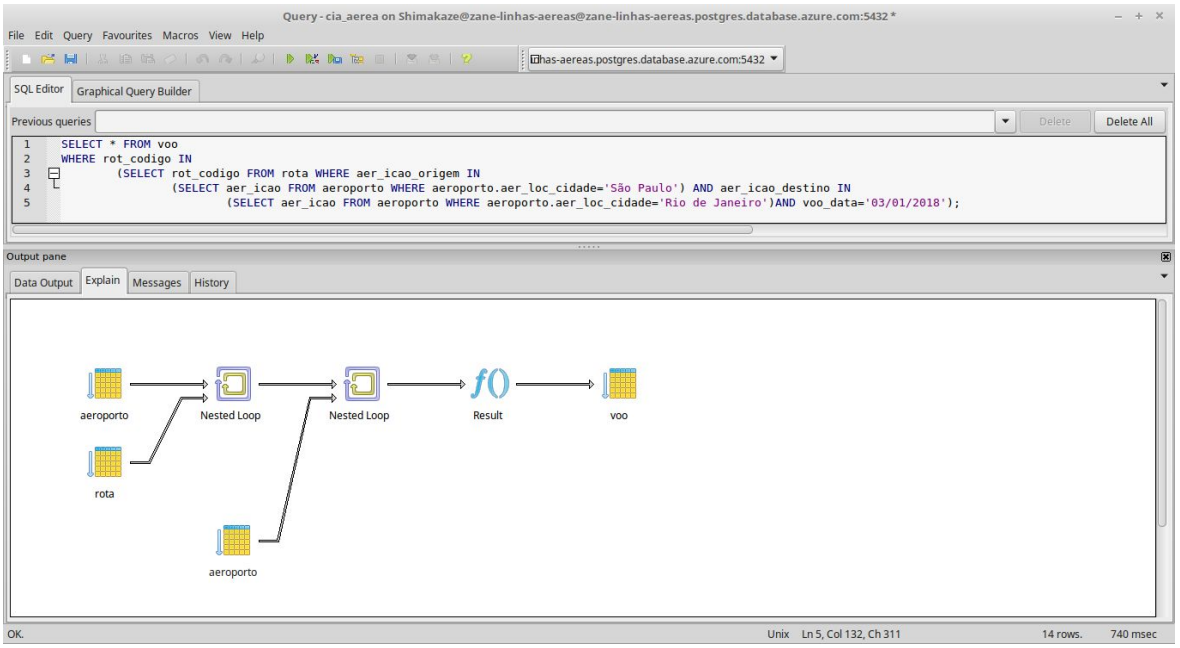
Turma 04

Marcos Henrique Monteiro da Silva	8802392
Marcos Vinicius Oliveira Lunardelli	9019302
Peterson Medeiros Santos	9004550

Relatório de performance de consultas

Consulta 1 - consulta de voos por cidade de origem, cidade de destino e data



Nesta consulta, são gerados dois joins, o primeiro para conciliar as informações da cidade de origem com quaisquer rotas que contenham esta cidade como partida. O segundo, concilia as informações obtidas no primeiro com as informações da cidade de destino. Por fim, o filtro pela data na relação resultante.

Um Nested-Loop é resultado da execução de um escaneamento sequencial no primeiro nó e então, para cada tupla que ele retorna, é executada a segunda operação de escaneamento sequencial no segundo nó.

