Otimização de Consultas no SQL Server

Sumário

Prefácio

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Commented [F1]: Convidar alguém para fazer o Prefácio do Livro

Sobre o Autor

Atualmente o Fabiano Neves Amorim trabalha como arquiteto na plataforma de dados na SolidQ, uma das maiores empresas em consultoria e treinamento na plataforma de dados Microsoft. Fabiano é autor de vários artigos e de um e-book na área de otimização de consultas, além de possuir várias certificações em SQL Server.

Membro ativo na comunidade SQL Server, já ministrou palestras e cursos em várias cidades do Brasil e vários eventos relacionados a Banco de Dados.

Quando não está otimizando alguma consulta, ou procurando erros no otimizador de consultas do SQL Server, Fabiano gosta de ir ao cinema com sua maravilhosa esposa, brincar com seu filho e andar com seus cachorros da raça beagle.

Agradecimentos

Commented [F2]: Escrever depois determinar o livro

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Introdução

Sobre o Livro

Commented [FA3]: Explicar sobre o que se trata o livro, e descrever todos os capítulos do livro com um breve resumo de no máximo 6 linhas por capítulo.

Falar sobre os exemplos, onde baixar o código, onde baixar o banco de dados de demo...

Este livro tem o objetivo de cobrir vários aspectos relacionados à Otimização de Consultas no banco de dados Microsoft SQL Server.

Aficionado por tecnologia, um estudante procurando aprimorar seus conhecimentos em banco de dados, um administrador de banco de dados tentando resolver um problema de performance, um desenvolvedor desejando escrever consultas mais eficientes. Não importa porque você está lendo este livro, todos os perfis que mencionei tem um ponto em comum. O desejo em fazer o ***melhor trabalho possível*** no ***menor tempo possível*** levando em conta os ***custos*** relacionados a tempo de estudo e complexidade de resolução de um determinado problema.

Pensando bem, caro leitor, você não é muito diferente do principal assunto deste livro.

Capítulos

Segue abaixo a lista de capítulos desde livro com um resumo dos tópicos que serão cobertos neste livro:

Capítulo 1 - Visão Geral

Neste capítulo veremos de forma amigável e utilizando exemplos de nosso cotidiano como a otimização de um processo é primordial para tomada de decisões. E como a arte de otimização de consultas em banco de dados funciona.

Capítulo 2 - Entendendo Planos de Execução

Entender como visualizar os planos de execução no SQL Server e como foram criados. Estes são pontos chave neste capítulo.

Entenderemos passos básicos da otimização de uma consulta, e, como o SQL Server trabalha para otimizar um código. Também é feita uma breve descrição sobre o uso de planos em cache, técnica utilizada pelo SQL Server para armazenar planos em memória a fim de evitar retrabalho. Por fim é apresentado ao leitor quais são as maneiras e algumas dicas para visualização dos planos de execução.

Capítulo 3 – Estatísticas

Estatísticas ou teorias probabilísticas são utilizadas pelo SQL Server como variáveis importantes para criação de um plano de execução. Neste capítulo veremos o que são e como as estatísticas são utilizadas pelo SQL Server.

Vários problemas e soluções relacionadas a estatísticas serão apresentadas neste capítulo, bem como técnicas avançadas de como o SQL Server utiliza as estatísticas.

Capítulo 4 – Pontos Chaves na Otimização

Itens importantes do tema otimização de consultas e planos de execução serão apresentados neste capítulo. São eles: Seletividade, Densidade, Cardinalidade, Search Arguments (SARG).

Estes pontos são de extrema importância para o otimizador de consultas e neste capítulo entenderemos como eles são utilizados pelo otimizador.

Capítulo 5 – Algoritmos de Join

Um join, ou junção em português é o ato de juntar os dados de tabela com outra tabela. Para executar este processo, o SQL Server pode optar por usar várias técnicas e diferentes algoritmos otimizados para efetuar um join.

Entender como os algoritmos funcionam, e analisar se o otimizador de consultas escolheu a melhor opção para juntar as tabelas, é uma importante ferramenta para otimização de uma consulta.

