busca sequencial paralela (busca completa) na tabela role onde o filtro vai ser o retorno de Unique (‘han solo’), os blocos da tabela serão divididos entre os processos cooperantes. Blocos são entregues um de cada vez, para que o acesso à tabela permaneça sequencial.

Nested Loop: vai iterar sobre os dados retornados no index scan e parallel seq scan.

Gather: a consulta paralela criará um plano de consulta que inclui um nó Gather. Esse nó terá exatamente um plano filho, que é a parte do plano que será executada em paralelo. Se o nó Gather estiver no topo da árvore do plano, toda a consulta será executada em paralelo. Se estiver em outro lugar na árvore do plano, somente a parte do plano abaixo será executada em paralelo.

Firebird - Join sem índice (otimizado):

```
SELECT first_name, last_name FROM actors JOIN roles on id = actor_id WHERE role = 'Han Solo';
```

```
Select Expression
    -> Nested Loop Join (inner)
        -> Filter
            -> Table "ROLES" Full Scan
        -> Filter
            -> Table "ACTORS" Access By ID
                -> Bitmap
                    -> Index "RDB$PRIMARY1" Unique Scan
```

```
Current memory = 11182536
Delta memory = -25880
Max memory = 11281648
Elapsed time= 1.596 sec
Cpu = 1.470 sec
Buffers = 2048
Reads = 60296
Writes = 0
Fetches = 3672887
```

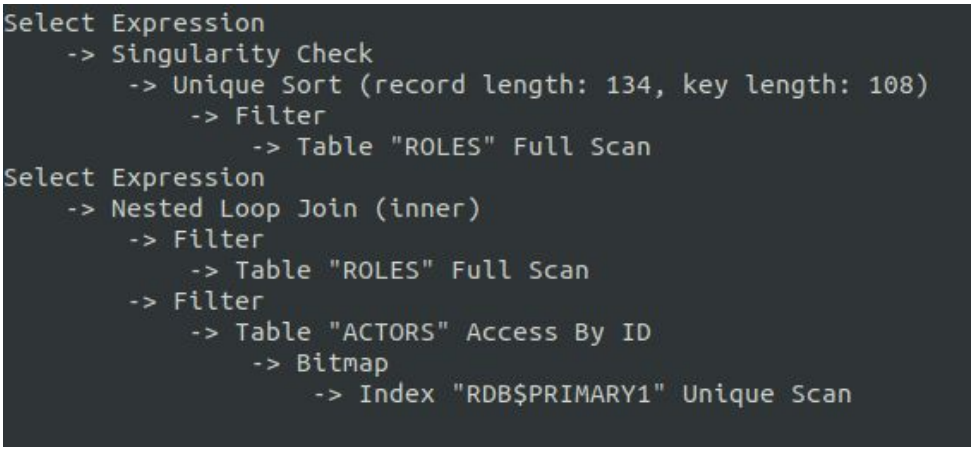
Filter -> Table "actors" Access By ID: vai realizar uma busca com index em actors e retornar as tuplas com name = 'Han Solo' formando um bitmap. E então vai filtrar acessando os IDs dessas tuplas do Bitmap.

Filter -> Table "roles" Full Scan: faz uma busca completa na tabela actors pegando os id = actor_id.

Nested Loop Join: vai interar em cima dos dados retornados de Filter (com os id = actor_id) e de Filter (com os id das tuplas com 'Han Solo').

Firebird - Join sem índice (não otimizado):

Query: SELECT first_name, last_name FROM actors JOIN roles on id = actor_id where role = (select distinct role from roles where role = 'Han Solo');



Elapsed time= 2.115 sec

Singularity Check: vai realizar um scan completo na tabela roles filtrando os que role = ‘Han Solo’. Após retornar todos as tuplas com ‘Han Solo’ vai aplicar um Unique Sort que elimina os repetidos, retornando apenas um ‘Han Solo’.

Filter -> Table “actors” Access By ID: vai realizar uma busca com index em actors e retornar as tuplas com name = ‘Han Solo’ (que é o retorno de Singularity Check) formando um bitmap. E então vai filtrar acessando os IDs dessas tuplas do Bitmap.

Filter -> Table “roles” Full Scan: faz uma busca completa na tabela actors pegando os id = actor_id.

Nested Loop Join: vai interar em cima dos dados retornados de Filter (com os id = actor_id) e de Filter (com os id das tuplas com ‘Han Solo’ (que é o retorno de Singularity Check)).

