Índices Clusterizados e Não-Clusterizados

Adaptado de Edson Dionísio e Diogo Souza

Dentre as estratégias de otimização para bancos de dados os índices são os mais usados e atuam diretamente no sequenciamento dos dados mapeados, criando links para as linhas das tabelas e aumentando a performance das consultas. Um problema comum ao usar índices é achar que só precisamos espalhar vários deles pelas tabelas e teremos alta performance automaticamente. Este artigo trata de desmistificar o assunto, esclarecendo as diferenças entre dois tipos de índices: clusterizados e não-clusterizados, além de compreender como e quando adotá-los em seus projetos, seus prós e contras, etc. Você também verá exemplos práticos com medidores de performance para avaliar quem se sai melhor em cada situação.

Os índices são estruturas usadas para armazenar de forma eficiente dados de uma ou mais colunas de tabelas e/ou views para que possam ser recuperados mais rapidamente quando consultas forem feitas às mesmas.

Existem vários tipos de índices para os bancos de dados, como hash, columnstore, xml, etc. Neste artigo trataremos de explicar os dois mais usados: índices **clusterizados** e **não-clusterizados**. Usaremos o SQL Server para construir os exemplos do artigo; o leitor poderá encontrar uma relação dos demais tipos na URL disponibilizada na seção **Links**.

Clusterizado vs Não-Clusterizado

Um índice clusterizado determina a ordem em que as linhas de uma tabela são armazenadas no disco. Se uma tabela tem um índice clusterizado, no momento de um INSERT as linhas dessa tabela serão armazenadas em disco na ordem exata do mesmo índice. Por exemplo, suponha que temos uma tabela chamada “Livro” que tem uma coluna de chave primária “livroID” e que criamos um índice clusterizado para essa mesma coluna. Ao fazer isso, todas as linhas dentro da tabela Livro serão **fisicamente** ordenadas (no disco atual em que estão inseridas) através dos valores que estão na coluna livroID.

Isso implicará em um ganho enorme na performance das pesquisas, pois as colunas da tabela estarão ordenadas na mesma ordem dos índices clusterizados por intermédio do modelo de armazenamento usado por esse tipo de índice.

Já os índices não-clusterizados não fazem esse trabalho de ordenação dos dados tal qual é feito com os índices clusterizados. Em outras palavras, enquanto os índices clusterizados ordenam fisicamente tanto as linhas da tabela quanto os próprios índices e

mantêm os mesmos próximos uns aos outros; os não-clusterizados ordenam somente o índice em si, e não as linhas (que são salvas sempre de forma aleatória no disco).

Numa tabela, quando é definida uma chave primária, um índice clusterizado é criado automaticamente pelo banco para a mesma. E o que ele faz basicamente é ordenar as informações pela coluna de chave daquele índice em disco.

Vejamos um exemplo mais próximo da realidade, considere uma tabela de CEP (CEP, Endereço, Bairro, Cidade, etc.) com uma enorme quantidade de dados e que será muito pouco atualizada. Essa tabela será usada por um sistema onde um usuário informa o CEP num campo da tela e os outros campos de endereço devem ser carregados automaticamente após um *Enter*.

O primeiro critério para avaliar se a nossa tabela precisa de um índice é analisar quais colunas recebem mais consultas, portanto a coluna CEP. Para melhorar a performance nas pesquisas, o DBA tem como opção mais viável a criação de um índice nessa coluna. Vejamos as duas realidades caso ele optasse por índices clusterizados e não-clusterizados:

- Com um índice **não-clusterizado,** o banco salvaria os dados de CEP de forma aleatória no disco e armazenaria no índice apenas um ponteiro para o local onde o dado real está.
- Com um índice **clusterizado**, o banco salvaria os dados de CEP ordenados fisicamente e, sempre que um novo dado for inserido ou atualizado, teremos de arcar com o custo de reescrever os dados no índice para que os mesmos continuem ordenados.

