FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

MESTRADO EM ENGENHARIA INFORMÁTICA

Sistemas de Gestão de Dados

TPCH benchmarking

Project 1

Hugo Abreu - 2018275541
Gui Costa - 2021186342

• U  C •

12 de abril de 2022

Conteúdo

1 Caraterísticas dos Computadores

PostgreSQL - Computador 1:

- Processador - AMD Ryzen 7 4800H with Radeon Graphics, 2.90 GHz, 8 Core(s)
- Gráfica - NVIDIA GeForce GTX 1650 Ti
- RAM - 16,0 GB DDR4 3200MHz
- Sistema Operativo - Windows 10

Oracle - Computador 2:

- Processador - Intel(R) Core(TM) i5-7300HQ 2.50 GHz
- Gráfica - NVIDIA GeForce GTX 1050
- RAM - 8,0 GB DDR4 2400MHz
- Sistema Operativo - Windows 10

2 Carregamento das tabelas

		Customer	Linei-tem	Nation	Orders	Part	Part-supp	Region	Sup-plier
Post-greSQL	Load Time (segundos)	27	1592	0	245	36	175	0	1
Oracle	Load Time (segundos)	11	443	0	105	31	49	0	1

Tabela 2.1: Tempos do carregamento das tabelas

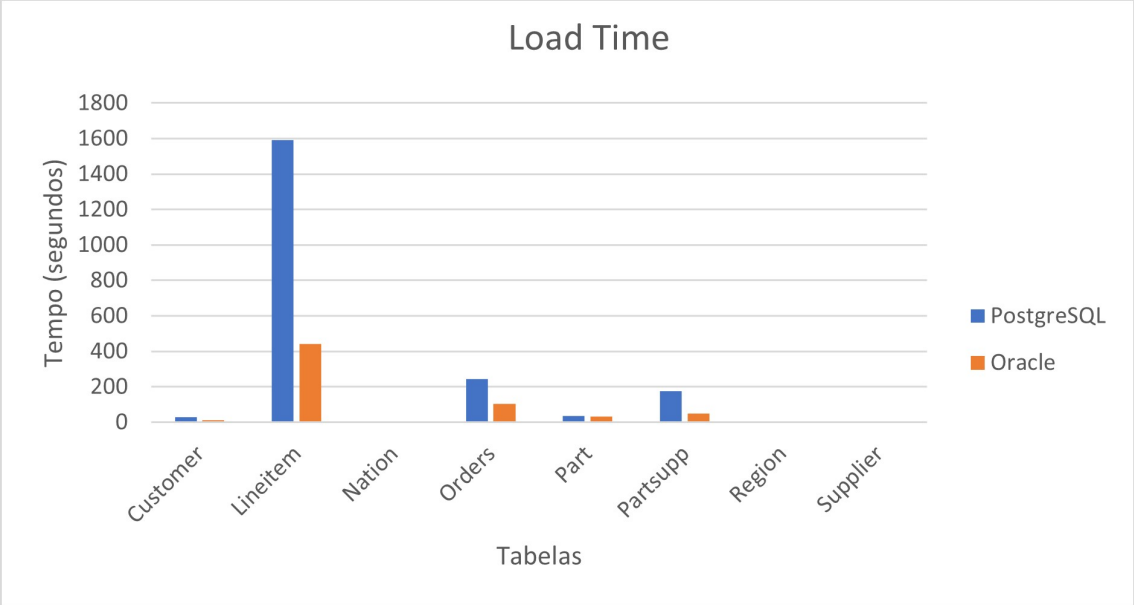

Figura 2.1: Tempos do carregamento das tabelas

Pelos tempos do carregamento das tabelas podemos observar que a tabela Lineitem é a que possui mais dados dado que foi a que demorou mais tempo. Com este conhecimento já é de prever que qualquer pesquisa que utilize dados desta tabela tenha tempos elevados comparados com os restantes.

Podemos também reparar numa diferença nos tempos entre os dois motores. O Oracle tem tempos mais baixos relativamente ao postgreSQL.

