FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Sistemas de Gestão de Dados

2020/2021 - 2º Semestre

SSB Perf Bench and Processing Design

Grupo 14
Francisco Miranda - 2015250592
Maria Paula Viegas - 2017125592

Index

Avaliação do Grupo

- O grupo foi capaz de gerar e carregar a SSB benchmark, avaliando a performance da mesma. Foram produzidos gráficos para:
    - tempo de carregamento sem chaves
    - tempo de query sem busca por chave
    - tempo de criação para cada chave PK e FK
    - tempo de query com busca por chaves
    - tempo médio de linhas por segundo e MB por segundo
    - query execution plans
- Autoavaliação do grupo
    - 20 Valores
- Contribuição individual
    - Francisco: Relatório, automação extração dados
    - Maria Paula: Relatório, produção automatizada de plots
- Autoavaliação individual
    - Francisco: 20 Valores
    - Maria Paula: 20 Valores
- Tempo de esforço
    - Francisco: 12 hrs
    - Maria Paula: 12 hrs

Introdução

Para este projeto da cadeira de Sistemas de Gestão de Dados, gaermos e carregamos a Star Schema Benchmark (SSB) para uma base de dados PostgreSQL e avaliamos a performance com o intuito de aprender sobre benchmarking de bases de dados e SSB.

Geramos a Star Schema Benchmark e criamos as tabelas no postgres conforme fizemos na aula prática a partir do https://github.com/electrum/ssb-dbgen.

Para a avaliação, temos dois ficheiros python: *collect_data.py*, para automatizar a coleção de dados experimentais conforme desejamos, e *load_time_plots.py*, para processar a informação coletada e gerar os gráficos pedidos.

Em *collect_data.py*, determinamos a base de dados para ser conectada do postgres, criamos uma base de dados SQLite *benchmark_data.db* para armazenar os resultados e inserimos quantas vezes ele deverá repetir o processo de coleta de dados se repete. Para cada repetição o script elimina todos os dados presentes fazendo um drop do schema e reconstruindo as tabelas. De seguida faz o processo de carregamento dos dados, execução dos queries sem keys, construção das keys e reexecução dos queries. Para cada uma das etapas mede o tempo demorado em cada passo.

A experiência foi realizada 50 vezes num portátil com as seguintes especificações:

- Windows 10 Home
- PostgreSQL 10
- 32GB RAM DDR4, 3200Mbps
- 1TB SSD 3400 MB/s
- Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz, 2304 Mhz, 8 Core(s)

Benchmarking

Benchmarking tem como objetivo medir a capacidade dos sistemas em quatro diferentes etapas: esquema e carga de trabalho, geração de dados, parâmetros e métricas e validação. Os três principais benchmarks para data warehouses são TCP-DS, TCP-H e SSB e cada um deles têm especificações que devem ser estritamente seguidas. Um benchmark deve:

- coletar métricas relevantes;
- ser facilmente adaptável a diferentes sistemas;
- ser escalável (considerar volumes de dados diferentes);
- apresentar resultados simples e fáceis de entender;
- gerar os mesmos dados quando executado múltiplas vezes em mesmas condições.

SSB

Star Schema Benchmark é um benchmark que define uma aplicação de data warehouse centralizado. Esse tipo de aplicação tem agilidade para captar e utilizar dados, visto que armazena os dados pelo esquema estrela. Além disso, são aplicações com grandes quantidades de dados de um determinado período de tempo, sendo assim capazes de oferecer análises históricas que preveem tendências e auxiliam no planejamento a longo e curto prazo.

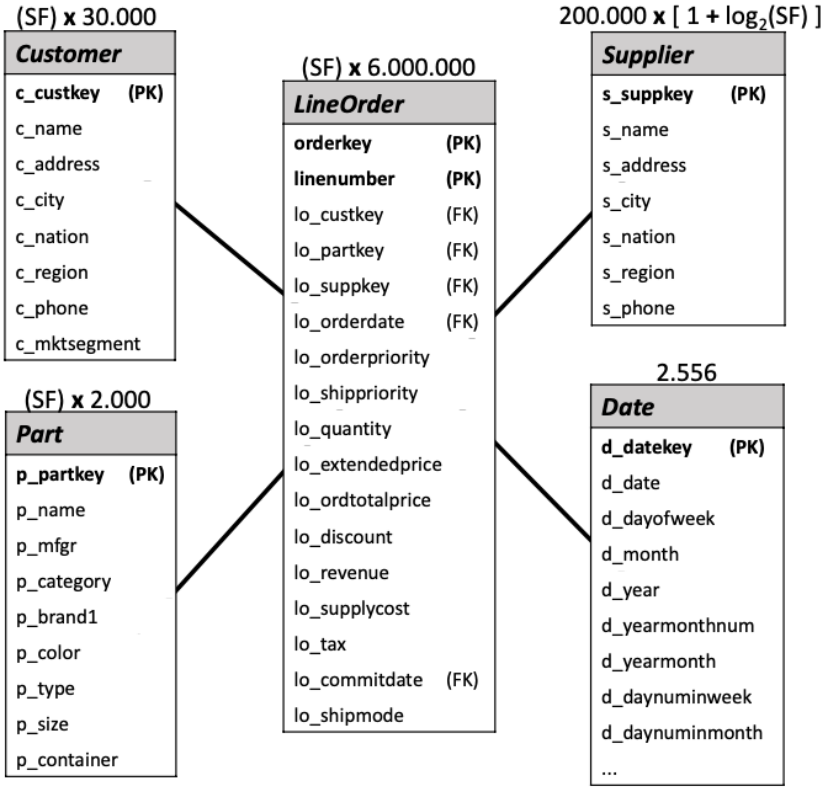

Imagem 1: Star Schema Benchmark

É uma extensão do TPC-H com esquema adaptado para estrela. Ele tem dimensões desnormalizadas (apresenta dados redundantes), permitindo melhor desempenho no processamento de consultas e facilidade de entendimento.