Portanto, como a coluna CEP sofrerá poucas ou nenhuma atualização, a melhor estratégia é criar um índice clusterizado para a mesma. Todavia, se a coluna sofresse muitas atualizações, criar um índice clusterizado poderia invalidar o custo x benefício que vimos, com mais sobrecarga e menos performance. Neste caso, o leitor deve analisar os cenários e, através de testes de desempenho e stress, avaliar qual a melhor alternativa. Mais adiante faremos uma análise comparativa entre o uso de ambos os índices num exemplo prático, com medidores de desempenho para que você entenda melhor as diferenças entre os mesmos.

Vantagens x Desvantagens

Os índices clusterizados têm como principal vantagem a performance nas pesquisas: elas são mais rápidas em relação aos não-clusterizados. Isso se deve ao fato de as informações daquele índice e sua coluna respectiva estarem ordenadas e próximas na memória física do banco.

A principal desvantagem é o "custo" de novas escritas em disco que isso trará para o mesmo. Por exemplo, se uma linha em específico tiver seu valor atualizado (*UPDATE*) em uma de suas colunas de índice (clusterizados), o banco de dados irá mover a linha

inteira para que a tabela continue a ser ordenada na mesma ordem da coluna de índice clusterizado.

Imagine quantas vezes isso poderá ser feito em um sistema que atualiza muitos dados na referida tabela. Uma opção pode ser não usar o índice clusterizado nesse caso, mas o problema retorna quando a mesma tabela também passar a receber muitas pesquisas.

Para manter a organização o banco de dados precisará sempre rearranjar as linhas quando um UPDATE for efetuado. Isso trará consequências diretas na performance da base inteira e deve ser algo planejado antes.

Tanto os índices clusterizados quanto os não-clusterizados podem ser exclusivos. Isso significa que duas linhas não podem ter o mesmo valor que a chave de índice. Caso contrário, o índice não será exclusivo e várias linhas poderão compartilhar o mesmo valor de chave. Além disso, para cada tabela podemos ter apenas um índice clusterizado, ao passo que essa restrição não se aplica aos não-clusterizados. Essa limitação caracteriza mais um ponto a ser avaliado com cuidado quando do uso dessas estruturas em suas tabelas.

É possível ainda criar índices clusterizados e não-clusterizados em tabelas no SQL Server usando o SQL Server Management Studio ou o Transact-SQL.

Exemplo Prático

Para exemplificar na prática, vamos criar uma implementação comparativa e analisar a diferença em performance no uso dos dois tipos de índices. Para isso, é necessário que o leitor já tenha o SQL Server – 2005 ou superior - instalado na sua máquina (cuja instalação não fará parte do escopo deste artigo), além de baixado o arquivo do banco de exemplo que a Microsoft disponibiliza no seu site oficial; utilizaremos a base chamada AdventureWorks2012 para facilitar os testes e não termos de criar um exemplo do zero. Certifique-se de baixar a versão correspondente ao seu SQL Server, no nosso exemplo usaremos a versão 2014.

Descompacte o arquivo e importe-o na ferramenta do SQL. Os índices são criados automaticamente quando se é definida uma constraintdo tipo Primary Key (chave primária) para uma ou mais colunas. Assim, o banco assume que estas colunas que formam a chave primária de uma tabela podem ser também usadas para definir seu índice clusterizado. Em outras palavras, se no momento da criação da chave não forem definidos detalhes, será automaticamente criado um índice clusterizado sobre a(s) mesma(s) coluna(s).

A Primary Key é um conceito lógico que se destina a melhorar a implementação do modelo do banco de dados, enquanto que o índice clusterizado é um conceito físico para organizar as páginas de dados de uma tabela. Por isso, lembre-se que se tivermos

muitas instruções de INSERT, UPDATE e DELETE, é recomendado não usar esse tipo de índice para não perder desempenho organizando os dados a cada atualização.

A criação de um índice clusterizado pode ser feita tanto no desenvolvimento da tabela quanto via T-SQL's (extensão do SQL proprietária da Microsoft e Sybase). Vejamos então como seria essa aplicação a partir de uma T-SQL, de acordo com a **Listagem 1**.