3 Criação das chaves

3.1 Chaves Primárias

		R_REGIONKEY	N_NATIONKEY)	C_CUSTKEY	S_SUPPKEY	P_PARTKEY	PS_PARTKEY,PS_SUPPKEY	O_ORDERKEY	L_ORDERKEY,L_LINENUMBER
PostgreSQL	Tempo (segundos)	0	0	8	0	11	44	72	371
Oracle	Tempo (segundos)	0	0	6	0	9	43	48	283

Tabela 3.1: Tempos de criação das chaves primárias

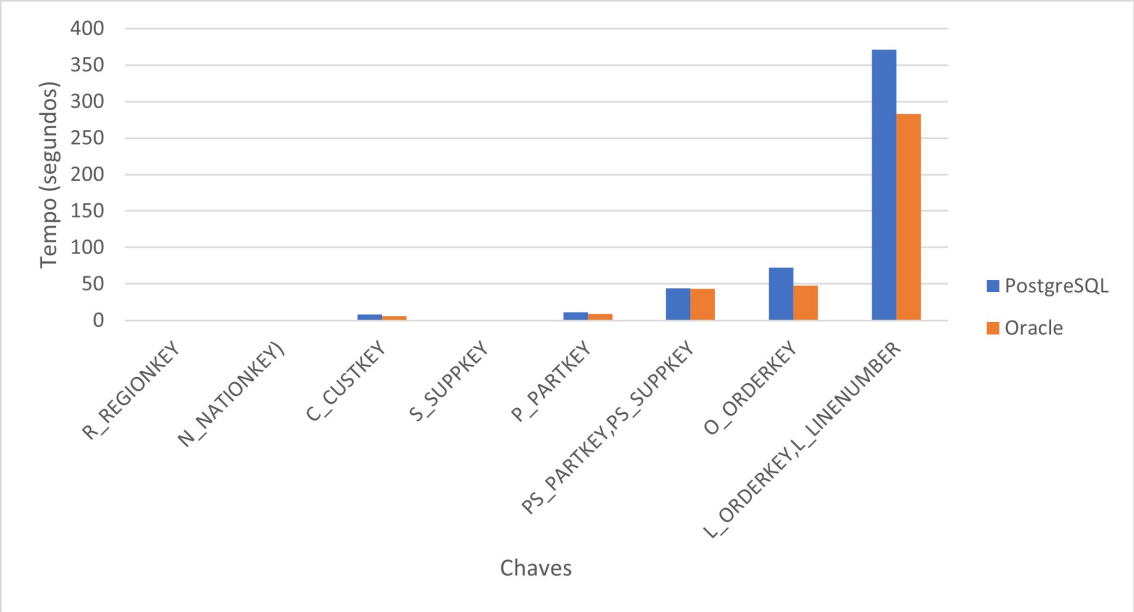

Figura 3.1: Tempos da criação das chaves primárias

3.2 Chaves Forasteiras

		N_REGIONKEY	S_NATIONKEY	C_NATIONKEY	PS_SUPPKEY	PS_PARTKEY	O_CUSTKEY	L_ORDERKEY	L_PARTKEY,L_SUPPKEY
PostgreSQL	Tempo (segundos)	0	0	5	5	2	37	209	257
Oracle	Tempo (segundos)	0	0	11	13	8	33	73	177

