12

Mãos na massa

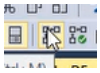
Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

1) Crie uma nova consulta olhando a base `SUCOS_VENDAS` no **Management Studio**.

2) Digite as seguinte consultas, mas não execute-as ainda:

```
SELECT N FROM Nums1 where N = '10001'
SELECT N FROM Nums2 where N = '10001'
```

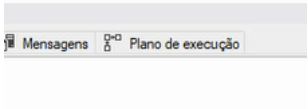
3) Clique no botão **Plano de Execução**, na barra de ferramentas do **Management Studio**:



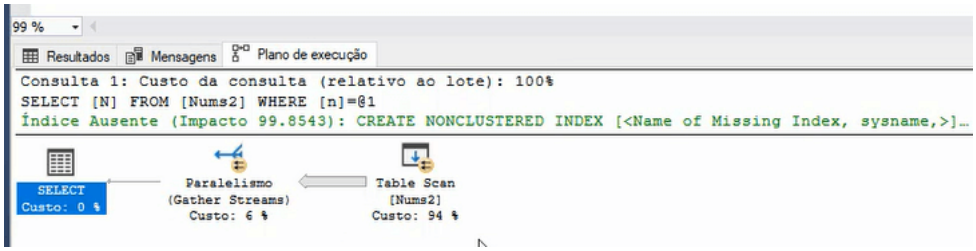
4) Execute a segunda consulta, com o plano de execução selecionado:

```
SELECT N FROM Nums2 where N = '10001'
```

5) Você verá uma aba chamada **Plano de Execução** na região de saídas do **SQL Server Management Studio**:



6) Selecione esta aba. Você verá o plano de execução da consulta:

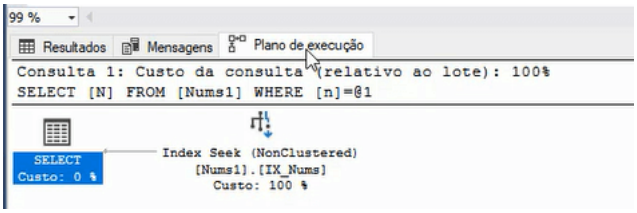


Note que usamos o `TABLE SCAN` porque a tabela não tem índice.

7) Execute agora a primeira consulta:

```
SELECT N FROM Nums1 where N = '10001'
```

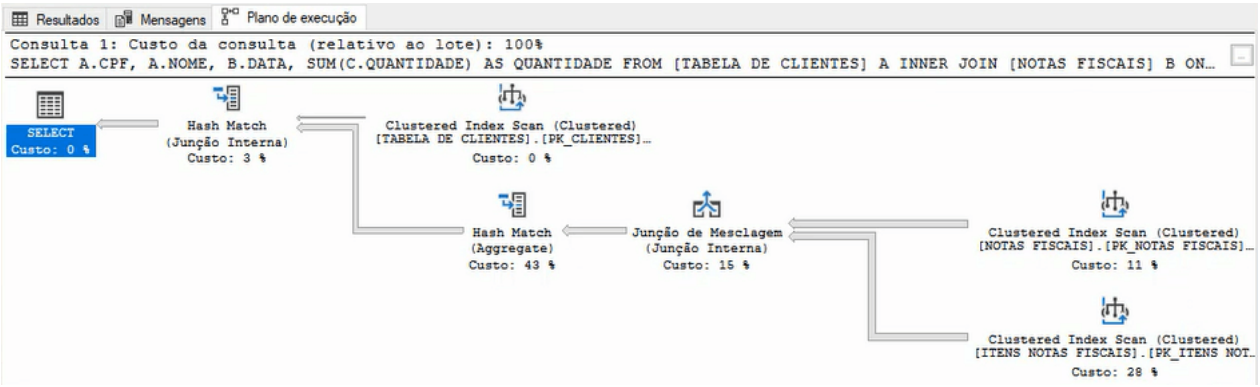
8) Olhando o plano de execução, você vê que, agora, o índice é utilizado:



9) Os planos de execução podem ser mais complexos. Executando:

```
SELECT A.CPF, A.NOME, B.DATA, SUM(C.QUANTIDADE) AS QUANTIDADE
FROM [TABELA DE CLIENTES] A
INNER JOIN [NOTAS FISCAIS] B ON A.CPF = B.CPF
INNER JOIN [ITENS NOTAS FISCAIS] C ON B.NUMERO = C.NUMERO
GROUP BY A.CPF, A.NOME, B.DATA
```