Listagem 1. Criando uma tabela com índice clusterizado via T-SQL.

```
USE AdventureWorks2014;
  GO
  -- Criação da tabela de testes.
  CREATE TABLE dbo.TesteDevmedia
      (Coluna1 int NOT NULL,
       Coluna2 nchar(10) NULL,
       Coluna3 nvarchar(50) NULL);
  GO
  -- Criação do índice clusterizado chamado de IX_TesteDevmedia_Coluna1
  -- na tabela dbo.TesteDevmedia usando a coluna1.
  CREATE CLUSTERED INDEX IX_TesteDevmedia_Coluna1
      ON dbo.TesteDevmedia (Coluna1);
  GO
```

Para entender melhor o cenário de testes, o dividiremos em três passos: consultaremos primeiro coluna "*LastName*" da tabela Person.Person sem nenhum índice criado, com um índice não-clusterizado e, por último, com um índice clusterizado. Adicionalmente, faremos as mesmas consultas, porém envolvendo duas colunas (*LastName* e *FirstName*) de índice ao mesmo tempo, dessa forma o leitor poderá ver como os mesmos se comportam em cenários de índices mais complexos.

Mas antes disso, precisamos efetuar uma cópia da tabela Person.Person, apenas para que não precisemos alterar a estrutura da original caso você necessite efetuar testes futuros. Para isso, execute o script da **Listagem 2** que verificará, primeiramente, se a tabela já existe na base (excluindo-a caso positivo) e depois criará a cópia.

Listagem 2. Script para clonar tabela de Person.Person.

```
 USE AdventureWorks2014;
  GO

  IF EXISTS (SELECT * FROM sys.tables WHERE OBJECT_ID =
OBJECT_ID('Person.Person_Teste'))
         DROP TABLE Person.Person_Teste;
  GO

  SELECT * INTO Person.Person_Teste FROM Person.Person;
  GO
```

Além disso, antes de executarmos a consulta para o primeiro caso de teste (sem índices), precisamos nos assegurar que nenhum índice de nenhum tipo esteja associado às colunas de nome. Para tanto, execute o código da **Listagem 3** que se encarregará de varrer a tabela de índices e remover o de nome "*Name_Index*", que é o que daremos para o nosso futuro índice. A segunda parte da listagem também trata de remover o índice não-

clusterizado "*IX_Person_LastName_FirstName_MiddleName*" que já existe na base por padrão e engloba as três colunas de nome da tabela Person_Teste.

Listagem 3. Código utilizado para saber se o(s) índice(s) já existe(m).

```
 IF EXISTS (SELECT * FROM sys.indexes WHERE OBJECT_ID = OBJECT_ID('Person.Person_Teste')
AND name = 'Name_Index')
           DROP INDEX Person.Person_Teste.Name_Index;
         IF EXISTS (SELECT * FROM sys.indexes WHERE OBJECT_ID =
OBJECT_ID('Person.Person_Teste')
           AND name = 'IX_Person_LastName_FirstName_MiddleName')
           DROP INDEX Person.Person_Teste.IX_Person_LastName_FirstName_MiddleName;
```

Agora que passamos pelo primeiro passo, o próximo será habilitarmos a impressão de dados estatísticos via TSQL no SQL Server Management Studio, como segue:

```
SET STATISTICS io ON
SET STATISTICS time ON
GO
```

Esse TSQL basicamente diz ao SQL Server que queremos que as nossas consultas retornem as informações de desempenho como parte da saída. Como usaremos o recurso de ***Table Scan*** do SQL Server para exibir as informações estatísticas de cada consulta, bem como o desempenho das mesmas, é preciso que você habilite a opção no Management Studio clicando no ícone demonstrado na **Figura 1**.

Figura 1. Habilitando Table Scan no SQL Management Studio.