Tabela 3.2: Tempos de criação das chaves forasteiras

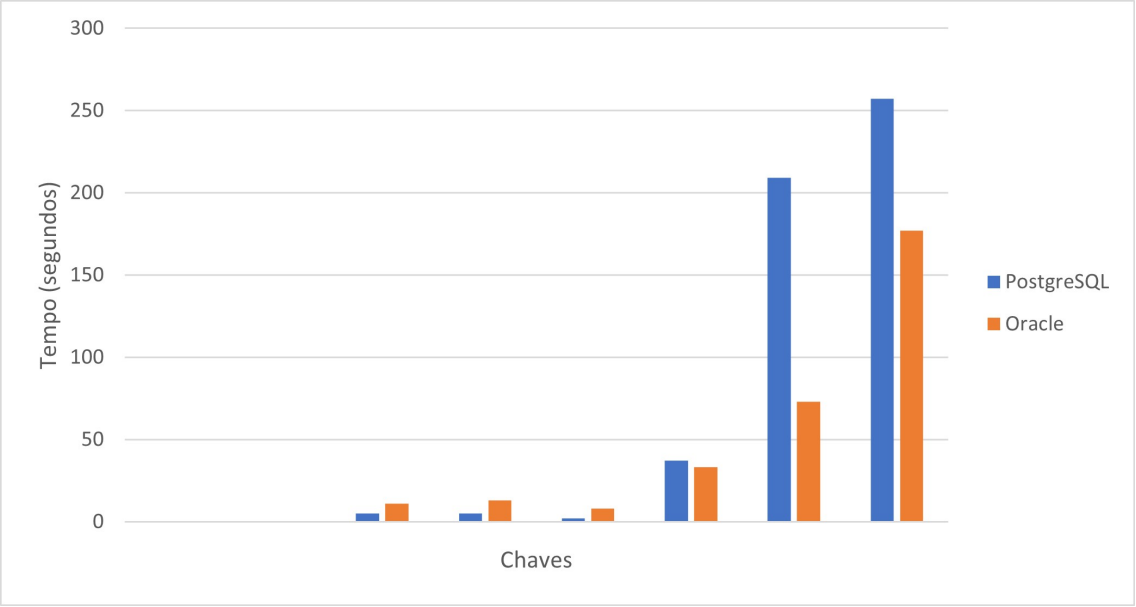

Figura 3.2: Tempos da criação das chaves forasteiras

4 Tempos das Pesquisas

- **PostgreSQL**

		1º Querie	2º Querie	3º Querie	4º Querie	5º Querie	6º Querie	7º Querie	8º Querie	9º Querie	10º Querie	11º Querie	12º Querie	13º Querie	14º Querie	15º Querie	16º Querie	17º Querie	18º Querie	19º Querie	20º Querie	21º Querie	22º Querie
PostgreSQL	S/chaves	553	14400	133	393	331	69	117	354	461	104	9	106	41	67	133	16	14400	465	67	14	14400	17
PostgreSQL	C/chaves	126	78	151	115	153	98	125	120	356	127	14	139	50	102	198	15	14400	389	66	0	250	18

Tabela 4.1: Tempos das pesquisas no PostgreSQL

- **Oracle**

		1º Querie	2º Querie	3º Querie	4º Querie	5º Querie	6º Querie	7º Querie	8º Querie	9º Querie	10º Querie	11º Querie	12º Querie	13º Querie	14º Querie	15º Querie	16º Querie	17º Querie	18º Querie	19º Querie	20º Querie	21º Querie	22º Querie
Oracle	S/chaves	113	14400	131	43	302	25	125	44	49	64	2	60	25	86	86	4	16	98	28	166	1533	4
Oracle	C/chaves	116	413	124	43	284	25	133	42	51	66	2	59	24	85	86	3	16	98	27	176	282	4

