Nome **: Angemydelson Saint-Bert**
Matrícula **: 2121101002**
Prof **: Guilherme Dal Bianco**
Disciplina **: Banco de Dados II**

CIÊNCIA DA COMPUTAÇÃO

TRABALHO TP3

ANGEMYDELSON SAINT-BERT, JOÃO LEONARDO COMIRAN FIGUEIRÓ

GUILHERME DAL BIANCO

BANCO DE DADOS II

CHAPECÓ

2023

1 Tarefa 1

1.1 Explain - Consulta 8:

- Listar os nomes dos colaboradores com salario maior que a média do seu departamento

Na consulta 8, tivemos os seguintes tempos de execução:

1. 8234.907 ms
2. 8406.566 ms
3. 8692.755 ms
4. 8452.878 ms
5. 8552.224 ms

A média foi de 8467.866 ms e o desvio foi de 289.85 ms.

```python
SELECT
    emp.nome,
    emp.salario
FROM
    departamentos d
JOIN (SELECT
        AVG(e.salario) OVER (PARTITION BY e.dep_id)
media_departamento,
        e.dep_id,
        e.nome,
        e.salario
      FROM
          empregados e) emp ON
    emp.dep_id = d.dep_id
WHERE
    emp.salario > emp.media_departamento;
```

1.1.1 Explain

Python

```
                                                                                QUERY PLAN
                                                                                    QUERY PLAN

------------------------------------------------------------------------------------------
-------------------------------------------------------------------------
 Merge Join  (cost=1671421.00..2036125.64 rows=3333333 width=11) (actual
time=2617.304..7832.312 rows=4902983 loops=1)
   Merge Cond: (d.dep_id = emp.dep_id)
   Buffers: shared read=66942, temp read=118444 written=120575
   ->  Sort  (cost=2.16..2.24 rows=33 width=4) (actual time=126.206..126.223
rows=33 loops=1)
         Sort Key: d.dep_id
         Sort Method: quicksort  Memory: 25kB
         Buffers: shared read=1
         ->  Seq Scan on departamentos d  (cost=0.00..1.33 rows=33 width=4)
(actual time=126.192..126.198 rows=33 loops=1)
               Buffers: shared read=1
   ->  Materialize  (cost=1671418.83..2004752.17 rows=3333333 width=15) (actual
time=2491.084..7308.416 rows=4902984 loops=1)
         Buffers: shared read=66941, temp read=118444 written=120575
         ->  Subquery Scan on emp  (cost=1671418.83..1996418.83 rows=3333333
width=15) (actual time=2491.080..6216.684 rows=4902984 loops=1)
               Filter: ((emp.salario)::numeric > emp.media_departamento)
               Rows Removed by Filter: 4793873
               Buffers: shared read=66941, temp read=118444 written=94707
               ->  WindowAgg  (cost=1671418.83..1846418.83 rows=10000000 width=47)
(actual time=2491.074..5289.478 rows=9696857 loops=1)
                     Buffers: shared read=66941, temp read=118444 written=94707
                     ->  Sort  (cost=1671418.83..1696418.83 rows=10000000
width=15) (actual time=2428.948..3078.651 rows=10000000 loops=1)
                           Sort Key: e.dep_id
                           Sort Method: external merge  Disk: 252128kB
                           Buffers: shared read=66941, temp read=63029
written=63234
                           ->  Seq Scan on empregados e  (cost=0.00..166941.00
rows=10000000 width=15) (actual time=0.304..664.901 rows=10000000 loops=1)
                                 Buffers: shared read=66941
 Planning:
   Buffers: shared hit=51 read=16
 Planning Time: 4.095 ms
 JIT:
   Functions: 16
   Options: Inlining true, Optimization true, Expressions true, Deforming true
```

```
   Timing: Generation 0.585 ms, Inlining 36.900 ms, Optimization 49.052 ms,
Emission 40.127 ms, Total 126.664 ms
 Execution Time: 8045.740 ms
(31 registros)
```

Merge Join (Merge Cond: d.dep_id = emp.dep_id): O Merge Join é o método de junção escolhido pelo otimizador de consultas. Isso implica que as duas tabelas estão ordenadas pelo campo dep_id e são mescladas com base nesse critério. Essa junção é usada para combinar linhas das tabelas "departamentos" e "emp" com base nos valores correspondentes de dep_id.