Consulta sem índices

Faremos agora uma busca pelo sobrenome (*LastName*) de uma pessoa dentro da base de exemplo e veremos o "esforço" com o qual a informação será retornada sem a utilização de índices. A seguir temos o nosso código para busca dos dados de uma pessoa em específico no banco:

```
SELECT *
    FROM Person.Person_Teste where LastName = 'Brown';
  GO
```

A execução trará um resultado de 92 linhas de um total de 19.972 linhas na tabela. Agora, basta acessar a aba "*Execution plan*" ao lado da aba "*Results*" e você verá algo semelhante ao que temos na **Figura 2**.

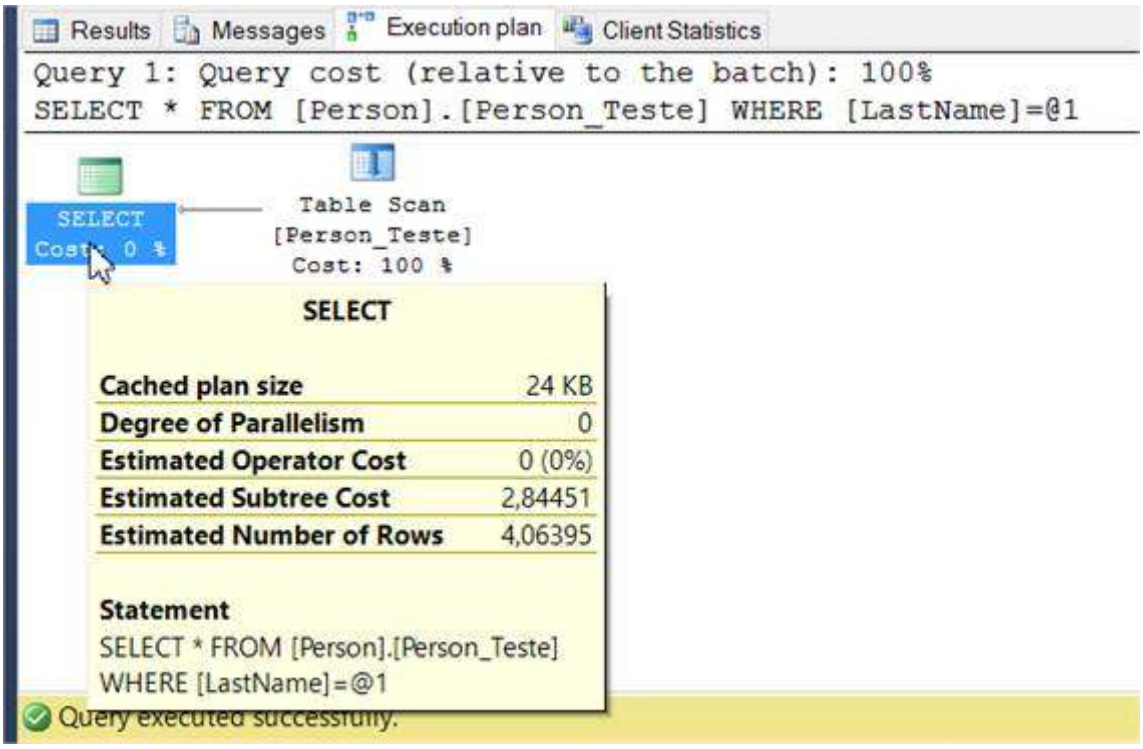

Figura 2. Tabela de custos para a execução do SELECT sem índices.

Como podemos ver na figura, essa query teve um "custo estimado de subárvore" de 2.84451. Esse valor representa o custo total do otimizador do SQL para executar não só essa query, mas todas as operações que a precederam na mesma subárvore. Quanto menor esse número, menor a intensidade da execução da referida query para o banco.

Consulta com índice não-clusterizado

Agora, para efetuar o mesmo teste, porém com um índice não-clusterizado, precisamos criar explicitamente um novo envolvendo a coluna usada na pesquisa. Crie, portanto, o índice tal como mostra a **Listagem 4**.

