25 de setembro

Reduzindo IO com as DMV´s de Missing Indexes – Projeto Real

Pessoal, estive num projeto bem legal esses tempos. O cenário era o seguinte :

O cliente tinha comprado uma infra de ultima geração (SAN e um Blade, 32 GB de RAM 16 processadores..etc) em suma, um show de servidor mas alguns contadores estavam completamente fora do normal. Todo o aparato de disco, foi configurado nos best practices da MS (blocagem de 64K, stripe size de 128k, conjuntos raid 0+1 separados pra log e data, vazão da HBA em 256). Em um determinado banco, o mais acessado (Aprox 400 gb – Ao todo quase 4 TB), as leituras eram completamente anormais. Vamos dar uma olhadinha como estavam :

Physical Disk

Data	% Disk Read Time	Avg Disk Queue lenght	Avg Disk Sec transfer
28/07/2009	3300	542	37
29/07/2009	2700	580	35
30/07/2009	3000	540	30

SQL SERVER

Data	Page Life Expectancy
28/07/2009	80
29/07/2009	60
30/07/2009	75

Bom, uma média diária dinossáurica de disk reads, fila de disco absurda e expectativa de vida das páginas ridiculamente baixos.

Essa coleta eu fiz automatizada. O Perfmom capturava os dados de 15 em 15 segundos pra um csv e de 1 em 1 hora eu subia via job do SQL server. depois estarei postando essa rotina, é bem simples e funcional. Sou a favor da simplicidade.

O pior de tudo que no dia 2 ia entrar mais 4 filiais e um armazém online. O disco não ia suportar. Eu tinha dois dias pra fazer alguma coisa.

Estava com o profiler ligado, mas não tinhamos tempo útil para atacar as queries (que estavam bem mal escritas, os cursores pareciam luzinhas em árvore de natal e quem me

conhece sabe o que penso deles). Enfim, tinha que ser uma solução milagrosa. E pra isso nós temos disponível as fantásticas DMV´s de “missing indexes”.

São elas :

DMV	Informações Retornadas
sys.dm_db_missing_index_group_stats	Retorna informações de resumos sobre grupos de índice ausentes, por exemplo, as melhorias de desempenho que podem ocorrer ao implementar um grupo específico de índices ausentes.
sys.dm_db_missing_index_groups	Retorna informações sobre um grupo específico de índices ausentes, como o identificador de grupo e os identificadores de todos os índices ausentes que estão contidos naquele grupo.
sys.dm_db_missing_index_details	Retorna informações detalhadas sobre um índice ausente; por exemplo, retorna o nome e identificador da tabela onde o índice está ausente, e os tipos de coluna e colunas que deveriam formar o índice ausente.
sys.dm_db_missing_index_columns	Retorna informações sobre as colunas de tabela de banco de dados em que está faltando um índice.

Retirado do BOL 2008. Para quem quiser estudar mais sobre elas segue o link :

http://msdn.microsoft.com/pt-br/library/ms345524.aspx

Bom partindo delas, eu completamente a favor de não reinventar a roda, fui ao minha bíblia de SQL SERVER.

O site www.sqlservercentral.com. Lá encontrei este script : Util_MissingIndexes do Jesse Roberge.

http://www.sqlservercentral.com/scripts/Index+Management/63937/

Esta procedure basicamente faz o seguinte. Passado uma tabela como parâmetro, ela me informa quais os índices sugeridos para as consultas feitas nela.Ela me retorna algumas colunas, mas as mais importantes são :

Coluna	Descrição
unique_compiles	Número de Compilações e Recompilações que seriam beneficiados por este índice faltante.
user_seeks	Numero de Vezes que uma consulta foi efetuada e teria sido beneficiada por este índice faltante.
last_user_seek	Data e hora da ultima vez que uma consulta foi efetuada que seria beneficiada por este índice faltante.

avg_total_user_cost	Custo médio que seria reduzido na consulta se este índice fosse usado.
avg_user_impact	Média em percentual de benefício na consulta se este índice fosse usado.
equality_column	Colunas a serem criadas no índice que foram usadas na consulta com operador =
inequality_columns	Colunas a serem criadas no índice que foram usadas na consulta com operador “não igual” >,<,<>..etc.
included_columns:	Colunas a serem colocadas no índice pela cláusula include.