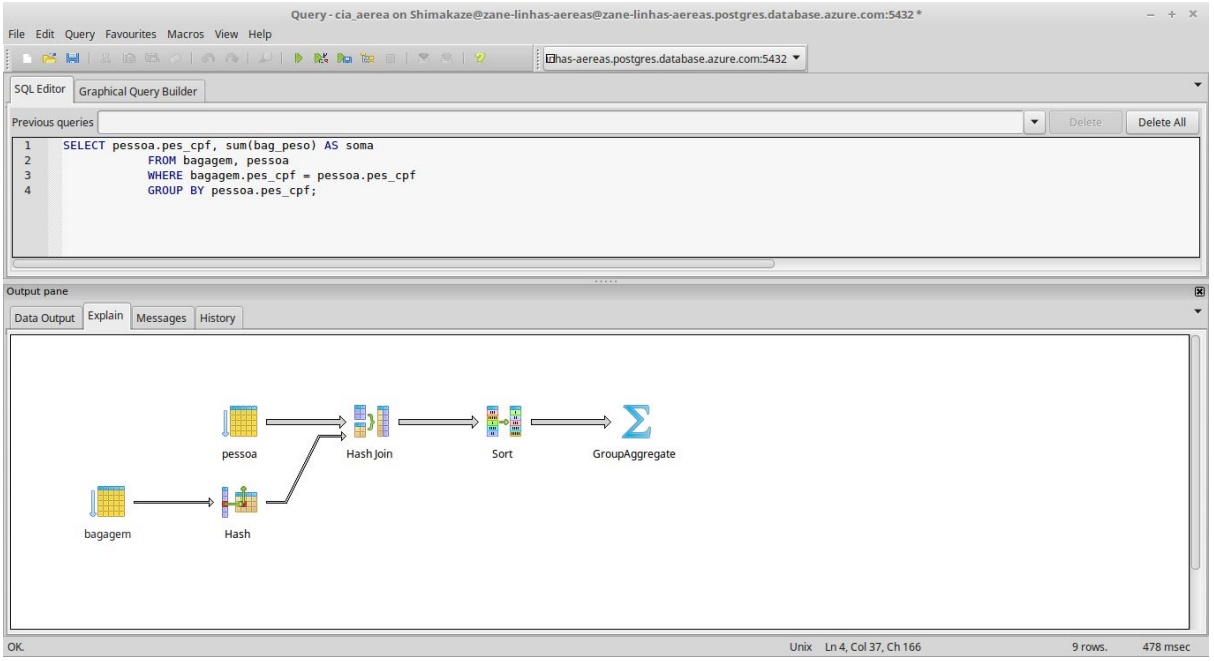
Em resumo, o algoritmo segue:

- Nested Loop executa em um lado do join, uma vez, criando uma relação "A";
- Para cada tupla em "A", executa-se uma segunda operação, criando uma relação "B";
- Se "B" não retornou nenhuma tupla, os dados de "A" são ignorados;
- Se "B" retornou tuplas, para cada tupla retornada uma nova tupla é retornada pelo Nested Loop, baseada na tupla atual de A e na tupla atual de B.

	QUERY PLAN text
1	Seq Scan on voo (cost=0.00..560.25 rows=150 width=36) (actual time=0.062..0.742 rows=1 loops=1)
2	Filter: (SubPlan 1)
3	Rows Removed by Filter: 299
4	SubPlan 1
5	-> Result (cost=0.00..3.69 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=300)
6	One-Time Filter: (voo.voo_data = '2018-01-03'::date)
7	-> Nested Loop (cost=0.00..3.69 rows=1 width=4) (actual time=0.038..0.038 rows=1 loops=1)
8	Join Filter: (rota.aer_icao_destino = aeroporto_1.aer_icao)
9	Rows Removed by Join Filter: 7
10	-> Nested Loop (cost=0.00..2.47 rows=1 width=24) (actual time=0.016..0.021 rows=3 loops=1)
11	Join Filter: (rota.aer_icao_origem = aeroporto.aer_icao)
12	Rows Removed by Join Filter: 19
13	-> Seq Scan on aeroporto (cost=0.00..1.20 rows=1 width=20) (actual time=0.008..0.008 rows=2 loops=1)
14	Filter: (aer_loc_cidade = 'São Paulo'::bpchar)
15	Rows Removed by Filter: 6
16	-> Seq Scan on rota (cost=0.00..1.12 rows=12 width=44) (actual time=0.003..0.003 rows=11 loops=2)
17	-> Seq Scan on aeroporto aeroporto_1 (cost=0.00..1.20 rows=1 width=20) (actual time=0.003..0.003 rows=3 loops=3
18	Filter: (aer_loc_cidade = 'Rio de Janeiro'::bpchar)
19	Rows Removed by Filter: 13
20	Planning time: 0.310 ms
21	Execution time: 0.780 ms

Neste formato, a leitura de requisições é feita de baixo pra cima, uma vez que para que o resultado desejado seja encontrado, é necessário fazer um nested loop, que por sua vez requisita outro nested loop e etc. No caso, ele começa fazendo uma busca sequencial. Os valores de custo mostrado pelo postgreSQL são valores estimados de custo calculados pelo próprio programa sendo o primeiro um custo inicial e o final um custo total. Por não precisar de preparo prévio, as buscas sequenciais possuem o valor inicial de 0.00. Vale notar que esses valores não são representados em segundos, mas sim em um valor especificado pelas configurações do Postgre que representam custo de processamento e tempo. O grande uso de buscas sequenciais no projeto provavelmente se dá pelo baixo número de tuplas nas tabelas procuradas. O custo inicial é nulo mas o custo total é alto em relação ao custo dos nós filhos. Isto se deve por esta consulta específica resultar em uma tupla única.

Consulta 2 - consulta do peso total das bagagens de um passageiro



Uma busca usando índice hash, indexado pelo CPF da pessoa é feita na relação de bagagem. A junção é feita com o resultado da busca da tupla do CPF buscado, e nesta relação resultante, é feita a soma dos valores.

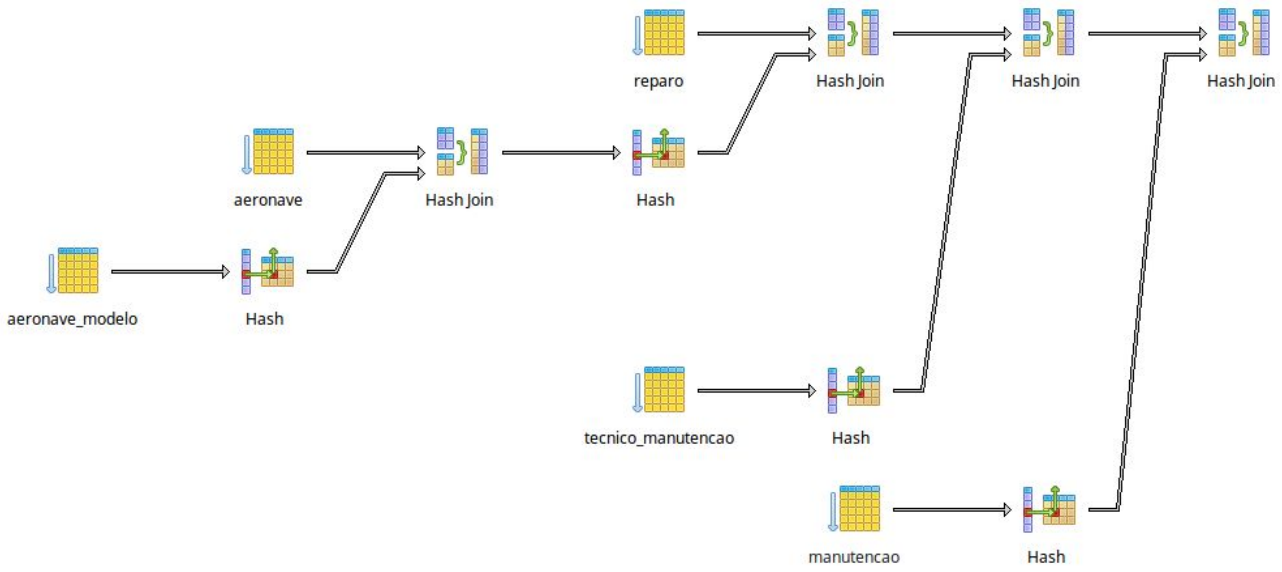
	QUERY PLAN text
1	GroupAggregate (cost=54.36..54.57 rows=12 width=12) (actual time=0.423..0.427 rows=6 loops=1)
2	Group Key: pessoa.pes_cpf
3	-> Sort (cost=54.36..54.39 rows=12 width=8) (actual time=0.412..0.413 rows=12 loops=1)
4	Sort Key: pessoa.pes_cpf
5	Sort Method: quicksort Memory: 25kB
6	-> Hash Join (cost=1.27..54.14 rows=12 width=8) (actual time=0.069..0.367 rows=12 loops=1)
7	Hash Cond: (pessoa.pes_cpf = bagagem.pes_cpf)
8	-> Seq Scan on pessoa (cost=0.00..49.00 rows=1000 width=4) (actual time=0.022..0.195 rows=1000 loops=1)
9	-> Hash (cost=1.12..1.12 rows=12 width=8) (actual time=0.024..0.024 rows=12 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 9kB
11	-> Seq Scan on bagagem (cost=0.00..1.12 rows=12 width=8) (actual time=0.009..0.013 rows=12 loops=1)
12	Planning time: 385.405 ms
13	Execution time: 0.572 ms

Um hash-join é a junção de duas sub-relações, uma relação de scan sequencial na tabela secundária e um hash criado a partir da tabela primária. O hash criado pode ser criado em memória ou em disco, dependendo da análise do tamanho das tabelas envolvidas. Pela quantidade de dados envolvidos na consulta, a utilização de hash-join com particionamento único em memória e de busca sequencial nas relações não causa impacto significativo no desempenho.

Em um caso de uso típico, é esperado que a quantidade de bagagens de cada cliente mantenha uma média entre 1 e 2 volumes, com essa quantidade multiplicada pela quantidade de passageiros em um voo. A quantidade de volumes para cada voo então, varia também de acordo com a capacidade da aeronave, o que, com uma frota variada pode variar de 150 até 880 passageiros, ou seja, de 300 a 1760 volumes.

Consulta 3 - consulta de reparos indicando o técnico responsável, a aeronave e detalhes do tipo de reparo

```
 SELECT aeronave.avi_serial_number,
      aeronave.avi_matricula,
      aeronave.avi_mod_modelo,
      aeronave.avi_categoria,
      aeronave_modelo.avi_mod_capacidade,
      tecnico_manutencao.pes_cpf,
      tecnico_manutencao.tec_anac,
      tecnico_manutencao.tec_tipo_contrato,
      manutencao.man_nome,
      manutencao.man_desricao,
      reparo.rep_orcamento
   FROM aeronave,
      aeronave_modelo,
      reparo,
      tecnico_manutencao,
      manutencao
  WHERE aeronave_modelo.avi_mod_modelo = aeronave.avi_mod_modelo AND
reparo.avi_serial_number = aeronave.avi_serial_number AND
tecnico_manutencao.pes_cpf = reparo.tec_cpf AND manutencao.man_codigo =
reparo.man_codigo;
```



	QUERY PLAN text
1	Hash Join (cost=5.24..7.58 rows=44 width=1004) (actual time=0.163..0.199 rows=45 loops=1)
2	Hash Cond: (reparo.man_codigo = manutencao.man_codigo)
3	-> Hash Join (cost=4.15..6.28 rows=44 width=360) (actual time=0.101..0.127 rows=45 loops=1)
4	Hash Cond: (reparo.tec_cpf = tecnico_manutencao.pes_cpf)
5	-> Hash Join (cost=3.02..4.92 rows=44 width=192) (actual time=0.076..0.091 rows=45 loops=1)
6	Hash Cond: (reparo.avi_serial_number = aeronave.avi_serial_number)
7	-> Seq Scan on reparo (cost=0.00..1.44 rows=44 width=16) (actual time=0.007..0.008 rows=45 loops=1)
8	-> Hash (cost=2.64..2.64 rows=30 width=180) (actual time=0.057..0.057 rows=30 loops=1)
9	Buckets: 1024 Batches: 1 Memory Usage: 11kB
10	-> Hash Join (cost=1.16..2.64 rows=30 width=180) (actual time=0.037..0.049 rows=30 loops=1)
11	Hash Cond: (aeronave.avi_mod_modelo = aeronave_modelo.avi_mod_modelo)
12	-> Seq Scan on aeronave (cost=0.00..1.30 rows=30 width=176) (actual time=0.007..0.009 rows=30 loops=1)
13	-> Hash (cost=1.07..1.07 rows=7 width=108) (actual time=0.012..0.012 rows=7 loops=1)
14	Buckets: 1024 Batches: 1 Memory Usage: 9kB
15	-> Seq Scan on aeronave_modelo (cost=0.00..1.07 rows=7 width=108) (actual time=0.006..0.009 rows=7 loops=1)
16	-> Hash (cost=1.06..1.06 rows=6 width=172) (actual time=0.014..0.014 rows=6 loops=1)
17	Buckets: 1024 Batches: 1 Memory Usage: 9kB
18	-> Seq Scan on tecnico_manutencao (cost=0.00..1.06 rows=6 width=172) (actual time=0.009..0.010 rows=6 loops=1)
19	-> Hash (cost=1.04..1.04 rows=4 width=652) (actual time=0.050..0.050 rows=4 loops=1)
20	Buckets: 1024 Batches: 1 Memory Usage: 9kB
21	-> Seq Scan on manutencao (cost=0.00..1.04 rows=4 width=652) (actual time=0.020..0.021 rows=4 loops=1)
22	Planning time: 0.479 ms
23	Execution time: 0.309 ms

Nesta consulta, são utilizados diversos hash-join para se montar a relação final. Pela quantidade de tuplas não ser significativa, o particionamento é realizado em memória em um único passo, não necessitando de particionamento recursivo. Em um caso de uso em que a companhia aérea tenha uma frota significativa de por exemplo, 150 aeronaves, é esperado que a tabela de reparos tenha uma quantidade estimada de 1500 tuplas por mês. O desempenho será variável com base na quantidade de reparos sob responsabilidade de um técnico e se terão filtros para filtrar por determinado período de tempo.