Listagem 4. Criação do índice não-clusterizado 'Name_Index'.

```
USE AdventureWorks2014;
  GO

  CREATE NONCLUSTERED INDEX Name_Index
      ON Person.Person_Teste (LastName);
  GO
```

Após isso, basta realizar uma nova consulta com base no mesmo script de SELECT com as mesmas informações usadas para a coluna de LastName e verificar o resultado na aba "*Execution plan*", tal como mostra a **Figura 3**. Veja que o custo caiu para 0.283268, isto é, uma redução mínima, principalmente por estarmos lidando com uma base não tão populosa assim, mas já é o suficiente para mostrar o ganho em qualquer tipo de consulta realizada com esse tipo de índice.

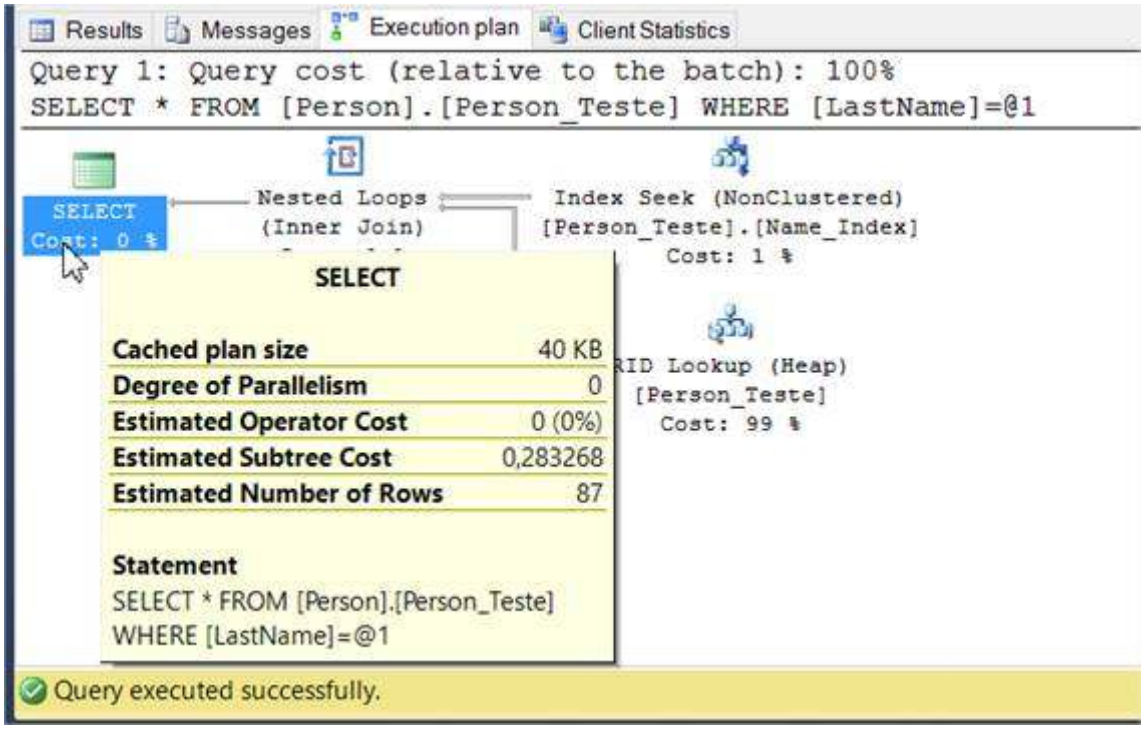

Figura 3. Tabela de custos para a execução do SELECT com um índice não-clusterizado.

Indice clusterizado

Para fazer o mesmo teste com um índice clusterizado, precisamos nos assegurar de remover o que criamos na listagem anterior, bem como criar o novo com a nova sintaxe. Para isso, execute o script contido na **Listagem 5**.

Listagem 5. Criação do índice clusterizado 'Name_Index'.

```
IF EXISTS (SELECT * FROM sys.indexes WHERE OBJECT_ID =
OBJECT_ID('Person.Person_Teste') AND name = 'Name_Index')
         DROP INDEX Person.Person_Teste.Name_Index;

  CREATE CLUSTERED INDEX Name_Index
      ON Person.Person_Teste (LastName);
  GO
```

Após a execução, rode mais uma vez o SELECT e acesse novamente a aba "*Execution plan*". Você verá algo semelhante à **Figura 4**.