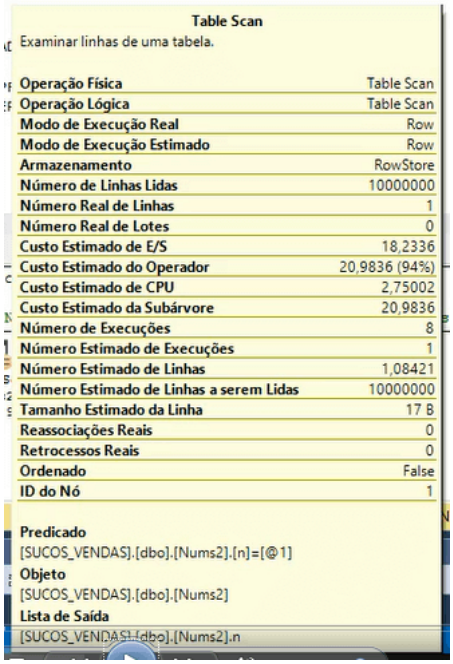
Você verá um plano de execução com várias ramificações:



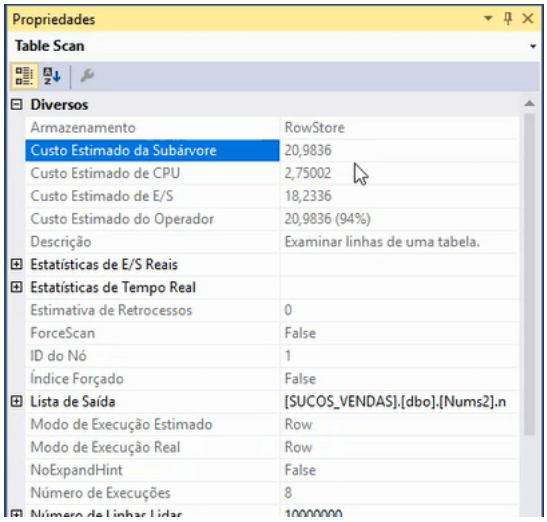
10) Execute o plano de execução da consulta:

```
SELECT N FROM Nums2 where N = '10001'
```

11) Ela vai usar o *iterator* `TABLE SCAN` , que percorre a tabela toda. Passando o mouse sobre cada fase do plano de execução, você pode ver as estatísticas de cada um:



12) Clicando sobre cada fase, você tem também, ao lado, uma janela com as mesmas estatísticas:

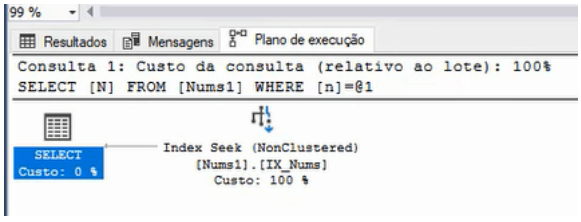


13) Clicando sobre a fase `SELECT` (final do plano de execução), você pode ver a estatística final:

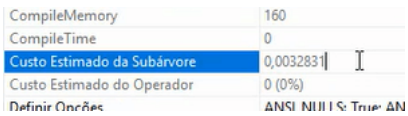
É aqui que você pode ver se uma consulta será mais ou não eficiente que a outra. O resultado final é visto em **Custo Estimado da Subárvore**.

14) Executando a primeira consulta, sabemos que ela usará *iterator* `SEEK`, ou seja, usando o índice:

```
SELECT N FROM Nums1 where N = '10001'
```



15) Comparando o resultado, com o `SEEK`, o valor do **Custo Estimado da Subárvore** será de 0,0032831:



Ou seja: bem mais eficiente.

16) Abra no **Management Studio** o arquivo **ConsultasIndices.sql** (caso você ainda não o tenha baixado, baixe-o aqui (https://caelum-online-public.s3.amazonaws.com/839-administracao-do-sql-server-2017/05/ConsultasIndices.sql)).

