Administração de Bases de Dados (4º ano de MIEI)

Configuração, Otimização e Avaliação do *benchmark* TPC-C

Relatório de Desenvolvimento

Francisco Reinolds (a82982)
Luís Alves (a80165)
Rafaela Rodrigues (a80516)

8 de Janeiro de 2021

Resumo

Este relatório inicia-se com uma breve contextualização, seguindo-se a declaração dos objetivos deste trabalho e em que é que este consiste. De seguida é explicitado o processo de obtenção da configuração de referência, bem como os resultados obtidos através da alteração dos parâmetros de configuração do PostgreSQL. No capítulo seguinte são abordadas as interrogações analíticas adicionais. São apresentados os planos de execução originais e as abordagens do grupo para melhorar o seu desempenho, sendo também expostos os resultados obtidos com as alterações propostas. Por fim, é descrito o processo de replicação testado, sendo feita a comparação com a versão de apenas um servidor. Termina-se este relatório com uma análise crítica sobre o trabalho efetuado, bem como com as dificuldades experienciadas no desenvolvimento deste projeto.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização

O presente relatório foi elaborado no âmbito do Trabalho Prático da Unidade Curricular de Administração de Bases de Dados, que se insere no 1º semestre do 4º ano do Mestrado Integrado em Engenharia Informática.

1.2 Objetivos e Trabalho Proposto

Tendo em conta a adaptação do *benchmark* TPC-C, pretende-se com este trabalho otimizar o desempenho da carga transacional através dos parâmetros de configuração do PostgreSQL, otimizar o desempenho das interrogações analíticas tendo em conta os respetivos planos e mecanismos de redundância utilizados e, por fim, testar uma configuração com recurso a replicação ou processamento distribuído.

Capítulo 2

Otimização do Desempenho da Carga Transacional

Neste capítulo é abordado o processo de obtenção da configuração de referência a partir da qual se otimizou a carga transacional. Para isso foram alterados os parâmetros de configuração do PostgreSQL.

2.1 Configuração de referência

Antes de iniciar a otimização, foi necessário encontrar uma configuração de referência. Começamos por criar uma máquina virtual do tipo *n1-standard-2* (2 vCPUs, 7.5 GB de memória), que possui um disco de arranque de 50GB e um disco extra de 100GB, ambos do tipo *SSD persistent*.

O disco extra é utilizado para armazenar a base de dados, sendo que o tamanho e o tipo explicitados acima são importantes pois a performance do disco tem um grande impacto nos resultados. Podemos observar nas figuras 2.1 e 2.2 que, para um mesmo tamanho, um *SSD persistent disk* tem o limite de IOPS (*Input/Output Operations Per Second*) 40 vezes superior a um disco *standard*.

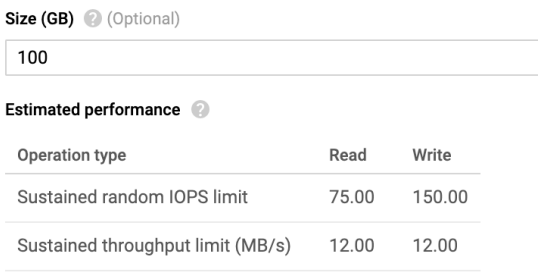

Figura 2.1: Desempenho estimado de um *standard persistent disk*

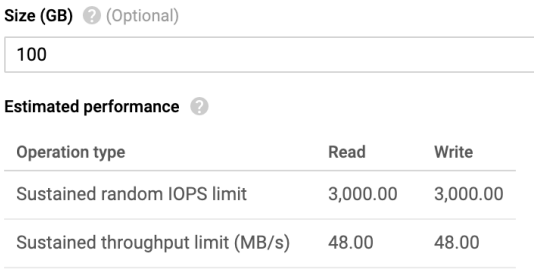

Figura 2.2: Desempenho estimado de um *SSD persistent disk*

Depois de escolhidas as configurações da máquina, foi necessário definir o *workload*. O número de *warehouses* foi estabelecido baseado no espaço estimado que ocupa na memória (4.9 GB, de acordo com a especificação do *tpc-c*). O número de clientes foi obtido através de testes, onde se pretendia que a carga do CPU fosse superior a 45%.

Relativamente ao tempo, e tendo em conta o orçamento que o grupo possuía para realizar os testes, foi estabelecida a seguinte solução: cada teste intermédio com a alteração de um parâmetro era realizado durante 5 minutos, enquanto que o teste final de comparação foi executado durante 10 minutos.

Os valores que constituem a configuração de referência encontram-se ilustrados na tabela 2.7.

Parâmetro	Valor
Warehouse	64
Client	40
Time	5/10 min

Tabela 2.1: Parâmetros do ficheiro *workload_config.properties*

2.2 Otimização da Carga Transacional

Primeiramente foi necessário executar o *benchmark* com a configuração de referência, para haver uma base de comparação com os testes futuros. É de ressalvar que todos os testes foram realizados com a base de dados no mesmo estado, ou seja, após efetuar o *./load* dos dados.

O resultado de ter executado o *benchmark* durante 10 minutos encontra-se na figura 2.3. O valor inicial de throughput obtido foi de 220.2 transações por segundo.

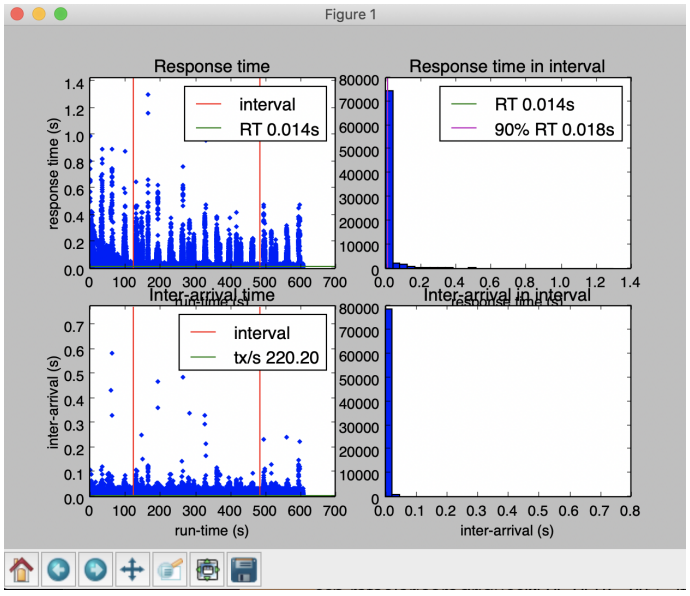

Figura 2.3: Resultados do *benchmark* sem modificações

2.2.1 Alterações

Dado o vasto número de parâmetros que o PostreSQL possui, foi necessário fazer uma seleção dos que consideramos mais importantes para cumprimos o objetivo definido. Dado que a finalidade é maximizar a carga transacional (ou *throughput*), os valores de *response time* e *abort rate* não serão tidos em consideração como valores a otimizar. De seguida, serão explicitados todos parâmetros que foram alterados e a motivação

para tal, o seu valor por defeito e o resultado obtido com a alteração.

1. work_mem = 256 MB
2. synchronous_commit = off
3. effective_io_concurrency = 2
4. checkpoint_timeout = 1h
5. random_page_cost = 1
6. fsync = off

Work_mem

Este parâmetro define a quantidade de memória que cada interrogação tem à sua disposição para operações de ordenação e para as tabelas de hash. Para evitar que sejam usados ficheiros temporários em disco (que tornam a interrogação mais lenta) e como as interrogações possuem um elevado número de operações deste tipo, aumentamos em 98% o valor deste parâmetro.

Alteração	***Throughput* (tx/s)**	**Utilização máx. de CPU (%)**
work_mem = 246MB	221.36	45.94
work_mem = 3GB	221.86	46.06
work_mem = 5GB	224.23	49.80

Tabela 2.2: Resultado das alterações do parâmetro *work_mem*

Synchronous_commit

Este parâmetro por defeito está ativo, ou seja, o utilizador só recebe a confirmação do *commit* da transação quando todos os registos forem guardados em disco. Desativando esta opção, o utilizador pode receber a confirmação antes da transação estar segura. Logo, caso haja alguma falha no sistema, os dados podem ser perdidos. Como não temos a perda de informação como prioridade, com este parâmetro a *off* evitamos que o CPU esteja parado à espera que tudo seja escrito no disco e passe mais tempo a executar as transações.

Alteração	***Throughput* (tx/s)**	**Utilização máx. de CPU (%)**
synchronous_commit = off	224.93	48.35

Tabela 2.3: Resultado da alteração do parâmetro *synchronous_commit*

Effective_io_concurrency

Este parâmetro define o número de operações de *I/O* que são expectáveis de ocorrer simultaneamente. O valor pré-definido é 1. No entanto, no decorrer dos testes, o resultado do *throughput* só foi superior à configuração base até ao valor 2, pelo que consideramos que poderá estar relacionado com o número de cores

do CPU. Como mais do que uma operação pode ser executada em simultâneo, o número de interrogações respondidas por segundo aumenta, tal como desejado.

