Escola de Artes, Ciências e Humanidades – Universidade de São Paulo

Parte 3: Orientação a Objetos e Estudo de PECs e Desempenho

Prof. Fátima Nunes

Matheus Percoraro de Carvalho Santos	Nº USP: 11917271
Ryan Brenno Ramos	Nº USP: 11215772
Sungwon Yoon	Nº USP: 9822261
Wendel Fernandes de Lana	Nº USP: 11796722

Relatório Estatístico

Artefato a)

O SQL Server, nativamente, não dá suporte a orientação a objetos. O único método seria utilizando uma linguagem de programação orientada a objetos e relacionando os objetos a tabelas no banco de dados.

- **Objeto Complexo:** o pedido de delivery contém como informação o endereço a ser entregue. Essa regra pode ser modelada como um atributo composto de um tipo Endereço definido previamente contido na tabela de deliveries.

```
create type Endereco as
(rua varchar (20),
numero int,
cidade varchar (15),
cep varchar (8),
complemento varchar(20))

create type Delivery (
  id_pedido int,
  id_cliente int,
 endereco Endereco
)

create table deliveries of Delivery
```

- **Tipo Referência:** o cadastro de pedidos de delivery deve se relacionar com o cliente que realizou aquele pedido específico. Para isso, podemos modelar o atributo cliente no tipo Delivery como um tipo referencial à tabela clientes.

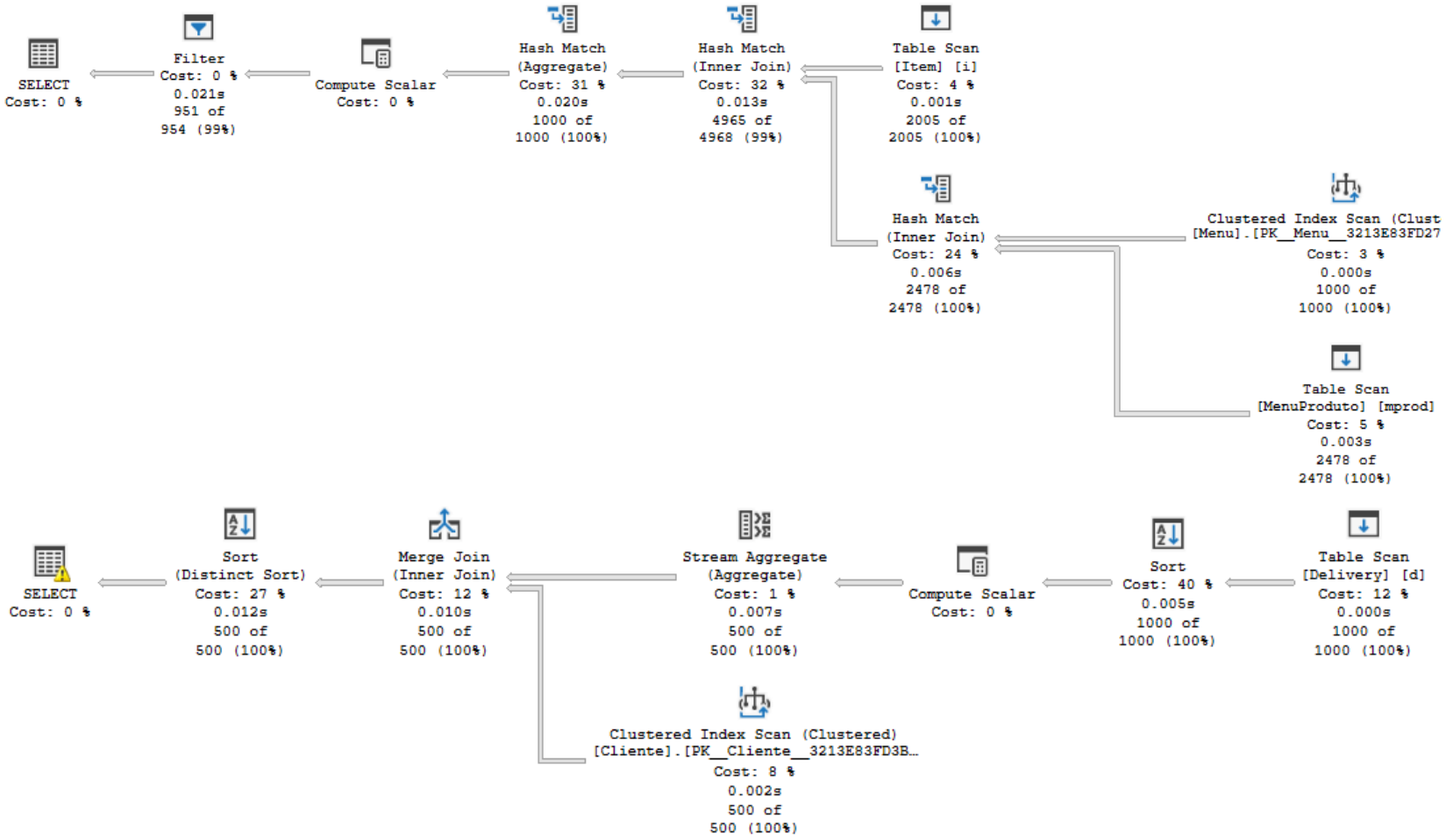
```
create type Cliente as
(nome varchar (30),
cpf varchar (15),
telefone int,
ref using int)

create table clientes of Cliente ref is id system generated

create type Delivery
(id_pedido int,
endereco Endereco,
id_cliente ref(Cliente) scope clientes)

create table deliveries of Delivery
```