Tabela 4.2: Tempos das pesquisas no Oracle

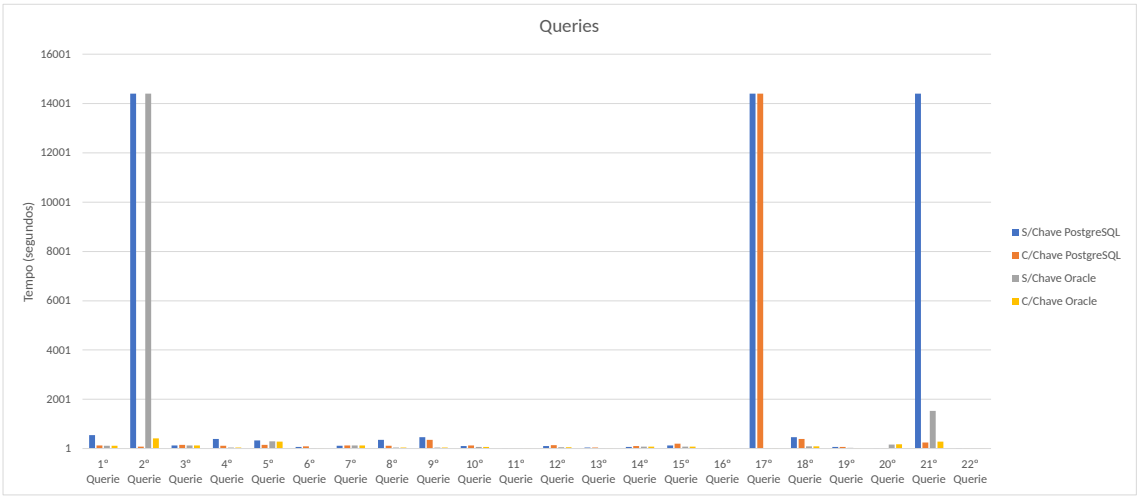

Figura 4.1: Tempos das pesquisas

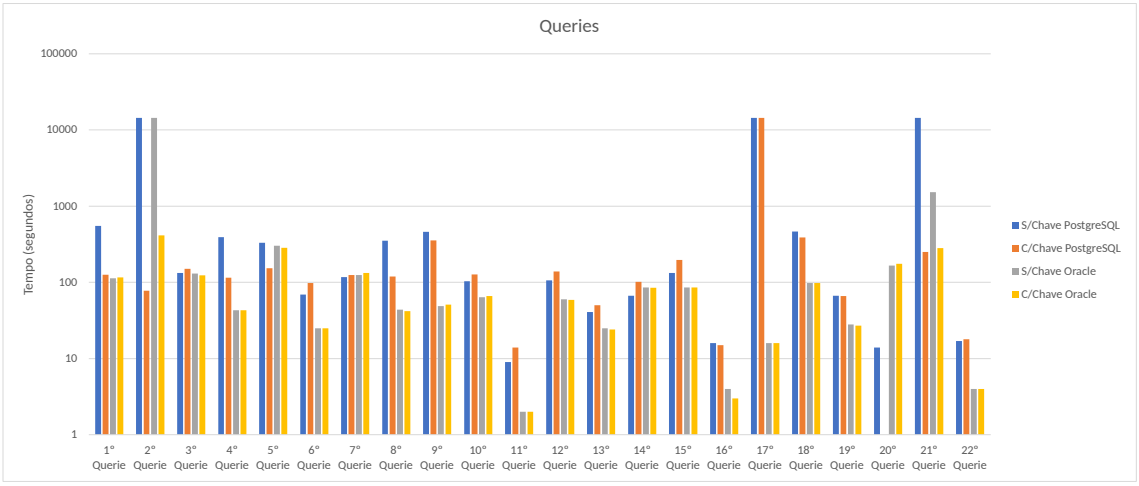

Figura 4.2: Tempos das pesquisas em escala logarítmica de base 10

Pela análise dos gráficos e das tabelas é possível verificar que os tempos após a criação das chaves primárias e forasteiras são bastante mais baixos do que antes da criação das mesmas. Isto acontece porque o motor de base de dados tem agora a possibilidade de efetuar joins entre tabelas utilizando as chaves forasteiras como também utilizar os indíces das chaves primárias aquando do scan das tabelas.

Em grande parte dos resultados o motor de base de dados da Oracle teve tempos mais baixos, isto pode-se dar ao facto dos dados gerados e carregados para as tabelas serem diferentes, como também pelo facto deste motor trabalhar de forma diferente do postgreSQL.

Na 17ª Querie e na 21ª Querie o postgreSQL não conseguiu devolver o resultado em menos de 4 horas , enquanto que o Oracle conseguiu devolver em pouco tempo. Novamente isto deve-se ao facto de estes motores terem formas diferentes de armazenar as tabelas no disco. Outro factor relevante é o da tabela Lineitem ter bastantes mais dados nos dados gerados para o postgreSQL.

5 Optimizações Possíveis

- **PostgreSQL**

  Para a otimização de pesquisas com elevado número de dados como é o caso de um dataset como o TCP-H é possível, através do ficheiro de configuração do postgreSQL, alterar alguns parâmetros que podem tornar estas pesquisas mais rápidas. Porém também podem trazer problemas quando alteramos erradamente.[1] [2] [3]

  Estes parâmetros são:

  - *Shared_buffers*: É a quantidade de memória que o motor de base de dados usa para memória partilhada de buffers. Por default o valor é de 32MB mas para máquinas com mais de 1GB de RAM e com o intuito de aumentar a performance pode-se alterar este valor para 25% da memória partilhada do sistema.
  - *Work_mem*: É a quantidade de memória que é usada para operações internas de classificação e para as tabelas de hash antes destas serem escritas para ficheiros temporários no disco. O valor default é 1MB porém o valor recomendado para melhor perfomance é dado pela seguinte fórmula ((Total RAM - shared_buffers)/(16 x CPU cores)).
  - *Maintenance_work_mem*: É a quantidade de memória que o motor de base de dados usa para operações de manutenção como criação de indíces , alteração de tabelas , etc.
  - *Random_page_cost*: É o valor que representa o custo de um acesso aleatório a uma página do disco. O valor recomendado é 1.1 para discos SSD.
  - *Effective_cache_size*: Define a suposição do motor de base de dados sobre o tamanho efetivo da cache que está disponível para uma única pesquisa.
  - *Cpu_tuple_cost*: É o valor que representa o custo da leitura de cada linha durante uma pesquisa. O valor default é 0.01 porém, de forma a obter uma melhor performance deve ser colocado a 0.03

  Para além destas alterações ao ficheiro de configuração também é possível utilizar outras técnicas de otimização como : criar indíces nos campos mais usados nas pesquisas o que evita o scan sequencial da tabela, criar vistas das pesquisas mais efetuadas, utilizar os comandos EXPLAIN e ANALYZE para avaliar e compreender o que o motor vai fazer com o intuito de verificar aspetos possíveis de otimizar.