Alteração	***Throughput* (tx/s)**	**Utilização máx. de CPU (%)**
effective_io_concurrency = 2	221.87	49.8
effective_io_concurrency = 4	145.33	44.3

Tabela 2.4: Resultado da alteração do parâmetro effective_io_concurrency

Checkpoint_timeout

Sendo o valor original 5 minutos, este parâmetro indica o tempo máximo entre *checkpoints* WAL (*Write Ahead Log*) automáticos. Um *checkpoint* é um momento na sequência de transações em que todos os ficheiros de dados em disco são atualizados para refletir a informação contida no *log*. Como esta operação diminui o tempo de CPU dedicado às transações, decidimos definir um período de tempo superior (1 hora), não ocupando tanto o CPU com esta tarefa.

Alteração	***Throughput* (tx/s)**	**Utilização máx. de CPU (%)**
checkpoin_timeout = 1h	222.18	40.02

Tabela 2.5: Resultados da alteração do parâmetro checkpoint_timeout

Random_page_cost

Este parâmetro estabelece o custo de obter páginas do disco não sequencialmente, sendo este valor 4 vezes superior ao valor de obter de forma sequencial. Igualando este custo ao custo sequencial pode levar a que o sistema opte por usar *index scans* mais regularmente, o que poderá aumentar o número de interrogações respondidas por segundo.

Alteração	***Throughput* (tx/s)**	**Utilização máx. de CPU (%)**
random_page_cost = 1	222.96	52.057

Tabela 2.6: Resultados da alteração do parâmetro random_page_cost

Fsync

Por fim, o parâmetro *fsync*, quando ativado, tentará garantir que todas as atualizações são gravadas no disco. No entanto, com o intuito de melhorar o desempenho, desativamos este parâmetro, de modo a não ocupar o CPU com processos desnecessários (não estamos preocupados com a integridade dos dados).

Alteração	***Throughput* (tx/s)**	**Utilização máx. de CPU (%)**
fsync = off	223.04	45.36

Tabela 2.7: Resultado da alteração do parâmetro *fsync*

2.2.2 Resultados obtidos

Juntando todas as alterações realizadas aos parâmetros anteriores, obtemos que o novo *throughput* é de 225.37 transações por segundo, ou seja, obtivemos uma otimização de pouco mais de 2%.

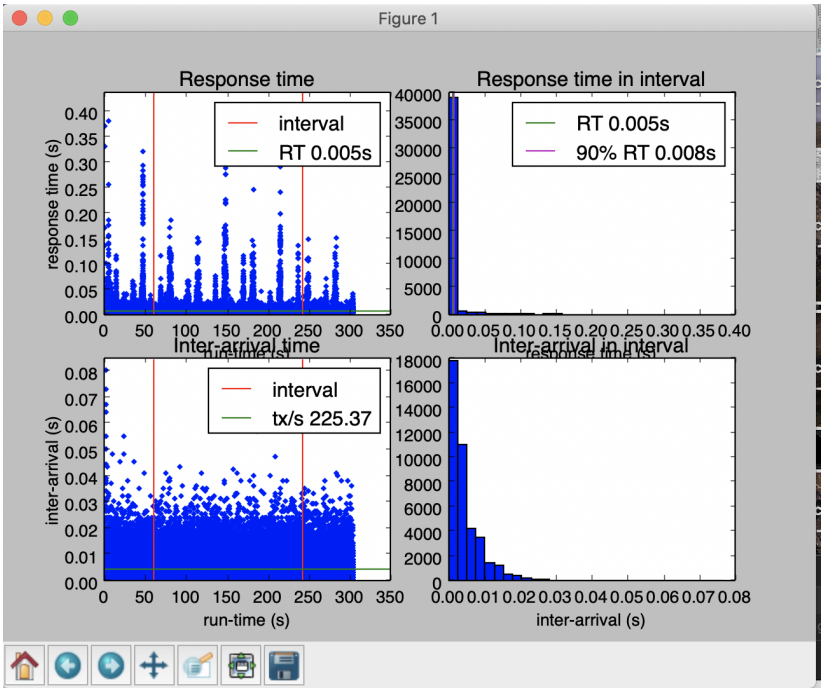

Figura 2.4: Resultado do *benchmark* com otimização dos parâmetros do PostgreSQL

Capítulo 3

Otimização do Desempenho das Interrogações Analíticas

3.1 Interrogação A1

A Interrogação A1 consiste em descobrir quais os fornecedores de uma dada nação que possuem determinadas partes que podem ser candidatas a uma oferta promocional caso a quantidade desse itens seja superior a 50% da quantidade total que foi encomendada após uma determinada faixa de segundos.

O plano inicial de execução é o que se apresenta nas figuras A.1 e A.2.

Deste plano, verificamos que é feita uma ordenação sobre 1.3M de registos, bem como percorrida a tabela *order_line* múltiplas vezes, para scan sequencial e *hash join* com a tabela *stock*. No entanto, esse *join* que depois é ordenado resulta numa agregação de apenas cerca de 30.000 registos. Isto pareceu indicar que seria benéfico realizar uma filtragem mais antecipada dos registos da tabela *order_line* para que apenas os que estão dentro da determinada faixa de segundos fossem selecionados para o *join* com a tabela *stock*. Isto estava a acontecer depois de um scan sequencial, mas como se trata de um intervalo, pensamos que poderia ser melhorado com recurso a um índice. Assim sendo, criamos um índice da seguinte forma:

```
create index ix_delivery on order_line(extract(second from ol_delivery_d));
```

Criado o índice, o plano de execução não se alterou. Assim, para tentar forçar o *Postgres* a escolher um plano que recorresse ao índice, alteramos a interrogação para que incluísse um intervalo, da seguinte forma:

```
select su_name, su_address from supplier, nation
where su_suppkey in (select mod(s_i_id * s_w_id, 10000) from stock, order_line
  where s_i_id in (select i_id from item
    where i_data like 'c%')
  and ol_i_id=s_i_id
  and extract(second from ol_delivery_d) between 50 and 60
  group by s_i_id, s_w_id, s_quantity
  having 2*s_quantity > sum(ol_quantity))
and su_nationkey = n_nationkey
```

	Tempo médio de resposta (em *ms*)	Melhoria face ao inicial
Interrogação Inicial	6115	—
Interrogação com índice	4179	31%

Tabela 3.1: Resultados obtidos para a Interrogação A1

```
and n_name = 'GERMANY'
order by su_name;
```

A inclusão do *between 50 and 60* permitiu que o *Postgres* optasse por um novo plano, que se apresenta nas figuras A.3 e A.4.

Com recurso ao índice, passou a ser possível realizar um *index* scan na tabela *order_line*, mais concretamente um *Bitmap Index Scan*. Assim, a união com a tabela *stock* foi menos custosa, resultando numa ordenação de apenas 60% dos registos em comparação com a versão original.

Os tempos de resposta médios obtidos após 10 medições sobre a configuração de referência são os que se apresentam na tabela 3.1.

Nota-se uma melhoria significativa no desempenho da interrogação, uma vez que o tempo de resposta reduziu 31% relativamente ao original. Isto valida a hipótese colocada inicialmente quanto à vantagem de utilizar um índice numa interrogação que recorre a filtragens por intervalos, neste caso de tempo.

3.2 Interrogação A2

A Interrogação A2 consiste em descobrir quantos fornecedores são capazes de fornecer itens com determinados atributos, por ordem decrescente. O resultado é agrupado pelo identificador do item.

O plano inicial de execução é o que se apresenta na figura B.1.

Tendo em conta que as transações realizadas durante o *benchmark* não alteram qualquer informação utilizada por esta interrogação, concluímos que poderia ser aplicada uma **vista materializada** de forma a melhorar o desempenho.

Assim, criou-se a vista materializada *stock_item_mv*, da seguinte forma:

```
create materialized view stock_item_mv as
    select i_price, i_data, i_id, i_name, mod((s_w_id * s_i_id),10000) as calc
        from item inner join stock on i_id = s_i_id;
```

Esta vista materializada permite consolidar numa só tabela informação relativa a preços e nomes de itens, bem como guardar em memória o resultado da computação `mod((s_w_id * s_i_id),10000)`. Pretende-se com esta tabela reduzir o número de travessias nas tabelas *stock* e *item*, por consolidarmos a sua união com as informações relevantes para a interrogação em memória. Isto permitirá evitar o *scan* da tabela *item* e o custoso *hash join*, por já ter sido guardado o seu resultado em memória.

Assim sendo, em vez de se executar a interrogação analítica conforme o plano da figura B.1, esta é alterada para tirar proveito desta vista materializada, passando a tomar a seguinte forma:

	Tempo médio de resposta (em *ms*)	Melhoria face ao inicial
Interrogação Inicial	15529	—
Interrogação com vista materializada	13296	14%
Interrogação com vista e índice	8327	46%

Tabela 3.2: Resultados obtidos para a Interrogação A2

```
select i_name, substr(i_data, 1, 3) as brand, i_price,
    count(distinct (calc)) as supplier_cnt
    from stock_item_mv where i_data not like 'z%'
        and (calc not in (select su_suppkey from supplier
            where su_comment like '%bean%'))
    group by i_name, substr(i_data, 1, 3), i_price order by supplier_cnt desc;
```

O plano de execução desta interrogação é o que se apresenta na figura B.2.