Para facilitar a leitura e no meu caso as que usei foram user_seeks, avg_user_impact e as colunas de sugestão do índice.

As filiais e o Armazém iriam começar a ser colocados online no dia 2 as 20:00 hrs. Neste mesmo dia, finalizado o trabalho, as 18:00 hrs os contadores eram estes :

Physical Disk

Data	% Disk Read Time	Avg Disk Queue lenght	Avg Disk Sec transfer
31/07/2009	1225	129	12,98293
01/08/2009	340	74	0,99393
02/07/2009	37	0,16785	0,38722
03/07/2009	20	0,15478	0,12452

SQL SERVER

Data	Page Life Expectancy
31/07/2009	900
01/07/2009	1400
02/07/2009	7000
03/07/2009	6800

Ou seja, tínhamos finalmente uma boa estrutura para a entrada das filiais.

Eu poderia ter usado as DMV´s de IO para me retornar as consultas com maior IO, mas como pelo profiler eu tinha visto que estava praticamente tudo com problema, peguei as maiores tabelas e fiz o trabalho. Me foquei nas colunas avg_user_impact e user_seek, pois pra mim era muito mais interessante melhorar 40% de uma consulta feitas 50 vezes do que 90% em uma feita 1 vez. Após estas estarem OK, fui para as com menor acesso.

Passado o sufoco, um segundo trabalho agora é revisar os índices que não estão sendo ou pouco usados. Revisar as consultas ..etc.

Pra falar a verdade eu esperava uma melhora, mas não desta maneira pois sou cético quanto a qualquer coisa sugerida automaticamente. Sou daqueles que usa o Index Tuning Wizard em alguns casos, pois prefiro manualmente revisar as consultas.

Óbvio que isso é uma coisa minha, mas em virtude desses números resolvi entender um pouco mais sobre estas DMV´s e algumas perguntas ficaram na minha cabeça :

1. Quando uma consulta usada por este índice é feita, a coluna user_seeks realmente é atualizada ?
2. Quando uma consulta usada por este índice é feita, a coluna last_user_seek realmente é atualizada ?
3. Quando as colunas sugeridas para o índice são exibidas, o otimizador leva em conta a seletividade, densidade enfim as informações estatísticas para esta sugestão ou ele somente coloca conforme a ordem do where ?
4. A coluna avg_user_impact informa o percentual real ?

Com base nessas questões resolvi fazer alguns testes.

Criei uma tabela e inseri 1000000 de linhas com seletividade diferente entre as colunas :

```
 CREATE TABLE [dbo].[TestesDmv](
   [Codigo1] [int] NOT NULL,
   [Codigo2] [int] NULL,
   [Codigo3] [int] NULL,
   [Codigo4] [int] NULL,
   [Campo1] [varchar](50) NULL,
   [Campo2] [varchar](50) NULL,
   [Campo3] [varchar](50) NULL,
   [Campo4] [varchar](50) NULL,
CONSTRAINT [PK_TestesDmv] PRIMARY KEY CLUSTERED
(
   [Codigo1] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) O
[PRIMARY]
) ON [PRIMARY]
```