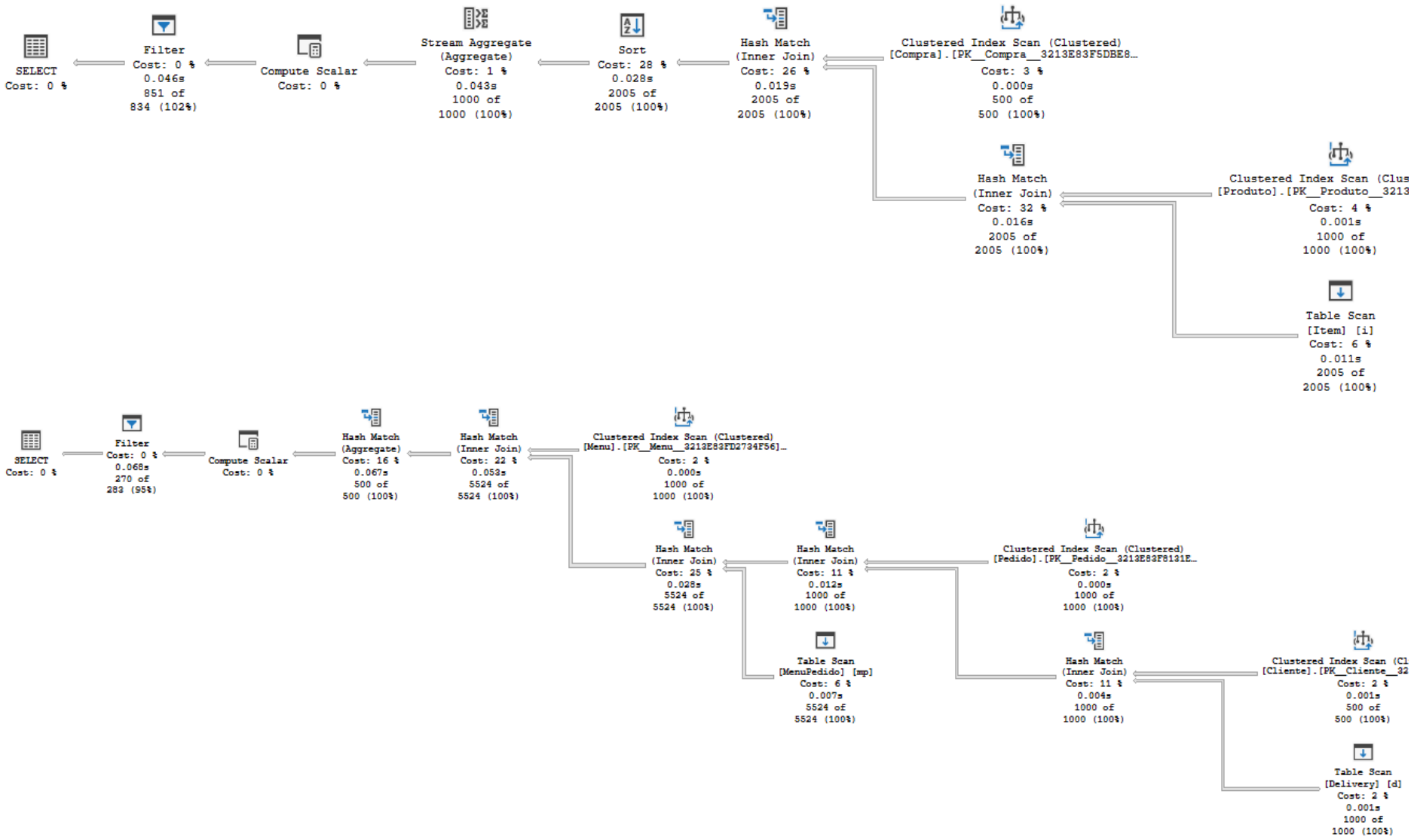
- **Herança:** os pedidos podem ser subdivididos em pedidos de delivery e pedidos locais (comandas de mesa). Para abranger essa regra, podemos usar o conceito de herança do modelo objeto-relacional. Os tipos Comanda e Delivery herdarão os atributos do tipo pai Pedido.

```
create type Pedido as
(id int,
data date,
codigo_cupom varchar(5),
forma_pagamento varchar(10),
status int)
not final

create type Comanda
under Pedido
(id_funcionario int,
mesa int,
gorjeta decimal(10,3))

create type Delivery
under Pedido
(endereco Endereco,
id_cliente ref(Cliente) scope clientes)

create table pedidos of Pedido
create table comandas of Comanda under pedidos
create table deliveries of Delivery under pedidos
```

