Artefato D

A consulta escolhida foi a primeira das quatro.

Portanto, observemos esta consulta:

Query Editor | Query History

```
1  SELECT DISTINCT o.nome
2  FROM obra o
3      INNER JOIN diretores d ON o.diretores_id IN (
4          SELECT id
5          FROM diretores
6          WHERE nome_diretor = 'Karole Pickens'
7      )
8      INNER JOIN avaliacao a ON o.id IN (
9          SELECT obra_id
10         FROM avaliacao
11         GROUP BY obra_id
12         HAVING AVG(nota) > 8
13     );
14
15
16
17
18
```

Data Output

	nome text
1	Senhor dos Aneis 24
2	Senhor dos Aneis 7

Observando esta consulta com EXPLAIN, temos que:

	Node
1.	→ Aggregate
2.	→ Hash Inner Join **Hash Cond**: (avaliacao.obra_id = o.id)
3.	→ Nested Loop Inner Join
4.	→ Seq Scan on diretores as d
5.	→ Materialize
6.	→ Aggregate **Filter**: (avg(avaliacao.nota) > '8'::numeric)
7.	→ Seq Scan on avaliacao as avaliacao
8.	→ Hash
9.	→ Merge Inner Join
10.	→ Nested Loop Inner Join
11.	→ Index Scan using diretores_pkey on diretores as diretores **Filter**: (nome_diretor = 'Karole Pickens'::text)
12.	→ Materialize
13.	→ Index Only Scan using avaliacao_pkey on avaliacao as a
14.	→ Sort
15.	→ Seq Scan on obra as o

Uma função que não usamos que se diferencia da query é a cláusula EXISTS. Portanto, fazendo a seguinte query:

```sql
SELECT nome
FROM obra WHERE EXISTS(
    SELECT obra_id FROM avaliacao
    GROUP BY obra_id
    HAVING AVG(nota) > 8
    AND obra.id = avaliacao.obra_id)
AND
EXISTS(
    SELECT d.id FROM diretores d
    WHERE d.nome_diretor = 'Karole Pickens'
    AND obra.diretores_id = d.id)
;
```

Data Output | Explain | Messages | Notifications

	nome text
1	Senhor dos Aneis 7
2	Senhor dos Aneis 24

Analisando através do EXPLAIN, temos:

#	Node
1.	→ Nested Loop Inner Join **Join Filter:** (obra.diretores_id = d.id)
2.	→ Seq Scan on obra as obra **Filter:** (SubPlan 1)
3.	→ Aggregate **Filter:** (avg(avaliacao.nota) > '8'::numeric)
4.	→ Seq Scan on avaliacao as avaliacao **Filter:** (obra.id = obra_id)
5.	→ Materialize
6.	→ Seq Scan on diretores as d **Filter:** (nome_diretor = 'Karole Pickens'::text)

Tudo será explicado no fim, portanto paciência.

Agora, para a terceira query, podemos ser menos óbvios e fazer algo como:

```sql
SELECT DISTINCT o.nome
FROM obra o WHERE(
    EXISTS (
                SELECT 1 FROM diretores d INNER JOIN avaliacao a
                ON((o.diretores_id IN (
                            SELECT diretores.id FROM diretores
                            WHERE diretores.nome_diretor = 'Karole Pickens'
                            )
                 ) AND (
                        o.id IN (
                            SELECT avaliacao.obra_id FROM avaliacao
                            GROUP BY avaliacao.obra_id
                            HAVING AVG(avaliacao.nota) > 8
                        )
                    )
                )
    )
)
```

Data Output | Explain | Messages | Notifications

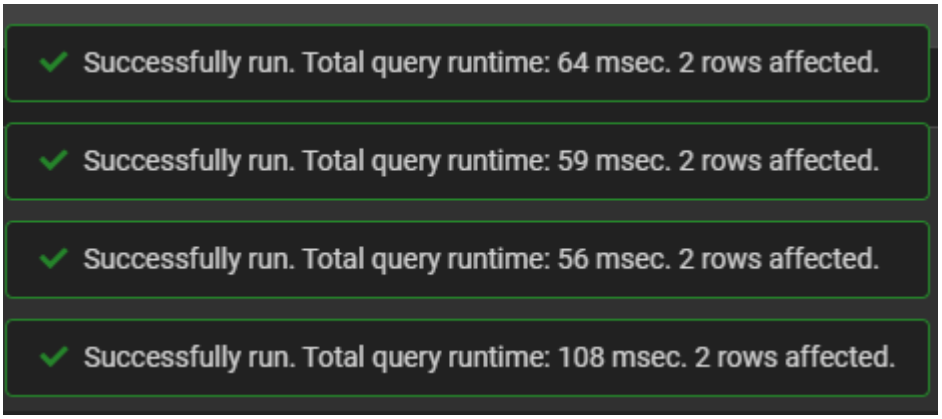
	nome text
1	Senhor dos Aneis 24
2	Senhor dos Aneis 7