Populei as colunas com as seletividades diferentes:

```
set nocount on


declare @codigo1 int = 0
declare @codigo2 int = 0
declare @codigo3 int = 0
declare @codigo4 int = 0


while @codigo1 < 1000000
begin
   if    (@codigo1 between 0 and 1000) or
      (@codigo1 between 2000 and 5000) or
      (@codigo1 between 7000 and 9000)or
      (@codigo1 between 10000 and 20000)  or
      (@codigo1 between 30000 and 40000)  or
      (@codigo1 between 40000 and 60000)  or
```

```
      (@codigo1 between 50000 and 80000)  or
      (@codigo1 between 80000 and 100000)  or
      (@codigo1 between 100000 and 200000) or
      (@codigo1 between 300000 and 400000) or
      (@codigo1 between 500000 and 500100) or
      (@codigo1 between 600000 and 700000) or
      (@codigo1 between 800000 and 900000)
         set @codigo2 = @codigo2
   else
         set @codigo2 = @codigo2 + 1


   if    (@codigo1 between 30000 and 60000)
   OR (@codigo1 between 200000 and 200100)
      set @codigo3 = @codigo3
   else
      set @codigo3 = @codigo3 + 1
   if    (@codigo1 between 10000 and 20000)
      set @codigo4 = @codigo4 + 1
   else
      set @codigo4 = @codigo4


   insert into TestesDmv(Campo1,Campo2,Campo3,Campo4,Codigo1,Codigo2,Codigo3,Codigo4)
   values ('campo1 ' + CAST(@codigo1 as CHAR(10)),
         'campo2 ' + CAST(@codigo2 as CHAR(10)),
         'campo3 ' + CAST(@codigo3 as CHAR(10)),
         'campo4 ' + CAST(@codigo4 as CHAR(10)),
         @codigo1,
         @codigo2,
         @codigo3,
         @codigo4
         )


   set @codigo1 = @codigo1 + 1


end
```

Então rodei a procedure (nao colocarei todos as colunas retornadas e sim somente as necessárias)

exec Util_MissingIndexes ",'testesdmv'

last_user_seek	user_seek	avg_user_impact	equality_columns	inequality_columns	included_columns

Nada ainda pra fazer. Rodei novamente com esta consulta :

```
Select    campo1,
          campo2,
          campo3,
          campo4
from       testesdmv
where    codigo1 = 10

go
```

exec Util_MissingIndexes ",'testesdmv'

Como a consulta era feita com base na PK (indice cluster criado nela), não me retornou nada ainda.

Passei para a próxima.

```
Select    campo1,
          campo2,
          campo3,
          campo4
from      testesdmv
where     codigo3 between 10 and 10000
and       codigo2 between 1 and 1000

go
exec Util_MissingIndexes ",'testesdmv'
```

last_user_seek	user_seek	avg_user_impact	equality_columns	inequality_columns	include_columns
2009-09-24 01:25:58.590	1	57,91	NULL	[Codigo2], [Codigo3]	[Campo1], [Campo2], [Campo3], [Campo4]

Bom, já temos alguma coisa.

User_seeks = 1
Last_User_seek = 2009-09-24 01:25:58.590

Rodei Novamente

```
Select    campo1,
          campo2,
          campo3,
          campo4
from      testesdmv
where     codigo3 between 10 and 10000
and       codigo2 between 1 and 1000

go
exec Util_MissingIndexes ",'testesdmv'
```

last_user_seek	user_seek	avg_user_impact	equality_columns	inequality_columns	include_columns
2009-09-24 01:27:41.653	2	57,91	NULL	[Codigo2], [Codigo3]	[Campo1], [Campo2], [Campo3], [Campo4]

User_seeks = 2
Last_User_seek = 2009-09-24 01:27:41.653

Bom, desta maneira podemos confiar na atualização do user_seek e last user_seek

Assim, minhas 2 primeiras perguntas foram respondidas :

1. Quando uma consulta usada por este índice é feita, a coluna user_seeks realmente é atualizada ?
2. Quando uma consulta usada por este índice é feita, a coluna last_user_seek realmente é atualizada ?

A resposta é SIM.

Usando o resultado da procedure podemos ver também as colunas que foram sugeridas como índice :

Equality_columns = NULL

Inequality_columns = [Codigo2], [Codigo3]

Included_columns = [Campo1], [Campo2], [Campo3], [Campo4]

Por que a coluna equality_columns ficou null ?