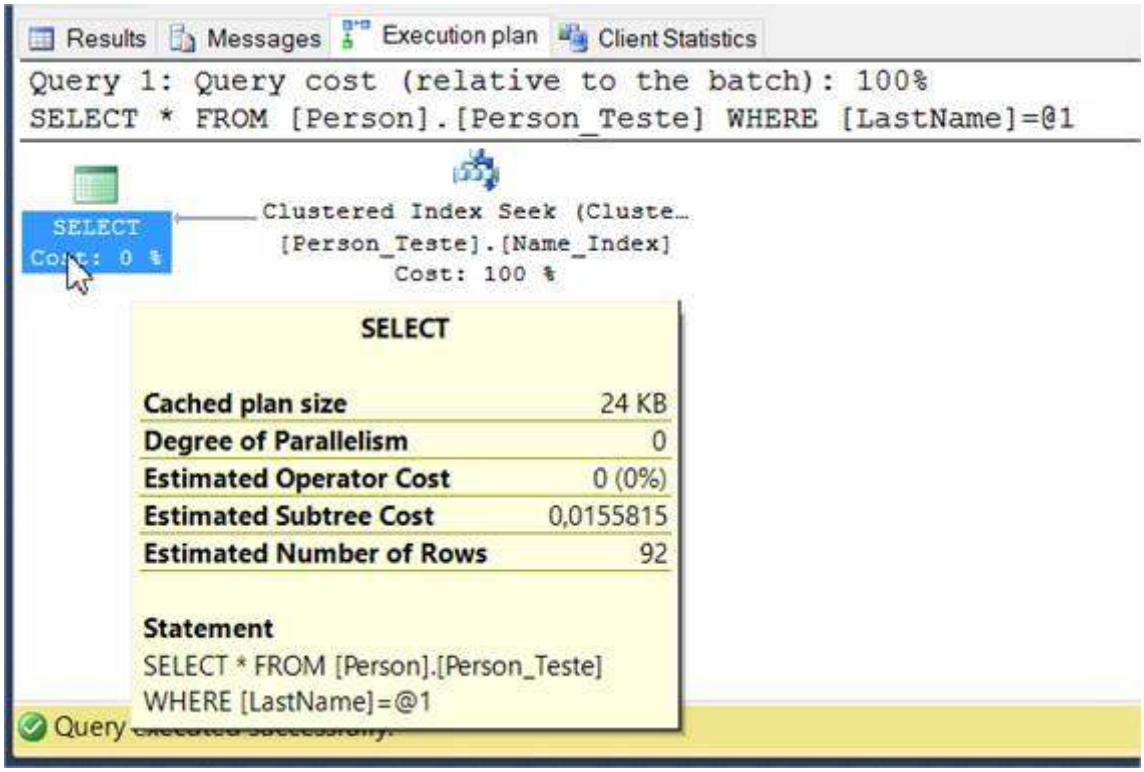

Figura 4. Tabela de custos para a execução do SELECT com um índice clusterizado.

Observe o enorme ganho que tivemos, mesmo com um exemplo bem simples e poucos dados na base. O valor de custo que era de 2.84451 caiu para 0.01558. Em termos de tempo final de execução, e levando em consideração uma massa expressiva de dados, teremos um ganho enorme usando a opção com índice clusterizado.

Consulta comparativa com índices de colunas compostas

Para entender ainda mais como o mecanismo de índices funciona veremos dois exemplos de uso dos dois tipos, porém agora com duas colunas de índice (índice composto). Para isso, execute o script demostrado na **Listagem 6**.