Verificamos ainda que é feito um scan sequencial, seguido de uma ordenação, de acordo com algumas colunas. Por isso, decidimos criar um índice composto por essas mesmas colunas, de forma a facilitar a ordenação. Foi criado da seguinte forma:

```
create index ix_stock_item_mv on stock_item_mv(i_name,
    (substr((i_data)::text, 1, 3)),i_price);
```

A introdução deste índice provocou uma nova alteração no plano da interrogação, que se apresenta na figura B.3.

Os tempos de resposta médios obtidos após 10 medições sobre a configuração de referência são os que se apresentam na tabela 3.2.

Nota-se uma melhoria significativa no tempo de resposta à interrogação analítica, especialmente quando combinadas as estratégias de materialização de vistas e de utilização de índices, sendo que o salto entre recorrer apenas à vista materializada e à utilização de um índice sobre ela é de 37%.

3.3 Interrogação A3

A interrogação A3 fornece informações sobre as receitas alcançadas pelos países de uma dada região, que são apresentados por ordem decrescente de receitas. O plano de execução desta interrogação encontra-se nas figuras C.1, C.2, C.3, C.4 e C.5.

A partir do plano inicial, constatamos que cerca de 10% do tempo da interrogação é gasto a unir as tabelas *customer*, *nation* e *region*, sendo que o resultado desta união, para a mesma região, é sempre o mesmo. Assim, consideramos que seria benéfico traduzir esta união para uma **vista materializada**, limitando as colunas desta vista àquelas que são necessárias para o resto da lógica da interrogação. Para além disso, reparamos também que o resultado final está ordenado pelo nome da *nation*, pelo que consideramos que a

introdução de um índice nesta **vista materializada** na coluna *n_nation_key* também poderia influenciar positivamente o desempenho da interrogação.

A vista materializada foi criada então da seguinte forma:

```
create materialized view nat as select c_id, c_w_id,
    c_d_id, c_state, n_name, n_nationkey from region
inner join nation on n_regionkey = r_regionkey
inner join customer on ascii(substr(c_state,1,1))-ascii('a') = n_nationkey
where r_name = 'EUROPE';
```

Assim sendo, em vez de a interrogação analítica executar conforme o plano da figura C.1, esta é alterada para recorrer à vista criada da seguinte forma:

```
select n_name, sum(ol_amount) as revenue
from orders, order_line, stock, supplier, nat
where c_id = o_c_id
and c_w_id = o_w_id
and c_d_id = o_d_id
and ol_o_id = o_id
and ol_w_id = o_w_id
and ol_d_id= o_d_id
and ol_w_id = s_w_id
and ol_i_id = s_i_id
and mod((s_w_id * s_i_id),10000) = su_suppkey
and su_nationkey = n_nationkey
group by n_name
order by revenue desc;
```

O plano de execução desta nova interrogação é o que se apresenta na figura C.6. É possível verificar que os *records* da vista estão a ser ordenados de acordo com três colunas: *c_w_id*, *c_d_id* e *c_id*. Por isso, seria razoável expectar que a introdução de um índice composto por estas três colunas na vista materializada poderia resultar numa melhoria do tempo de resposta da interrogação. Foi criado o índice da seguinte forma:

```
create index nat_ix on nat(c_w_id, c_d_id, c_id);
```

Tendo já obtido algumas melhorias face ao desempenho inicial, decidimos validar a introdução de mais um índice. Foi de seguida adicionado o índice na coluna *n_nation_key*, da seguinte forma:

```
create index nat_name on nat(n_name);
```

O plano de execução sofreu uma nova alteração, que se encontra refletida na figura C.7.

Os tempos de resposta médios obtidos após 10 medições sobre a configuração de referência são os que se apresentam na tabela 3.3.

É possível constatar uma ligeira melhoria com a introdução da vista materializada (cerca de 13%). No entanto, com o acréscimo dos índices, foi possível melhorar o tempo médio de resposta em cerca de 45%, face ao tempo médio inicial.

Alteração	**Tempo médio de resposta (em *ms*)**
Interrogação Inicial	9880
Vista materializada	8590
Vista materializada + Índice nat_x	6670
Vista materializada + Ambos os índices	5390

Tabela 3.3: Resultados obtidos para a Interrogação A3

3.4 Interrogação A4

A interrogação A4 consiste em ordenar decrescentemente, os clientes que gastaram mais de 200 unidades monetárias em encomendas.

O plano inicial de execução é o seguinte:

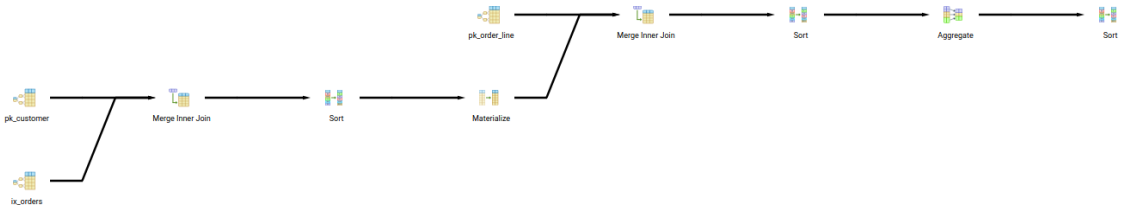

Figura 3.1: Plano inicial de execução da Interrogação A2

A interrogação é iniciada com *index scans* nas tabelas *customer* e *orders*, seguida de um *inner join*, uma ordenação da tabela criada e finalmente uma materialização.

É efetuado seguidamente um index scan na tabela *order_line* seguido de um *inner join* com a tabela previamente materializada de modo a conjugar os campos necessários, seguido de uma ordenação que visa facilitar a agregação que será efetuada depois e uma filtragem que removerá as entradas que não respeitam a restrição de ter gasto 200 unidades monetárias. Finalmente, é realizado uma ordenação de modo a organizar decrescentemente as entradas.

As partes mais custosas desta interrogação são os 3 últimos passos:

1. Ordenação de todas as linhas de encomendas
2. Respetiva agregação e filtragem
3. Ordenação final de entradas

isto deve-se ao grande volume de dados envolvidos nestas operações. Corremos a interrogação 10 vezes obtendo um tempo médio de execução de 2 minutos e 5 segundos.

3.4.1 Possíveis melhorias e premissas

Como já foi detalhado anteriormente, temos neste plano alguns passos que dificilmente verão a sua performance melhorada devido a diversos fatores, como por exemplo o elevado volume de dados que a interrogação necessita para ser realizada.

No entanto, também não temos acesso a algumas informações importantes que nos poderiam ajudar a tomar decisões que viriam a melhorar o desempenho da interrogação, como por exemplo:

1. Frequência da utilização da interrogação (se é utilizada várias vezes por hora, ou apenas uma vez por dia)
2. Importância da atualidade dos dados apresentados pela interrogação (qual é a tolerância permitida, *i.e.* se necessitamos de dados corretos na última hora ou no momento em que a interrogação foi executada)

Tendo em conta estes fatores, de seguida são apresentados alguns cenários e explicitadas as decisões que seriam tomadas em cada um deles.

Atualidade dos dados é importante

Neste caso, a abordagem atual seria provavelmente a melhor a seguir visto que esta satisfaz o requisito da atualidade dos dados. Esta atualidade dos dados, pode ser uma obrigatoriedade em várias situações, como por exemplo a monitorização de certos atributos para propósitos de *Business Intelligence*, o que pode fazer com que esta seja imprescindível. Tem no entanto a desvantagem de ser uma interrogação custosa a nível computacional, o que pode ser uma desvantagem considerável, como por exemplo na situação seguinte.

Interrogação é utilizada frequentemente

Neste caso torna-se necessário determinar a frequência de utilização da interrogação.

Caso a interrogação fosse executada diariamente, os cerca de 2 minutos que esta demora a executar tornar-se-iam negligenciáveis. No entanto, caso a interrogação fosse utilizada a cada 5 minutos, estes 2 minutos que esta demora a executar tornam-se muito mais significativos visto que irão impedir o normal funcionamento do sistema. Nesta situação, um possível compromisso podia ser a atualidade dos dados. Caso esta não fosse crucial, poderíamos tomar partido de vistas materializadas e atualizarmo-la-íamos periodicamente, assegurando assim uma execução muito mais rápida da interrogação sem, no entanto, garantir a atualidade dos dados.

3.4.2 Testes e Resultados

Foram realizados testes com diversos índices e vistas materializadas de modo a testar a diferença que estes teriam na execução normal da interrogação.

Índices

Testamos o desempenho da interrogação não só com os índices que já se encontravam na base de dados mas testamos também com outros, de modo a testar se a adição de alguns campos fazia alguma diferença

significativa na performance da interrogação. Os resultados são os que se encontram nas tabelas 3.4 e 3.5.