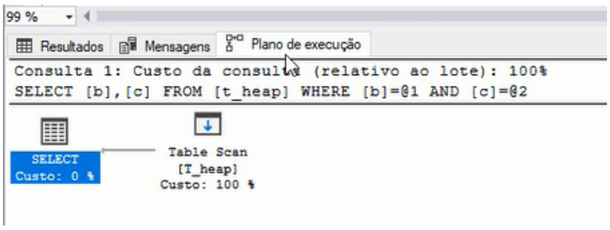
17) Execute, inicialmente, a criação de uma tabela `HEAP` :

```
CREATE TABLE T_heap (a int NOT NULL, b int NOT NULL, c int NOT NULL, d int NOT NULL, e int NOT NULL,
insert into T_heap (a,b,c,d,e,f) values ([dbo].[NumeroAleatorio](1,100), [dbo].[NumeroAleatorio](1,1
insert into T_heap (a,b,c,d,e,f) values ([dbo].[NumeroAleatorio](1,100), [dbo].[NumeroAleatorio](1,1
insert into T_heap (a,b,c,d,e,f) values ([dbo].[NumeroAleatorio](1,100), [dbo].[NumeroAleatorio](1,1
insert into T_heap (a,b,c,d,e,f) values ([dbo].[NumeroAleatorio](1,100), [dbo].[NumeroAleatorio](1,1
insert into T_heap (a,b,c,d,e,f) values ([dbo].[NumeroAleatorio](1,100), [dbo].[NumeroAleatorio](1,1
insert into T_heap (a,b,c,d,e,f) values ([dbo].[NumeroAleatorio](1,100), [dbo].[NumeroAleatorio](1,1
insert into T_heap (a,b,c,d,e,f) values ([dbo].[NumeroAleatorio](1,100), [dbo].[NumeroAleatorio](1,1
insert into T_heap (a,b,c,d,e,f) values ([dbo].[NumeroAleatorio](1,100), [dbo].[NumeroAleatorio](1,1
insert into T_heap (a,b,c,d,e,f) values ([dbo].[NumeroAleatorio](1,100), [dbo].[NumeroAleatorio](1,1
insert into T_heap (a,b,c,d,e,f) values ([dbo].[NumeroAleatorio](1,100), [dbo].[NumeroAleatorio](1,1
```

18) Execute diversos `SELECT` s e verifique o plano de execução, mesclando com criação de diversos índices. Primeiro execute, com a geração do plano de execução:

```
SELECT b, c FROM t_heap where b = 68 and c= 55
```

Vemos que a decisão do SQL Server foi de usar o `SCAN` :



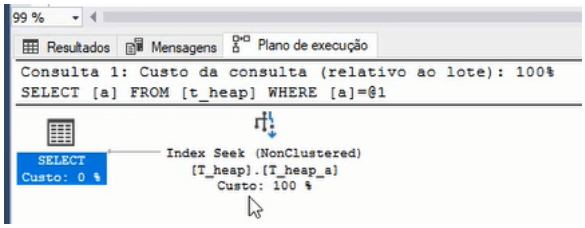
19) Agora, crie o índice conforme o comando abaixo:

```
CREATE NONCLUSTERED INDEX T_heap_a ON T_heap (a);
```

Executando o comando de seleção abaixo:

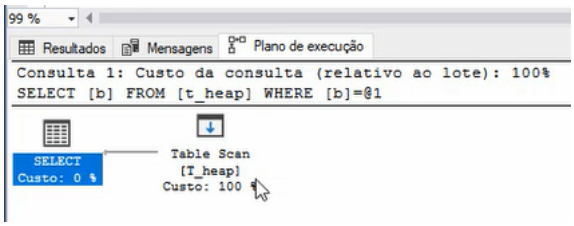
```
SELECT a FROM t_heap WHERE a = 1
```

Você verá o plano de execução usando o índice. Logo, o SQL Server opta pelo `SEEK` :



20) Já se você usar uma seleção que não envolve os campos que fazem parte do índice, volta ao `SCAN`:

```
SELECT b FROM t_heap WHERE b = 1
```

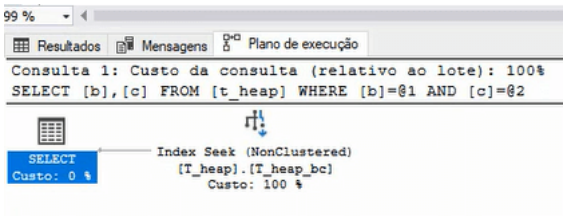


21) Crie um novo índice para os campos `b` e `c`:

```
CREATE INDEX T_heap_bc ON T_heap (b, c);
```