Gráficos de avaliação

tempo de carregamento (load time graph)

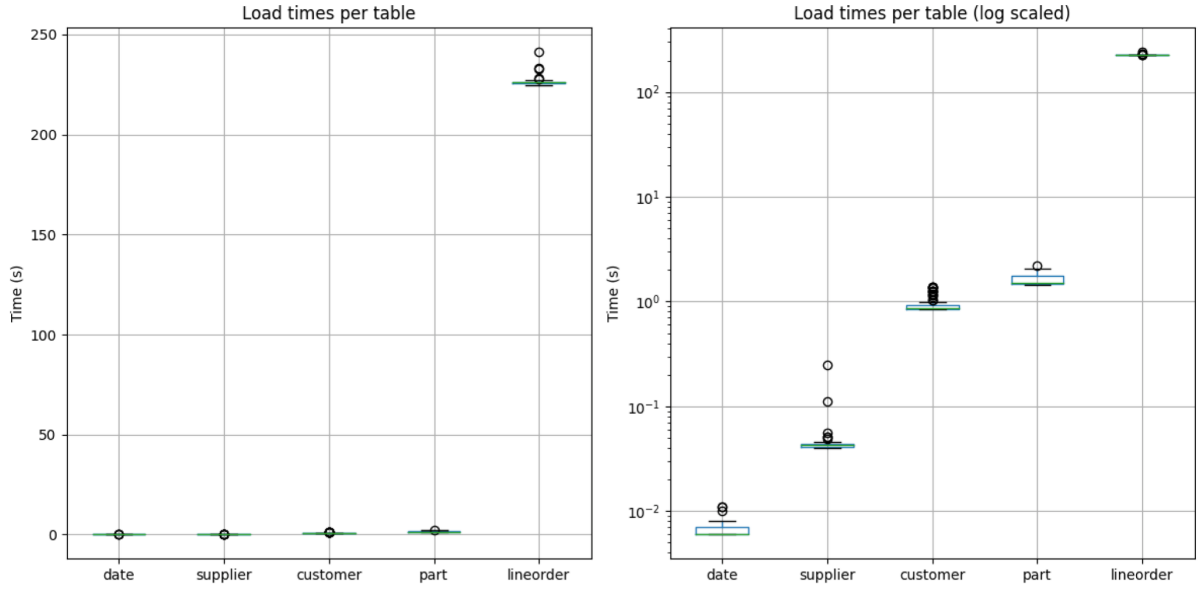

Gráfico 1 - load time graph (without keys)

Mean load time (s)	
date	0.006603
supplier	0.046974
customer	0.943135
part	1.621631
lineorder	226.311698

Imagem 2 - tempo de carregamento das tabelas

Os gráficos acima apresentam o tempo de carregamento de cada tabela sem o uso de chaves. À direita temos o mesmo gráfico porém com uma escala logarítmica para melhor visualização.

Percebemos que a tabela *lineorder* tem um tempo de carregamento muito maior que as outras tabelas. Isto acontece porque é o centro do esquema de estrela e, consequentemente, a maior tabela da base de dados. A tabela *date* tem o menor tempo por ser a menor tabela.