Listagem 6. Criação do índice não-clusterizado nas colunas de nome e sobrenome.

```
IF EXISTS (SELECT * FROM sys.indexes WHERE OBJECT_ID =
OBJECT_ID('Person.Person_Teste') AND name = 'Name_Index')
        DROP INDEX Person.Person_Teste.Name_Index;
```

```
  CREATE NONCLUSTERED INDEX Name_Index
      ON Person.Person_Teste (FirstName, LastName);
  GO
```

Após isso, vamos executar a mesma query (**Listagem 7**), incluindo agora a coluna de *FirstName*. Além disso, vamos complicar um pouco mais a consulta incluindo na busca também a coluna de primeiro nome (*FirstName*), e uma cláusula LIKE na condição para exigir que o banco trabalhe mais para encontrar os resultados.

Listagem 7. Consulta envolvendo as duas colunas de nome, com uma cláusula LIKE.

```
SELECT FirstName, LastName
    FROM Person.Person_Teste
         where FirstName like '%Jo%' and LastName = 'Brown';
  GO
```

O resultado pode ser visualizado na **Figura 5**. Para efetuar a comparação com o índice clusterizado composto, execute a query da **Listagem 8,** reexecute o mesmo SELECT e veja o respectivo resultado na **Figura 6**.

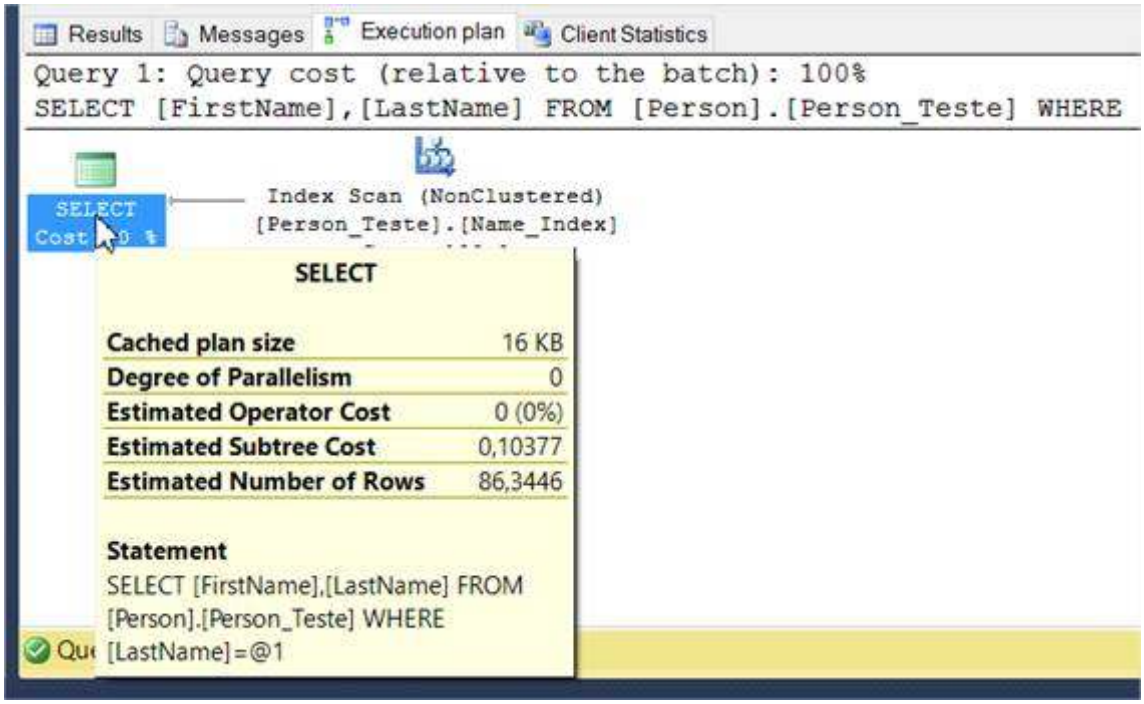

Figura 5. Tabela de custos para a execução do SELECT com um índice não-clusterizado composto.