Índice	Existe Nativamente	Tabelas e Colunas
1. ix_customer	Sim	c_w_id, c_d_id, c_last
2. pk_customer	Sim	c_w_id, c_d_id, c_id
3. ix_customer_several	Não	c_id, c_w_id, c_d_id, c_last
4. ix_orders	Sim	o_w_id, o_d_id, o_c_id
5. ix_orders_several	Não	o_id, o_c_id, o_d_id, o_w_id, o_entry_d
6. pk_order_line	Sim	ol_w_id, ol_d_id, ol_o_id, ol_number

Tabela 3.4: Índices usados na Interrogação A4

Índices Utilizados	Tempo de Resposta (em s)	Melhoria face ao Inicial
—	186	—
2, 4 e 6	121	34,9%
1, 4 e 6	117	37,1%
2, 5 e 6	121	34,9%

Tabela 3.5: Impacto da utilização de combinações de índices na Interrogação A4

Graças aos resultados acima obtidos podemos constatar algumas coisas:

1. Muitos dos índices criados não foram escolhidos por defeito pelo *Postgres*, o que nos leva a crer que não iriam melhorar significativamente o desempenho da interrogação
2. Não tendo sido escolhidos, o plano de execução da interrogação manteve-se o mesmo, até ao momento em que foi forçada a escolha dos índices
3. Os índices, quando escolhidos, não apresentaram melhorias expressivas face aos resultados obtidos com os índices que existem nativamente na Base de Dados. Os planos de execução não foram alterados com a introdução dos índices, resultando apenas na utilização de outro índice "semelhante".

Vistas materializadas

Foi criada uma Vista Materializada de forma a testar a diferença de performance obtida na Interrogação. Esta Vista contém todos os dados necessários à resolução da interrogação, o que resulta obviamente numa melhoria considerável a nível do desempenho da mesma.

```
create materialized view query_4 as (select c_last, c_id o_id, o_entry_d,
o_ol_cnt, sum(ol_amount) from customer, orders, order_line where c_id =
o_c_id and c_w_id = o_w_id and c_d_id = o_d_id and ol_w_id = o_w_id and
ol_d_id = o_d_id and ol_o_id = o_id group by o_id, o_w_id, o_d_id, c_id,
c_last, o_entry_d, o_ol_cnt having sum(ol_amount) > 200 order by
sum(ol_amount) desc, o_entry_d)
```

Após serem executados 5 testes, obtivemos os resultados que se apresentam na figura 3.2

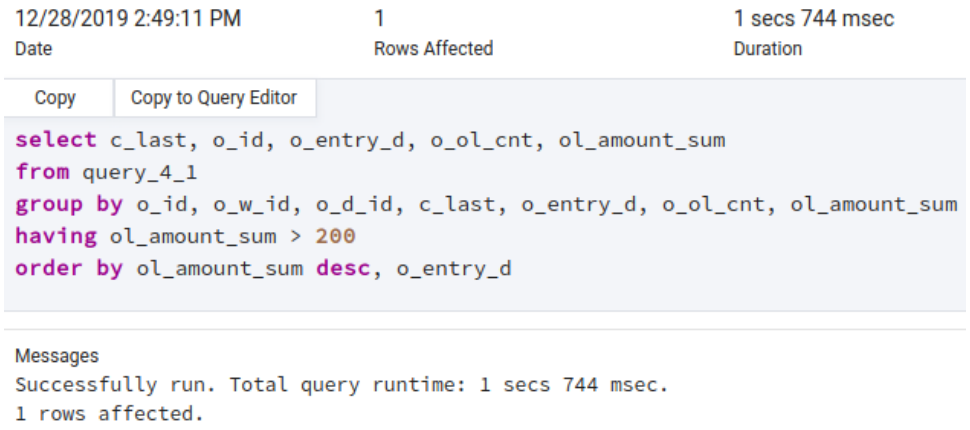

Figura 3.2: Tempo de Execução da Interrogação A4

Graças à Vista Materializada, foi possível diminuir o tempo de execução da interrogação em cerca de 120 segundos, uma melhoria extremamente significativa.

Tal como foi mencionado anteriormente, não é possível obter estes resultados sem compromissos, sendo esses o facto de não termos garantida a atualidade dos dados, o que dependendo das circunstâncias nas quais pretendemos utilizar a Interrogação, pode não ser necessariamente mau. No entanto, são levantados outros problemas, nomeadamente a necessidade de atualizar a Vista periodicamente. Infelizmente, a componente periódica da atualização da vista é algo que não é garantido nativamente pelo Postgresql, não sendo possível agendar uma atualização que ocorra de algum em algum tempo, no entanto, é possível efetuar este agendamento com agentes externos.

Capítulo 4

Replicação/Processamento Distribuído

Alta disponibilidade e redundância são duas características altamente desejáveis na nossa Base de Dados sendo estas imprescindíveis em sistemas atuais, já que qualquer tempo de inatividade inutiliza os recursos disponíveis.

Foi-nos proposto que estudássemos uma possível configuração que tomasse partido de Replicação ou de Processamento Distribuído com qualquer uma das ferramentas estudadas, sendo que o grupo decidiu optar pela *Replicação*.

O mecanismo utilizado para implementar a Replicação foi o de *Transaction Log Shipping*. Este pode ser utilizado para criar um *High Availability Cluster* que tem um ou mais *Servidores Standby (SS)* (que estão prontos para assumir a posição de *Servidor Primário (SP)* caso este falhe). Este serviço depende de um trabalho mútuo entre o *SS* e o *SP* de modo a que o primeiro possa receber os ficheiros *WAL (Write Ahead Log)* enviados pelo *SP*. O envio destes registos é chamado de *Log Shipping*, e consiste no envio de *Segmentos WAL*, um de cada vez. O tamanho destes ficheiros é variável e pode ser ajustado para melhor se adequar às condições de largura de banda disponíveis na infraestrutura. É de notar também que o *Log Shipping* é assíncrono, o que pode resultar na perda de dados, caso o *SP* falhe e não tenha enviado o *segmento WAL*.

É devido a esta desvantagem que decidimos implementar o *Transaction Log Shipping* utilizando o *Streaming Replication*, já que com este mecanismo, o *SS* liga-se ao *SP*. Desta maneira o *SP* envia os registos *WAL* para o *SS* à medida que estes são gerados, evitando assim que o ficheiro *WAL* atinja o tamanho predefinido para que seja enviado. Graças a esta estratégia, o risco de perda de dados, graças a uma falha do *SP*, torna-se inferior, comparadamente ao que se poderia esperar de *File Based Log Shipping*.

4.1 Testes e Resultados

Para testar a implementação, foi executado o benchmark em 2 cenários distintos, todos eles numa máquina com 7,5GB de RAM e um disco extra com 50GB **Standard Persistent**. São os que se descrevem de seguida.

1. *Benchmark* executado no *SP*, sendo este o único ativo, não ocorrendo replicação
2. *Benchmark* executado no *SP*, sendo que o *SS* se encontra ativo, estando a ocorrer replicação

Cenário 1

Número do Teste	Tempo de Resposta (em s)	Transações por Segundo
1	1.453	12.52
2	1.369	13.95
3	1.224	15.64
4	1.897	10.10
5	1.935	9.91
Média	1.5756	12.426

Tabela 4.1: Resultados da execução do *benchmark* no Cenário 1

Cenário 2

Número do Teste	Tempo de Resposta (em s)	Transações por Segundo
1	1.812	9.82
2	1.493	12.32
3	1.397	13.51
4	1.301	14.72
5	2.526	7.59
Média	1.706	11.592

Tabela 4.2: Resultados da execução do *benchmark* no Cenário 2

Concluídos os testes podemos verificar que os resultados da execução do *benchmark* no *SP* enquanto este se encontra a enviar segmentos *WAL* para o *SS* são ligeiramente piores quando comparados com os do *Cenário 1*.

No entanto, é necessário efetuar algumas considerações, nomeadamente o facto da amostra de testes efetuados ser reduzida e o seu alcance não ser extenso, visto que só cobrimos 2 cenários. Algo que poderíamos testar seria a execução do *benchmark* no *SS* enquanto este recebe ficheiros por parte do *SP* e também a execução simultânea dos *benchmarks* em ambos os *Servidores*.

Abordando agora o *Processamento Distribuído de Interrogações*, o facto de não o ser possível fazer nativamente no *PostgreSQL*, foi uma das razões que levou o grupo a optar pela *Replicação*. No entanto, teria sido interessante testar os efeitos que seriam tidos na execução das Interrogações, não só a nível do planos de execução mas também dos tempos que estas demoram a executar.

Capítulo 5

Conclusão

Com este trabalho foi possível efetuar investigação prática relativamente ao desempenho de um sistema de gestão de bases de dados. Tendo em conta os objetivos definidos no enunciado, foi possível alcançar melhorias no desempenho da base de dados, quer ao nível da carga transacional, quer ao nível do tempo de resposta das interrogações analíticas. Para além disso, foi possível implementar uma solução que recorresse a um mecanismo de replicação.

No entanto, o grupo teve algumas dificuldades em estimar o saldo que iria necessitar para efetuar todos os testes. Por isso, optou por uma abordagem conservadora, de forma a que fosse possível realizar testes a todo o trabalho realizado, e não correr o risco de realizar testes a apenas algumas partes com uma configuração superior.