average rates

nº rows/second

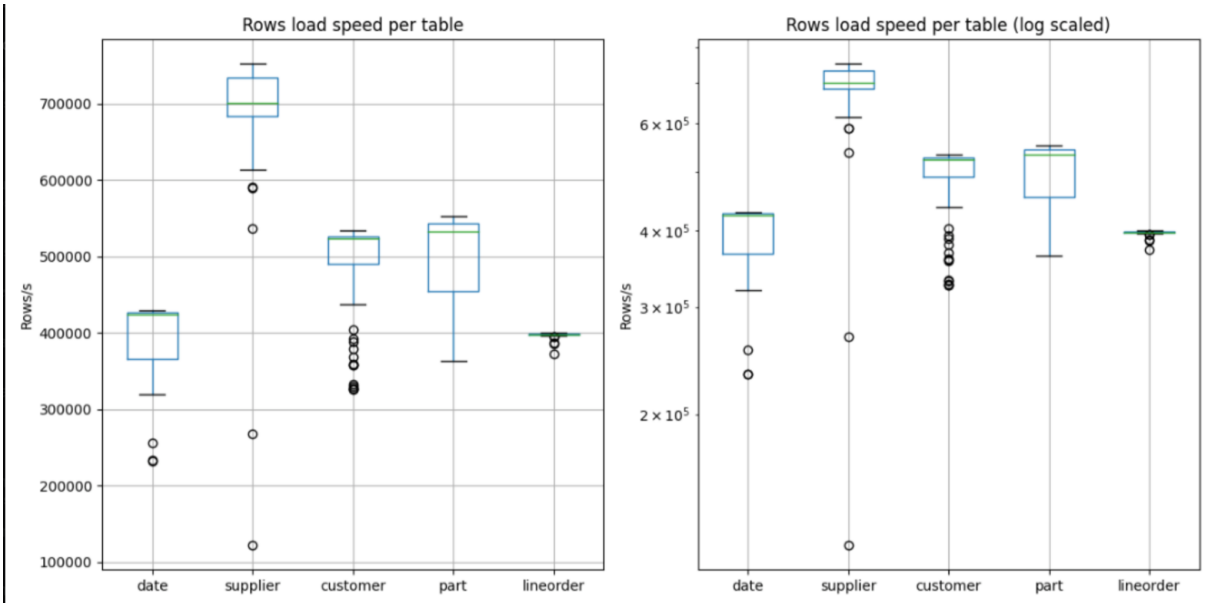

Gráfico 4 - rows/table

Mean Rows Load Speed(Rows/s)		Standard Deviation Rows Load Speed(Rows/s)	
date	394319.960216	date	46476.950885
supplier	45684.924653	supplier	6383.207568
customer	32580.941727	customer	4489.125244
part	125000.550696	part	13765.291167
lineorder	26519.770665	lineorder	254.431092

Imagem 4 - n linhas/segundo

Nesse gráfico, temos uma média de quantas linhas são carregadas por segundo por tabela, sendo o gráfico da direita com uma escala diferente. Observamos que o supplier apresenta variações mais drásticas e a média mais alta por ter uma cardinalidade em função de uma escala logarítmica. A diferente escala impacta no número de linhas geradas pois quanto maior o fator de escala, mais linhas são adicionadas.

MB/second

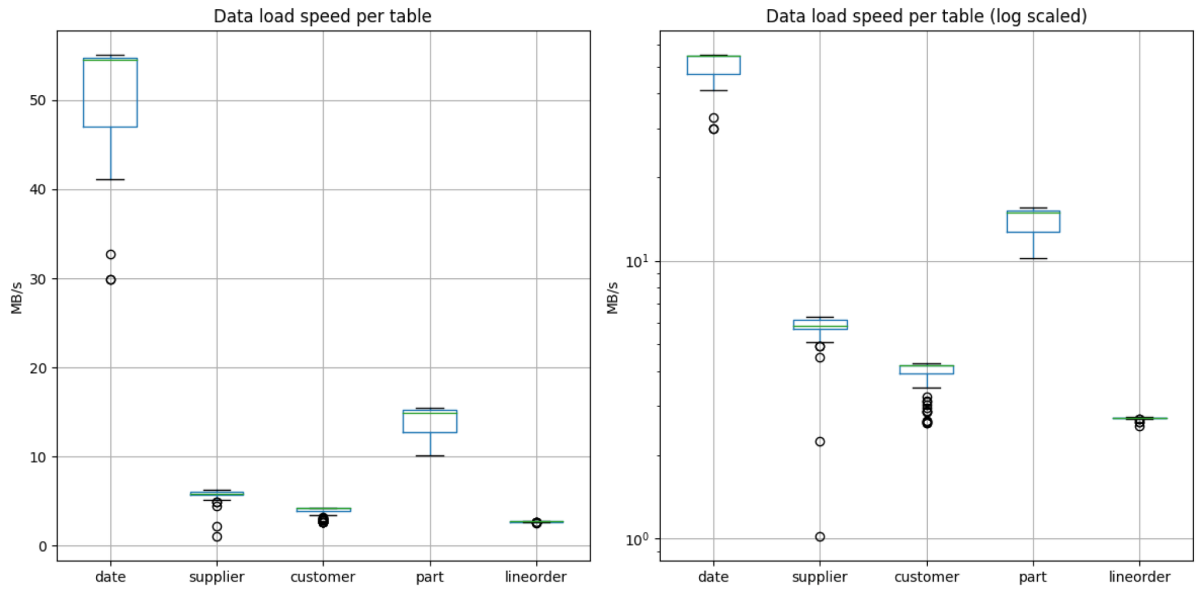

Gráfico 5 - MB/s

Mean Data Transfer Speed(MB/s)		Std Dev Data Transfer Speed(MB/s)	
date	50.600797	date	5.964118
supplier	5.710616	supplier	0.797901
customer	3.911410	customer	0.538929
part	14.00885	part	1.542680
lineorder	2.708911	lineorder	0.025989

Imagem 5 - velocidade transferência

Aqui observamos as velocidades de transferência de cada tabela em MB/segundo. A tabela *date* tem maior velocidade, seguida pela tabela *part*. Essa informação está dentro do esperado visto que são as menores tabelas e *date* inclusive tem um valor fixo de dados gerados (7 anos em dias), ou seja, não tem um fator de escala SF (Imagem 1). A variação da average rate também ocorre por causa do tempo de inicialização do processo que prevalece em tabelas menores. De maneira oposta, *lineorder* tem menores variações no diagrama porque tabelas maiores demoram mais tempo para serem processadas e então o impacto dos overheads deixa de ser notável.

tempo de query (query time graph)

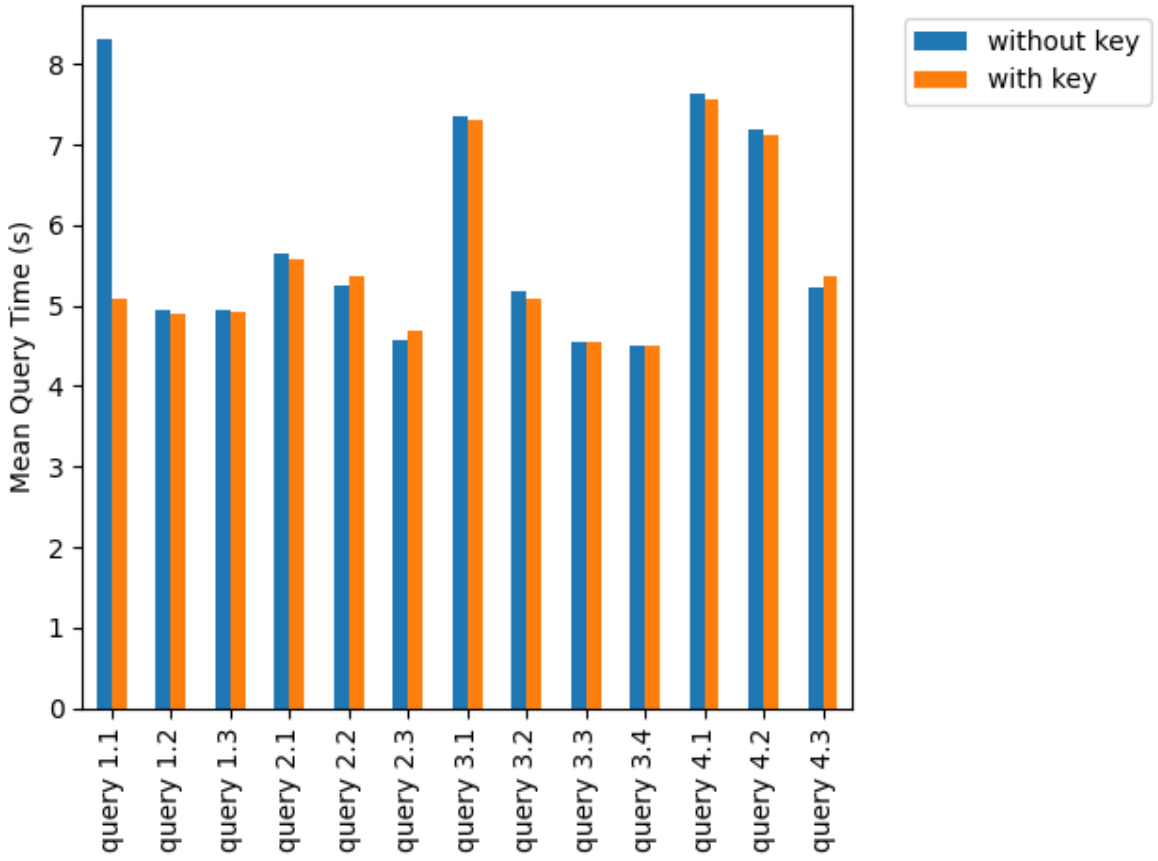

Gráfico 2 - query time graph

Sendo query time o tempo a espera da base de dados retornar a informação pedida, temos no Gráfico 2 as query times para as diferentes queries e comparamos o tempo com e sem busca por chave. As queries são numeradas de acordo com os slides fornecidos pelo professor (*lab_Slides1_SSB.pdf*).

Observamos que a maioria dos tempos são semelhantes com ou sem chave com exceção do query 0.

tempo de construção das chaves (key times)

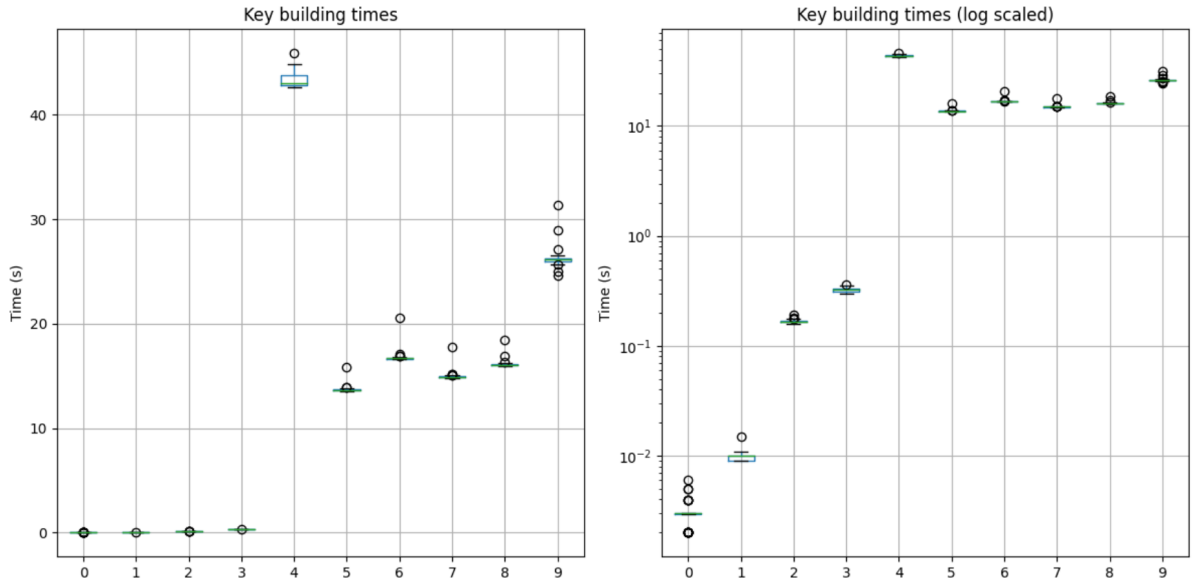

Gráfico 3 - key times

Mean key building time(s)		Standard Deviation key building time(s)	
key 0	0.002932	key 0	0.000721
key 1	0.009758	key 1	0.000819
key 2	0.167253	key 2	0.004646
key 3	0.323817	key 3	0.012889
key 4	43.265808	key 4	0.585887
key 5	13.698145	key 5	0.280330
key 6	16.767135	key 6	0.476296
key 7	14.977188	key 7	0.345306
key 8	16.103751	key 8	0.314345
key 9	26.208902	key 9	0.783280

Imagem 3 - tempo de construção das chaves

Os gráficos acima apresentam um diagrama de caixas com os tempos de construção de cada uma das 10 chaves. O gráfico da direita tem uma escala logarítmica para uma melhor visualização das primeiras chaves. As chaves de 0 a 4 são primárias (PK) e as chaves de 5 a 9 são estrangeiras (FK) - informação encontrada no ficheiro *ssb.ri*.

Como esperado, as chaves PK têm um tempo de criação menor que as FK, visto que as FKs são da grande tabela central *lineorder* (Imagem 1). Porém, a chave PK 4 tem um valor muito alto mesmo para FKs. Isso pode ser explicado por também ser da tabela *lineorder* e por se tratar de uma primary key complexa (*lo_orderkey*, *lo_linenumber*).

query execution plans

query 1.1

```
+-----------------------------------------------------------------------------------------------------+
|QUERY PLAN                                                                                           |
+-----------------------------------------------------------------------------------------------------+
|Finalize Aggregate  (cost=1847913.28..1847913.29 rows=1 width=8)                                     |
|  ->  Gather  (cost=1847913.06..1847913.27 rows=2 width=8)                                           |
|        Workers Planned: 2                                                                           |
|        ->  Partial Aggregate  (cost=1846913.06..1846913.07 rows=1 width=8)                          |
|              ->  Hash Join  (cost=74.53..1845188.48 rows=689833 width=4)                            |
|                    Hash Cond: (lineorder.lo_orderdate = date.d_datekey)                             |
|                    ->  Parallel Seq Scan on lineorder  (cost=0.00..1832407.33 rows=4832611 width=8)|
|                          Filter: ((lo_discount >= 1) AND (lo_discount <= 3) AND (lo_quantity < 25))|
|                    ->  Hash  (cost=69.96..69.96 rows=365 width=4)                                   |
|                          ->  Seq Scan on date  (cost=0.00..69.96 rows=365 width=4)                  |
|                                Filter: (d_year = 1993)                                              |
+-----------------------------------------------------------------------------------------------------+
```

Vemos pelo query execution plan que é feito uma agregação parcial em paralelo de 2 *workers*, em que cada worker faz:

1. Hash de todas entradas *year* da tabela *date* para depois pesquisar entradas com *year=1993.* Foram encontradas 365 linhas com essa condição
2. Filter de todas as entradas da tabela *lineorder* seguindo o filtro da query ((lo_discount >= 1) AND (*lo_discount* <= 3) AND (*lo_quantity* < 25)). 4832611 linhas respeitam essa regra.
3. Join das tabelas *lineorder* e *date* com o hash de *lineorder.lo_orderdate* = *date.d_datekey*, retornando 689833 linhas.
4. A agregação parcial, ou seja, o *sum(lo_revenue)* que reduz o número de linhas para 1.

Finalmente, temos a agregação final ao juntar as duas agregações parciais que retorna 1 linha de 8 bytes.

query 2.1

```
+-----------------------------------------------------------------------------------------------------------------+
|QUERY PLAN                                                                                                       |
+-----------------------------------------------------------------------------------------------------------------+
|Finalize GroupAggregate  (cost=1683186.70..1683396.70 rows=7000 width=21)                                        |
|  Group Key: date.d_year, part.p_brand1                                                                          |
|  ->  Sort  (cost=1683186.70..1683221.70 rows=14000 width=21)                                                    |
|        Sort Key: date.d_year, part.p_brand1                                                                     |
|        ->  Gather  (cost=1680752.58..1682222.58 rows=14000 width=21)                                            |
|              Workers Planned: 2                                                                                 |
|              ->  Partial HashAggregate  (cost=1679752.58..1679822.58 rows=7000 width=21)                       |
|                    Group Key: date.d_year, part.p_brand1                                                        |
|                    ->  Hash Left Join  (cost=22843.27..1677373.19 rows=317252 width=17)                         |
|                          Hash Cond: (lineorder.lo_orderdate = date.d_datekey)                                   |
|                          ->  Hash Join  (cost=22747.74..1676443.49 rows=317252 width=17)                        |
|                                Hash Cond: (lineorder.lo_suppkey = supplier.s_suppkey)                           |
|                                ->  Hash Join  (cost=21878.00..1671449.80 rows=1570811 width=21)                 |
|                                      Hash Cond: (lineorder.lo_partkey = part.p_partkey)                         |
|                                      ->  Parallel Seq Scan on lineorder  (cost=0.00..1551161.27                 |
|                                            rows=37489527 width=16)                                              |
|                                      ->  Hash  (cost=21459.00..21459.00 rows=33520 width=13)                    |
|                                            ->  Seq Scan on part  (cost=0.00..21459.00 rows=33520 width=13)   |
|                                                  Filter: ((p_category)::text = 'MFGR#12'::text)                 |
|                                ->  Hash  (cost=794.00..794.00 rows=6059 width=4)                                |
|                                      ->  Seq Scan on supplier  (cost=0.00..794.00 rows=6059 width=4)            |
|                                            Filter: ((s_region)::text = 'AMERICA'::text)                         |
|                          ->  Hash  (cost=63.57..63.57 rows=2557 width=8)                                        |
|                                ->  Seq Scan on date  (cost=0.00..63.57 rows=2557 width=8)                       |
+-----------------------------------------------------------------------------------------------------------------|
```