Sort (departamentos): A tabela "departamentos" é ordenada pelo campo dep_id usando o método de classificação rápida (quicksort). Isso é feito para facilitar a junção posterior.

Seq Scan (departamentos): Uma leitura sequencial é realizada na tabela "departamentos" após a ordenação. Isso é eficiente porque a tabela já está ordenada, e a leitura sequencial é mais rápida nesse caso.

Materialize (emp): A subconsulta (emp) é materializada, o que significa que os resultados da subconsulta são armazenados temporariamente em memória ou disco. Isso é feito para evitar a reexecução da subconsulta durante a junção.

Subquery Scan (emp): A subconsulta (emp) é executada, calculando a média do salário por departamento usando a função de janela (WindowAgg). A subconsulta também filtra os resultados para incluir apenas linhas em que o salário do empregado é maior que a média do departamento.

WindowAgg (emp): A função de janela é aplicada para calcular a média do salário por departamento. Essa é uma etapa crucial para a comparação entre o salário do empregado e a média do departamento.

Sort (empregados): A tabela "empregados" é ordenada pelo campo dep_id usando o método de ordenação externa (external merge). Esta ordenação é necessária para a função de janela, que requer que os dados estejam ordenados pela coluna de partição (dep_id).

Seq Scan (empregados): Uma leitura sequencial é realizada na tabela "empregados" após a ordenação. Esta leitura sequencial é possível porque a tabela está ordenada.

Filter (emp): A subconsulta (emp) filtra os resultados para incluir apenas as linhas em que o salário do empregado é maior que a média do departamento.

1.2 Tempos - Consulta 9:

Na consulta 9, tivemos os seguintes tempos de execução:

1. 8742.614 ms
2. 8775.055 ms
3. 8565.363 ms
4. 8701.901 ms
5. 8616.049 ms

A média foi de 8680.1964 ms e o desvio foi de 766.669 ms.

1.2.1 Explain

```python
SELECT
  e.dep_id,
  e.nome,
  e.salario,
  AVG(e.salario) OVER (PARTITION BY e.dep_id)::INTEGER
media_departamento
FROM
  empregados e
JOIN departamentos d ON
  e.dep_id = d.dep_id;
```

1.2.1 Explain

```python
WindowAgg  (cost=1671542.29..2019290.87 rows=10000000 width=19) (actual
time=2860.402..8393.315 rows=9696856 loops=1)
   Buffers: shared read=66942, temp read=123608 written=131283
   ->  Merge Join  (cost=1671542.29..1844290.87 rows=10000000 width=15) (actual
time=2720.261..5238.818 rows=9696856 loops=1)
         Merge Cond: (d.dep_id = e.dep_id)
         Buffers: shared read=66942, temp read=65851 written=99027
```

```
         ->  Sort  (cost=123.19..127.61 rows=1770 width=4) (actual
time=124.128..124.162 rows=33 loops=1)
               Sort Key: d.dep_id
               Sort Method: quicksort  Memory: 25kB
               Buffers: shared read=1
               ->  Seq Scan on departamentos d  (cost=0.00..27.70 rows=1770
width=4) (actual time=124.114..124.120 rows=33 loops=1)
                     Buffers: shared read=1
         ->  Materialize  (cost=1671418.83..1721418.83 rows=10000000 width=15)
(actual time=2596.093..4318.880 rows=9696857 loops=1)
               Buffers: shared read=66941, temp read=65851 written=99027
               ->  Sort  (cost=1671418.83..1696418.83 rows=10000000 width=15)
(actual time=2596.087..3323.388 rows=9696857 loops=1)
                     Sort Key: e.dep_id
                     Sort Method: external merge  Disk: 266408kB
                     Buffers: shared read=66941, temp read=65851 written=66805
                     ->  Seq Scan on empregados e  (cost=0.00..166941.00
rows=10000000 width=15) (actual time=0.315..707.199 rows=10000000 loops=1)
                           Buffers: shared read=66941
 Planning:
   Buffers: shared hit=52 read=17
 Planning Time: 3.282 ms
 JIT:
   Functions: 15
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 0.532 ms, Inlining 36.812 ms, Optimization 47.393 ms,
Emission 39.805 ms, Total 124.543 ms
 Execution Time: 8742.614 ms
(27 registros)
```