Ainda assim, consideramos que os resultados obtidos permitem validar as estratégias definidas pelo grupo para abordar cada problema proposto, pelo que consideramos que os objetivos foram cumpridos.

Apêndice A

Interrogação A1

A.1 Interrogação SQL

```
select  su_name, su_address
from  supplier, nation
where  su_suppkey in
(select  mod(s_i_id * s_w_id, 10000)
from     stock, orderline
where    s_i_id in
(select i_id
 from item
 where i_data like 'c%')
 and ol_i_id=s_i_id
 and extract(second from ol_delivery_d) > 50
group by s_i_id, s_w_id, s_quantity
having   2*s_quantity > sum(ol_quantity))
 and su_nationkey = n_nationkey
 and n_name = 'GERMANY'
order by su_name
```

A.2 Figuras

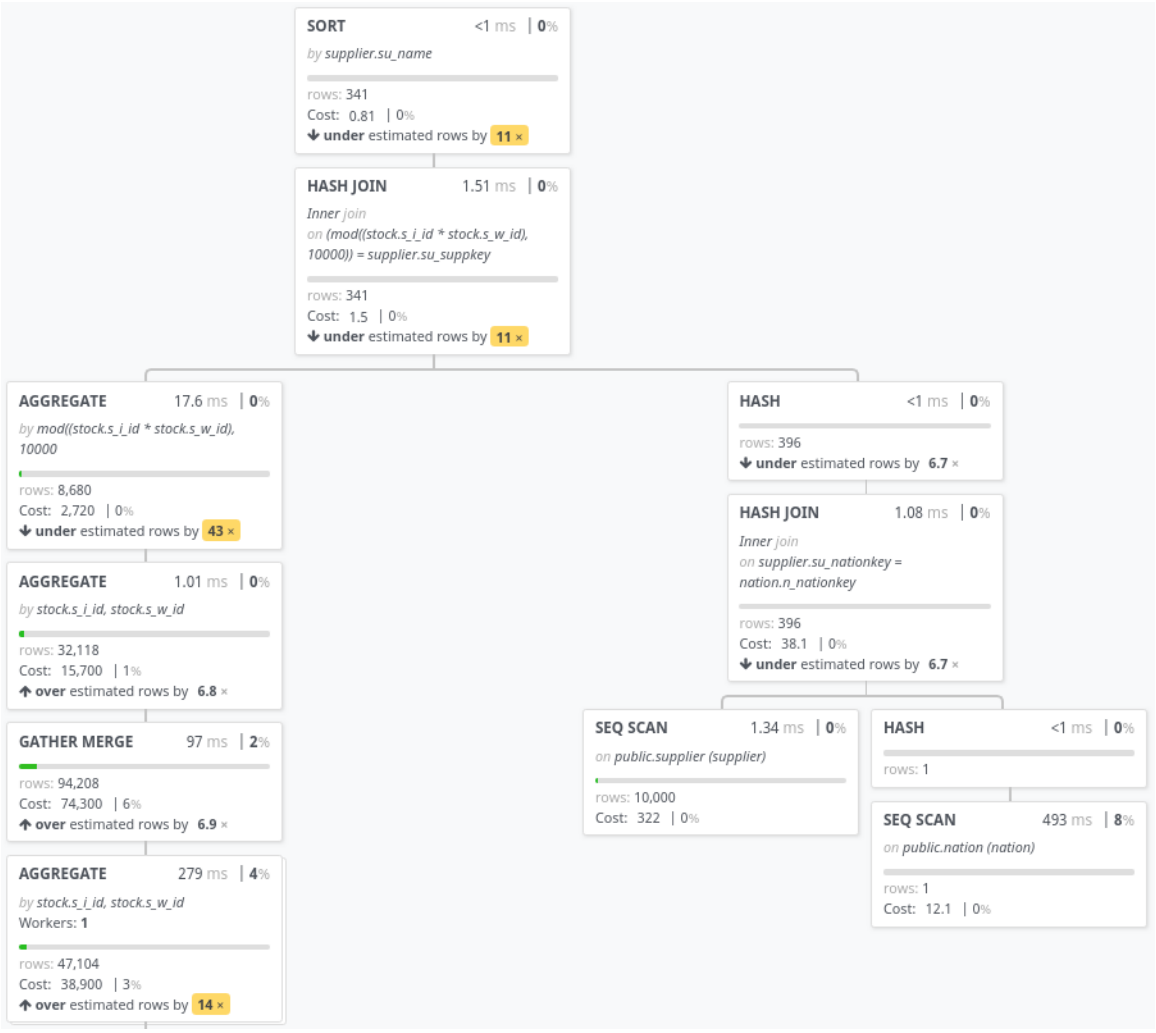

Figura A.1: Plano inicial de execução da Interrogação A1 (topo)

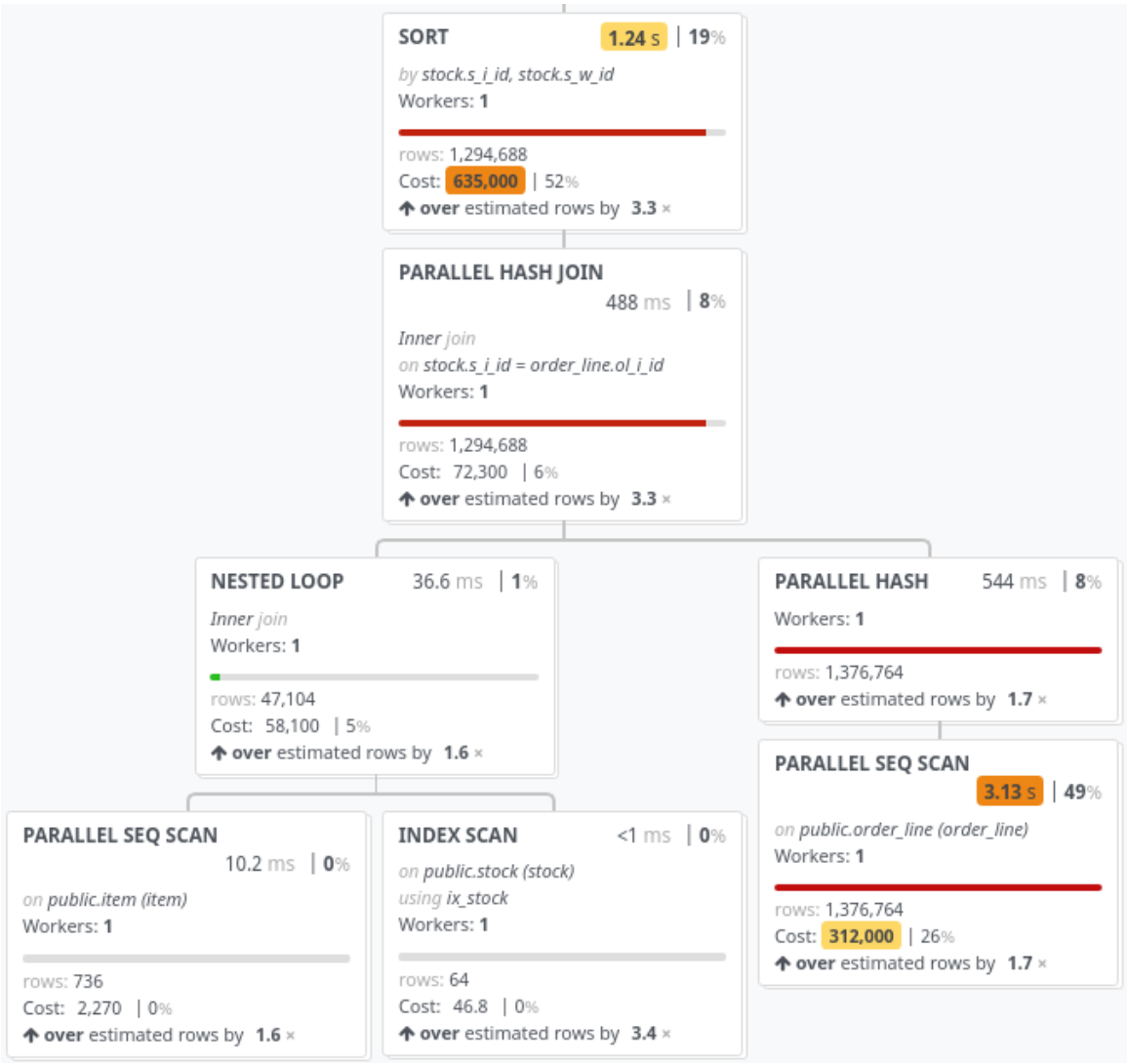

Figura A.2: Plano inicial de execução da Interrogação A1 (fundo)