Vemos pelo query execution plan que é feito uma *HashAggregate* parcial em paralelo de 2 workers. Cada worker realiza as seguintes operações:

1. Filtra partes da tabela em que *p_category* = 'MFGR#12' com um hash e scan sequencial. 6059 linhas foram retornadas
2. Junta as tabelas *lineorder* e *parts* com um Hash Join em que *lineorder.lo_partkey* = *part.p_partkey.* Retorna-se 1570811 linhas.
3. Filtra a tabela *supplier* (*s_region* = 'AMERICA') com um hash e scan sequencial paralelo para termos 6059 linhas.
4. Junta a tabela resultante do ponto 2 acima com a tabela *supplier* em que *lineorder.lo_suppkey* = *supplier.s_suppkey*. 317252 linhas foram retornadas
5. Junta a tabela do ponto 4 com a tabela *date* em que *lineorder.lo_orderdate* = *date.d_datekey.* 317252 linhas foram retornadas.
6. Calcula as partial aggregates para os hash dos grupos de pares distintos (*date.d_year*, *part.p_brand1*). Tem um retorno de 7000 linhas.

Finalmente, temos a *GroupAggregate* final ao somar os resultados dos 2 workers, que retorna 7000 linhas de 21 bytes.

query 2.2

```
+-----------------------------------------------------------------------------------------------------------------+
|QUERY PLAN                                                                                                       |
+-----------------------------------------------------------------------------------------------------------------+
|Finalize GroupAggregate  (cost=1674789.60..1674795.46 rows=21 width=21)                                          |
|  Group Key: date.d_year, part.p_brand1                                                                          |
|  ->  Gather Merge  (cost=1674789.60..1674794.94 rows=42 width=21)                                               |
|        Workers Planned: 2                                                                                       |
|        ->  Partial GroupAggregate  (cost=1673789.58..1673790.07 rows=21 width=21)                               |
|              Group Key: date.d_year, part.p_brand1                                                              |
|              ->  Sort  (cost=1673789.58..1673789.65 rows=28 width=17)                                           |
|                    Sort Key: date.d_year, part.p_brand1                                                         |
|                    ->  Hash Left Join  (cost=23554.86..1673788.90 rows=28 width=17)                             |
|                          Hash Cond: (lineorder.lo_orderdate = date.d_datekey)                                   |
|                          ->  Nested Loop  (cost=23459.32..1673693.30 rows=28 width=17)                          |
|                                ->  Hash Join  (cost=23459.04..1673030.84 rows=140 width=21)                     |
|                                      Hash Cond: (lineorder.lo_partkey = part.p_partkey)                         |
|                                      ->  Parallel Seq Scan on lineorder  (cost=0.00..1551161.27                 |
|                                           rows=37489527 width=16)                                               |
|                                      ->  Hash  (cost=23459.00..23459.00 rows=3 width=13)                        |
|                                            ->  Seq Scan on part  (cost=0.00..23459.00 rows=3 width=13)          |
|                                                  Filter: (((p_brand1)::text >= 'MFGR#2221'::text) AND           |
|                                                     ((p_brand1)::text <= 'MFGR#2228'::text))                    |
|                                ->  Index Scan using supplier_pkey on supplier  (cost=0.29..4.73 rows=1          |
|                                      width=4)                                                                   |
|                                      Index Cond: (s_suppkey = lineorder.lo_suppkey)                             |
|                                      Filter: ((s_region)::text = 'ASIA'::text)                                  |
|                          ->  Hash  (cost=63.57..63.57 rows=2557 width=8)                                        |
|                                ->  Seq Scan on date  (cost=0.00..63.57 rows=2557 width=8)                       |
+-----------------------------------------------------------------------------------------------------------------+
```

Vemos pelo query execution plan que é feito um *GroupAggregate* em paralelo de 2 workers. Cada worker realiza as seguintes operações:

1. Um hash e um scan sequencial que filtra resultados da tabela *part* em que "*p_brand1 between* 'MFGR#2221' *and* 'MFGR#2228'. Após esse filtro, temos 3 linhas.
2. Juntamos a tabela *lineorder* com os resultados do ponto 1 em que *lineorder.lo_partkey = part.p_partkey.* Retornamos 140 linhas
3. Com a chave primária (PK) da tabela *supplier* como index, filtramos a tabela *supplier* em que s_region = 'ASIA' e temos 1 linha.
4. Juntamos os resultados dos pontos 2 e 3 acima com um nested loop e *lineorder.lo_orderdate = date.d_datekey*. O retorno é de 28 linhas.
5. Juntamos a tabela *date* com os resultados de 4, estabelecendo que *lineorder.lo_orderdate = date.d_datekey.* Ainda temos 28 linhas de resposta.
6. Organizamos o resultado anterior por *date.d_year* e *part.p_brand.*
7. Um GroupAggregate parcial que produz 21 linhas com o group key *date.d_ year, part.p_brand1*

Finalmente, temos um *Gather Merge* e um GroupAggregate final dos 2 workers, que retorna 21 linhas de 21 bytes.

query 3.1

```
+-----------------------------------------------------------------------------------------------------------------+
|QUERY PLAN                                                                                                       |
+-----------------------------------------------------------------------------------------------------------------+
|Sort  (cost=1793542.97..1793553.91 rows=4375 width=28)                                                           |
|  Sort Key: date.d_year, (sum(lineorder.lo_revenue)) DESC                                                        |
|  ->  Finalize GroupAggregate  (cost=1793125.27..1793278.39 rows=4375 width=28)                                  |
|        Group Key: customer.c_nation, supplier.s_nation, date.d_year                                             |
|        ->  Sort  (cost=1793125.27..1793147.14 rows=8750 width=28)                                               |
|              Sort Key: customer.c_nation, supplier.s_nation, date.d_year                                        |
|              ->  Gather  (cost=1791633.61..1792552.36 rows=8750 width=28)                                       |
|                    Workers Planned: 2                                                                           |
|                    ->  Partial HashAggregate  (cost=1790633.61..1790677.36 rows=4375 width=28)                  |
|                          Group Key: customer.c_nation, supplier.s_nation, date.d_year                           |
|                          ->  Hash Join  (cost=15048.19..1777405.82 rows=1322779 width=24)                       |
|                                Hash Cond: (lineorder.lo_orderdate = date.d_datekey)                             |
|                                ->  Hash Join  (cost=14944.44..1773244.88 rows=1543040 width=24)                 |
|                                      Hash Cond: (lineorder.lo_custkey = customer.c_custkey)                     |
|                                      ->  Hash Join  (cost=869.50..1650454.54 rows=7547891 width=20)             |
|                                            Hash Cond: (lineorder.lo_suppkey = supplier.s_suppkey)               |
|                                            ->  Parallel Seq Scan on lineorder  (cost=0.00..1551161.27       |
|                                                  rows=37489527 width=16)                                        |
|                                            ->  Hash  (cost=794.00..794.00 rows=6040 width=12)                   |
|                                                  ->  Seq Scan on supplier  (cost=0.00..794.00 rows=6040     |
|                                                        width=12)                                                |
|                                                        Filter: ((s_region)::text = 'ASIA'::text)                |
|                                      ->  Hash  (cost=12475.00..12475.00 rows=91995 width=12)                    |
|                                            ->  Seq Scan on customer  (cost=0.00..12475.00 rows=91995        |
|                                              width=12)                                                          |
|                                              Filter: ((c_region)::text = 'ASIA'::text)                          |
|                                ->  Hash  (cost=76.35..76.35 rows=2192 width=8)                                  |
|                                      ->  Seq Scan on date  (cost=0.00..76.35 rows=2192 width=8)                 |
|                                            Filter: ((d_year >= 1992) AND (d_year <= 1997))                      |
|+----------------------------------------------------------------------------------------------------------------+
```

Vemos pelo query execution plan que é feito um *HashAggregate* parcial em paralelo com 2 workers. Cada worker realiza as seguintes operações:

1. Filtra a tabela *supplier* de maneira que *s_region* = 'ASIA' com um hash e scan sequencial. São produzidos 6040 resultados.
2. Junta as tabelas *lineorder* e *supplier* tal que *lineorder.lo_suppkey* = *supplier.s_suppkey* com o Parallel Seq Scan. Isto resulta em 7547891 linhas.
3. Filtra os resultados da tabela *customer* tal que *c_region* = 'ASIA' com um Hash Sequential Scan, que retorna 91995 linhas.
4. Junta as tabelas produzidas nos pontos 2 e 3 tal que *lineorder.lo_custkey* = *customer.c_custkey*, que soma 1543040 linhas.
5. Filtra a tabela *date* ((*d_year* >= 1992) AND (*d_year* <= 1997)). O hash desse filtro resulta em 2192 linhas.
6. Junta as tabelas dos pontos 4 e 5 tal que *lineorder.lo_orderdate* = *date.d_datekey* que resulta em 1322779 linhas.
7. Agrupa o resultado do ponto 6 por *customer.c_nation*, *supplier.s_nation* and *date.d_year* que retorna 4375 linhas.