A consulta SQL envolve uma junção entre as tabelas "empregados" e "departamentos" através do algoritmo Merge Join. A primeira fase é a WindowAgg, com um custo estimado entre 886.47 e 786371.61. Essa operação processa 10 milhões de linhas, resultando em uma largura de 19 unidades. O tempo de execução varia de 663.617 a 9635.029 unidades de tempo, e o acesso a buffers inclui 2,128,502 leituras compartilhadas, 57,758 leituras temporárias, e 32,255 gravações temporárias.

A etapa subsequente é o Merge Join, cujo custo se situa entre 886.47 e 611371.61. Nessa fase, ocorre a junção propriamente dita, relacionando as tabelas "empregados" e "departamentos". A condição de junção é estabelecida pela igualdade entre os valores de "dep_id" nas duas tabelas. O tempo de execução varia de 133.280 a 6537.452 unidades de tempo, e são realizadas 2,128,502 leituras compartilhadas.

Dentro do Merge Join, uma parte relevante é o Index Scan na tabela "empregados" usando o índice "idx_dep_id". Este índice é crucial para otimizar a busca nas 10 milhões de linhas da tabela "empregados", reduzindo o custo da operação.

A consulta também envolve operações de ordenação, como indicado pela presença de Sort na tabela "departamentos". Esse processo contribui para o custo total da execução.

1.3 Explain - Consulta 10:

Na consulta 10, tivemos os seguintes tempos de execução:

1. 8759.480 ms
2. 8803.186 ms
3. 8563.460 ms
4. 8707.760 ms
5. 8728.879 ms

A média foi de 8712.553 ms e o desvio foi de 8233.67945 ms.

```python
SELECT
  emp.nome,
  emp.salario,
  emp.dep_id,
  emp.media_departamento
FROM
  departamentos d
JOIN (SELECT
    AVG(e.salario) OVER (PARTITION BY e.dep_id)
media_departamento,
    e.dep_id,
    e.nome,
    e.salario
   FROM
     empregados e) emp ON
  emp.dep_id = d.dep_id
WHERE
  emp.salario >= emp.media_departamento;
```

1.3.1 Explain

```
Python
 Merge Join  (cost=1671542.29..2036255.35 rows=3333333 width=47) (actual
time=2813.259..8581.881 rows=4902983 loops=1)
   Merge Cond: (d.dep_id = emp.dep_id)
   Buffers: shared read=66942, temp read=118444 written=120575
   ->  Sort  (cost=123.19..127.61 rows=1770 width=4) (actual time=139.050..139.070
rows=33 loops=1)
         Sort Key: d.dep_id
         Sort Method: quicksort  Memory: 25kB
         Buffers: shared read=1
         ->  Seq Scan on departamentos d  (cost=0.00..27.70 rows=1770 width=4)
(actual time=139.034..139.041 rows=33 loops=1)
               Buffers: shared read=1
   ->  Materialize  (cost=1671418.83..2004752.17 rows=3333333 width=47) (actual
time=2674.192..8008.653 rows=4902984 loops=1)
         Buffers: shared read=66941, temp read=118444 written=120575
         ->  Subquery Scan on emp  (cost=1671418.83..1996418.83 rows=3333333
width=47) (actual time=2674.187..6819.903 rows=4902984 loops=1)
               Filter: ((emp.salario)::numeric >= emp.media_departamento)
               Rows Removed by Filter: 4793873
               Buffers: shared read=66941, temp read=118444 written=94707
               ->  WindowAgg  (cost=1671418.83..1846418.83 rows=10000000 width=47)
(actual time=2674.179..5795.298 rows=9696857 loops=1)
                     Buffers: shared read=66941, temp read=118444 written=94707
                     ->  Sort  (cost=1671418.83..1696418.83 rows=10000000
width=15) (actual time=2602.318..3317.914 rows=10000000 loops=1)
                           Sort Key: e.dep_id
                           Sort Method: external merge  Disk: 252128kB
                           Buffers: shared read=66941, temp read=63029
written=63234
                           ->  Seq Scan on empregados e  (cost=0.00..166941.00
rows=10000000 width=15) (actual time=0.319..708.556 rows=10000000 loops=1)
                                 Buffers: shared read=66941
 Planning:
   Buffers: shared hit=53 read=16
 Planning Time: 2.666 ms
 JIT:
   Functions: 16
   Options: Inlining true, Optimization true, Expressions true, Deforming true
   Timing: Generation 0.644 ms, Inlining 39.147 ms, Optimization 52.860 ms,
Emission 46.909 ms, Total 139.559 ms
 Execution Time: 8803.186 ms
```