Através do explain, também teremos:

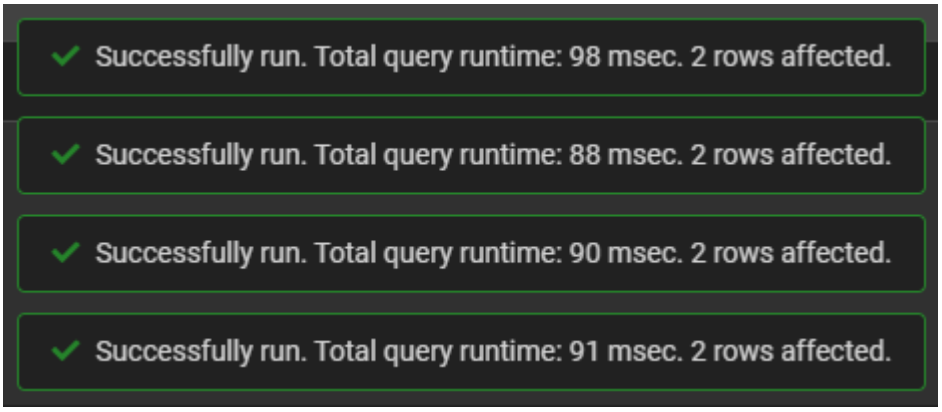
#	Node
1.	→ Unique
2.	→ Sort
3.	→ Seq Scan on obra as o Filter: (SubPlan 3)
4.	→ Result
5.	→ Nested Loop Inner Join
6.	→ Index Only Scan using avaliacao_pkey on avaliacao as a
7.	→ Materialize
8.	→ Seq Scan on diretores as d
9.	→ Seq Scan on diretores as diretores Filter: (nome_diretor = 'Karole Pickens'::text)
10.	→ Aggregate Filter: (avg(avaliacao.nota) > '8'::numeric)
11.	→ Seq Scan on avaliacao as avaliacao

Como primeiro passo da análise, vamos ver os tempos de resposta de cada query.

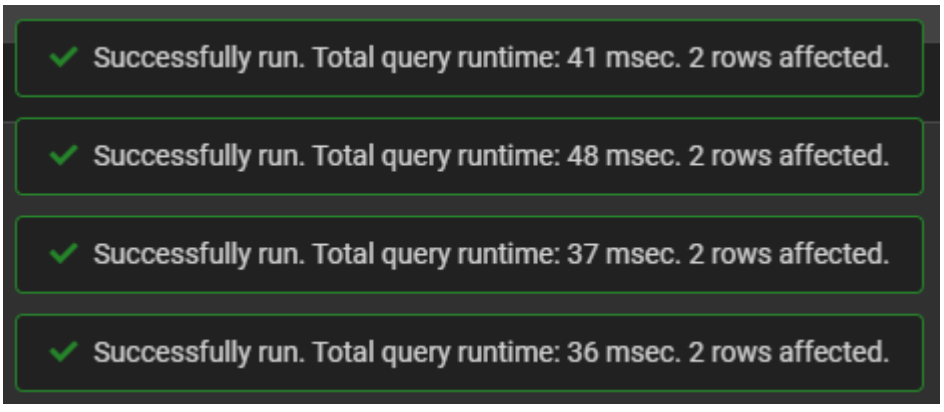
A query original:



A primeira query alternativa:



A segunda query alternativa:



Com isso, podemos ter a conclusão de que, em termos de custo para este caso:

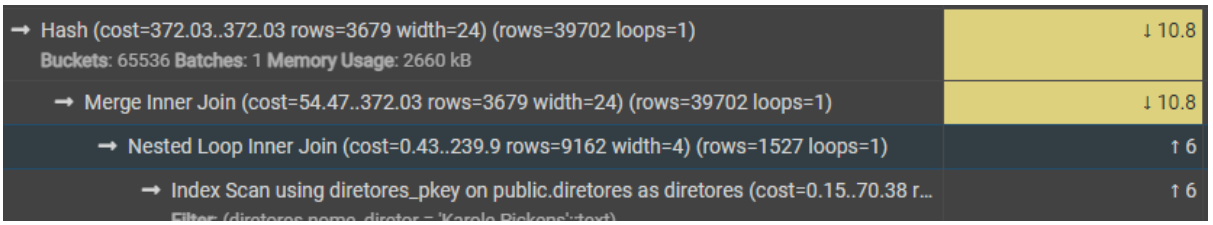
Segunda query alternativa < Query original < Primeira query alternativa

A partir daqui, começaremos a analisar o por quê:

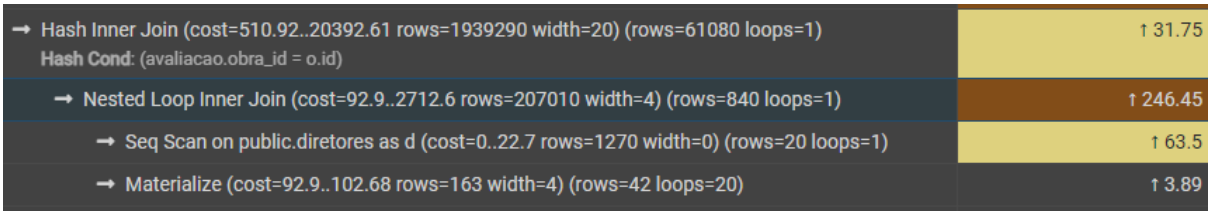
Query Original VS Primeira query alternativa

Ambos os métodos usam os exatos mesmos selects, portanto a diferença de custo está entre fazer dois INNER JOINs ou dois EXISTS com uma cláusula AND.

Graças ao pgAdmin, podemos ver o custo dessas operações, de certa forma.



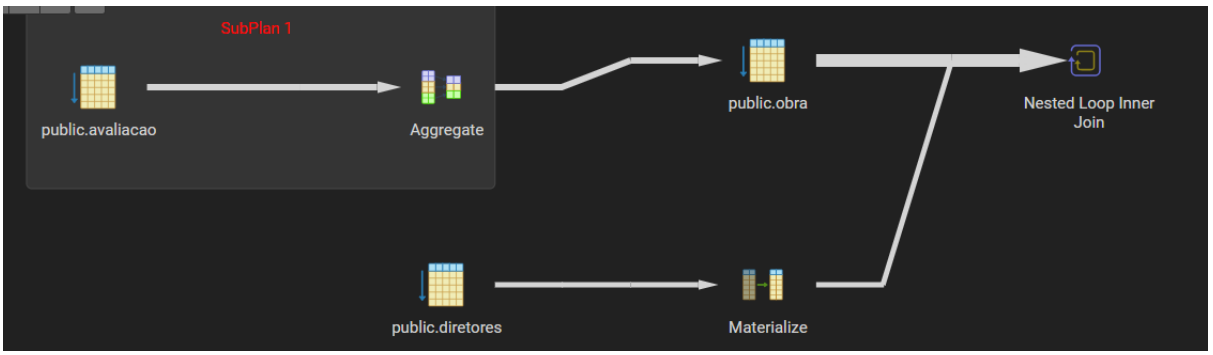
O custo do inner join em relação à tabela de diretores é baixo porém relevante, enquanto que:



O custo do inner join em relação às avaliações é elevado. A segunda operação mais custosa da query atrás apenas do Aggregate dos INNER JOINs. O custo elevado é causado pela expectativa de muitas linhas nesta parte do plano SQL de consulta, quando não é o caso:

Rows X	Actual	Plan
↑ 255	2	510
↑ 31.75	61080	1939290
↑ 246.45	840	207010

Porém, no primeiro caso alternativo:



Todo o custo está acumulado no scan sequencial de obra (que faz sentido, já que é a maior tabela do banco de dados) e do loop de INNER JOIN, que acontece em:

```
obra.diretores_id = d.id
```

Nesta consulta, sem nenhum filtro, duas tabelas enormes são comparadas, sem contar que obra, a maior tabela do modelo, é escaneada inteira também.