Artefato b)

Planos de execução de consulta para as consultas formuladas na parte 1 do trabalho, na ordem em que foram apresentadas anteriormente:




SELECT
Cost: 0 %
Filter
Cost: 0 %
0.046s
851 of
834 (102%)
Compute Scalar
Cost: 0 %
Stream Aggregate
(Aggregate)
Cost: 1 %
0.043s
1000 of
1000 (100%)
Sort
Cost: 28 %
0.028s
2005 of
2005 (100%)
Hash Match
(Inner Join)
Cost: 26 %
0.019s
2005 of
2005 (100%)
Clustered Index Scan (Clustered)
[Compra].[PK__Compra__3213E83F5DBE8...
Cost: 3 %
0.000s
500 of
500 (100%)
Hash Match
(Inner Join)
Cost: 32 %
0.016s
2005 of
2005 (100%)
Clustered Index Scan (Clus
[Produto].[PK__Produto__3213
Cost: 4 %
0.001s
1000 of
1000 (100%)
Table Scan
[Item] [i]
Cost: 6 %
0.011s
2005 of
2005 (100%)
SELECT
Cost: 0 %
Filter
Cost: 0 %
0.068s
270 of
283 (95%)
Compute Scalar
Cost: 0 %
Hash Match
(Aggregate)
Cost: 16 %
0.067s
500 of
500 (100%)
Hash Match
(Inner Join)
Cost: 22 %
0.053s
5524 of
5524 (100%)
Clustered Index Scan (Clustered)
[Menu].[PK__Menu__3213E83FD2734F56]...
Cost: 2 %
0.000s
1000 of
1000 (100%)
Hash Match
(Inner Join)
Cost: 25 %
0.028s
5524 of
5524 (100%)
Hash Match
(Inner Join)
Cost: 11 %
0.012s
1000 of
1000 (100%)
Clustered Index Scan (Clustered)
[Pedido].[PK__Pedido__3213E83F8131E...
Cost: 2 %
0.000s
1000 of
1000 (100%)
Table Scan
[MenuPedido] [mp]
Cost: 6 %
0.007s
5524 of
5524 (100%)
Hash Match
(Inner Join)
Cost: 11 %
0.004s
1000 of
1000 (100%)
Clustered Index Scan (Cl
[Cliente].[PK__Cliente__32
Cost: 2 %
0.001s
500 of
500 (100%)
Table Scan
[Delivery] [d]
Cost: 2 %
0.001s
1000 of
1000 (100%)

O otimizador de consultas, sempre quando há alguma junção que envolva a chave primária de alguma tabela (utilizada como chave de ordenação), lê a tabela por meio de um índice clusterizado. O índice clusterizado, no SQL Server, corresponde à organização física da tabela em árvore B.

A maioria das junções são realizadas por meio do algoritmo de hash-junção visto em aula, cujo desempenho tende a ser o melhor dos algoritmos quando as tabelas se encontram em necessidade de classificação. Contudo, em uma das junções na segunda consulta, o otimizador optou por realizar a classificação da tabela que se encontrava desordenada e posteriormente utilizar o merge-junção, julgando que o custo da ordenação mais o custo do algoritmo seriam inferiores ao custo do hash-junção.

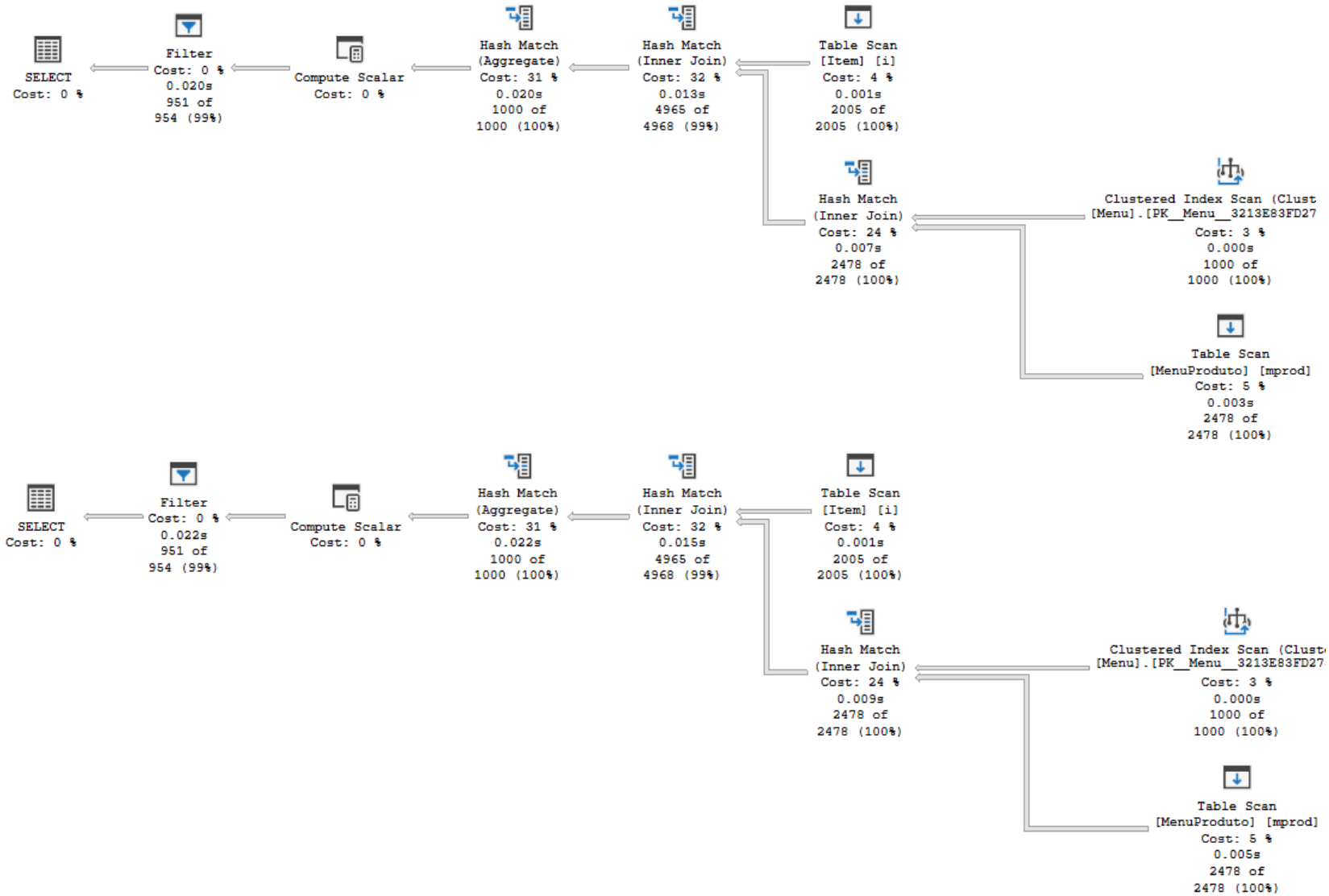
A seleção foi realizada após as junções em todos os planos de consulta pois ela é realizada por agrupamentos. Em uma situação em que a seleção pudesse ser realizada antes das junções, seria preferível fazê-lo a fim de reduzir a quantidade de tuplas das tabelas antes da operação de junção.

Artefato d)

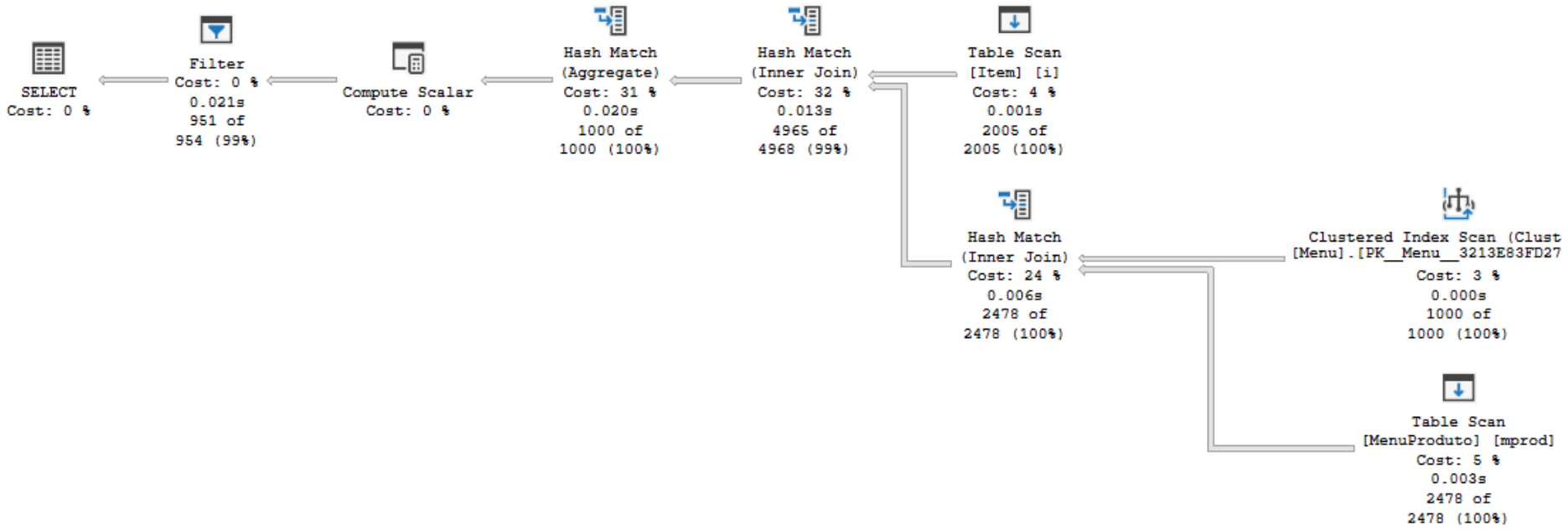
A consulta a ser reescrita utilizando duas estratégias equivalentes é a primeira consulta entregue na parte 1 do trabalho. A seguir, seguem as duas consultas reescritas e os seus planos de execução respectivamente:

```
SELECT m.nome FROM Item AS i
JOIN MenuProduto AS mprod ON i.id_produto = mprod.id_produto
JOIN Menu AS m ON mprod.id_menu = m.id
GROUP BY m.nome
HAVING SUM(i.quantidade) >= 10
```

```
SELECT m.nome FROM (SELECT m.nome, i.quantidade FROM Menu AS m
JOIN MenuProduto AS mprod ON mprod.id_menu = m.id
JOIN Item AS i ON i.id_produto = mprod.id_produto) AS m
GROUP BY m.nome
HAVING SUM(m.quantidade) >= 10
```


SELECT
Cost: 0 %
Filter
Cost: 0 %
0.020s
951 of
954 (99%)
Compute Scalar
Cost: 0 %
Hash Match
(Aggregate)
Cost: 31 %
0.020s
1000 of
1000 (100%)
Hash Match
(Inner Join)
Cost: 32 %
0.013s
4965 of
4968 (99%)
Table Scan
[Item] [i]
Cost: 4 %
0.001s
2005 of
2005 (100%)
Hash Match
(Inner Join)
Cost: 24 %
0.007s
2478 of
2478 (100%)
Clustered Index Scan (Clust
[Menu].[PK__Menu__3213E83FD27
Cost: 3 %
0.000s
1000 of
1000 (100%)
Table Scan
[MenuProduto] [mprod]
Cost: 5 %
0.003s
2478 of
2478 (100%)
SELECT
Cost: 0 %
Filter
Cost: 0 %
0.022s
951 of
954 (99%)
Compute Scalar
Cost: 0 %
Hash Match
(Aggregate)
Cost: 31 %
0.022s
1000 of
1000 (100%)
Hash Match
(Inner Join)
Cost: 32 %
0.015s
4965 of
4968 (99%)
Table Scan
[Item] [i]
Cost: 4 %
0.001s
2005 of
2005 (100%)
Hash Match
(Inner Join)
Cost: 24 %
0.009s
2478 of
2478 (100%)
Clustered Index Scan (Clust
[Menu].[PK__Menu__3213E83FD27
Cost: 3 %
0.000s
1000 of
1000 (100%)
Table Scan
[MenuProduto] [mprod]
Cost: 5 %
0.005s
2478 of
2478 (100%)

Para fins de comparação, a seguir está o plano de consulta gerado para a consulta original da parte 1 (apresentado anteriormente no artefato b):