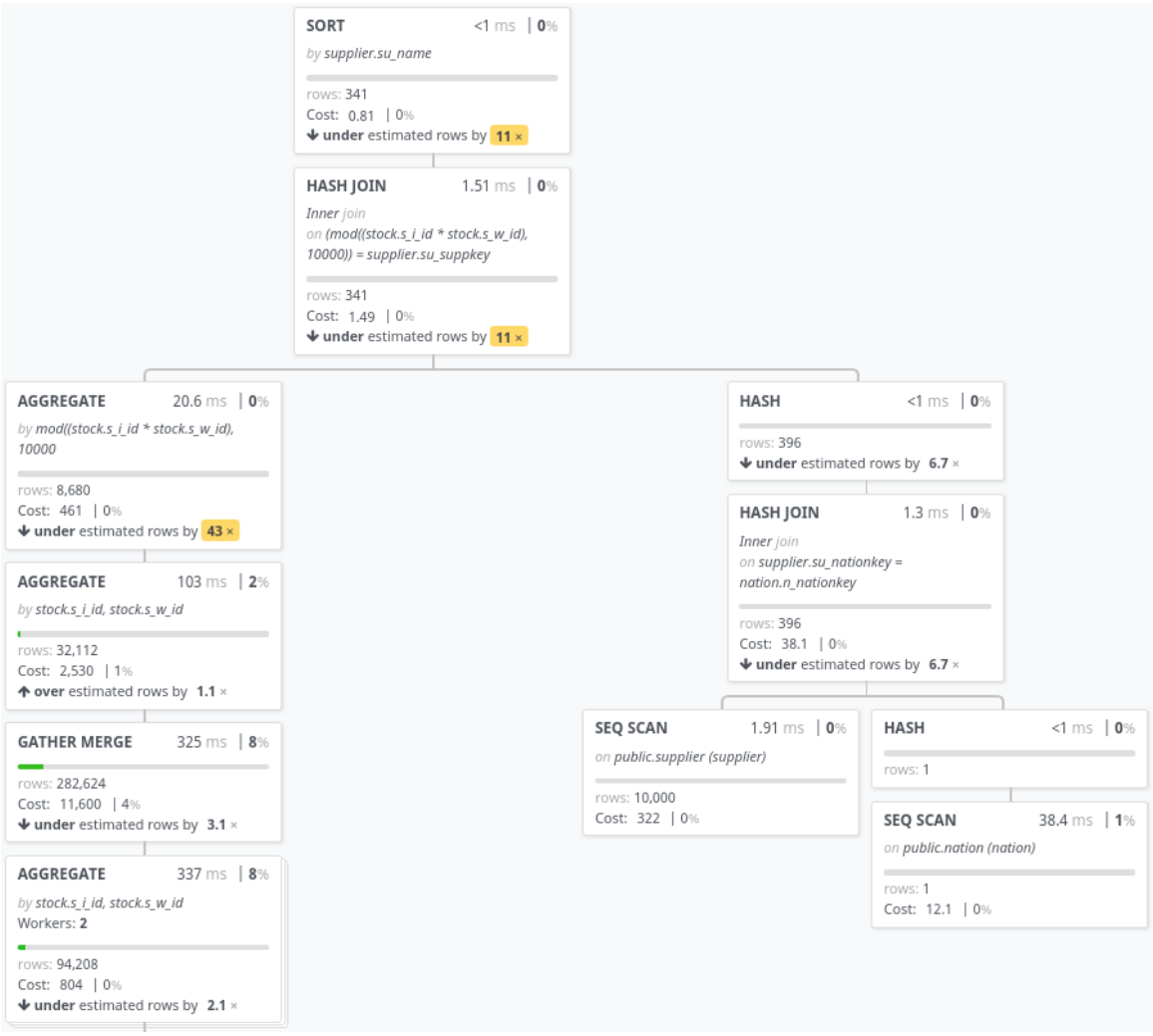

Figura A.3: Plano execução da Interrogação A1 com índice (topo)

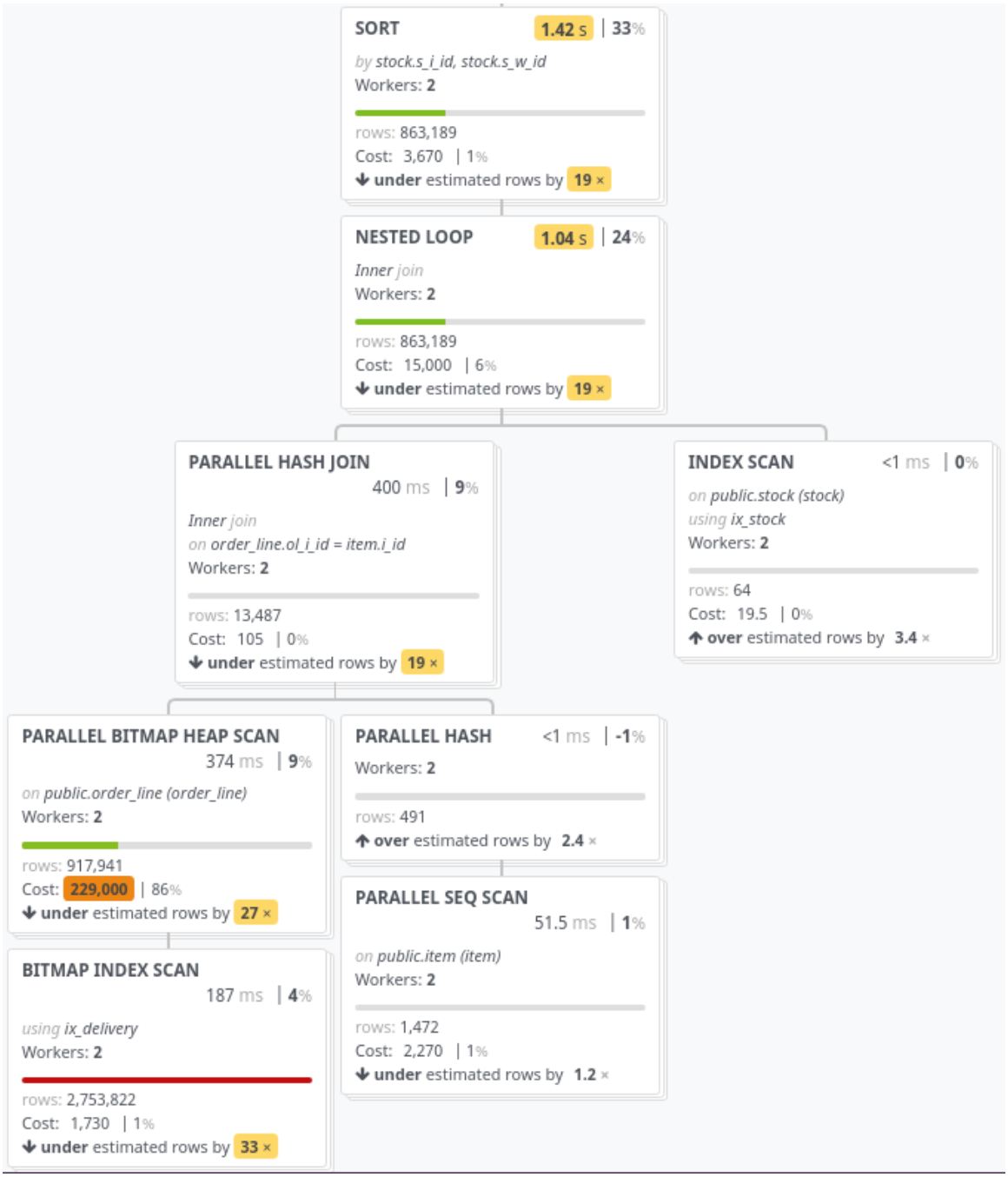

Figura A.4: Plano execução da Interrogação A1 com índice (fundo)

Apêndice B

Interrogação A2

B.1 Interrogação SQL

```
select  i_name,
 substr(i_data, 1, 3) as brand,
 i_price,
 count(distinct (mod((s_w_id * s_i_id),10000))) as supplier_cnt
from  stock, item
where  i_id = s_i_id
 and i_data not like 'z%'
 and (mod((s_w_id * s_i_id),10000) not in
(select su_suppkey
 from supplier
 where su_comment like '%bean%'))
group by i_name, substr(i_data, 1, 3), i_price
order by supplier_cnt desc
```

B.2 Figuras

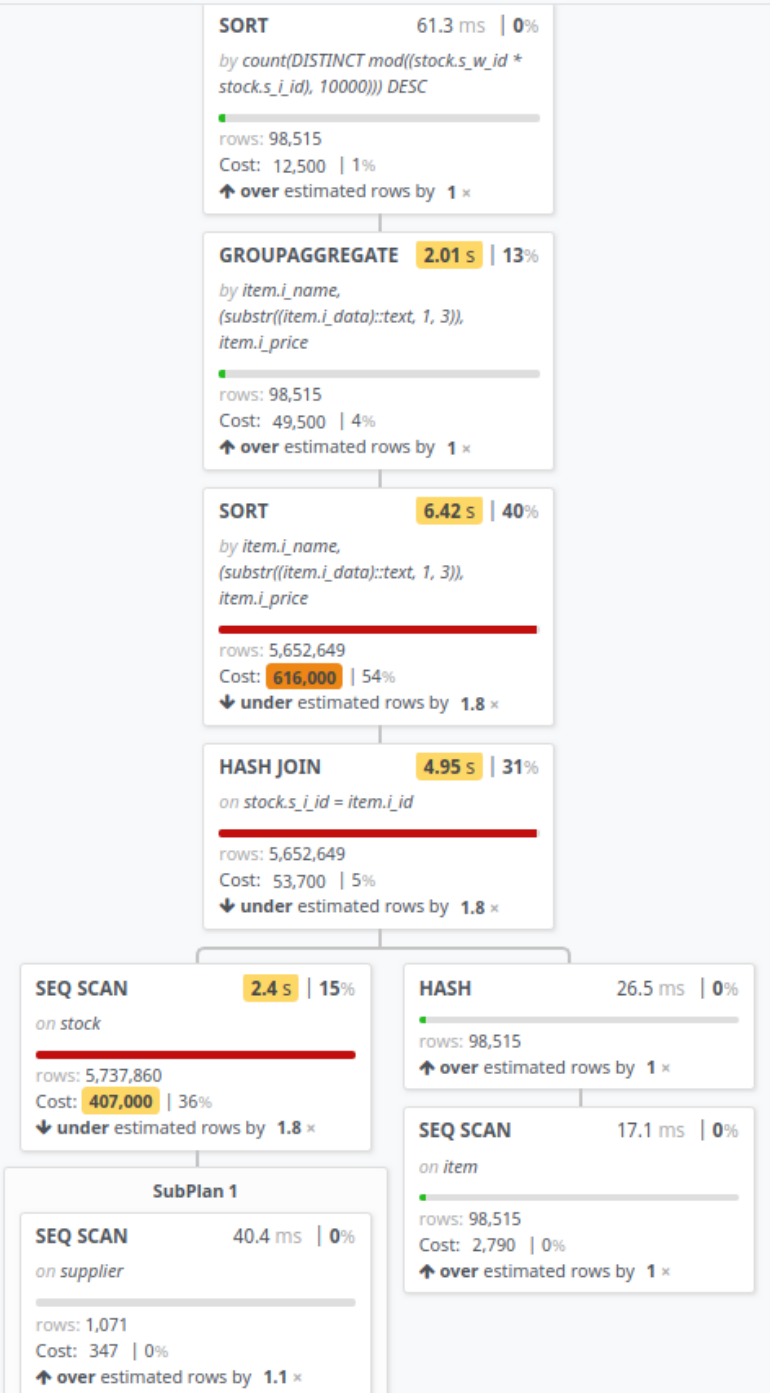

Figura B.1: Plano inicial de execução da Interrogação A2

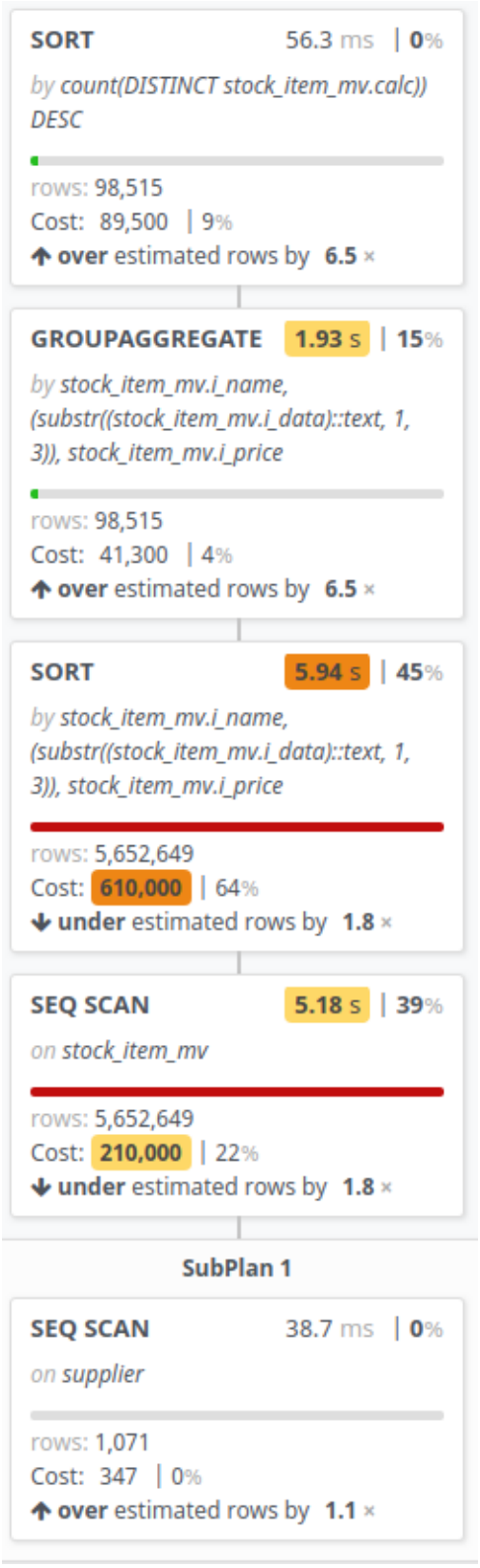

Figura B.2: Plano de execução da Interrogação A2 com vista materializada

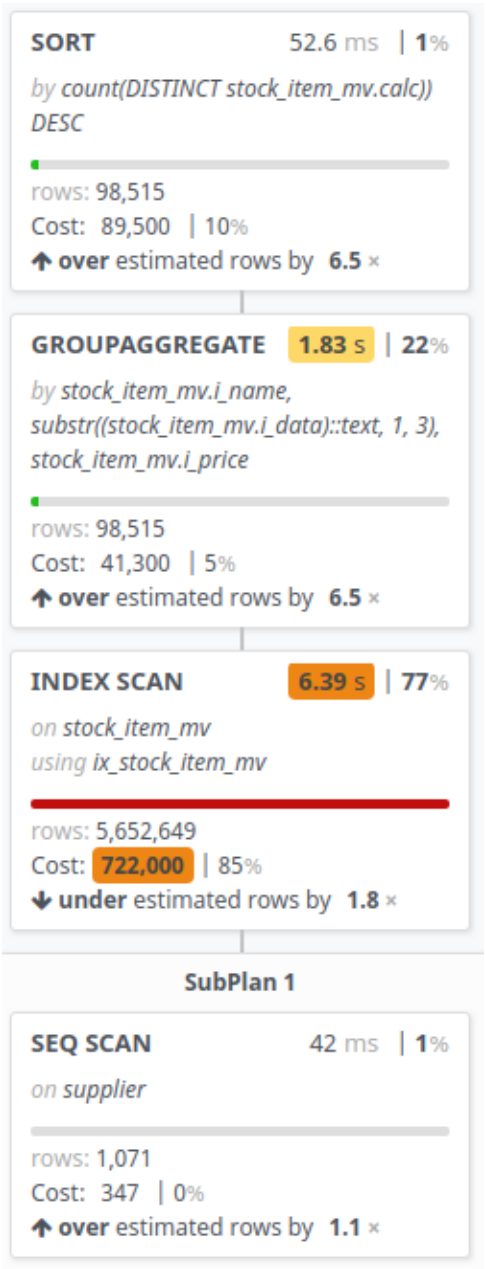

Figura B.3: Plano de execução da Interrogação A2 com vista materializada e índice

Apêndice C

Interrogação A3

C.1 Interrogação SQL

```
select n_name, sum(ol_amount) as revenue
from customer, orders, order_line, stock, supplier, nation, region where c_id = o_c_id
and c_w_id = o_w_id
and c_d_id = o_d_id
and ol_o_id = o_id
and ol_w_id = o_w_id
and ol_d_id= o_d_id
and ol_w_id = s_w_id
and ol_i_id = s_i_id
and mod((s_w_id * s_i_id),10000) = su_suppkey
and ascii(substr(c_state,1,1))-ascii('a') = su_nationkey and su_nationkey = n_nationkey
and n_regionkey = r_regionkey
and r_name = 'EUROPE' group by n_name
order by revenue desc;
```

C.2 Figuras

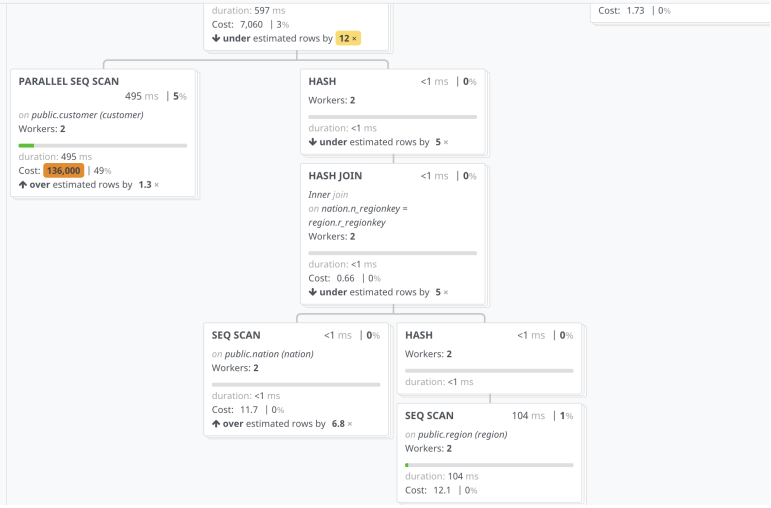

Figura C.1: Plano inicial de execução da Interrogação A3 (Parte 1)

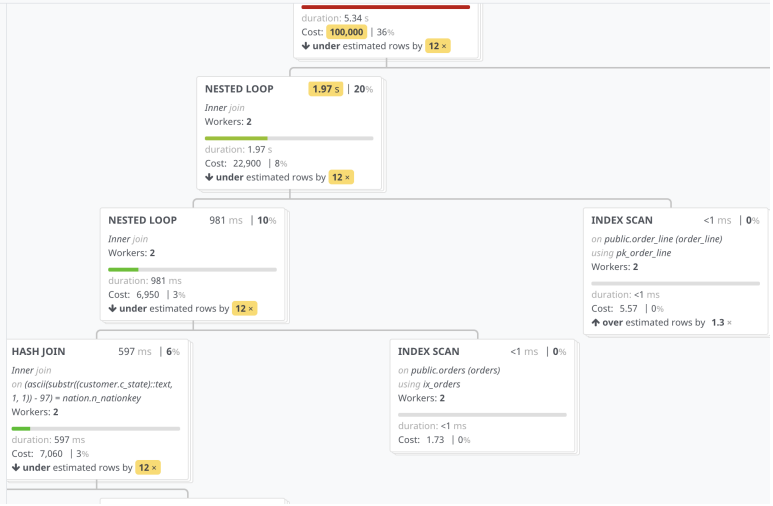

Figura C.2: Plano inicial de execução da Interrogação A3 (Parte 2)

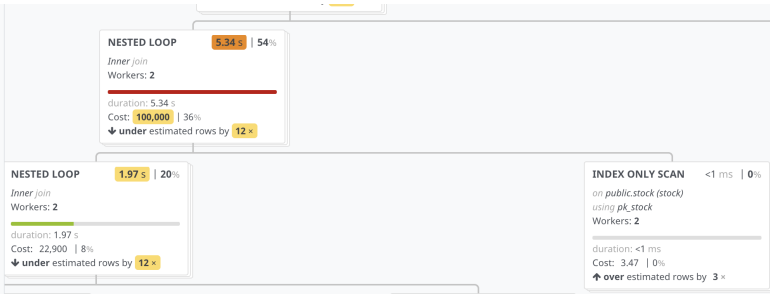

Figura C.3: Plano inicial de execução da Interrogação A3 (Parte 3)

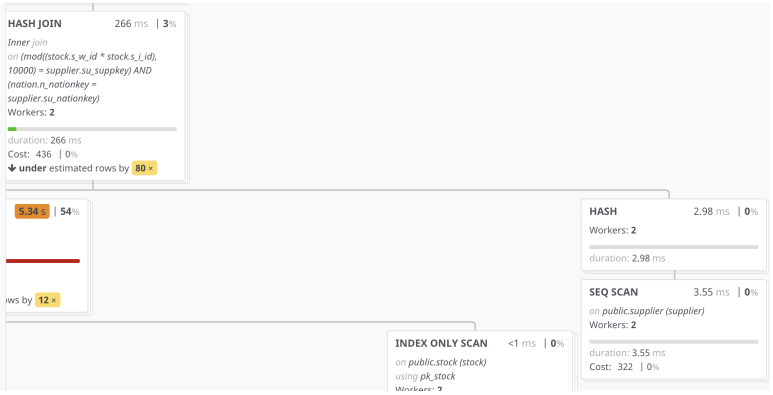

Figura C.4: Plano inicial de execução da Interrogação A3 (Parte 4)

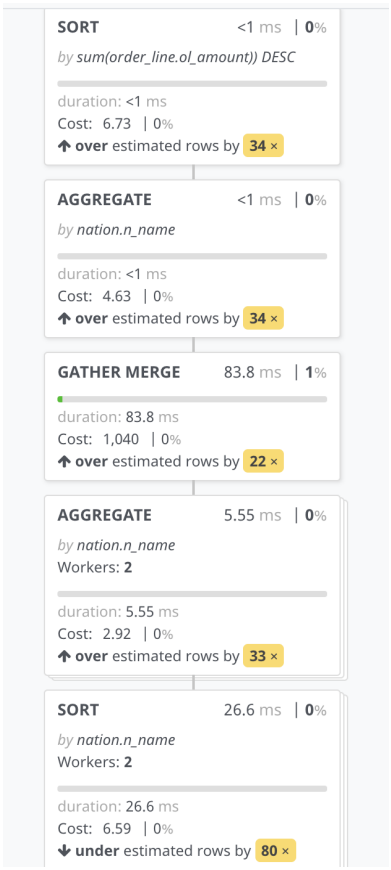

Figura C.5: Plano inicial de execução da Interrogação A3 (Parte 5)

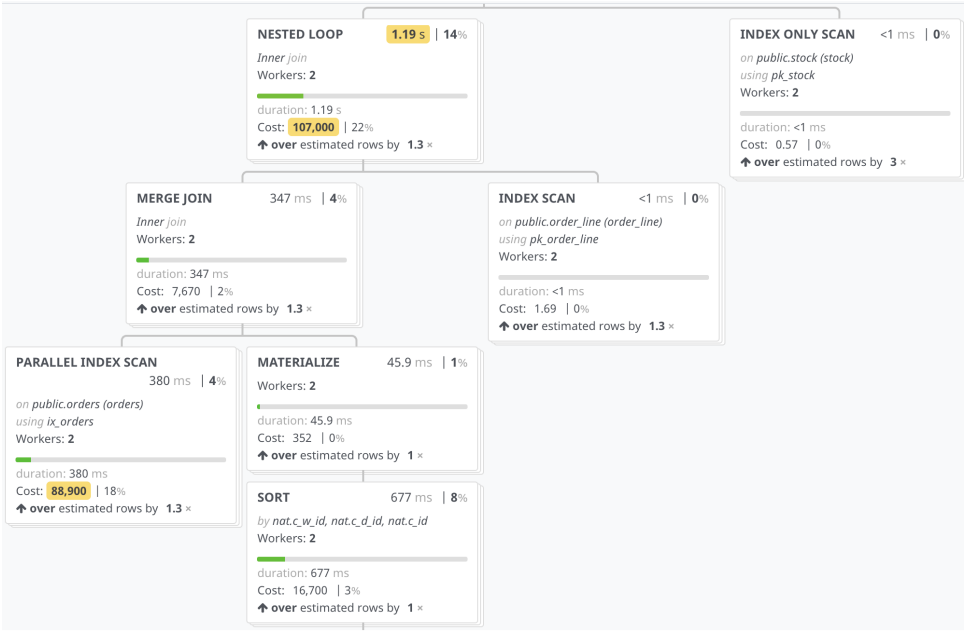

Figura C.6: Plano de execução da Interrogação A3 após adição de vista materializada

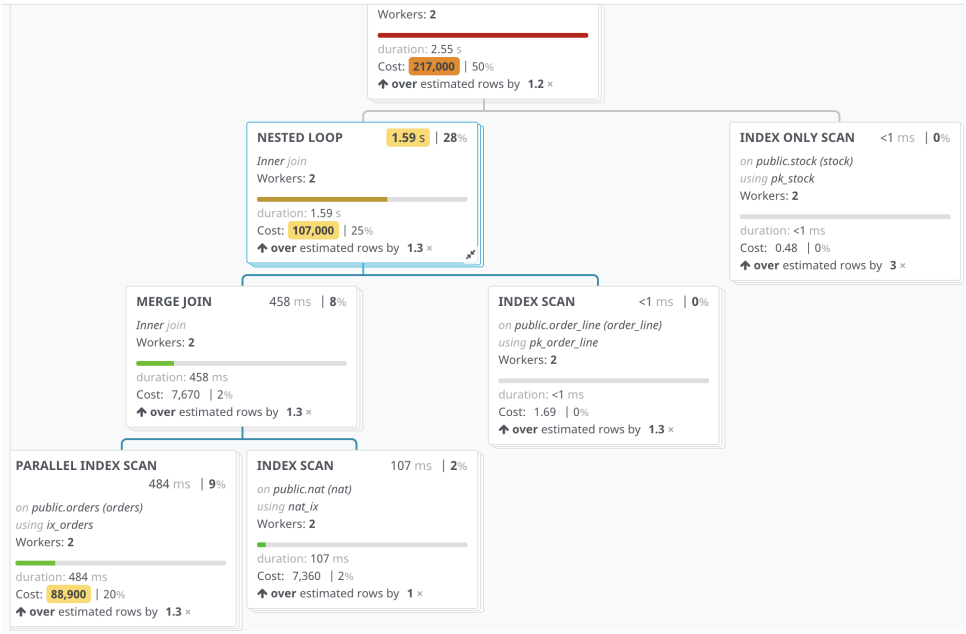

Figura C.7: Plano inicial de execução da Interrogação A3 após introdução de índices