Capítulo 6 – Operadores

Este é o maior capítulo do livro, e são demonstrados vários exemplos de operadores utilizados pelos planos de execução de consultas.

Após entender o que é um operador, o que ele faz, e como ele é processado, você poderá entender se o otimizador de consultas do SQL Server escolheu o melhor operador para processar ou acessar uma determinada informação.

A lista de operadores que são cobertas por este capítulo é a seguinte: FullScan, Clustered Index Scan, Non-Clustered Index Scan, Index Seek, Bookmark Lookup, RID Lookup, Table Spool (Eager e Lazy), Index Spool, RowCount Spool, Stream Aggregate, Assert, Concatenation, Compute Scalar, Sort, Merge Interval e Split/Sort/Collapse.

Capítulo 7 – Influenciando o Otimizador de Consultas

Técnicas para influenciar e entender melhor como um plano de execução funciona são apresentadas neste capítulo.

Dicas e boas práticas para desenvolvimento de software são apresentadas neste capítulo.

Material

Para replicar e testar os códigos e exemplos utilizados neste livro é recomendado que você utilize o Microsoft SQL Server 2008 R2. Uma versão de avaliação de 180 dias pode ser baixada no seguinte endereço: http://www.microsoft.com/sqlserver/2008/en/us/trial-software.aspx.

Todos os scripts, e o banco de dados utilizados para teste neste livro podem ser baixados na seguinte página de internet: www.ALGUMACOISA.com.br/Livro/.

No website você pode fazer o download de todos os scripts e do banco de dados “LivroQO” utilizado extensamente neste livro.

As instruções para criação do banco de dados estarão no arquivo leia-me junto aos scripts de criação do banco.

Futuras erradas do livro serão publicadas no mesmo website.

Capítulo 1 - Visão Geral

Commented [F4]: Estimativa de 2 páginas

Arte da otimização de consultas em SGBDs

Desde a criação do primeiro banco de dados relacional em meados de 1970, um dos maiores desafios relacionados a banco de dados permeia no tempo, a pergunta que eleva este desafio é a seguinte:

- Como identificar qual é a melhor forma de acesso aos dados de uma tabela na maneira mais rápida possível?

Para ilustrar melhor esta questão, vou mudar o cenário de banco de dados para nosso cotidiano ato de “pegar o metro”.

Certa vez eu estava em Londres fazendo um treinamento de T-SQL com um colega de trabalho, e todos os dias brincávamos dizendo: - Hoje vamos criar um novo ***“plano de execução”*** para escolher qual será a rota que iremos fazer do apartamento para o centro de treinamento.

De primeira vista isso parece uma tarefa ***trivial***, mas antes de ter certeza disso, vejamos o mapa de metro de Londres:

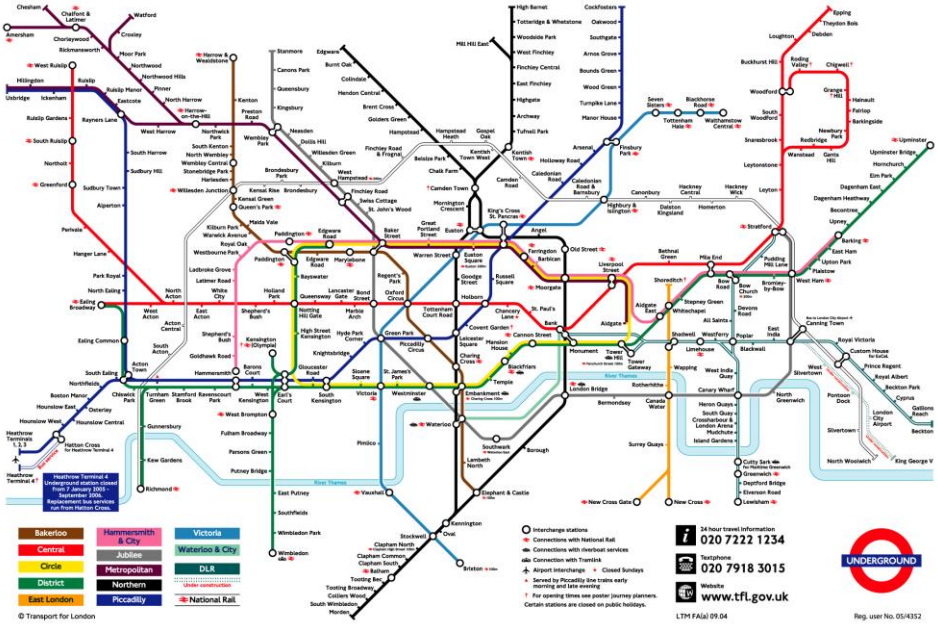


Figura1: Mapa Metro Londres

Nosso percurso requeria pegar o metro na estação mais próxima de nosso apartamento e ir para a estação Morgate. Primeiro problema, havia duas opções como saída, as estações Waterloo ou WestMinster ambas na linha cinza.

Você ainda não viu qual a dificuldade nisso? A dificuldade inicia quando você leva em conta todas as variáveis que podem influenciar em nossa decisão. Vamos analisar algumas:

1. A estação Waterloo é bem maior que a estação Westminster.
    a. A ***concorrência*** para comprar tickets ou ficar na fila para entrar nos vagões era maior.
    b. A quantidade de entradas e saídas era maior, o que aumentava a ***probabilidade*** de pegarmos um caminho errado.
2. Na estação WestMinster havia menos concorrência, porém era um pouco mais longe que a estação Waterloo.
    a. Eu gostava mais porque a estação é ao lado do BigBen e a vista era mais bonita.
    b. Meu colega não gostava porque ele precisava passar na Cafeteria para comprar um "café latte", e no caminho para WestMinster não havia nenhuma.
3. Ao chegar á estação, qual a linha anda mais rápido?
    a. E se a estação estiver fechada? (Por 2 dias a estação WestMinster ficou fechada por causa de alguns protestos de estudantes)
4. Vamos da estação "A" para o "B" depois para a "C" e depois para a "D", ou vamos da estação "A" para o "C" direto?

Eu poderia continuar e citar dezenas de variantes, fatos ou desejos que poderiam influenciar em nossa decisão em chegar ao destino.

Você pode estar se perguntando, o que quero contar com essa história?

O fato é que, para decidir qual é o melhor caminho de acesso aos dados de sua tabela, o SQL Server precisa planejar bem como ele fará para acessar os dados.

A quantidade de variantes que podem influenciar nesta decisão é enorme. O otimizador de consultas não pode validar todas as variações de planos possíveis, pois isso levaria muito tempo. Lembre-se, o ponto chave aqui é criar um plano que irá retornar os dados da maneira mais rápida possível, levando o menor tempo possível para criar este plano. O segredo é ***nunca gastar mais tempo analisando as variantes, do que o tempo que você irá gastar para executar o plano***.

Capítulo 2 - Entendendo Planos de Execução

Commented [F5]: Estimativa de 30 páginas

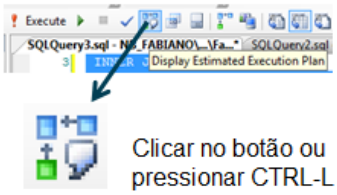
Visualizando planos de execução

Os planos de execução são um "mapa" de como o SQL Server fará para acessar os dados de suas tabelas. Este plano é criado pelo otimizador de consultas. Veremos mais detalhes sobre o otimizador de consultas no fim deste capítulo. Por hora vejamos como visualizar este mapa de acesso aos dados no SQL Server.

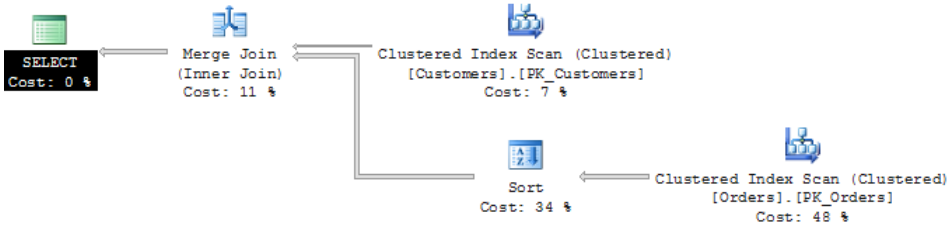
Para começar a entender como ler um plano vejamos um exemplo. Inicie o SQL Server Managment Studio, abra uma nova consulta digite o seguinte comando T-SQL:

```
USE NorthWind
GO
SELECT *
  FROM Orders
 INNER JOIN Customers
    ON Orders.CustomerID = Customers.CustomerID
```

Após digitar a consulta, clique no botão *display estimated execution plans* ou pressione CTRL+L:

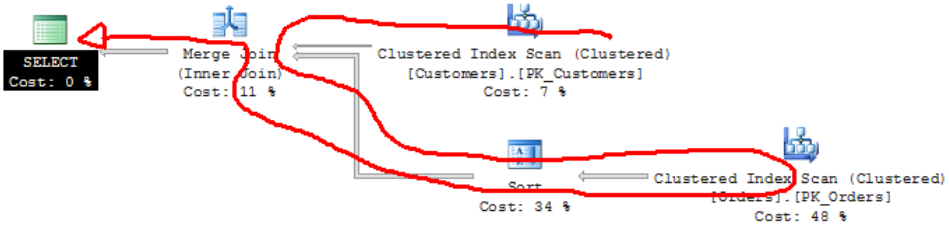


Após clicar no botão, um plano em modo gráfico será exibido na parte inferior do SSMS.



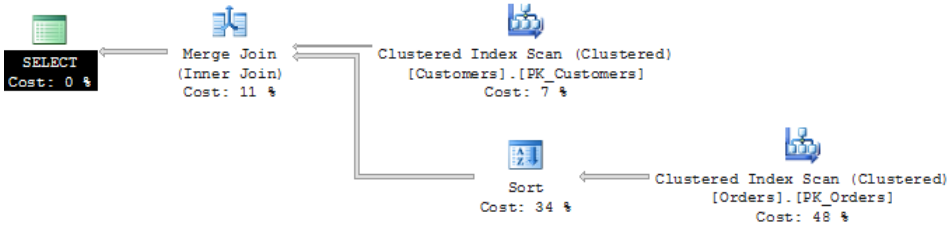
O plano acima contem 5 operadores, sendo que o operador de SELECT não é um operador físico e apenas representa a operação que esta sendo executada.

Os planos são lidos da direita para a esquerda e de cima para baixo, por exemplo, o primeiro passo a ser executado no plano acima é o Clustered Index Scan (em Customers), depois o Merge Join irá chamar o Sort que irá chamar o Clustered Index Scan (em Orders). Abaixo podemos visualizar uma representação gráfica de como o plano é lido.



Os operadores (ícones no plano) são responsáveis por executar as tarefas necessárias para ler e processar uma consulta. Veremos mais sobre os operadores em um capítulo reservado para isso.

Ainda que o modo que lemos o plano seja o exibido acima, quando executado ele roda da esquerda para a direita, mais ou menos assim:



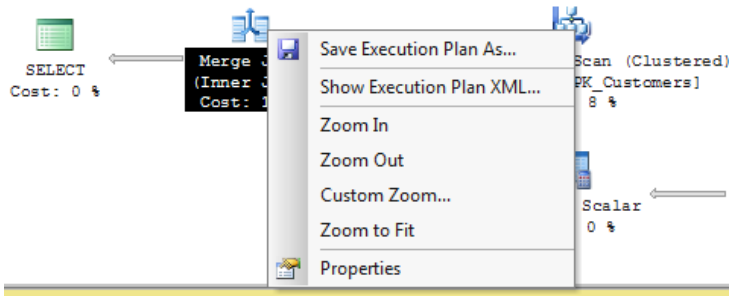
Existem ainda mais 2 tipos de visualização de um plano de execução o modo texto e o XML. Este livro irá sempre focar em planos em modo gráfico, que é o plano que podemos visualizar na figura acima. Você pode consultar o help do SQL Server para mais detalhes em relação aos planos em modo texto e XML.

Dicas e propriedades dos operadores

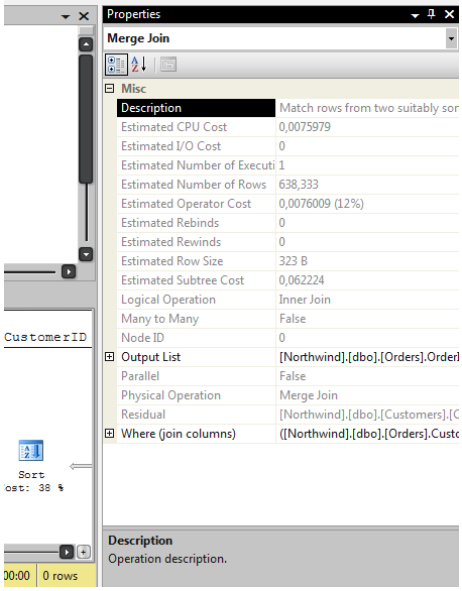
Cada operador dos planos de execução contêm diversas informações adicionais sobre o processo que esta sendo executado. Estas informações são exibidas quando paramos o mouse em cima de um operador, ou quando clicamos com o botão do lado direito do mouse no operador e selecionamos a opção *properties.*

Propriedades

Abaixo podemos visualizar as opções de um plano ao clicar com o botão do lado direito do mouse em algum operador.



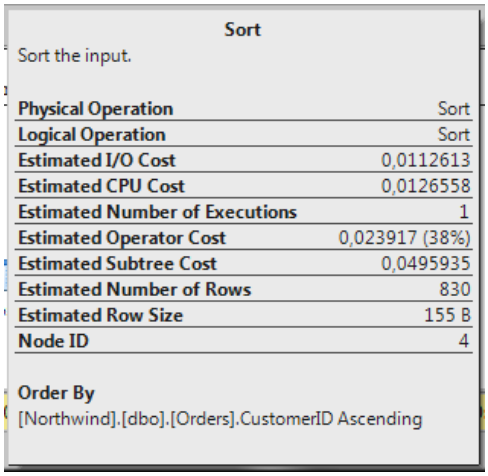
Após clicar no botão *properties* uma janela com mais informações em relação ao operador é exibida.



Como podemos observar na imagem acima, várias informações relacionadas ao operador são exibidas. As propriedades exibidas podem variar dependendo do operador selecionado.

Hints

Outra alternativa para visualização de algumas propriedades dos operadores, consiste em parar o cursor do mouse em cima de um operador. Abaixo podemos visualizar dicas relacionadas ao operador de sort.



Como podemos observar nas dicas, o operador de sort está ordenando os dados por CustomerID de forma ascendente.

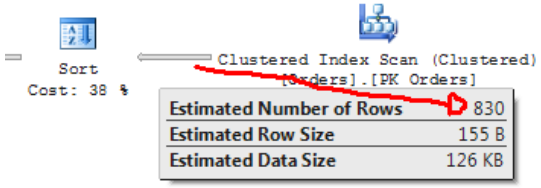
Segue abaixo uma tabela com a descrição de algumas propriedades exibidas no plano de execução.

Propriedade	Descrição
Estimated CPU Cost	Custo de uso de CPU pelo operador. Este número deve ser o menor possível.
Estimated I/O Cost	Custo de toda atividade de I/O realizada pelo operador. Este número deve ser o menor possível.
Estimated Operator Cost	Custo para o otimizador de consultas executar esta operação. Mostra entre parênteses o percentual total de custo do operador em relação a todo o plano.
Estimated Number of Executions	Estimativa de número de vezes que o operador será executado no plano.
Estimated Number of Rows	Estimativa do número de linhas que o operador irá retornar.
Estimated Row Size	Média estimada do tamanho de cada linha (em bytes) lida pelo operador.
Estimated SubTree Cost	Soma do custo de todos os operadores executados antes deste operador.

Nota: O valor numérico exibido nos custos representava o tempo estimado para execução do operador em uma máquina antiga de um funcionário da Microsoft responsável por calcular e documentar o custo dos operadores. Atualmente em nossas máquinas muito melhores este valor não representa nada. Devemos apenas nos atentar de que quanto menor o valor menor o custo de execução.

Setas

Outro ponto extremamente importante na leitura dos planos é a espessura das setas. Todos os operadores são ligados de um ao outro por setas, estas setas contêm informações sobre a quantidade de dados que serão processados de um operador ao outro.

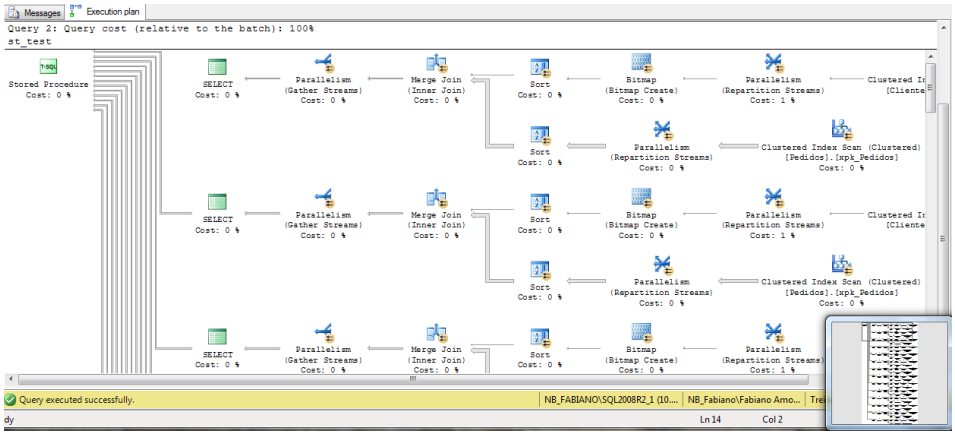


Na imagem acima podemos observar que o otimizador estimou que 830 linhas seriam retornadas após a execução do operador Clustered Index Scan.

Dicas para leitura de um plano muito grande

Um ponto importante e que nos ajuda bastante na leitura dos planos é o botão "+" do SSMS. Ele fica localizado na parte inferior a direita de um plano de execução, e com ele conseguimos navegar pelo plano em uma mini janela.

Alguns planos são muito grandes e complexos de serem analisados e este botão pode nos ajudar na leitura do plano.



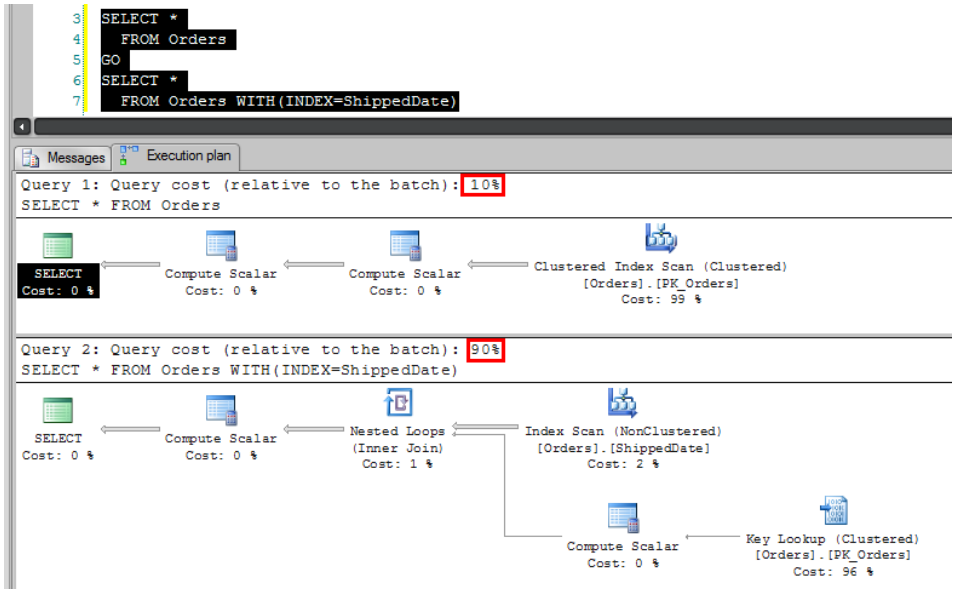
Acima podemos observar um plano complexo e uma mini janela com todo o plano aberta na parte inferior direita da imagem.