22) Executando a seleção usando `b` e `c`, você terá:

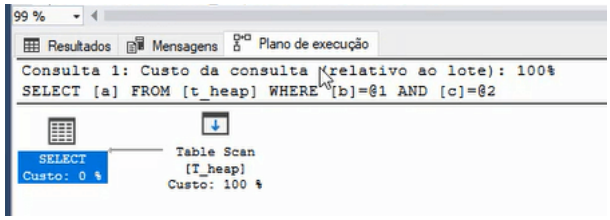
```
SELECT b, c  FROM t_heap WHERE b = 1 and c = 1
```



O SQL usou o novo índice para o `SEEK`.

23) Se você tirar o `b` ou o `c` da cláusula `SELECT`, ele volta ao `SCAN` porque, como o índice não é *clusterizado*, ele só vai usar o índice se os mesmos estiverem presentes na seleção:

```
SELECT a  FROM t_heap WHERE b = 1 and c = 1
```



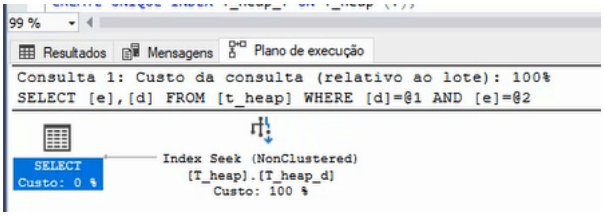
24) Foi criado um índice onde usa como critério de busca apenas um campo mas, no nível mais baixo do índice, associa com um outro campo. Isso se aplica somente a índices `NON CLUSTERED`:

```
CREATE INDEX T_heap_d ON T_heap (d) INCLUDE (e);
```

Se você executar a consulta:

```
SELECT d, e FROM t_heap WHERE d = 1 and e = 1
```

O resultado será:



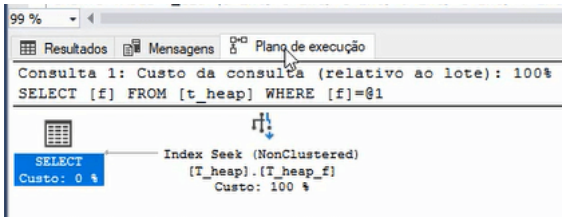
25) Você pode criar um índice único. Veja o comando:

```
CREATE UNIQUE INDEX T_heap_f ON T_heap (f);
```

E executando o comando:

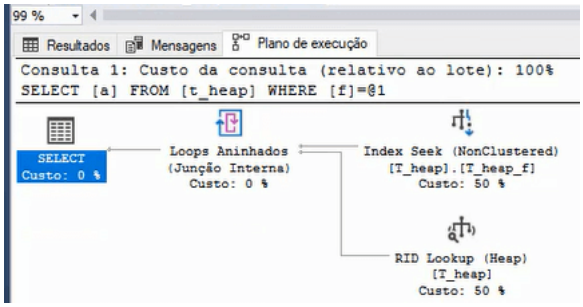
```
SELECT f FROM t_heap WHERE f = 1
```

Você verá o plano de execução:



26) Mas, se foi colocado um campo que não faz parte do índice na seleção, se ele não fosse `UNIQUE`, teria um `SCAN` (exemplo parecido como mostrado no passo 20). Mas aqui o plano de execução será diferente:

```
SELECT a FROM t_heap WHERE f = 1
```



O SQL Server vai usar uma técnica chamada **Consulta de Marcadores** (***Bookmark Lookup***), que é uma técnica onde o SQL vai buscar o resultado pelo índice e depois percorrer as páginas de memória do campo desejado. Ele divide o processo em dois:

um na busca do índice e o outro apenas nas páginas de memórias associadas ao elemento do índice que foi achado. Isso torna o processo mais rápido que o `SCAN`.