A consulta SQL em análise apresenta um plano de execução complexo, centrado em uma junção entre as tabelas "departamentos" e uma subconsulta derivada (alias "emp"). O objetivo é selecionar os empregados cujos salários são maiores ou iguais à média salarial de seus respectivos departamentos. Vamos analisar cada parte do plano detalhadamente.

A operação começa com um Merge Join, que combina os resultados da subconsulta com a tabela de departamentos. Este algoritmo de junção é eficiente quando ambas as tabelas estão ordenadas pelo critério de junção, que neste caso é a coluna "dep_id". O custo estimado é entre 123.89 e 778354.22, processando 3.3 milhões de linhas e produzindo uma saída com uma largura de 47 unidades.

Dentro da subconsulta, há um processo de Sort na tabela departamentos (d), que visa otimizar a junção ordenando as linhas por "dep_id". Este passo contribui para um melhor desempenho na etapa de junção. Uma leitura sequencial (Seq Scan) na tabela "departamentos" é realizada, custando 27.70 unidades de processamento.

A operação de Materialize é subsequente, visando armazenar temporariamente os resultados da subconsulta para serem utilizados posteriormente. A subconsulta (alias "emp") calcula a média de salários por departamento e filtra os empregados cujos salários são maiores ou iguais à média do seu departamento. Esse processo é crucial para a condição de filtro posterior.

Dentro da subconsulta, destaca-se um WindowAgg (Agregação de Janela), que realiza a agregação dos salários médios por departamento. Além disso, um Index Scan utilizando o índice "idx_dep_id" na tabela "empregados (e)" é efetuado para buscar as informações necessárias. A utilização de índices otimiza a busca na tabela, contribuindo para uma execução mais eficiente.

O tempo total de execução médio da consulta foi de 11,008.249 ms, com um desvio padrão de 482.9995522 ms. Esse desvio padrão indica uma variação considerável nos tempos de execução, sugerindo que a consulta pode ser sensível a diferentes circunstâncias. A análise do plano destaca a complexidade da consulta e aponta para possíveis áreas de otimização, como estratégias de junção mais eficientes ou ajustes nos índices utilizados, para melhorar o desempenho geral do processamento.

2 Tarefa 2

2.1 Consulta 8:

```python
WITH md AS (
SELECT
    d.dep_id,
    AVG(e.salario) AS media_departamento
FROM
    departamentos d
JOIN empregados e ON
    d.dep_id = e.dep_id
GROUP BY
    d.dep_id
)
SELECT
    e.nome,
    e.salario
FROM
    empregados e
JOIN md ON
    e.dep_id = md.dep_id
WHERE
    e.salario > md.media_departamento;
```