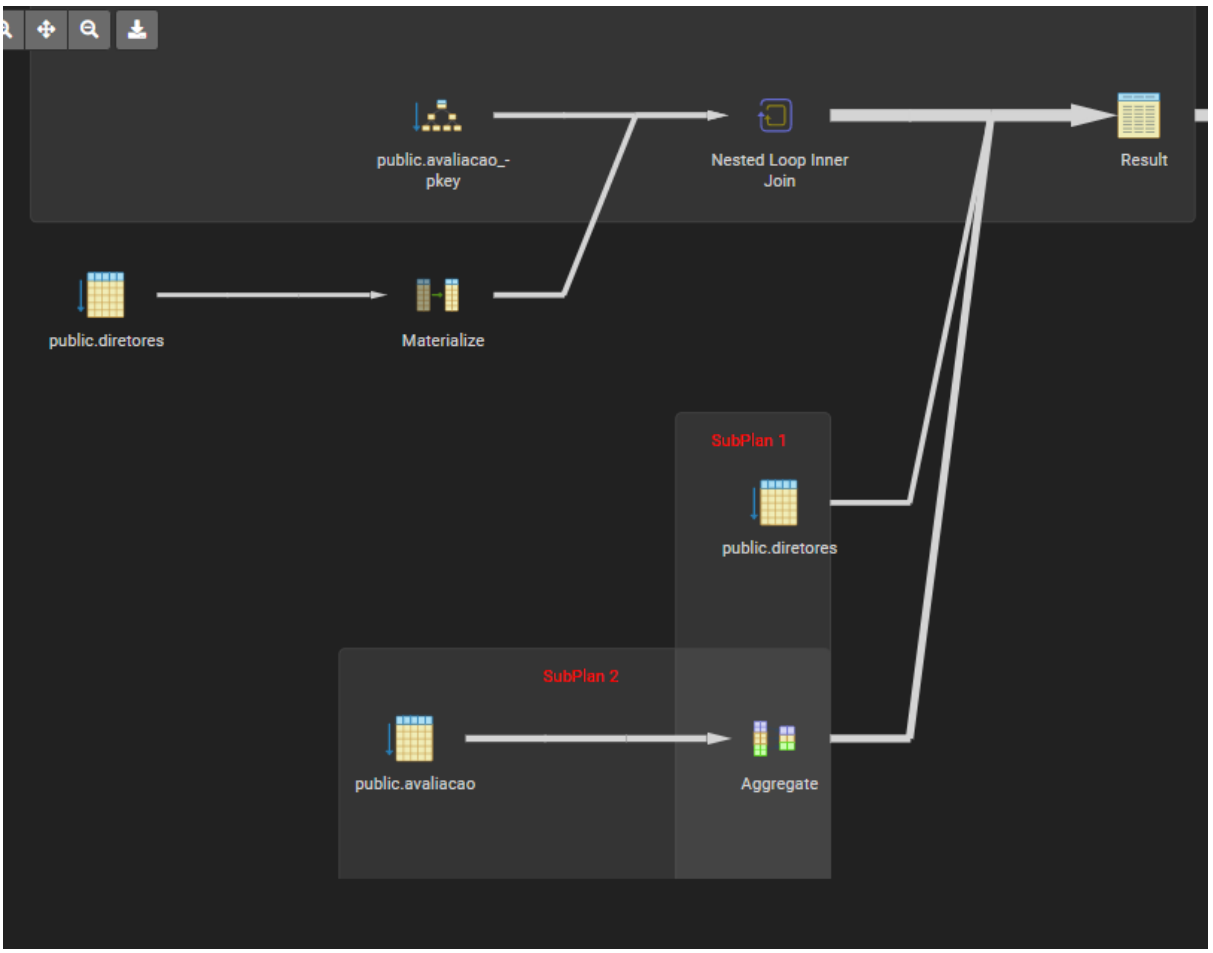
Portanto, mesmo que o plano diga que menos operações são executadas para este plano, o custo das operações do plano estão muito pior distribuídos.

Assim, concluímos que a má distribuição leva a esta query ser mais custosa.

Query Original VS Segunda query alternativa

Como, no caso original, usamos INNER JOINs que são usados apenas para o WHERE, neste caso, o uso esperto de EXISTS não apenas dá um resultado equivalente, como possui uma performance superior.

Neste plano, os índices da resposta já são obtidos na ponto de RESULT:



Aqui, não temos outputs certos dados pelo plano, mas temos nós que já possuem os índices:

parent_node	1_1_1_1_1_2
_serial	8
rowsx	1270
rowsx_direction	positive
rowsx_flag	4

Portanto, quando chega na busca sequencial da tabela obra, os resultados já estão lá, sendo procurados por índice, reduzindo o custo total.

Assim, podemos concluir que o plano esperto da segunda query alternativa é superior ao da query original.