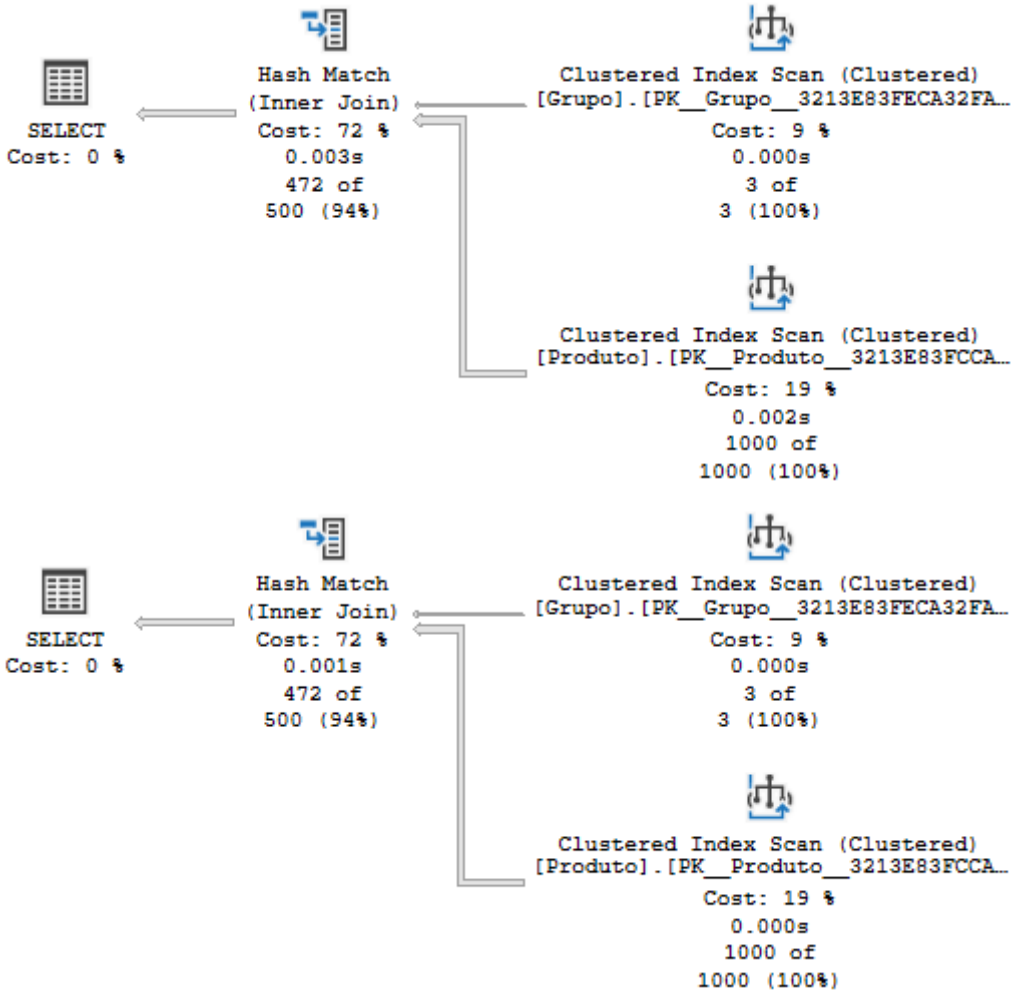
Para a primeira consulta projetada, foi utilizada a estratégia de equivalência da comutação de junções. Já para a segunda, a estratégia utilizada foi a distribuição da projeção pela operação de junção. O otimizador de consultas do SQL Server, tratando-se de consultas diferentes porém equivalentes, conseguiu chegar no mesmo plano de execução otimizado para as três consultas vistas anteriormente.

Artefato e)

A seguir, a consulta projetada que faz uso da estratégia de *subquery*, a mesma consulta reescrita sem utilização de *subquery* e seus respectivos planos de execução:

```
SELECT p.nome
FROM Produto AS p
WHERE p.id_grupo = (SELECT g.id FROM Grupo AS g
WHERE (g.modo_conservacao = 'Geladeira' OR g.modo_conservacao = 'Freezer')
AND g.id = p.id_grupo)
```

```
SELECT p.nome
FROM Produto AS p
JOIN Grupo AS g ON p.id_grupo = g.id
WHERE g.modo_conservacao = 'Geladeira' OR g.modo_conservacao = 'Freezer'
```



Como visto anteriormente no artefato d), mesmo na consulta projetada com utilização de *subquery*, o otimizador do SGBD optou por adotar uma estratégia diferente, mais simples e otimizada, do que a intencionada pela *subquery*. Desse modo, ambas as consultas geraram o mesmo plano de execução.