27) Execute novos comandos, criando uma outra tabela, igual a anterior. Mas nesta você irá criar índices *clusterizados*:

```
CREATE TABLE T_clu (a int, b int, c int, d int, e int, f int);

insert into T_clu (a,b,c,d,e,f) values (1, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](1,

insert into T_clu (a,b,c,d,e,f) values (2, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](1,

insert into T_clu (a,b,c,d,e,f) values (3, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](1,

insert into T_clu (a,b,c,d,e,f) values (4, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](1,

insert into T_clu (a,b,c,d,e,f) values (5, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](1,

insert into T_clu (a,b,c,d,e,f) values (6, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](1,

insert into T_clu (a,b,c,d,e,f) values (7, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](1,

insert into T_clu (a,b,c,d,e,f) values (8, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](1,

insert into T_clu (a,b,c,d,e,f) values (9, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](1,

insert into T_clu (a,b,c,d,e,f) values (10, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](1
```

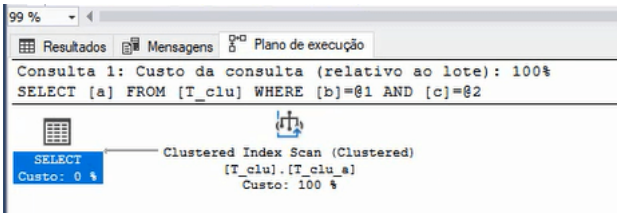
28) Crie, mas agora um índice *clusterizado*:

```
CREATE UNIQUE CLUSTERED INDEX T_clu_a ON T_clu (a);
```

29) Execute a consulta:

```
SELECT a FROM T_clu where b = 68 and c= 55
```

Se o índice não fosse *clusterizado*, não haveria o `SEEK` e sim o `SCAN`. Mas, como ele é *clusterizado*, teremos como plano de execução:



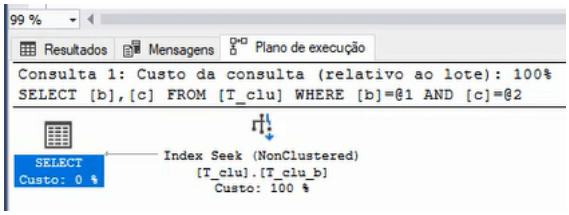
30) Crie um índice composto `NON CLUSTERED` nesta mesma tabela:

```
CREATE INDEX T_clu_b ON T_clu (b, c);
```

31) Se você executar a consulta:

```
SELECT b, c FROM T_clu where b = 50 and c = 50
```

O plano de execução dará preferência ao índice `NON CLUSTERED`, porque coincide com a seleção:



32) No **Management Studio**, abra o arquivo **Joins.sql** (caso você ainda não o tenha baixado, baixe-o [aqui (https://caelum-online-public.s3.amazonaws.com/839-administracao-do-sql-server-2017/05/Joins.sql)](https://caelum-online-public.s3.amazonaws.com/839-administracao-do-sql-server-2017/05/Joins.sql)).

33) Execute, inicialmente, a criação de uma tabela `HEAP`:

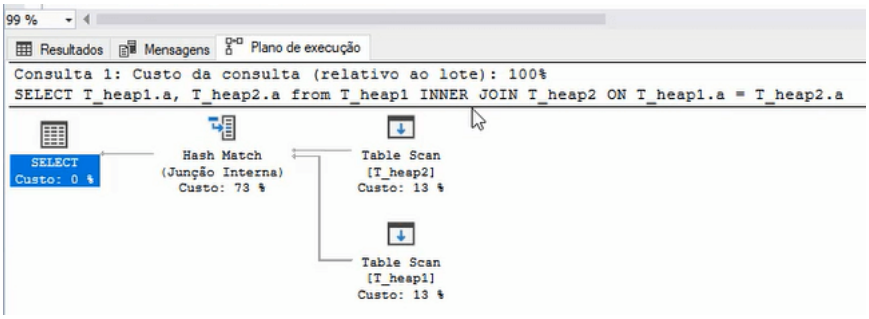
```
DROP TABLE T_heap1
DROP TABLE T_heap2
CREATE TABLE T_heap1 (a int NOT NULL, b int NOT NULL, c int NOT NULL, d int NOT NULL, e int NOT NULL

insert into T_heap1 (a,b,c,d,e,f) values (1, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](

insert into T_heap1 (a,b,c,d,e,f) values (2, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](

insert into T_heap1 (a,b,c,d,e,f) values (3, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](

insert into T_heap1 (a,b,c,d,e,f) values (4, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](

insert into T_heap1 (a,b,c,d,e,f) values (5, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](

insert into T_heap1 (a,b,c,d,e,f) values (6, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](

insert into T_heap1 (a,b,c,d,e,f) values (7, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](

insert into T_heap1 (a,b,c,d,e,f) values (8, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](

insert into T_heap1 (a,b,c,d,e,f) values (9, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](

insert into T_heap1 (a,b,c,d,e,f) values (10, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio]

insert into T_heap1 (a,b,c,d,e,f) values (11, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio]

insert into T_heap1 (a,b,c,d,e,f) values (12, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio]

insert into T_heap1 (a,b,c,d,e,f) values (13, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio]

insert into T_heap1 (a,b,c,d,e,f) values (14, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio]

insert into T_heap1 (a,b,c,d,e,f) values (15, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio]

insert into T_heap1 (a,b,c,d,e,f) values (16, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio]

insert into T_heap1 (a,b,c,d,e,f) values (17, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio]

insert into T_heap1 (a,b,c,d,e,f) values (18, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatori
```

```
insert into T_heap1 (a,b,c,d,e,f) values (19, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio]
insert into T_heap1 (a,b,c,d,e,f) values (20, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio]
insert into T_heap1 (a,b,c,d,e,f) values (21, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio]
CREATE TABLE T_heap2 (a int NOT NULL, b int NOT NULL, c int NOT NULL, d int NOT NULL, e int NOT NULL
insert into T_heap2 (a,b,c,d,e,f) values (1, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](
insert into T_heap2 (a,b,c,d,e,f) values (2, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](
insert into T_heap2 (a,b,c,d,e,f) values (3, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](
insert into T_heap2 (a,b,c,d,e,f) values (4, [dbo].[NumeroAleatorio](1,100),[dbo].[NumeroAleatorio](
```