Quando cada worker finaliza, juntamos eles com um Gather e ordenamos segundo *customer.c_nation*, *supplier.s_nation*, *date.d_year.* Finalmente, temos um GroupAggregate com o group key *customer.c_nation*, *supplier.s_nation*, *date.d_year* e ordenamos os resultados por *date.d_year* de forma ascendente e (sum(lineorder.lo_revenue)) de forma descendente. São retornadas 4375 linhas de 28 bytes.

query 4.1

```
+-----------------------------------------------------------------------------------------------------------------------+
|QUERY PLAN                                                                                                             |
+-----------------------------------------------------------------------------------------------------------------------+
|Finalize GroupAggregate  (cost=1832680.12..1832686.68 rows=175 width=20)                                               |
|  Group Key: date.d_year, customer.c_nation                                                                            |
|  ->  Sort  (cost=1832680.12..1832680.99 rows=350 width=28)                                                            |
|        Sort Key: date.d_year, customer.c_nation                                                                       |
|        ->  Gather  (cost=1832628.58..1832665.33 rows=350 width=28)                                                    |
|              Workers Planned: 2                                                                                       |
|              ->  Partial HashAggregate  (cost=1831628.58..1831630.33 rows=175 width=28)                              |
|                    Group Key: date.d_year, customer.c_nation                                                          |
|                    ->  Hash Left Join  (cost=43177.90..1826176.29 rows=545229 width=20)                               |
|                          Hash Cond: (lineorder.lo_orderdate = date.d_datekey)                                         |
|                          ->  Hash Join  (cost=43082.36..1824647.16 rows=545229 width=20)                              |
|                                Hash Cond: (lineorder.lo_partkey = part.p_partkey)                                     |
|                                ->  Hash Join  (cost=14916.24..1773551.01 rows=1520384 width=24)                       |
|                                      Hash Cond: (lineorder.lo_custkey = customer.c_custkey)                           |
|                                      ->  Hash Join  (cost=869.74..1650454.77 rows=7571635 width=20)                   |
|                                            Hash Cond: (lineorder.lo_suppkey = supplier.s_suppkey)                     |
|                                            ->  Parallel Seq Scan on lineorder  (cost=0.00..1551161.27                 |
|                                                  rows=37489527 width=24)                                              |
|                                            ->  Hash  (cost=794.00..794.00 rows=6059 width=4)                          |
|                                                  ->  Seq Scan on supplier  (cost=0.00..794.00 rows=6059               |
|                                                        width=4)                                                       |
|                                                        Filter: ((s_region)::text = 'AMERICA'::text)                   |
|                                      ->  Hash  (cost=12475.00..12475.00 rows=90360 width=12)                          |
|                                            ->  Seq Scan on customer  (cost=0.00..12475.00 rows=90360                  |
|                                                  width=12)                                                            |
|                                                  Filter: ((c_region)::text = 'AMERICA'::text)                         |
|                                ->  Hash  (cost=23459.00..23459.00 rows=286890 width=4)                                |
|                                      ->  Seq Scan on part  (cost=0.00..23459.00 rows=286890 width=4)                  |
|                                            Filter: (((p_mfgr)::text = 'MFGR#1'::text) OR ((p_mfgr)::text =            |
|                                              'MFGR#2'::text))                                                         |
|                          ->  Hash  (cost=63.57..63.57 rows=2557 width=8)                                              |
|                                ->  Seq Scan on date  (cost=0.00..63.57 rows=2557 width=8)                             |
+-----------------------------------------------------------------------------------------------------------------------+
```

Vemos pelo query execution plan que é feito um Partial HashAggregate em paralelo de 2 workers com as group key *date*.*d_year*, *customer*.*c_nation*. Cada worker realiza as seguintes operações:

1. Filtra a tabela *supplier* tal que *s_region* = 'AMERICA'. É feito um hash que resulta em 6059 linhas.
2. Junta as tabelas *lineorder* e *supplier* tal que *lineorder*.*lo_custkey* = *customer*.*c_custkey.* O retorno é de 7571635 linhas.
3. Filtra a tabela *customer* tal que c_region = 'AMERICA' por meio de um scan e um hash que resulta em 90360 linhas.
4. Junta os resultados dos pontos 2 e 3 acima tal que *lineorder*.*lo_custkey* = *customer*.*c_custkey* em 1520384 linhas.
5. Filtra a tabela *part* tal que (p_mfgr = 'MFGR#1' OR *p_mfgr* = 'MFGR#2') com um scan e um hash que produz 286890 linhas.
6. Junta os resultados dos pontos 4 e 5 tal que *lineorder*.*lo_partkey* = *part*.*p_partkey*. Retorna 545229 linhas.
7. Junta os resultados do ponto 6 com a tabela *date* tal que *lineorder*.*lo_orderdate* = *date*.*d_datekey*
8. Agrupa os resultados de 7 por *date*.*d_year* e *customer*.*c_nation* em 175 linhas.

Quando cada worker finaliza, juntamos eles com um Gather e ordenamos segundo *date.d_year*, *customer.c_nation*. Finalmente, temos um GroupAggregate com o group key *date.d_year*, *customer.c_nation* que retorna 175 linhas de 20 byte

Conclusão

Neste trabalho, entendemos como funcionam os benchmarks e em especial, Star Schema Benchmark. Aprendemos a analisar os gráficos de performance, olhar outliers e ler query execution plans. Observamos que chaves FK e PK podem melhorar determinados aspectos, apesar de nem sempre serem necessários e que o tempo de criação delas também é relevante. Concluímos que é importante fazer uma profunda análise de performance da base de dados para entender o processo, decidir o melhor sistema para as nossas necessidades e como otimizá-la.