- **Oracle**

  No caso do Oracle, ao contrario do PostgreSQL, existe a possibilidade de se utilizar um "Automatic Memory Management"(AMM). Isto é a maneira mais simples de optimizar o uso de memória. No entanto o utilizador tem sempre a possibilidade, dentro do AMM, de definir um *MEMORY_TARGET* e, opcionalmente definir um *MEMORY_MAX_TARGET* que impede o utilizador de definir um *memory target* maior que o *max target*. Ao definirmos estes valores, o sistema automaticamente redistribui a memória entre a "System Global Area"(SGA) e a "Program Global Area"(PGA).

Tabela 5.1: Exemplo de Target Advice para o SGA depois da última pesquisa executada

SGA_SIZE	SGA_SIZE_FACTOR	ESTD_DB_TIME	ESTD_DB_TIME_FACTOR	ESTD_BUFFER_CACHE_SIZE
1216	0.5	21	1	480
1824	0.75	21	1	1120
2432	1	21	1	1744
3040	1.25	21	1	2240
3648	1.5	21	1	2880
4256	1.75	21	1	3520
4864	2	21	1	4000

Apesar de existir um AMM teríamos de decidir quanta memória atribuir, para isto, o Oracle define uma tabela com os valores "aconselhados"para o target, que se podem visualizar ao executar *select * from v$memory_target_advice* que retorna uma tabela com os valores esperados factor de tempo para vários tamanhos de *memory target*, 1 sendo o valor atual e 2 sendo o dobro. É possível visualizar os "advices"também para outros valores como o do SGA e PGA. [4].

6 Evolução dos tempos com pesquisas simultâneas

	8ª Querie	15ª Querie
Tempo (segundos)	102	152

Tabela 6.1: Tempos de 2 pesquisas simultâneas

	8ª Querie	15ª Querie	9ª Querie	7ª Querie	18ª Querie
Tempo (segundos)	191	371	303	215	756

Tabela 6.2: Tempos de 5 pesquisas simultâneas

	8ª Querie	15ª Querie	9ª Querie	7ª Querie	18ª Querie	1ª Querie	14ª Querie	21ª Querie	3ª Querie	12ª Querie
Tempo (segundos)	255	499	379	264	940	509	220	1216	327	404

Tabela 6.3: Tempos de 10 pesquisas simultâneas

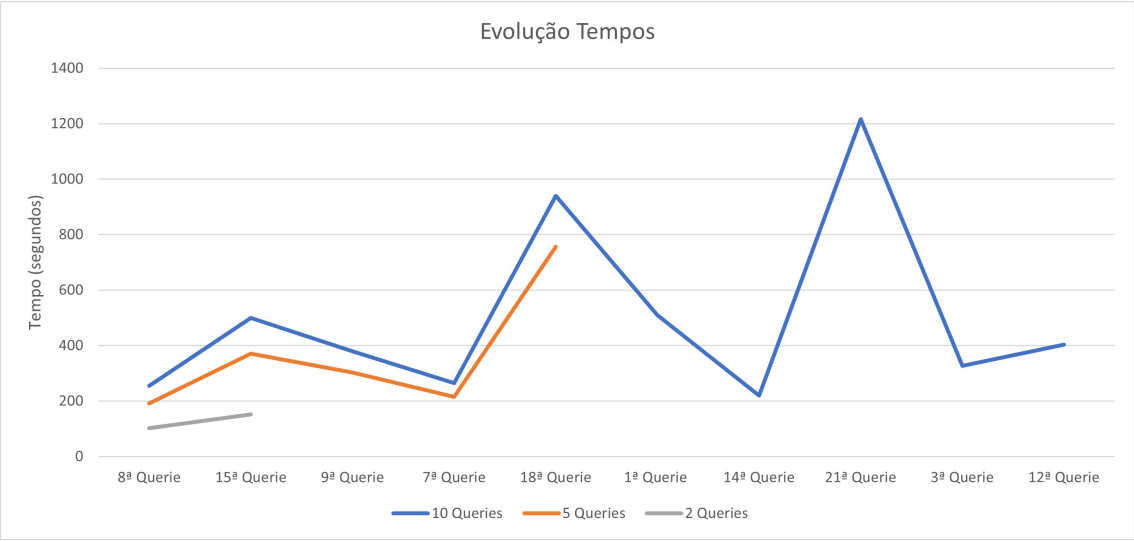

Figura 6.1: Tempos das pesquisas simultâneas

Pela análise do gráfico e das tabelas podemos verificar que quantas mais pesquisas simultâneas estiverem a correr maior é o tempo que cada uma demora. Isto deve-se ao facto

do motor de base de dados precisar de gerir os recursos necessários para correr cada uma das pesquisas.