Isso se deve ao fato da minha condição where não usar o operador de igualdade (=) e sim between (Valor >= e Valor <=)

where codigo3 between 10 and 10000

and codigo2 between 1 and 1000

E porque existem valores nas colunas include ?

Para que o SQL server não precise ir nas páginas de dados as colunas da consulta teriam que ser contempladas no índice. Isso se chama covered indexes. Caso elas não estejam, o SQL server tem que ir até a pagina de dados para retornar as colunas faltantes. Este processo se chama BookMark Lookup (SQL2K) e RID Lookup (SQL2K5) e é dispendioso para ser feito.(quem quiser saber mais sobre isso, procure o webcast do Luti sobre índices).

Mas a ordem sugerida é codigo2,codigo3 e não codigo3,codigo2 como é feito no where. Nós bem sabemos que neste caso, a ordem no where não afetará o uso do índice pois o otimizador é inteligente o bastante para fazer a troca caso seja necessária.

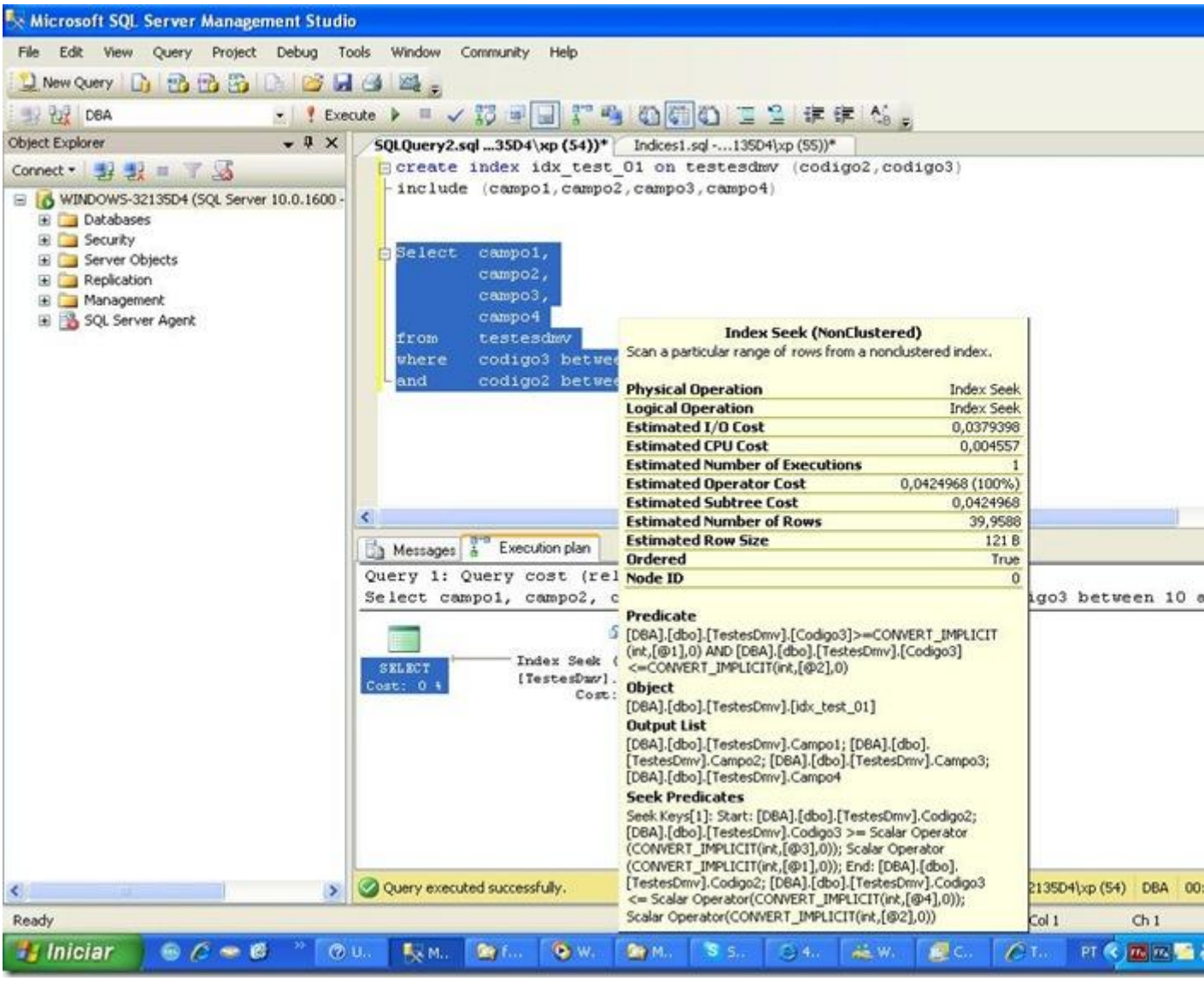
Deve ter algum motivo. Vamos criar como ele pediu :

```
create    index idx_test_01 on testesdmv (codigo2,codigo3)
include (campo1,campo2,campo3,campo4)
go
exec Util_MissingIndexes '','testesdmv'
```

last_user_seek	user_seek	avg_user_impact	equality_columns	inequality_columns	include_columns

Bom, o índice pedido já não existe mais. Mas fiquei intrigado quanto a ordem. Resolvi ver as estatísticas e o plano de execução :

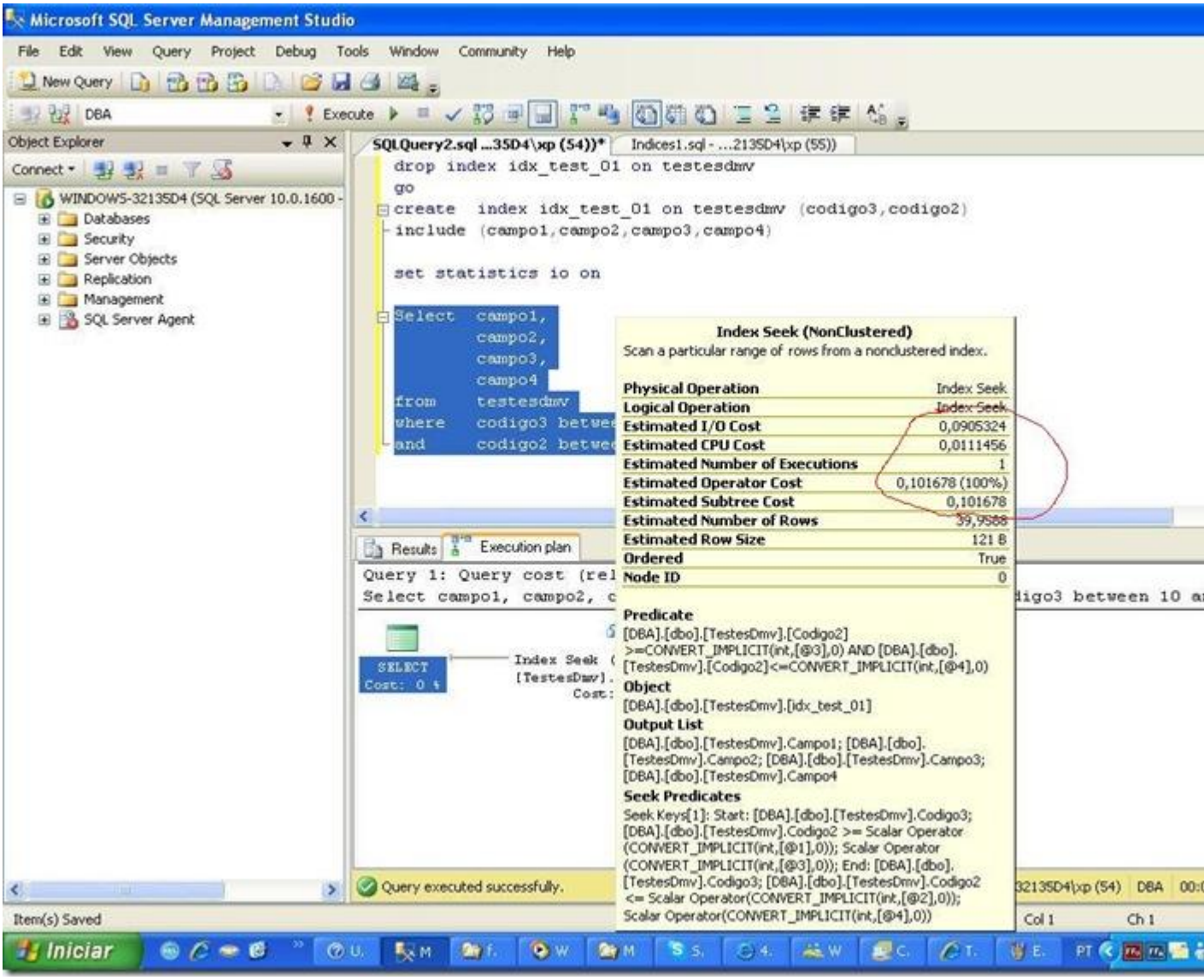
Table 'TestesDmv'. Scan count 1, **logical reads 52**, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.



Agora vamos criar na ordem da cláusula where e fazer a mesma comparação :

drop index idx_test_01 on testesdmv
go
create index idx_test_01 on testesdmv (codigo3,codigo2)
include (campo1,campo2,campo3,campo4)

Table 'TestesDmv'. Scan count 1, **logical reads 123**, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.



Podemos ver que além do logical reads, o custo de IO e cpu aumentaram. Ou seja, o otimizador sabe o que faz. Mas eu precisava fazer mais um teste, usando where com operador =

```
Select    campo1,
          campo2,
          campo3,
          campo4
from      testesdmv
where     codigo4 = 10001
and       codigo3 = 30000

go
exec Util_MissingIndexes '','testesdmv'
```

last_user_seek	user_seek	avg_user_impact	equality_columns	inequality_columns	include_columns
2009-09-25 13:42:04.043	1	98,39	[Codigo3], [Codigo4]	Null	[Campo1], [Campo2], [Campo3], [Campo4]

Por que não temos dados na coluna inequality. Isto se deve pelo fato que meu where só tem igualdade.

Novamente a coluna sugerida foi na ordem diferente da condição where.

Agora vamos mudar um pouco, vamos ver a seletividade destas colunas :

Este script calcula a seletividade das colunas (Agradecimentos ao Nilton Pinheiro. Você pode achar este script em http://www.mcdbabrasil.com.br/modules.php?name=News&file=article&sid=176)

```
SELECT  [Total Lines] = COUNT(*),
          [Distinct Lines] = COUNT(DISTINCT <Column>),
       -- quanto mais perto de 1 melhor
          [selectivity] = COUNT(DISTINCT <Column>)/CAST( COUNT(*) AS
DEC(10,2)) FROM    <yourtable>

 SELECT  [Total Lines] = COUNT(*),
       [Distinct Lines] = COUNT(DISTINCT codigo3),
       -- even more close to 1 better
       [selectivity] = COUNT(DISTINCT codigo3)/CAST( COUNT(*) AS DEC(10,2))
FROM    testesdmv

SELECT  [Total Lines] = COUNT(*),
       [Distinct Lines] = COUNT(DISTINCT codigo4),
       -- even more close to 1 better
       [selectivity] = COUNT(DISTINCT codigo4)/CAST( COUNT(*) AS DEC(10,2))
FROM    testesdmv

Codigo3
Total Lines Distinct Lines selectivity
-----------  -------------  ---------------------------------------
1000000     969898         0.96989800000

(1 row(s) affected)

Codigo4
Total Lines Distinct Lines selectivity
-----------  -------------  ---------------------------------------
1000000     10002          0.01000200000
```

(1 row(s) affected)

Como a gente pode ver, a coluna codigo3 é bem mais seletiva que a codigo4. Outro Gol do otimizador.