Na consulta 8, em razão das partes da query que mais demoraram para executar, sendo elas a comparação com o salário e a window function, empregamos as seguintes soluções respectivamente: criando um índice na coluna e refatorando a query como mostrado acima.

```python
      Hash Join  (cost=187744.79..382564.57 rows=3333333 width=11)
(actual time=1008.851..3462.947 rows=4902983 loops=1)
   Hash Cond: (e.dep_id = d.dep_id)
   Join Filter: ((e.salario)::numeric > (avg(e_1.salario)))
```

```
   Rows Removed by Join Filter: 4793873
   Buffers: shared hit=626 read=133273
   ->  Seq Scan on empregados e  (cost=0.00..166941.00
rows=10000000 width=15) (actual time=0.057..487.315 rows=10000000
loops=1)
         Buffers: shared hit=352 read=66589
   ->  Hash  (cost=187744.37..187744.37 rows=33 width=36) (actual
time=1008.768..1008.825 rows=32 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 10kB
         Buffers: shared hit=274 read=66684
         ->  Finalize GroupAggregate  (cost=187735.60..187744.04
rows=33 width=36) (actual time=1008.719..1008.815 rows=32 loops=1)
               Group Key: d.dep_id
               Buffers: shared hit=274 read=66684
               ->  Gather Merge  (cost=187735.60..187743.30 rows=66
width=36) (actual time=1008.706..1008.780 rows=96 loops=1)
                     Workers Planned: 2
                     Workers Launched: 2
                     Buffers: shared hit=274 read=66684
                     ->  Sort  (cost=186735.58..186735.66 rows=33
width=36) (actual time=993.117..993.121 rows=32 loops=3)
                           Sort Key: d.dep_id
                           Sort Method: quicksort  Memory: 27kB
                           Buffers: shared hit=274 read=66684
                           Worker 0:  Sort Method: quicksort
Memory: 27kB
                           Worker 1:  Sort Method: quicksort
Memory: 27kB
                           ->  Partial HashAggregate
(cost=186734.42..186734.75 rows=33 width=36) (actual
time=993.094..993.100 rows=32 loops=3)
                                 Group Key: d.dep_id
                                 Batches: 1  Memory Usage: 24kB
                                 Buffers: shared hit=260 read=66684
                                 Worker 0:  Batches: 1  Memory
Usage: 24kB
                                 Worker 1:  Batches: 1  Memory
Usage: 24kB
```

```
                                  ->  Hash Join
(cost=1.74..165901.08 rows=4166667 width=8) (actual
time=10.784..650.133 rows=3232285 loops=3)
                                        Hash Cond: (e_1.dep_id =
d.dep_id)
                                        Buffers: shared hit=260
read=66684
                                        ->  Parallel Seq Scan on
empregados e_1  (cost=0.00..108607.67 rows=4166667 width=8) (actual
time=0.018..185.577 rows=3333333 loops=3)
                                              Buffers: shared
hit=257 read=66684
                                        ->  Hash  (cost=1.33..1.33
rows=33 width=4) (actual time=10.754..10.755 rows=33 loops=3)
                                              Buckets: 1024
Batches: 1  Memory Usage: 10kB
                                              Buffers: shared hit=3
                                              ->  Seq Scan on
departamentos d  (cost=0.00..1.33 rows=33 width=4) (actual
time=10.734..10.739 rows=33 loops=3)
                                                    Buffers: shared
hit=3
 Planning Time: 0.412 ms
 JIT:
   Functions: 62
   Options: Inlining false, Optimization false, Expressions true,
Deforming true
   Timing: Generation 3.410 ms, Inlining 0.000 ms, Optimization
1.605 ms, Emission 30.655 ms, Total 35.670 ms
 Execution Time: 3600.092 ms
(45 registros)
```

Execution Time: 3836.744 ms

2.2 Consulta 9:

Nesse estudo de otimização de consultas em banco de dados, partimos de uma consulta original que demorava 10.194,972 ms para executar. Identificamos que a

complexidade da função de janela era um dos principais gargalos. A solução proposta envolveu a criação de uma Common Table Expression (CTE) para calcular a média salarial por departamento, resultando em uma consulta final mais eficiente. O tempo de execução foi significativamente reduzido para 2.912,956 ms, evidenciando os benefícios da otimização. A abordagem adotada não apenas melhorou o desempenho, mas também minimizou o impacto no banco de dados, destacando a importância da otimização para a eficiência operacional.

```
WITH media_dep AS (
SELECT
 AVG(e.salario) AS media_dep,
d.dep_id
FROM
departamentos d
Join empregados e on
d.dep_id = e.dep_id
Group by d.dep_id
) select
e.dep_id,
    e.nome,
    e.salario,
md.media_dep
From media_dep md
Join Empregados e on
  Md.dep_id = e.dep_id;
```

Execution Time: 2912.956 ms

Python

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------
 Hash Join  (cost=188388.39..382135.64 rows=88500000 width=47) (actual
time=1275.845..2879.299 rows=9696856 loops=1)
  Hash Cond: (e.dep_id = d.dep_id)
  Buffers: shared hit=113 read=133786
  -> Seq Scan on empregados e  (cost=0.00..166941.00 rows=10000000 width=15) (actual
time=0.262..490.652 rows=10000000 loops=1)
     Buffers: shared hit=96 read=66845
  -> Hash  (cost=188366.26..188366.26 rows=1770 width=36) (actual
time=1275.564..1275.625 rows=32 loops=1)
     Buckets: 2048  Batches: 1  Memory Usage: 18kB
     Buffers: shared hit=17 read=66941
```

```
    -> Finalize GroupAggregate (cost=187895.71..188348.56 rows=1770 width=36)
(actual time=1275.522..1275.616 rows=32 loops=1)
       Group Key: d.dep_id
       Buffers: shared hit=17 read=66941
       -> Gather Merge (cost=187895.71..188308.74 rows=3540 width=36) (actual
time=1275.511..1275.586 rows=96 loops=1)
          Workers Planned: 2
          Workers Launched: 2
          Buffers: shared hit=17 read=66941
          -> Sort (cost=186895.69..186900.11 rows=1770 width=36) (actual
time=1265.283..1265.286 rows=32 loops=3)
             Sort Key: d.dep_id
             Sort Method: quicksort Memory: 27kB
             Buffers: shared hit=17 read=66941
             Worker 0: Sort Method: quicksort Memory: 27kB
             Worker 1: Sort Method: quicksort Memory: 27kB
             -> Partial HashAggregate (cost=186782.50..186800.20 rows=1770 width=36)
(actual time=1265.251..1265.260 rows=32 loops=3)
                Group Key: d.dep_id
                Batches: 1 Memory Usage: 73kB
                Buffers: shared hit=3 read=66941
                Worker 0: Batches: 1 Memory Usage: 73kB
                Worker 1: Batches: 1 Memory Usage: 73kB
                -> Hash Join (cost=49.83..165949.16 rows=4166667 width=8) (actual
time=12.926..957.032 rows=3232285 loops=3)
                   Hash Cond: (e_1.dep_id = d.dep_id)
                   Buffers: shared hit=3 read=66941
                   -> Parallel Seq Scan on empregados e_1 (cost=0.00..108607.67
rows=4166667 width=8) (actual time=0.156..508.786 rows=3333333 loops=3)
                      Buffers: shared hit=1 read=66940
                   -> Hash (cost=27.70..27.70 rows=1770 width=4) (actual
time=12.758..12.759 rows=33 loops=3)
                      Buckets: 2048 Batches: 1 Memory Usage: 18kB
                      Buffers: shared hit=2 read=1
                      -> Seq Scan on departamentos d (cost=0.00..27.70 rows=1770
width=4) (actual time=12.739..12.744 rows=33 loops=3)
                         Buffers: shared hit=2 read=1
 Planning:
  Buffers: shared hit=91 read=23
 Planning Time: 4.847 ms
 JIT:
  Functions: 60
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 1.817 ms, Inlining 0.000 ms, Optimization 2.258 ms, Emission 35.921
ms, Total 39.996 ms
Execution Time: 3170.478 ms
(45 registros)
```

2.3 Consulta 10:

```python
WITH md AS (
SELECT
    d.dep_id,
    AVG(e.salario) AS media_departamento
FROM
    departamentos d
JOIN empregados e ON
    d.dep_id = e.dep_id
GROUP BY
    d.dep_id
)
SELECT
    e.nome,
    e.salario,
    md.dep_id,
    md.media_departamento
FROM
    empregados e
JOIN md ON
    e.dep_id = md.dep_id
WHERE
    e.salario >= md.media_departamento;
```

Na consulta 10, em razão das partes da query que mais demoraram para executar, sendo elas a comparação com o salário e a window function, empregamos as seguintes soluções respectivamente: criando um índice na coluna e refatorando a query como mostrado acima.

Execution Time: 3913.309 ms

Python

```
                                                                QUERY PLAN
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
----------
 Hash Join  (cost=188388.39..381848.14 rows=29500000 width=47) (actual
time=1251.588..3733.809 rows=4902983 loops=1)
  Hash Cond: (e.dep_id = d.dep_id)
  Join Filter: ((e.salario)::numeric >= (avg(e_1.salario)))
  Rows Removed by Join Filter: 4793873
  Buffers: shared hit=113 read=133786
  ->  Seq Scan on empregados e  (cost=0.00..166941.00 rows=10000000 width=15) (actual
time=0.269..496.130 rows=10000000 loops=1)
        Buffers: shared hit=96 read=66845
  ->  Hash  (cost=188366.26..188366.26 rows=1770 width=36) (actual
time=1251.290..1251.345 rows=32 loops=1)
        Buckets: 2048  Batches: 1  Memory Usage: 18kB
        Buffers: shared hit=17 read=66941
        ->  Finalize GroupAggregate  (cost=187895.71..188348.56 rows=1770 width=36)
(actual time=1251.249..1251.335 rows=32 loops=1)
              Group Key: d.dep_id
              Buffers: shared hit=17 read=66941
              ->  Gather Merge  (cost=187895.71..188308.74 rows=3540 width=36) (actual
time=1251.237..1251.305 rows=96 loops=1)
                    Workers Planned: 2
                    Workers Launched: 2
                    Buffers: shared hit=17 read=66941
                    ->  Sort  (cost=186895.69..186900.11 rows=1770 width=36) (actual
time=1241.279..1241.282 rows=32 loops=3)
                          Sort Key: d.dep_id
                          Sort Method: quicksort  Memory: 27kB
                          Buffers: shared hit=17 read=66941
                          Worker 0:  Sort Method: quicksort  Memory: 27kB
                          Worker 1:  Sort Method: quicksort  Memory: 27kB
                          ->  Partial HashAggregate  (cost=186782.50..186800.20 rows=1770 width=36)
(actual time=1241.252..1241.261 rows=32 loops=3)
                                Group Key: d.dep_id
                                Batches: 1  Memory Usage: 73kB
                                Buffers: shared hit=3 read=66941
                                Worker 0:  Batches: 1  Memory Usage: 73kB
                                Worker 1:  Batches: 1  Memory Usage: 73kB
                                ->  Hash Join  (cost=49.83..165949.16 rows=4166667 width=8) (actual
time=13.039..948.152 rows=3232285 loops=3)
                                      Hash Cond: (e_1.dep_id = d.dep_id)
                                      Buffers: shared hit=3 read=66941
```

```
                  -> Parallel Seq Scan on empregados e_1  (cost=0.00..108607.67
rows=4166667 width=8) (actual time=0.113..514.931 rows=3333333 loops=3)
                     Buffers: shared hit=1 read=66940
                  -> Hash  (cost=27.70..27.70 rows=1770 width=4) (actual
time=12.915..12.915 rows=33 loops=3)
                     Buckets: 2048  Batches: 1  Memory Usage: 18kB
                     Buffers: shared hit=2 read=1
                     -> Seq Scan on departamentos d  (cost=0.00..27.70 rows=1770
width=4) (actual time=12.894..12.898 rows=33 loops=3)
                        Buffers: shared hit=2 read=1
 Planning:
  Buffers: shared hit=150 read=22
 Planning Time: 4.498 ms
 JIT:
  Functions: 63
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 1.809 ms, Inlining 0.000 ms, Optimization 2.275 ms, Emission 36.368
ms, Total 40.451 ms
 Execution Time: 3899.545 ms
(47 registros)
```