De forma a optimizar estes tempos poderíamos criar indíces nas tabelas cujos campos são mais acedidos. Com isto o tempo de scan das tabelas iria diminuir. Poderíamos também utilizar as técnicas mencionadas anteriormente, como : shared_buffers, work_mem, etc.

7 Planos de execução

7.1 Pesquisa no tempo

Pesquisa sem indíces:

- cost = 1000.00..1126986.10
- rows = 1
- width = 108

Figura 7.1: Plano de execução da pesquisa

Pesquisa com um indíce no campo o_orderdate:

- cost = 172.62..54867.23
- rows = 1
- width = 108

Figura 7.2: Plano de execução da pesquisa

Para as seguintes experiências o resultado foi o mesmo:

- Indíce no campo o_orderdate
- Indíce no campo o_orderdate e o_totalprice
- Indíce no campo o_orderdate , o_totalprice e o_comment
- Indíce no campo o_orderdate , o_totalprice e o_comment e Indíce no campo o_clerk

Resultados:

- cost = 0.56..8.60
- rows = 1
- width = 108

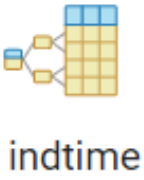

Figura 7.3: Plano de execução da pesquisa

7.2 Pesquisa nos produtos

Pesquisa sem indíces:

- cost = 1000.00..167015.10
- rows = 1
- width = 132

Para as seguintes experiências o resultado foi o mesmo:

- Indíce no campo p_name
- Indíce no campo p_name , p_mfgr e p_brand
- Indíce no campo p_name e Indíce no campo p_retailprice

Figura 7.4: Plano de execução da pesquisa

Resultados:

- cost = 0.56..8.59
- rows = 1
- width = 132

Figura 7.5: Plano de execução da pesquisa

7.3 Pesquisa nos suppliers

Pesquisa sem indíces:

- cost = 0.00..8176.17
- rows = 1
- width = 146

Figura 7.6: Plano de execução da pesquisa

Pesquisa com índice no campo s_name:

- cost = 0.42..8.45
- rows = 1
- width = 146

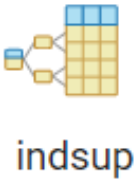

Figura 7.7: Plano de execução da pesquisa

Pesquisa com um índice nos campos s_name e s_comment e outro índice nos campos s_nationkey , s_address e s_phone:

- cost = 0.42..8.45
- rows = 1
- width = 146

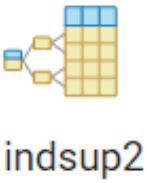

Figura 7.8: Plano de execução da pesquisa

7.4 Duas pesquisas mais rápidas e mais lentas

Algoritmos usados nas pesquisas mais lentas :

- 18 ª Querie

Figura 7.9: Plano de execução da 18ª Querie

  - Hash Join
  - Parallel Hash Join
  - Parallel Hash
  - Parallel Sec Scan
  - Finalize GroupAggregate
  - Gather Merge
  - Partial GroupAggregate
  - Sort