Mas e se eu tiver um where usando colunas com igual e “não igual” (><,<>..)

```
Select    campo1,
          campo2,
          campo3,
          campo4
from      testesdmv
where     codigo2 between 100 and 100000
and       codigo4 = 10001
and       codigo3 = 30000

go
exec Util_MissingIndexes ",'testesdmv'
```

last_user_seek	user_seek	avg_user_impact	equality_columns	inequality_columns	include_columns
2009-09-25 13:55:04.043	1	98,81	[Codigo3], [Codigo4]	[Codigo2]	[Campo1], [Campo2], [Campo3], [Campo4]

Equality_columns = [Codigo3],[Codigo4]
Inequality_columns = [Codigo2]
Included_columns = [Campo1], [Campo2], [Campo3], [Campo4]

Como a gente bem viu o otimizador sabe o que faz, então neste caso temos que criar o indice colocando equality columns primeiro e inequality logo após. Ficari a ssim :

```
Create index Idx_test_02 on testesdmv(codigo3,codigo4,codigo2)
include (campo1,campo2,campo3,campo4)
```

Bom, minha terceira pergunta estava respondida

- Quando as colunas sugeridas para o índice são exibidas, o otimizador leva em conta a seletividade, densidade enfim as informações estatísticas para esta sugestão ou ele somente coloca conforme a ordem do where ?

Não. O otimizador não usa a ordem do where. Ele coloca a ordem das colunas trabalhando com os dados estatisticos que ele tem (seletividade, densidade..etc).

Bom, eu não esperava menos do **optimization team** do SQL Server.

Finalmente me faltou ver se o avg_user_impact é realmente condizente. Vamos lá ?

```
dbcc dropcleanbuffers
go
set statistics time on
go
Select    campo1,
       campo2,
       campo3,
       campo4
from    testesdmv
where    codigo2 between 100 and 1000000
and     codigo4 = 10001
and       codigo3 = 30000
```

The result was :

```
(30002 row(s) affected)

SQL Server Execution Times:
   CPU time = 423 ms,  elapsed time = 4471 ms.
```

Agora criamos o índice

```
CREATE NONCLUSTERED INDEX Idx_test_03
ON [dbo].[TestesDmv] ([Codigo3],[Codigo4],[Codigo2])
include (campo1,campo2,campo3,campo4)

go
```

```
dbcc dropcleanbuffers
go
set statistics time on
go
Select    campo1,
       campo2,
       campo3,
       campo4
from    testesdmv
where    codigo2 between 100 and 1000000
and     codigo4 = 10001
and       codigo3 = 30000
```

The result set

(30002 row(s) affected)

SQL Server Execution Times:
  CPU time = 31 ms, elapsed time = 1308 ms.

Bom, o tempo foi reduzido de 4471 para 1308. Minha pergunta está respondida.

- A coluna avg_user_impact informa o percentual real ?

Sim. Muito próximo do real.

Bom, como a gente pode ver esta feature incluida no SQL SERVER é muito poderosa. Claro que a análise tem que ser bem feita, pois não saimos criando índices assim, como compramos pão. Mas se precisar, podemos confiar nas DMV´s.

Os dados das DMV´s são apagados no restart do SQL server e elas tem algumas limitações. Acessem o link que coloquei no início do artigo para entender melhor.

Bom galera é isso aí. Espero que como essa feature me ajudou muito, possa ajudar vocês também.

Abraços !!!!
Laerte Junior
http://laertejuniordba.spaces.live.com/blog/