34) Execute uma consulta que envolva um `JOIN`:

```
SELECT T_heap1.a, T_heap2.a from T_heap1 INNER JOIN T_heap2 ON T_heap1.a = T_heap2.a
```

Você terá como plano de consulta:



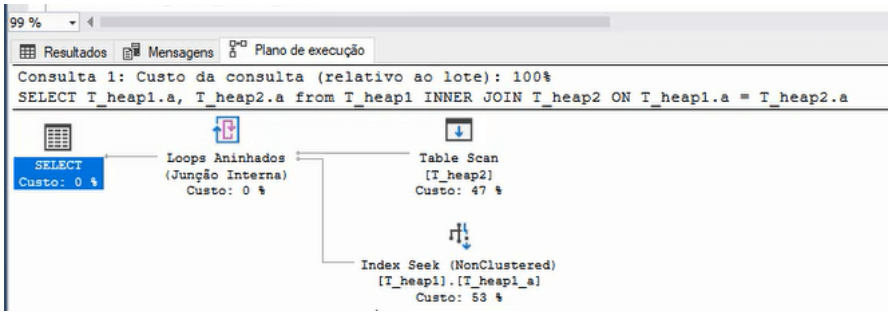
Note que haverá um `SCAN` nas duas tabelas.

35) Crie um índice para uma das tabelas:

```
CREATE INDEX T_heap1_a ON T_heap1 (a);
```

36) Voltando com a seleção, você terá um novo plano de execução:

```
SELECT T_heap1.a, T_heap2.a from T_heap1 INNER JOIN T_heap2 ON T_heap1.a = T_heap2.a
```



Somente um dos lados do `JOIN` terá índice.

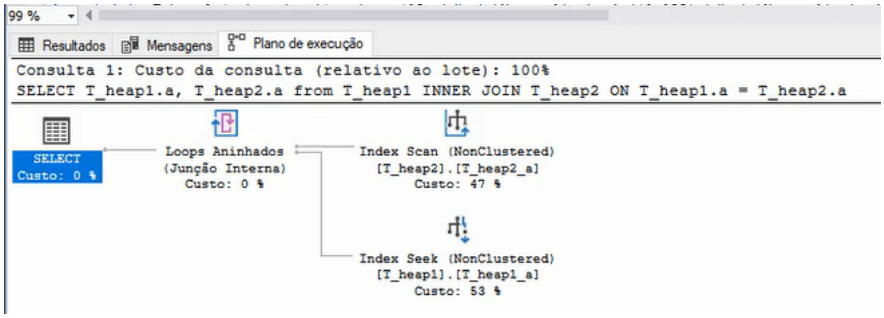
37) Para melhorar a consulta, crie o índice para a outra tabela:

```
CREATE INDEX T_heap2_a ON T_he,p2 (a);
```

Executando novamente a consulta:

```
SELECT T_heap1.a, T_heap2.a from T_heap1 INNER JOIN T_heap2 ON T_heap1.a = T_heap2.a
```

Você terá como resultado no plano de execução:



38) No **Management Studio**, abra o arquivo **SugestaoIndices.sql** (caso você ainda não o tenha baixado, baixe-o aqui (https://caelum-online-public.s3.amazonaws.com/839-administracao-do-sql-server-2017/05/SugestaoIndices.sql)).