- 9 ª Querie

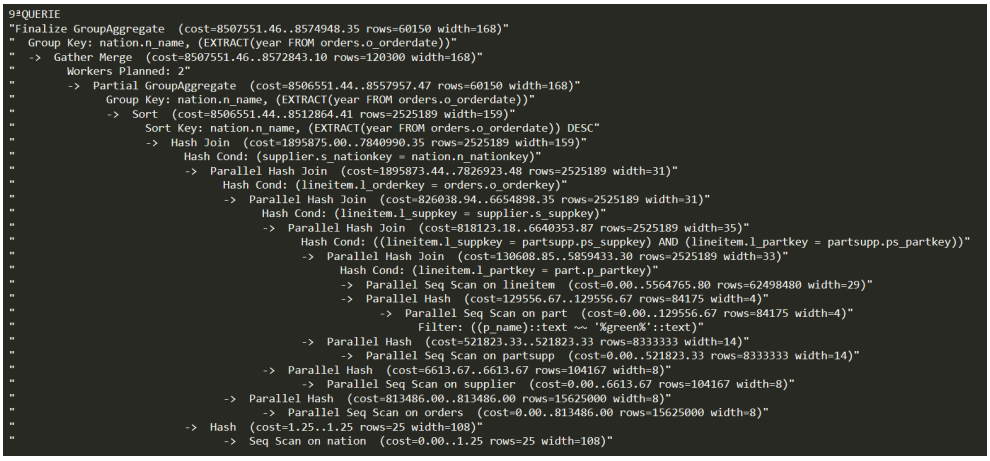

Figura 7.10: Plano de execução da 9ª Querie

- Parallel Hash Join
- Parallel Hash
- Seq Scan
- Parallel Index Only Scan
- GroupAggregate
- Sort
- Gather Merge

Algoritmos usados nas pesquisas mais rápidas :

- 16 ª Querie

```
QUERIE 16

"Sort  (cost=924191.62..924653.39 rows=184708 width=44)"
"  Sort Key: (count(DISTINCT partsupp.ps_suppkey)) DESC, part.p_brand, part.p_type, part.p_size"
"  ->  GroupAggregate  (cost=715930.55..902350.40 rows=184708 width=44)"
"        Group Key: part.p_brand, part.p_type, part.p_size"
"        ->  Gather Merge  (cost=715930.55..885908.72 rows=1459460 width=40)"
"              Workers Planned: 2"
"              ->  Sort  (cost=714930.53..716450.80 rows=608108 width=40)"
"                    Sort Key: part.p_brand, part.p_type, part.p_size"
"                    ->  Parallel Hash Join  (cost=170472.51..639881.19 rows=608108 width=40)"
"                          Hash Cond: (partsupp.ps_partkey = part.p_partkey)"
"                          ->  Parallel Index Only Scan using partsupp_pkey on partsupp  (cost=8697.50..432238.67 rows=4166667 width=8)"
"                                Filter: (NOT (hashed SubPlan 1))"
"                                SubPlan 1"
"                                  ->  Seq Scan on supplier  (cost=0.00..8697.00 rows=25 width=4)"
"                                        Filter: ((s_comment)::text ~~ '%Customer%Complaints%'::text)"
"                          ->  Parallel Hash  (cost=155598.33..155598.33 rows=304054 width=40)"
"                                ->  Parallel Seq Scan on part  (cost=0.00..155598.33 rows=304054 width=40)"
"                                      Filter: ((p_brand <> 'Brand#45'::bpchar) AND ((p_type)::text !~~ 'MEDIUM POLISHED%'::text) AND (p_size = ANY
('{49,14,23,45,19,3,36,9}'::integer[])))"
```

Figura 7.11: Plano de execução da 16ª Querie

- Parallel Hash Join
- Parallel Hash
- Seq Scan
- Parallel Index Only Scan
- GroupAggregate
- Sort
- Gather Merge

- 11 ª Querie

```
QUERIE 11

"Sort  (cost=1277635.70..1278011.07 rows=150147 width=36)"
"  Sort Key: (sum((partsupp.ps_supplycost * (partsupp.ps_availqty)::numeric))) DESC"
"  InitPlan 1 (returns $1)"
"    ->  Finalize Aggregate  (cost=564900.93..564900.94 rows=1 width=32)"
"          ->  Gather  (cost=564900.71..564900.92 rows=2 width=32)"
"                Workers Planned: 2"
"                ->  Partial Aggregate  (cost=563900.71..563900.72 rows=1 width=32)"
"                      ->  Parallel Hash Join  (cost=6986.88..561400.71 rows=333333 width=10)"
"                            Hash Cond: (partsupp_1.ps_suppkey = supplier_1.s_suppkey)"
"                            ->  Parallel Seq Scan on partsupp partsupp_1  (cost=0.00..521823.33 rows=8333333 width=14)"
"                            ->  Parallel Hash  (cost=6934.79..6934.79 rows=4167 width=4)"
"                                  ->  Hash Join  (cost=1.32..6934.79 rows=4167 width=4)"
"                                        Hash Cond: (supplier_1.s_nationkey = nation_1.n_nationkey)"
"                                        ->  Parallel Seq Scan on supplier supplier_1  (cost=0.00..6613.67 rows=104167 width=8)"
"                                        ->  Hash  (cost=1.31..1.31 rows=1 width=4)"
"                                              ->  Seq Scan on nation nation_1  (cost=0.00..1.31 rows=1 width=4)"
"                                                    Filter: (n_name = 'GERMANY'::bpchar)"
"  ->  Finalize GroupAggregate  (cost=598676.38..695716.11 rows=150147 width=36)"
"        Group Key: partsupp.ps_partkey"
"        Filter: (sum((partsupp.ps_supplycost * (partsupp.ps_availqty)::numeric)) > $1)"
"        ->  Gather Merge  (cost=598676.38..683959.50 rows=666666 width=36)"
"              Workers Planned: 2"
"              ->  Partial GroupAggregate  (cost=597676.35..606009.68 rows=333333 width=36)"
"                    Group Key: partsupp.ps_partkey"
"                    ->  Sort  (cost=597676.35..598509.68 rows=333333 width=14)"
"                          Sort Key: partsupp.ps_partkey"
"                          ->  Parallel Hash Join  (cost=6986.88..561400.71 rows=333333 width=14)"
"                                Hash Cond: (partsupp.ps_suppkey = supplier.s_suppkey)"
"                                ->  Parallel Seq Scan on partsupp  (cost=0.00..521823.33 rows=8333333 width=18)"
"                                ->  Parallel Hash  (cost=6934.79..6934.79 rows=4167 width=4)"
"                                      ->  Hash Join  (cost=1.32..6934.79 rows=4167 width=4)"
"                                            Hash Cond: (supplier.s_nationkey = nation.n_nationkey)"
"                                            ->  Parallel Seq Scan on supplier  (cost=0.00..6613.67 rows=104167 width=8)"
"                                            ->  Hash  (cost=1.31..1.31 rows=1 width=4)"
"                                                  ->  Seq Scan on nation  (cost=0.00..1.31 rows=1 width=4)"
"                                                        Filter: (n_name = 'GERMANY'::bpchar)"
```

Figura 7.12: Plano de execução da 11ª Querie

- Partial Aggregate
- Parallel Hash Join
- Parallel Seq Scan
- Parallel Hash
- Hash
- Sequential Scan
- Partial Group Aggregate
- Hash Join
- Finalize Aggregate
- Finalize Group Aggregate
- Sort

Onde as pesquisas mais lentas demoraram mais tempo :

- 18 ª Querie
  - Parallel Sequential Scan da tabela lineitem
  - Partial HashAggregate da tabela lineitem
  - Sequential Scan da tabela lineitem

- 9 ª Querie
  - Parallel Sequential Scan on lineitem

  - Parallel Sequential Scan on orders
  - Sort com sort key nation.n_name

Onde as pesquisas mais rápidas demoraram mais tempo :

- 16 ª Querie
  - Parallel Index Only Scan using partsupp_pkey on partsupp
  - Parallel Hash Join
  - Parallel Sequential Scan on part

- 11 ª Querie
  - Parallel Sequential Scan on partsupp

Ao analisar os planos de execução das pesquisas mais lentas percebemos que estas demoram bastante porque perdem muito tempo no sequential scan da tabela lineitem. Isto deve-se ao facto desta tabela ter uma quantidade enorme de dados em comparação com as outras. Devido a isto todos os algoritmos usados aseguir pelo motor de base de dados têm de lidar com uma quantidade enorme de dados retirada desta tabela. Outro factor para o tempo elevado é a utilização da chave forasteira da tabela lineitem para fazer o join com outras tabelas. Já as pesquisas mais rápidas usam as tabelas com menor quantidade de dados.

8 Reconhecimentos

Para o desenvolvimento e facilitação da obtenção dos resultados para este trabalho, foi utilizado um script em python fornecido e desenvolvido por Cláudia Campos e Beatriz Madeira, com os números 2018285941 e 2018280169 respectivamente.

References

[1] *An Introduction to PostgreSQL Performance Tuning and Optimization*. URL: https://www.enterprisedb.com/postgres-tutorials/introduction-postgresql-performance-tuning-and-optimization.

[2] *Resource Consumption - Memory*. URL: https://www.postgresql.org/docs/9.1/runtime-config-resource.html.

[3] *An Introduction to PostgreSQL Performance Tuning and Optimization*. URL: https://www.postgresql.org/docs/current/runtime-config-query.html.

[4] *Using AMM*. URL: https://docs.oracle.com/cd/B28359_01/server.111/b28310/memory003.htm#ADMIN11201.