PostgresSQL - Join com índice (otimizado):

```
SELECT * FROM movies_genres JOIN movies on id = movie_id WHERE name = 'Titanic';
```

```
Nested Loop  (cost=0.84..16.92 rows=1 width=7) (actual time=0.032..0.066 rows=10 loops=1)
  ->  Index Scan using movies_name on movies  (cost=0.42..8.44 rows=1 width=4) (actual time=0.020..0.023 rows=8 loops=1)
        Index Cond: ((name)::text = 'Titanic'::text)
  ->  Index Scan using movies_genres_movie_id on movies_genres  (cost=0.42..8.46 rows=2 width=11) (actual time=0.003..0.004 rows=1 loops=8)
        Index Cond: (movie_id = movies.id)
Planning Time: 0.296 ms
Execution Time: 0.092 ms
```

PostgresSQL - Join com índice (não otimizado):

Query: EXPLAIN ANALYZE SELECT genre FROM (select * from movies, movies_genres) as tabela WHERE name = ‘Titanic’ and tabela.id = tabela.movie_id ;

```
Nested Loop  (cost=0.84..16.92 rows=1 width=7) (actual time=0.039..0.077 rows=10 loops=1)
  ->  Index Scan using movies_name on movies  (cost=0.42..8.44 rows=1 width=4) (actual time=0.023..0.026 rows=8 loops=1)
        Index Cond: ((name)::text = 'Titanic'::text)
  ->  Index Scan using movies_genres_movie_id on movies_genres  (cost=0.42..8.46 rows=2 width=11) (actual time=0.004..0.005 rows=1 loops=8)
        Index Cond: (movie_id = movies.id)
Planning Time: 0.330 ms
Execution Time: 0.106 ms
```

Index Scan: vai realizar uma busca de index (b-tree) na tabela movies_genres onde move_id = movies.id.

Index Scan: vai fazer uma busca por Titanic na b-tree, onde vai buscar onde começa por T e trazer todas as tuplas com Titanic.

Nested Loop: loops aninhados. Nesse caso existe uma “tabela” com dados criados a partir do Index Scan que vai ser os id de movies_genres e outra com Index Scan que vai ser os id de movies onde name = ‘Titanic’.

Firebird - Join com índice (otimizado):

```
SELECT * FROM movies_genres JOIN movies on id = movie_id WHERE name = 'Titanic';
```

```
Select Expression
    -> Nested Loop Join (inner)
        -> Table "MOVIES_GENRES" Full Scan
        -> Filter
            -> Table "MOVIES" Access By ID
                -> Bitmap
                    -> Index "RDB$PRIMARY4" Unique Scan
```

```
Current memory = 11199456
Delta memory = 9328
Max memory = 11281648
Elapsed time= 1.085 sec
Cpu = 1.030 sec
Buffers = 2048
Reads = 14264
Writes = 0
Fetches = 2404514
```

Table “movies” Access By ID: vai realizar uma busca com index em movies e retornar as tuplas com name = ‘Titanic’ formando um bitmap. E então vai filtrar acessando os IDs dessas tuplas do Bitmap.

Table “movies_genres” Full Scan: faz uma busca completa na tabela movies_genres pegando os id = movie_id.

Nested Loop Join: vai interar em cima dos dados retornados de Full scan com os do Filter.

Firebird - Join com índice (não otimizado):

Query: EXPLAIN ANALYZE **SELECT** genre **FROM** (select * from movies, movies_genres) as tabela **WHERE** name = 'Titanic' and tabela.id = tabela.movie_id ;

```
Select Expression
    -> Nested Loop Join (inner)
        -> Table "MOVIES_GENRES" as "TABELA MOVIES_GENRES" Full Scan
        -> Filter
            -> Table "MOVIES" as "TABELA MOVIES" Access By ID
                -> Bitmap
                    -> Index "RDB$PRIMARY2" Unique Scan
```

Elapsed time= 0.813 sec

Filter: vai realizar uma busca com index em movies e retornar as tuplas com name = 'Titanic' formando um bitmap. E então vai filtrar acessando os IDs dessas tuplas do Bitmap.

Table "movies_genres" Full Scan: vai realizar um scan completo na tabela movies_genres verificando todos os id = movie_id.

Nested Loop Join: vai interar em cima dos dados retornados de Full scan com os do Filter formando uma nova tabela.

PostgresSQL - Com subconsultas (otimizado):

```
SELECT directors.first_name, directors.last_name FROM directors
join movies_directors on directors.id = movies_directors.director_id
join movies on movies.id = movies_directors.movie_id
WHERE movies.rank = (SELECT max(rank) from movies);
```

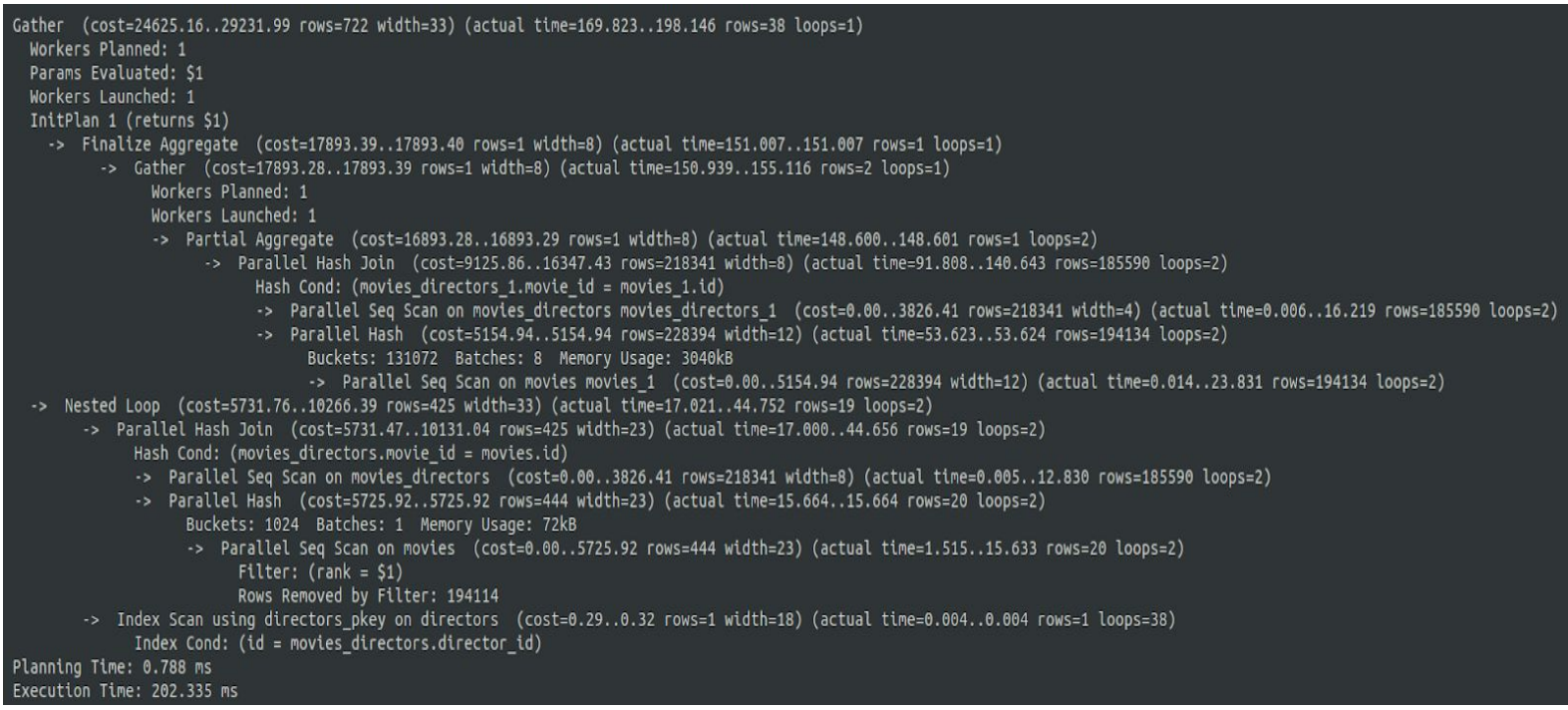
```
Gather  (cost=13457.80..18064.63 rows=722 width=33) (actual time=54.151..80.594 rows=38 loops=1)
  Workers Planned: 1
  Params Evaluated: $1
  Workers Launched: 1
  InitPlan 1 (returns $1)
    ->  Finalize Aggregate  (cost=6726.03..6726.04 rows=1 width=8) (actual time=34.468..34.468 rows=1 loops=1)
          ->  Gather  (cost=6725.92..6726.03 rows=1 width=8) (actual time=34.406..34.506 rows=2 loops=1)
                Workers Planned: 1
                Workers Launched: 1
                ->  Partial Aggregate  (cost=5725.92..5725.93 rows=1 width=8) (actual time=32.359..32.360 rows=1 loops=2)
                      ->  Parallel Seq Scan on movies movies_1  (cost=0.00..5154.94 rows=228394 width=8) (actual time=0.011..16.274 rows=194134 loops=2)
  ->  Nested Loop  (cost=5731.76..10266.39 rows=425 width=33) (actual time=17.214..42.360 rows=19 loops=2)
        ->  Parallel Hash Join  (cost=5731.47..10131.04 rows=425 width=23) (actual time=17.190..42.268 rows=19 loops=2)
              Hash Cond: (movies_directors.movie_id = movies.id)
              ->  Parallel Seq Scan on movies_directors  (cost=0.00..3826.41 rows=218341 width=8) (actual time=0.006..11.681 rows=185590 loops=2)
              ->  Parallel Hash  (cost=5725.92..5725.92 rows=444 width=23) (actual time=15.720..15.720 rows=20 loops=2)
                    Buckets: 1024  Batches: 1  Memory Usage: 72kB
                    ->  Parallel Seq Scan on movies  (cost=0.00..5725.92 rows=444 width=23) (actual time=0.950..15.684 rows=20 loops=2)
                          Filter: (rank = $1)
                          Rows Removed by Filter: 194114
        ->  Index Scan using directors_pkey on directors  (cost=0.29..0.32 rows=1 width=18) (actual time=0.004..0.004 rows=1 loops=38)
              Index Cond: (id = movies_directors.director_id)
Planning Time: 0.746 ms
Execution Time: 80.700 ms
```

O select mais interno é executado, sendo feito um scan completo na tabela movies para encontrar o maior valor de rank, esse valor é retornado para ser usado no select mais externo.

O select mais externo realiza um scan completo na tabela movies para filtrar as tuplas que atendem a igualdade ao maior rank, com o resultado do scan é feito uma tabela hash sobre o atributo de join, então cada tupla da tabela movies_directors é juntada com as tuplas do seu bucket na hash, com o resultado do join hash é feito um index nested loop com os índices da tabela movies_directors;

PostgresSQL - Com subconsultas (não otimizado):

SELECT directors.first_name, directors.last_name, movies.name FROM directors join movies_directors on directors.id = movies_directors.director_id join movies on movies.id = movies_directors.movie_id WHERE movies.rank = (SELECT max(rank) from movies JOIN movies_directors on id = movie_id);



O select mais interno é executado, sendo feito um scan completo na tabela movies para gerar uma hash que será usada para a junção com a tabela movies_directors, a partir dai é feito outro scan completo para encontrar o maior valor de rank, esse valor é retornado para ser usado no select mais externo. Ou seja o join desnecessário não foi retirado.

O select mais externo realiza um scan completo na tabela movies para filtrar as tuplas que atendem a igualdade ao maior rank, com o resultado do scan é feito uma tabela hash sobre o atributo de join, então cada tupla da tabela movies_directors é juntada com as tuplas do seu bucket na hash, com o resultado do join hash é feito um index nested loop com os índices da tabela movies_directors;

Firebird - Com subconsultas (otimizado):

```
SELECT directors.first_name, directors.last_name FROM directors
join movies_directors on directors.id = movies_directors.director_id
join movies on movies.id = movies_directors.movie_id
WHERE movies.rank = (SELECT max(rank) from movies);
```

```
Select Expression
    -> Singularity Check
        -> Aggregate
            -> Table "MOVIES" Full Scan
Select Expression
    -> Nested Loop Join (inner)
        -> Table "MOVIES_DIRECTORS" Full Scan
        -> Filter
            -> Table "DIRECTORS" Access By ID
                -> Bitmap
                    -> Index "RDB$PRIMARY3" Unique Scan
        -> Filter
            -> Table "MOVIES" Access By ID
                -> Bitmap
                    -> Index "RDB$PRIMARY2" Unique Scan
```

Elapsed time: 2.781 sec

O select mais interno faz um full scan para encontrar o maior valor de rank.

O select mais externo faz um index nested loop join entre três tabelas: movies_directors (feito scan completo), director (indexado), movies (indexado e com o filtro de rank = max(rank) também usando indexamento para a realização da filtragem).

Firebird - Com subconsultas (não otimizado):

SELECT directors.first_name, directors.last_name, movies.name FROM directors join movies_directors on directors.id = movies_directors.director_id join movies on movies.id = movies_directors.movie_id WHERE movies.rank = (SELECT max(rank) from movies JOIN movies_directors on id = movie_id);

```
Select Expression
    -> Singularity Check
        -> Aggregate
            -> Nested Loop Join (inner)
                -> Table "MOVIES_DIRECTORS" Full Scan
                -> Filter
                    -> Table "MOVIES" Access By ID
                        -> Bitmap
                            -> Index "RDB$PRIMARY2" Unique Scan
Select Expression
    -> Nested Loop Join (inner)
        -> Table "MOVIES_DIRECTORS" Full Scan
        -> Filter
            -> Table "DIRECTORS" Access By ID
                -> Bitmap
                    -> Index "RDB$PRIMARY3" Unique Scan
        -> Filter
            -> Table "MOVIES" Access By ID
                -> Bitmap
                    -> Index "RDB$PRIMARY2" Unique Scan
```

Elapsed time= 3.688 sec

O select mais interno faz um index nested loop join entre a tabela movies_directors (utilizando full scan) e a tabela movies (indexada), a partir disso faz um full scan no resultado do join para encontrar o maior valor de rank. Não retirou o join desnecessário.

O select mais externo faz um index nested loop join entre três tabelas: movies_directors (feito scan completo), director (indexado), movies (indexado e com o filtro de rank = max(rank) também usando indexamento para a realização da filtragem).