39) Execute a consulta abaixo na base `SUCOS_VENDAS` :

```
SELECT A.CPF, B.NUMERO, C.QUANTIDADE FROM
[TABELA DE CLIENTES] A
INNER JOIN [NOTAS FISCAIS] B ON A.CPF = B.CPF AND A.CPF = '7771579779'
INNER JOIN [ITENS NOTAS FISCAIS] C ON B.NUMERO = C.NUMERO
```

40) E compare o plano de execução com a consulta:

```
SELECT A.CPF, B.NUMERO, C.QUANTIDADE FROM
[ITENS NOTAS FISCAIS] C
INNER JOIN [NOTAS FISCAIS] B ON B.NUMERO = C.NUMERO
INNER JOIN [TABELA DE CLIENTES] A ON A.CPF = B.CPF AND A.CPF = '7771579779'
```

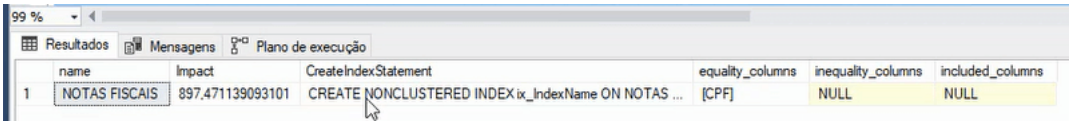
O resultado das duas consultas são os mesmos. Mas nos `JOIN` s, a ordem das tabelas é alterada. Você vê que o SQL Server monta o mesmo plano de consulta. Logo, a ordem dos `JOIN` s não altera o resultado final da consulta.

41) Para uma consulta, o SQL Server pode sugerir um índice para melhorar a consulta. Se você executar a consulta:

```
SELECT
    sys.objects.name,
    (avg_total_user_cost * avg_user_impact) * (user_seeks + user_scans) AS Impact,
    'CREATE NONCLUSTERED INDEX ix_IndexName ON ' + sys.objects.name COLLATE DATABASE_DEFAULT + ' (
```

```
        WHEN mid.inequality_columns IS NULL THEN ''
        ELSE CASE
                WHEN mid.equality_columns IS NULL THEN ''
                ELSE ','
            END + mid.inequality_columns
    END + ' ) ' + CASE
        WHEN mid.included_columns IS NULL THEN ''
        ELSE 'INCLUDE (' + mid.included_columns + ')'
    END + ';' AS CreateIndexStatement,
    mid.equality_columns,
    mid.inequality_columns,
    mid.included_columns
FROM sys.dm_db_missing_index_group_stats AS migs
INNER JOIN sys.dm_db_missing_index_groups AS mig
    ON migs.group_handle = mig.index_group_handle
INNER JOIN sys.dm_db_missing_index_details AS mid
    ON mig.index_handle = mid.index_handle
    AND mid.database_id = DB_ID()
INNER JOIN sys.objects WITH (NOLOCK)
    ON mid.OBJECT_ID = sys.objects.OBJECT_ID
WHERE (
    migs.group_handle IN (
        SELECT TOP (500) group_handle
        FROM sys.dm_db_missing_index_group_stats WITH (NOLOCK)
        ORDER BY (avg_total_user_cost * avg_user_impact) * (user_seeks + user_scans) DESC
    )
)
AND OBJECTPROPERTY(sys.objects.OBJECT_ID, 'isusertable') = 1
ORDER BY 2 DESC, 3 DESC
```

Você verá a lista de índices que podem ser criados para melhorar a performance do banco:



42) Se você criar o índice sugerido:

```
CREATE NONCLUSTERED INDEX Ind_00 on [NOTAS FISCAIS] ([CPF])
```

Executando a consulta novamente, você terá outro plano de consulta: