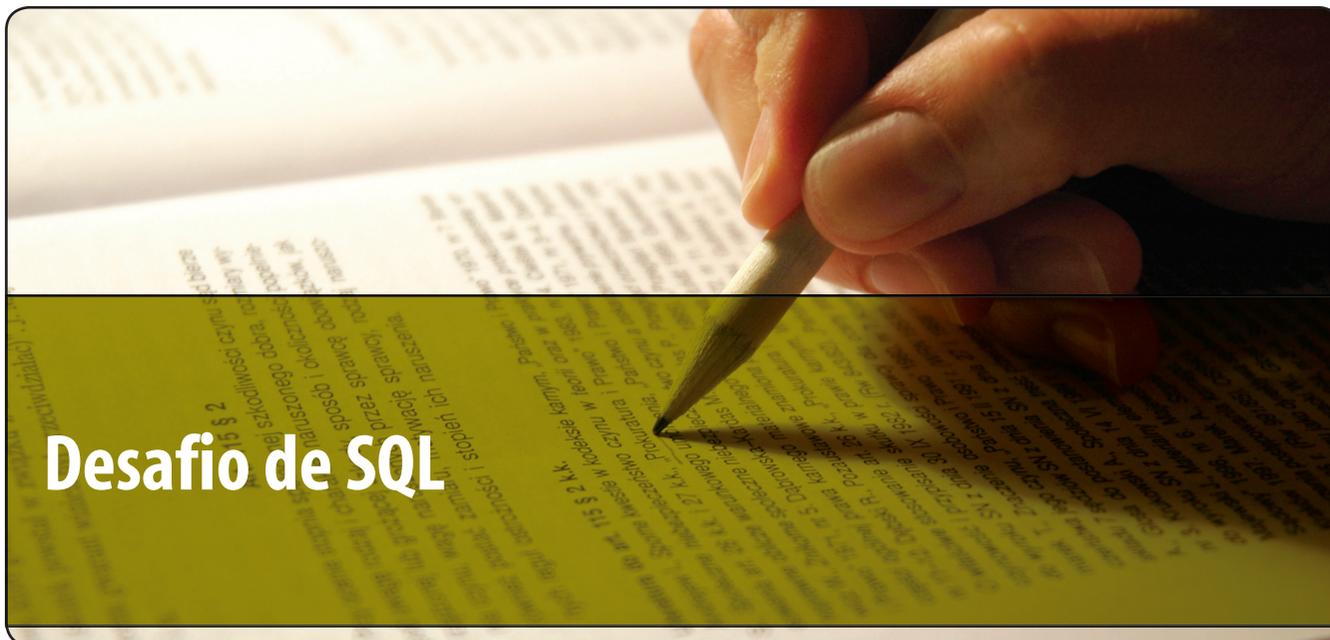


Nesta seção você encontra artigos sobre banco de dados, SQL ou persistência



## Desafio de SQL

Em continuidade à série de desafios que Wagner Crivelini nos brinda, desta vez o desafio é identificar qual a solução adotada no desafio anterior que nos traz melhor desempenho.

Um constante fantasma na vida de todo DBA/Desenvolvedor é a análise do plano de execução de uma consulta e é justamente este fantasma que Wagner vem exorcisar. E, novamente, a solução deste desafio usará o padrão ANSI, o que permite a implementação da mesma solução e vários SGBDs.

E caso você tenha um desafio que gostaria de compartilhar conosco, envie sua sugestão para [webeditor@sqlmagazine.com.br](mailto:webeditor@sqlmagazine.com.br). Divirtam-se.

Ricardo Rezende  
Editor técnico



**Wagner Crivelini**

[wcrivelini@gmail.com](mailto:wcrivelini@gmail.com)

Atua a 15 anos na área de Business Intelligence. Engenheiro formado pela UNICAMP, trabalha na IBM na unidade de Global Business Services. Atualmente ocupa a posição de líder técnico em projeto internacional na área de telecomunicações.

No Desafio SQL da edição 57 apresentamos dois métodos diferentes para se identificar quais valores não aparecem numa tabela que deveria ter códigos seqüenciais.

O desafio tratava de testes de uma aplicação de ativação de linhas telefônicas. Os casos de teste usavam números de telefones (ou TNs, na sigla em inglês) seqüenciais, mas alguns testes haviam falhado. Nosso trabalho foi identificar quais TNs não constavam na tabela de destino.

No portal da SQL Magazine você encontrará disponível para download o script completo do desafio anterior. O script inclui as instruções para reprodução do ambiente (tabelas usadas) e mais as duas soluções apresentadas.

Ambas soluções usavam a tabela de destino tbITN. A diferença residia na forma de emular a lista de TNs testadas. Nós criamos uma tabela de origem usando estratégias diferentes:

- 1 - a primeira solução usou uma variável de tabela, inserindo os TNs através de um laço;
- 2 a segunda solução usou um contador

### De que se trata o artigo?

Desenvolvimento de soluções para problemas cotidianos enfrentados por DBAs e desenvolvedores de aplicações para banco dados.

### Para que serve?

Fornecer conceitos de utilização de funcionalidades do padrão SQL ANSI na resolução de problemas enfrentados no dia-a-dia na recuperação de informações do banco de dados.

### Em que situação o tema é útil?

O desafio deste mês é útil na análise de performance de implementações de soluções de recuperação de informações do banco de dados através da análise do plano de execução.

de registros atuando numa tabela pré-existente.

A **Listagem 1** apresenta a primeira solução, enquanto que a **Listagem 2** mostra a segunda.

Mas restou uma questão a ser resolvida.

Se tínhamos ao menos duas soluções possíveis, qual delas deveríamos usar? Qual delas tinha melhor performance?

Este é o seu desafio neste mês: como fazer para comparar duas soluções. Mãos à obra.

### Resposta do desafio

Avaliar a performance de uma ou mais declarações SQL é uma necessidade muito comum. Mas apesar da enorme importância, pouca coisa se fala e se escreve a respeito da análise do plano de execução de consultas (**Nota DevMan**).

Como veremos aqui, isto não é nenhum bicho de sete cabeças. Vários SGBDs (inclusive os gratuitos) possuem ferramentas para ajudá-lo a entender o plano de execução.

Existem ferramentas gráficas e outras textuais, que inclusive geram arquivos XML. Nós mostraremos aqui os recursos básicos do SQL SERVER 2005.

Quando executamos uma declaração SQL ou um lote de declarações, a primeira ação que ocorre no SGBD é submeter a(s) sentença(s) a um processo de análise (ou parse, na expressão em inglês) para verificar se existem erros.

Uma vez que a declaração é validada, o banco de dados gera o que chamamos de árvore de consulta, que é uma seqüência de instruções que informam ao mecanismo do SGBD como lidar com a(s) sentença(s) em questão.

Finalmente, a árvore de consulta é analisada pelo otimizador de consultas do SGBD (ou Query Optimizer, para quem preferir). Este “cara” define se e quais índices serão utilizados, quais os tipos de junções de tabelas serão usados, etc. O objetivo é executar a declaração SQL da maneira mais eficiente possível (leia-se: a mais rápida).

Como qualquer software, nem sempre o otimizador de consultas consegue resultados satisfatórios. O resultado depende muito da maneira como nós informamos o que desejamos fazer, ou seja, da declaração SQL que escrevemos.

Todas as decisões do otimizador de consultas são baseadas no conceito de “custo” de cada operação. Este custo se baseia no tempo de uso de CPU e quantidade de I/O requerida. Portanto, quanto menor o custo

#### Listagem 1. Solução usando variáveis de tabela (T-SQL).

```
1 --1 definindo as variáveis principais e valores iniciais
2 DECLARE @intTNI inicial INT
3 DECLARE @intQtdTN INT
4 DECLARE @intContador INT
5
6 SET @intTNI inicial = 1131334101
7 SET @intQtdTN = 10
8
9 --2 definindo variável de tabela
10 DECLARE @tblVIRTUAL TABLE (codTN INT)
11
12 --3 laço para popular a variável de tabela
13 SET @intContador = 1
14 INSERT INTO @tblVIRTUAL (codTN) VALUES (@intTNI inicial)
15
16 WHILE @intContador < @intQtdTN
17 BEGIN
18 SET @intTNI inicial = @intTNI inicial + 1
19
20 INSERT INTO @tblVIRTUAL (codTN) VALUES (@intTNI inicial)
21 SET @intContador = @intContador + 1
22 END
23
24 --4 comparando as tabelas
25 SELECT codTN AS Ausente
26 FROM @tblVIRTUAL
27 WHERE codTN NOT IN
28 (SELECT codTN FROM tblTN)
29;
```

#### Listagem 2. Solução usando contador de registros

```
1 --1 definindo as variáveis principais e valores iniciais
2 DECLARE @intTNI inicial INT
3 DECLARE @intQtdTN INT
4
5 SET @intTNI inicial = 1131334101
6 SET @intQtdTN = 10
7
8 SELECT @intTNI inicial
9 + (
10 SELECT COUNT(*)
11 FROM tblNotaFiscal X
12 WHERE X.codNotaFiscal < N.codNotaFiscal
13 )
14 FROM tblNotaFiscal N
15 WHERE (
16 SELECT COUNT(*)
17 FROM tblNotaFiscal X
18 WHERE X.codNotaFiscal < N.codNotaFiscal
19 ) < @intQtdTN
20
21 AND @intTNI inicial + (
22 SELECT COUNT(*)
23 FROM tblNotaFiscal X
24 WHERE X.codNotaFiscal < N.codNotaFiscal
25 )
26 NOT IN (SELECT codTN FROM tblTN);
```



### Nota do DevMan

#### Plano de Execução

Para executar uma declaração DML, o banco de dados necessita executar várias etapas. Cada uma dessas etapas é necessária tanto para recuperar registros fisicamente no banco de dados ou prepará-los de alguma forma para o usuário que envia a declaração.

A combinação de todos os passos que o banco de dados usa para executar uma declaração SQL é chamada de plano de execução. Um plano de execução inclui também o método de acesso para cada tabela que a declaração referencia e a ordem das tabelas (a ordem de junção – join order).

de execução, melhor a performance.

E aqui chegamos ao ponto que nos interessa: o plano de execução da consulta vai nos mostrar exatamente este custo para execução da nossa declaração SQL.

Eu costumo usar a apresentação gráfica do plano de execução de uma consulta. Para ativar sua exibição, basta abrir uma janela de consulta no SQL Management Studio e clicar no botão SHOW ACTUAL

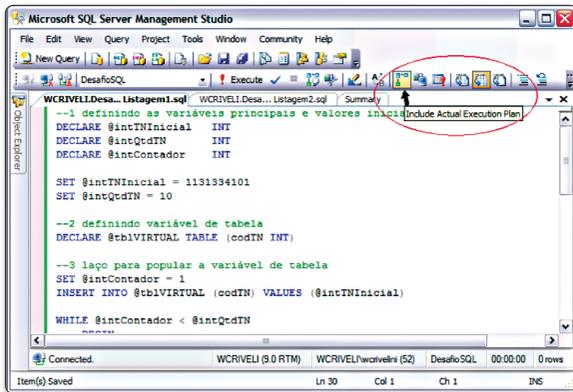


Figura 1. Ativação do plano de execução de uma consulta.

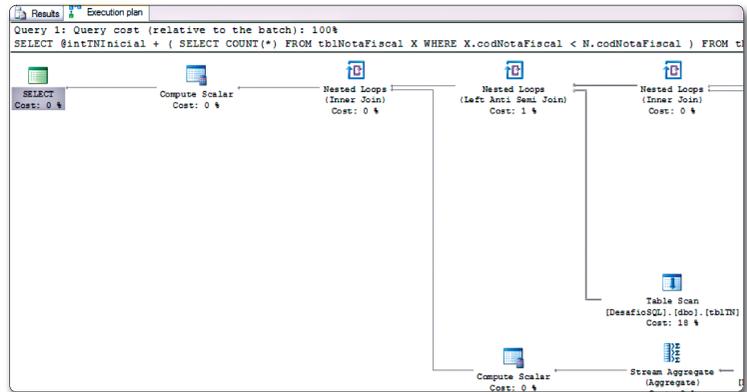


Figura 2. Apresentando o plano de execução (parcial) de uma consulta.

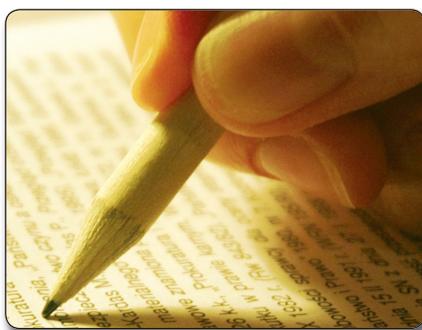
EXECUTION PLAN na barra de ferramentas. Quando rodarmos a consulta, o plano de execução será exibido. Veja a Figura 1.

A Figura 2 mostra graficamente o plano de execução do script da Listagem 2. O objeto exibido no canto esquerdo do diagrama representa a consolidação de todas as tarefas realizadas para execução do lote de declarações em questão. Os objetos à direita do diagrama são progressivamente mais detalhados, até chegarmos às tarefas mais básicas previstas no plano de execução.

O diagrama do plano de execução inclui várias informações. Basta passar o cursor do mouse sobre cada objeto apresentado para ter informações tais como número de registros afetados, grau de parelismo, custo total da sub-árvore, que é o custo de execução do lote que desejamos analisar. Obviamente, buscamos a consulta que apresenta menor custo de execução.

Mas teremos um problema para usar o diagrama do plano de execução no nosso estudo em particular. Como você pode observar na Figura 2, o diagrama é bastante grande e não coube na tela do computador. Seria difícil lhe mostrar separadamente as várias telas do diagrama.

Felizmente, existem outros meios de



apresentar os detalhes do plano de execução de uma consulta. Eu optei por mostrar estas informações de forma tabular.

Para ativar a exibição do plano de execução da consulta na forma de tabela (ou texto, como dissemos no início desta seção), devemos executar o seguinte comando na janela ativa:

```
SET STATISTICS PROFILE ON;
```

Após a ativação do STATISTICS PROFILE, quando rodarmos qualquer consulta nesta janela, será executado o comando e em seguida apresentado o plano de execução usado nesta operação.

Mas lembre-se: a ativação da exibição do plano de execução é válida para cada janela de consulta. Portanto, como eu abri duas janelas independentes, chamadas "Listagem1.sql" e "Listagem2.SQL", tive que repetir o processo de ativação do plano nas duas janelas (veja novamente a Figura 1).

Em seguida, devemos executar os scripts das duas janelas. Nós veremos o plano de execução detalhado de cada um deles.

Vamos ver agora como analisar o plano de execução das declarações apresentadas nas Listagens 1 e 2.

Em primeiro lugar, você vai observar que o plano de execução exibe uma tabela com 20 colunas! É verdade que ali tem muita informação que não vamos utilizar na maioria das análises. A Tabela 1 apresenta um resumo das informações do plano de execução da consulta exibida na Listagem 2, incluindo as informações mais relevantes para a nossa análise.

Veja que "NodeID" identifica cada ação do plano de execução e "Parent"

identifica qual é a ação que depende da tarefa executada. Analisando o custo total ("TotalSubtreeCost") para a ação final (NodeID 1), vemos o valor de 0,01842782. Isto é um indicador composto, por isso ele não tem uma unidade de medida (seja milissegundos ou número de I/Os necessários).

Mas esta métrica ("TotalSubtreeCost") é única. Você verá o mesmo valor para o nosso script, não importa em qual modelo da máquina em que você tente rodá-lo e nem qual versão do SQL SERVER.

Como você verá, o plano de execução do script "Listagem1.sql" é bem mais complicado. O que acontece ali é que este script contém diversas declarações SQL. E o que nós vemos é o plano de execução e o custo de cada uma destas declarações. Não há em lugar algum o cálculo do custo estimado total para todo o script.

Então vamos pôr a mão na massa e resolver esta questão. Observe na Listagem 1 que executamos um laço que repete 10 vezes uma operação de INSERT e depois temos um SELECT. O custo de definir o valor da variável e controlar a execução do laço é desprezível e não é exibido no plano de execução.

Ainda assim, a listagem resultante para o plano de execução deste script tem cerca de 50 linhas! Por isso não vamos apresentá-la aqui.

Na Tabela 2, vemos um resumo com as informações mais importantes deste plano. Temos apenas o custo total ("TotalSubtreeCost") para execução de cada declaração deste script.

Observe agora a linha de TOTAL das Tabelas 1 e 2. Veja que o custo total do script "Listagem1.sql" é 0,10661622, ao

NodeId	Parent	StmtText	Rows	PhysicalOp	EstimateIO	EstimateCPU	TotalSubtreeCost
1	0	SELECT @intTNInicial + ( SELECT C...	2	NULL	NULL	NULL	0,01842782
2	1	--Compute Scalar(DEFINE:([Expr1018]=	0	Compute Scalar	0	0,00000010	0,01842782
3	2	--Nested Loops(Inner Join, OUTER F	2	Nested Loops	0	0,00000418	0,01842772
4	3	--Nested Loops(Left Anti Semi Joi	2	Nested Loops	0	0,00008360	0,01513594
5	4	--Nested Loops(Inner Join, OU	10	Nested Loops	0	0,00001254	0,01171594
6	5	--Filter(WHERE:([Expr1004	10	Filter	0	0,00000480	0,00809060
7	6	--Nested Loops(Inner Jo	10	Nested Loops	0	0,00004180	0,00808580
8	7	--Clustered Index Sca	10	Clustered Index Scan	0,00312500	0,00016800	0,00329300
9	7	--Compute Scalar(DE	0	Compute Scalar	0	0,00000230	0,00475100
10	9	--Stream Aggregat	10	Stream Aggregate	0	0,00000230	0,00475100
11	10	--Clustered Ind	45	Clustered Index Seek	0,00312500	0,00016030	0,00472800
17	5	--Compute Scalar(DEFINE:([	0	Compute Scalar	0	0,00000230	0,00361280
18	17	--Stream Aggregate(DEF	10	Stream Aggregate	0	0,00000230	0,00361280
19	18	--Clustered Index See	45	Clustered Index Seek	0,00312500	0,00016030	0,00360590
24	4	--Table Scan(OBJECT:([Desaf	52	Table Scan	0,00320350	0,00008730	0,00330840
26	3	--Compute Scalar(DEFINE:([Expr	0	Compute Scalar	0	0,00000230	0,00328760
27	26	--Stream Aggregate(DEFINE:(	2	Stream Aggregate	0	0,00000230	0,00328760
28	27	--Clustered Index Seek(OB	10	Clustered Index Seek	0,00312500	0,00016030	0,00328530
<b>TOTAL</b>							<b>0,01842782</b>

Tabela 1. Informações do plano de execução do script "Listagem2.sql".

Declaração	#Declaração	NodeId	Parent	StmtText	TotalSubtreeCost
INSERT	1	1	1	INSERT INTO @tbVIRTUA	0,01000216
INSERT	2	1	1	INSERT INTO @tbVIRTUA	0,01000216
INSERT	3	1	1	INSERT INTO @tbVIRTUA	0,01000216
INSERT	4	1	1	INSERT INTO @tbVIRTUA	0,01000216
INSERT	5	1	1	INSERT INTO @tbVIRTUA	0,01000216
INSERT	6	1	1	INSERT INTO @tbVIRTUA	0,01000216
INSERT	7	1	1	INSERT INTO @tbVIRTUA	0,01000216
INSERT	8	1	1	INSERT INTO @tbVIRTUA	0,01000216
INSERT	9	1	1	INSERT INTO @tbVIRTUA	0,01000216
INSERT	10	1	1	INSERT INTO @tbVIRTUA	0,01000216
SELECT	11	1	0	SELECT codTN AS Ausente	0,00659462
<b>TOTAL</b>					<b>0,10661622</b>

Tabela 2. Cálculo do custo total de execução do script "Listagem1.sql".

passo que para o script "Listagem2.sql", este custo cai para 0,01842782.

Portanto, em bom português: o script "Listagem2.sql" é quase 6 vezes mais rápido do que o script "Listagem1.sql"!!! (a conta exata é 0,10661622 / 0,01842782 = 5,78561219).

Mais um detalhe importante. Existe uma segunda vantagem do script exibido na Listagem 2. O custo da consulta independente do número de registros pesquisados, porque ele depende primeiramente do tamanho da tabela auxiliar usada. Naquele script, usamos a tabela tblNotaFiscal.

No caso da Listagem 1, o custo do script aumentará linearmente conforme o número de registros pesquisados, pois haverá

um novo INSERT para cada registro!

Assim fechamos este estudo! Eu considero este tema importante para qualquer desenvolvedor. Portanto, fica aqui a recomendação: vale a pena se aprofundar no assunto!

Mês que vem tem mais. Até lá! ●

**Dê seu feedback sobre esta edição!**

A SQL Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

[www.devmedia.com.br/sqlmagazine/feedback](http://www.devmedia.com.br/sqlmagazine/feedback)



**Links**

FRAITEUR, Gael. SQL Tuning Tutorial - Understanding a Database Execution Plan. THE CODE PROJECT. Mar, 2005. (<http://www.codeproject.com/KB/database/sql-tuning-tutorial-1.aspx>)

FRITCHEY, Grant. Dissecting SQL Server Execution Plans. May, 2008. (<http://www.simple-talk.com/sql/performance/execution-plan-basics>)

McGEHEE Brad. SQL Server Query Execution Plan Analysis ([http://www.sql-server-performance.com/tips/query\\_execution\\_plan\\_analysis\\_p1.aspx](http://www.sql-server-performance.com/tips/query_execution_plan_analysis_p1.aspx))