Listagem 8. Criação do índice clusterizado nas colunas de nome e sobrenome.

```
IF EXISTS (SELECT * FROM sys.indexes WHERE OBJECT_ID =
OBJECT_ID('Person.Person_Teste') AND name = 'Name_Index')
         DROP INDEX Person.Person_Teste.Name_Index;

  CREATE NONCLUSTERED INDEX Name_Index
```

```
    ON Person.Person_Teste (FirstName, LastName);
GO
```

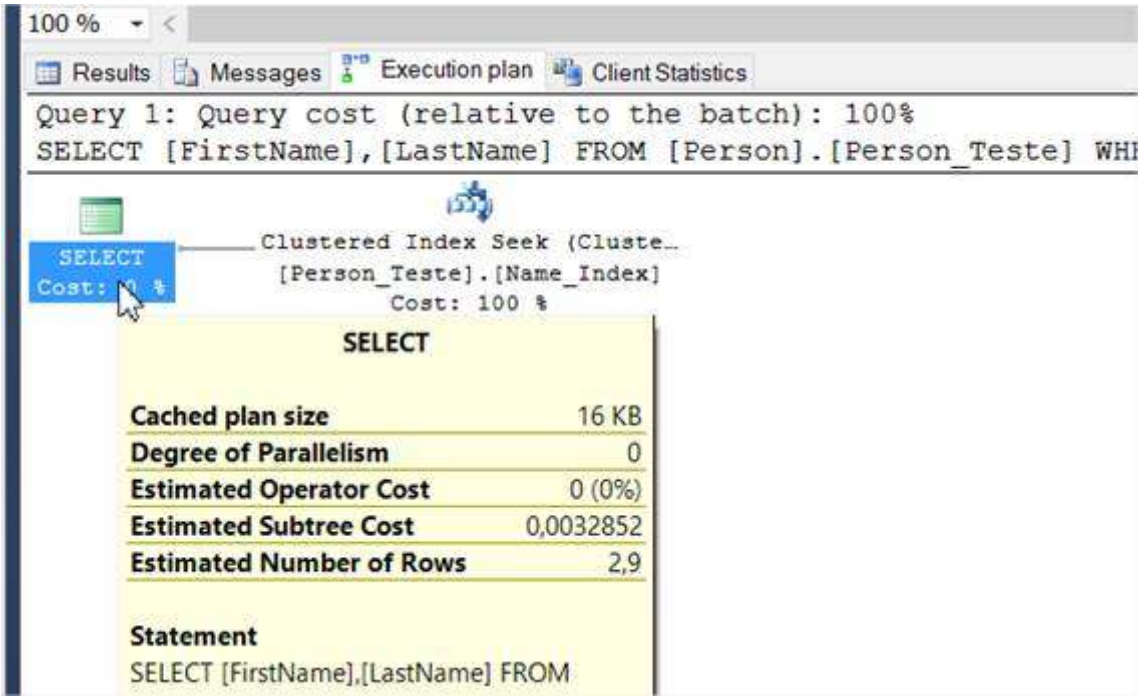

Figura 6. Tabela de custos para a execução do SELECT com um índice clusterizado composto.

Há momentos em que a utilização de índices clusterizados se torna um prejuízo, isso quando está relacionada às instruções de INSERT, UPDATE e DELETE. Em geral, as vantagens de recuperação superam os inconvenientes de manutenção, tornando um índice clusterizado bem visto pela comunidade de DBA's.

Além disso, lembre-se que cada tabela de um esquema pode ter vários índices não-clusterizados ao mesmo tempo, porém somente um clusterizado. Além disso, avalie sempre bem cada caso e verifique se o overhead gerado valerá a pena na performance final da sua base dados, fazendo sempre uso dos mecanismos de medição de performance que vimos no artigo. Até a próxima!

Links

Lista de índices SQL Server
https://msdn.microsoft.com/pt-br/library/ms175049.aspx

Índices Clusterizados
https://technet.microsoft.com/en-us/library/ms186342(v=sql.110).aspx

Índices Não-Clusterizados
https://technet.microsoft.com/en-us/library/ms189280(v=sql.110).aspx

Índices Clusterizados e Não-Clusterizados
https://msdn.microsoft.com/en-us/library/ms190457.aspx

AdventureWorks2014
http://msftdbprodsamples.codeplex.com/releases/view/93587