Emanuel Victor Peres Mendes - 1911100030
Tiago Cassol - 11911100059

Relatório de Otimização de Consultas

3) Essa consulta atual usa uma CTE (Common Table Expression) para criar uma classificação dos funcionários dentro de cada departamento com base no salário, utilizando a função window `ROW_NUMBER()`. A seguir, é feita uma seleção dos funcionários que possuem a classificação igual a 1, ou seja, os funcionários com os salários mais altos dentro de cada departamento.

Para otimizar essa consulta, podemos reduzir a carga de trabalho e evitar cálculos desnecessários. Passo a passo para a otimização:

Remover colunas desnecessárias da CTE:

- Não é necessário selecionar todos os campos da tabela de funcionários na CTE se apenas `dep_id`, `nome`, e `salario` serão utilizados posteriormente.

Utilizar o comando `DISTINCT ON`:

- Em vez de classificar todos os funcionários e, em seguida, filtrar por `rank_salario = 1`, podemos usar a cláusula `DISTINCT ON` para selecionar diretamente o funcionário com o salário mais alto em cada departamento.

Então a ideia foi:

`DISTINCT ON` **com** `ORDER BY`**:**

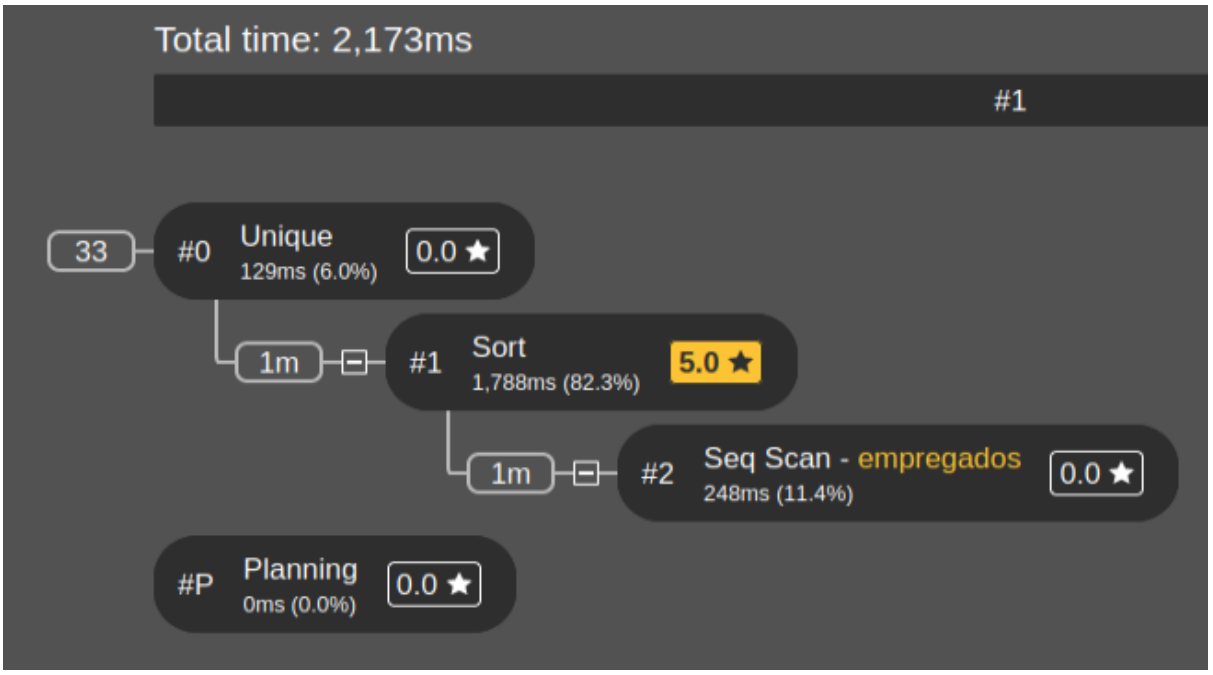
- Usamos `DISTINCT ON` junto com `ORDER BY dep_id, salario DESC`.
- Isso seleciona diretamente a primeira linha (com o salário mais alto) de cada departamento, pois está ordenado por departamento e salário em ordem decrescente.

Esta otimização simplifica a lógica da consulta ao usar `DISTINCT ON` em vez de calcular a posição de cada funcionário dentro do departamento. Isso pode melhorar a eficiência, especialmente em conjuntos de dados maiores, eliminando a necessidade de calcular números de linha para cada funcionário.

A query demorou 2396ms não otimizada, com desvio de 83,59 e 2063ms otimizada, com desvio de 97,57

Nessa questão, sacrificamos um pouco da velocidade para ordenar, facilitando o uso do `DISTINCT ON`, então para uma execução ainda mais rápida, poderia ter sido usado um

index, além de operações na memória ao invés de operações no disco, como mostra a seguir o analyze feito no site recomendado:



5.0 ★ Operation on Disk

This operation used enough memory that it had to write out to disk to store temporary data.

Total sort space used: 26 MB

If you can reduce the amount of memory consumed by the operation it would be faster, especially if you can get it to fit in memory.

You can do this by reducing the number of rows, with a LIMIT clause, or reducing the size of each row, by operating on fewer columns.

Decreasing the size of the rows is also possible in other ways, for example by using data types with a smaller footprint.

If reducing the memory required is not possible, you could consider increasing the amount of memory available for this operation by adjusting parameters like work_mem.

If you choose to do so, bear in mind that in-memory operations often use more memory than their on-disk equivalents.

5.0 ★ Index Potential

Sorts can often be avoided altogether by adding an index with the correct order.

Consider adding an index with the desired order to the table which contains the sort key(s) if possible, or restructure the query in order to allow Postgres to take advantage of an existing index.

The sort order is:

dep_id, salario DESC

```
Não otimizado:

WITH ranked_employees AS (
    SELECT
        dep_id,
        nome,
        salario,
        ROW_NUMBER() OVER (PARTITION BY dep_id ORDER BY salario DESC) AS rank_salario
    FROM
        empregados
)

SELECT
    dep_id,
    nome,
    salario
FROM
    ranked_employees
WHERE
    rank_salario = 1;

Otimizado:

SELECT
    DISTINCT ON (dep_id) dep_id,
    nome,
    salario
FROM
    empregados
ORDER BY
    dep_id, salario DESC;
```

8) Essa consulta compara o salário de cada funcionário com a média de salário do seu próprio departamento. Vamos otimizar essa consulta para melhorar sua eficiência:

Usar a junção e agrupamento para calcular a média por departamento:

- Em vez de calcular a média para cada linha na subconsulta, podemos usar uma junção e agrupamento para calcular a média do salário para cada departamento separadamente.

Evitar subconsulta correlacionada:

- A consulta atual possui uma subconsulta que referencia a tabela externa (`e.dep_id`) o que pode ser menos eficiente. Vamos evitar isso usando uma junção e uma média agrupada.

Filtrar os resultados com uma única condição `WHERE`:

- Vamos incorporar a condição de comparação do salário diretamente na cláusula `WHERE`, reduzindo a necessidade de uma subconsulta.

Essa foi a ideia:

CTE `media_dep` para calcular a média por departamento:

- Uma CTE é usada para calcular a média de salário para cada departamento.
- `SELECT dep_id, AVG(salario) AS media_por_departamento FROM empregados GROUP BY dep_id`.

Junção com a CTE `media_dep`:

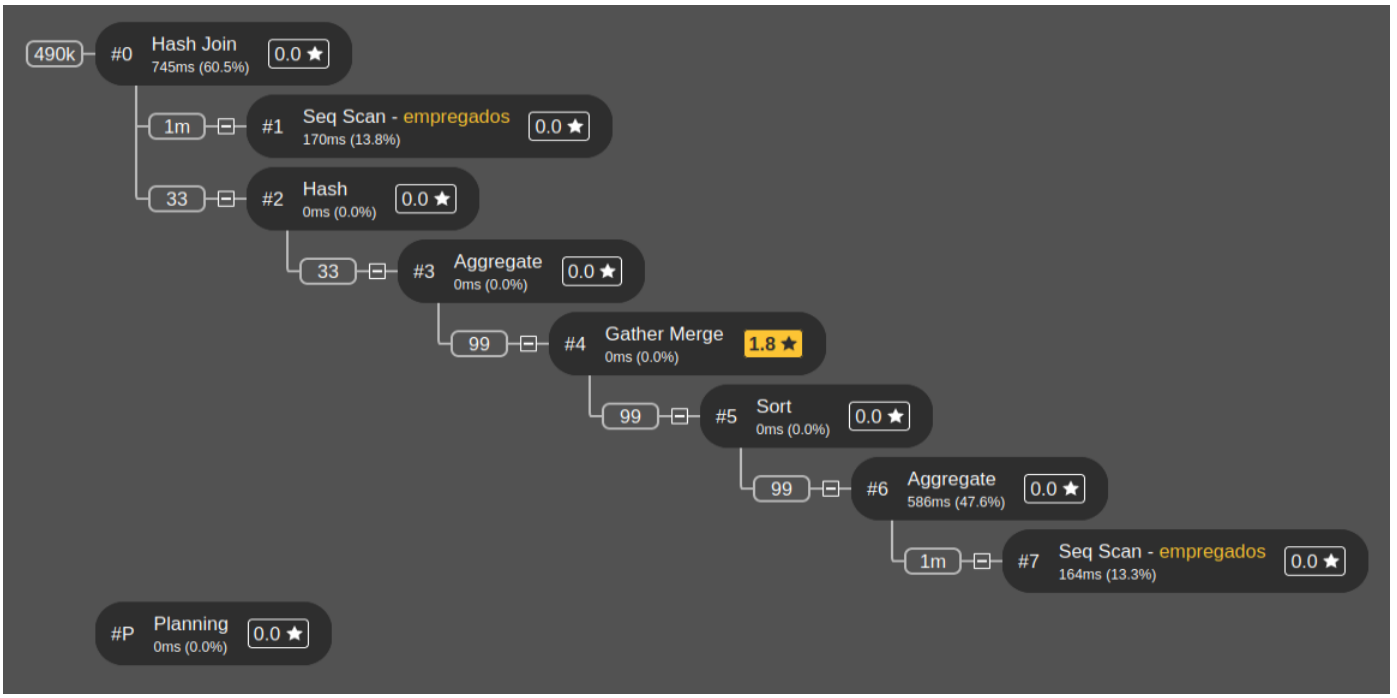
- Fazemos uma junção entre a tabela de funcionários (`empregados`) e a CTE `media_dep` com base no `dep_id`.
- Isso nos dá acesso à média de salário por departamento para cada funcionário.

Condição `WHERE`:

- Comparamos diretamente o salário de cada funcionário com a média de salário do seu departamento na cláusula WHERE.
- Isso retorna apenas os funcionários cujos salários são maiores que a média do seu próprio departamento. O que se esperava desde o início.

A query demorou mais de 3 horas não otimizada e 1146ms otimizada, com desvio de 29,08. O `JOIN` é o mesmo da número 9, um `HASH JOIN`, feito utilizando a coluna `dep_id` como chave de ligação entre a tabela principal empregados e a subconsulta `media_dep`. A condição `e.salario > media_dep.media_por_departamento` é aplicada no `WHERE`, comparando o salário de cada empregado com a média de salário de seu respectivo departamento.

Nesse caso, não havia muito o que fazer, a máquina virtual só tinha 2 cores pra usar, pelo que entendemos esse foi o problema.



1.8 ★ Scope for increased parallelism

This operation used the apparent maximum number of worker threads of 2 (default).

For transactional queries, where sub-100ms query times are required, accepting the overhead of thread management is unlikely to provide desirable results.

In this case, it is often better to reduce the amount of work being performed rather than increasing the resources available.

However, for analytical queries, where large amounts of data must sometimes be processed over longer periods of time, increasing the amount of workers available can be worthwhile, as long as the database server is still able to provide acceptable performance for any other simultaneous workloads.

If you want to allow Postgres to use more workers for a parallel operation, you can do so by increasing the value of the max_parallel_workers_per_gather setting. If you choose to do so, bear in mind that you may also need to increase the max_parallel_workers and max_worker_processes settings.

Não otimizado:

```sql
SELECT
    nome,
    salario
FROM empregados e
WHERE salario > (
        SELECT AVG(salario)
        FROM empregados
        WHERE dep_id = e.dep_id
    );
```

Otimizado:

```sql
SELECT
    e.nome,
    e.salario
FROM
    empregados e
JOIN (
    SELECT
        dep_id,
        AVG(salario) AS media_por_departamento
    FROM
        empregados
    GROUP BY
        dep_id
) media_dep ON e.dep_id = media_dep.dep_id
WHERE
    e.salario > media_dep.media_por_departamento;
```

9) A consulta realiza a junção entre as tabelas `empregados` e `departamentos` e calcula a média de salário para cada departamento usando uma função de janela `AVG` sobre a coluna `salario`. Vamos otimizar essa consulta:

Remover a seleção de colunas desnecessárias:

- No caso da média de salário por departamento, a mesma informação é replicada para cada linha. Podemos calcular a média separadamente e então juntá-la aos dados dos funcionários.

Calcular a média do salário por departamento separadamente:

- Isso reduz a carga de trabalho durante a junção de tabelas e evita cálculos repetitivos.

Juntar apenas as informações necessárias:

- Vamos unir apenas as informações dos funcionários e a média de salário por departamento, evitando a repetição dos dados.

Então, a ideia foi:

CTE (`media_salario_departamento`):

- Cria uma CTE que calcula a média de salário por departamento na tabela `empregados`.
- `SELECT dep_id, AVG(salario) AS media_salario FROM empregados GROUP BY dep_id`.

Consulta Principal:

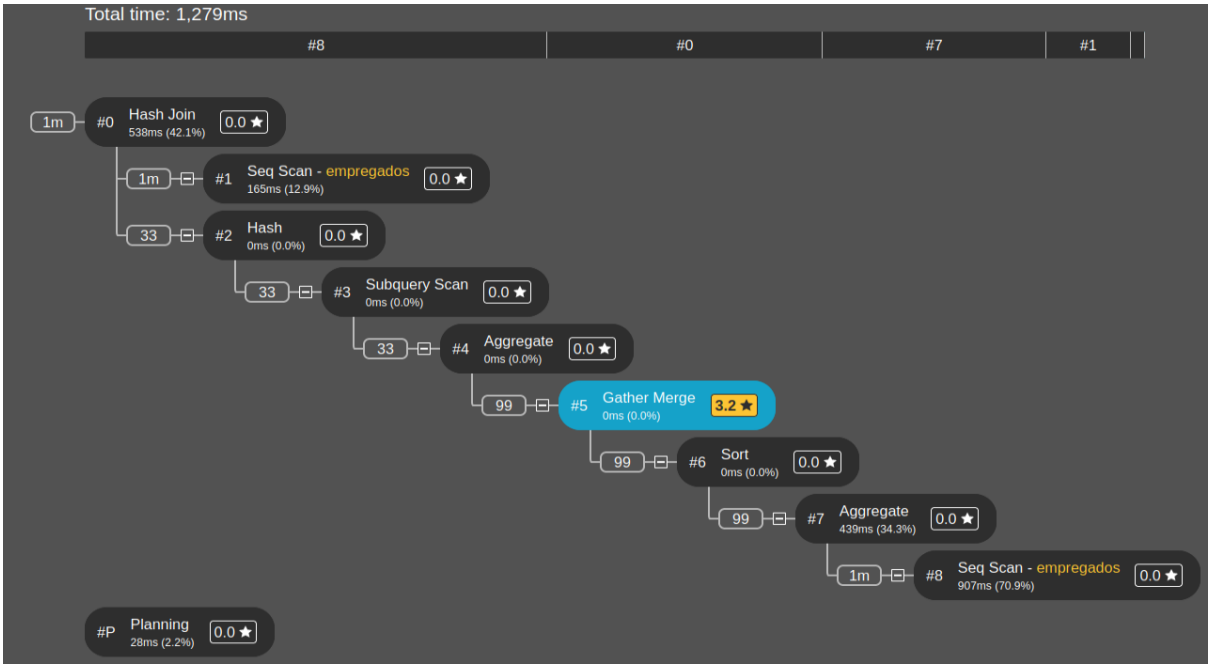
- Seleciona `dep_id`, `nome`, `salario` dos funcionários da tabela `empregados`.
- Realiza uma junção com a CTE `media_salario_departamento` usando o `dep_id`.

Esta otimização reduz a carga de trabalho ao calcular a média de salário separadamente antes de unir as informações dos funcionários, eliminando a repetição dos cálculos da média para cada linha de funcionário.

A query demorou 2583ms não otimizada, com desvio de 73,65 e 989ms otimizada, com desvio de 37,87

O algoritmo de `JOIN` utilizado nesse caso é o algoritmo mais comum e genérico para junção de tabelas em SQL, que é o `HASH JOIN`. O `HASH JOIN` é usado para combinar linhas de duas tabelas com base em uma condição de junção. Nesse contexto, estamos fazendo um `JOIN` entre a tabela de `empregados` e a CTE `media_salario_departamento` usando a coluna `dep_id`.

No caso da questão 9, aparentemente foi o mesmo problema da número 8.



3.2 ★ Scope for increased parallelism

This operation used the apparent maximum number of worker threads of 2 (default).

For transactional queries, where sub-100ms query times are required, accepting the overhead of thread management is unlikely to provide desirable results.

In this case, it is often better to reduce the amount of work being performed rather than increasing the resources available.

However, for analytical queries, where large amounts of data must sometimes be processed over longer periods of time, increasing the amount of workers available can be worthwhile, as long as the database server is still able to provide acceptable performance for any other simultaneous workloads.

If you want to allow Postgres to use more workers for a parallel operation, you can do so by increasing the value of the max_parallel_workers_per_gather setting. If you choose to do so, bear in mind that you may also need to increase the max_parallel_workers and max_worker_processes settings.

Não otimizado:

```sql
SELECT
    e.dep_id,
    e.nome AS nome_empregado,
    e.salario,
    AVG(e.salario) OVER (PARTITION BY e.dep_id) AS media_salario_departamento
FROM empregados e
JOIN departamentos d ON e.dep_id = d.dep_id;
```

Otimizado:

```sql
WITH media_salario_departamento AS (
    SELECT
        dep_id,
        AVG(salario) AS media_salario
    FROM
        empregados
    GROUP BY
        dep_id
)

SELECT
    e.dep_id,
    e.nome AS nome_empregado,
    e.salario,
    d.media_salario
FROM
    empregados e
JOIN
    media_salario_departamento d ON e.dep_id = d.dep_id;
```