Outra alternativa para leitura de um plano muito grande, é ligar a opção "*Include actual execution plan*" *(CTRL-M)* no SSMS. Isso fará com que o plano de execução atual seja exibido.

Vamos supor uma stored procedure que executa vários comandos. Ao executar a procedure com o CTRL-M ativado, no SSMS será exibido um plano de execução para cada comando executado. Desta forma também conseguimos fazer uma comparação e identificar qual é o comando com o maior custo na procedure usando a comparação dos planos.

Comparando planos

Comparar um plano com outro é uma maneira muito fácil de identificar qual é o melhor plano de execução. O SQL Server mostra um percentual com o custo de um plano comparado a outro. Por exemplo:



Para criar a comparação dos planos acima, selecionei as duas consultas, e pressionei CTRL-L. Na imagem conseguimos facilmente visualizar que a primeira consulta tem um custo de 10% em relação ao batch, sendo que o batch, são as 2 consultas selecionadas.

Esta prática é bem simples e pode nos ajudar a responder se um plano é melhor que o outro. Lembre-se de que este é um plano estimado, ou seja, esta estimativa não é 100% confiável.

Otimizador de consultas

O otimizador de consultas do SQL Server é o responsável por decidir como os dados de uma consulta serão acessados e retornados. A grosso modo o podemos dizer que o otimizador de consultas traduz um código T-SQL para um plano de execução que é executado pelo Query Processor.

Um plano de execução contém os detalhes de todas as etapas necessárias para localizar e processar os dados solicitados pelo comando T-SQL da maneira mais eficiente possível, ou ao menos, a melhor encontrada em tempo hábil.

Utilizando a linguagem ANSI/SQL dizemos ao SQL Sever o que queremos, e não como fazer. O otimizador de consultas é responsável por decidir como fazer.

Utilizando este princípio como base, vamos supor o seguinte código T-SQL:

```
SELECT a.Nome, e.Endereco
  FROM Alunos_Classe AS a
 INNER JOIN Endereco AS e
    ON a.ID_End = e.ID_End
 WHERE a.Ramo_Atividade = 'TI'
   AND a.Sexo = 'F'
```

Com o código acima estamos solicitando ao SQL Server, que ele retorne endereço de todos os alunos que trabalham com informática e são do sexo feminino.

Poderíamos executar o código da Consulta 1 utilizando o seguinte pseudo-código:

```
FOR EACH(Alunos_Classe)
{
  IF (Alunos_Classe.Ramo_Atividade = 'TI')
    IF (Alunos_Classe.Sexo = 'F')
      FOR EACH (Endereco)
        IF (Endereco.ID_End = Alunos_Classe.ID_End)
        {
          PRINT ('Aluno: ' + Alunos_Classe.Nome);
          PRINT ('Endereço: ' + Endereco. Endereco);
        }
}
```

No código acima, varremos a tabela "Aluno_Classe" e para cada linha lida, validamos se o aluno lido é igual ao ramo de atividade "TI", depois validamos se o sexo do aluno é "F". Caso ambas as validações sejam verdadeiras, iniciamos uma varredura na tabela de "Enderecos" procurando o endereço correspondente ao aluno lido.

Agora vejamos outro pseudo-código:

```
FOR EACH(Alunos_Classe)
{
  IF (Alunos_Classe.Sexo = 'F')
    IF (Alunos_Classe.Ramo_Atividade = 'TI')
      FOR EACH (Endereco)
        IF (Endereco.ID_End = Alunos_Classe.ID_End)
        {
          PRINT ('Aluno: ' + Alunos_Classe.Nome);
          PRINT ('Endereço: ' + Endereco. Endereco);
        }
}
```

No pseudo-código 2 temos uma alteração importante no código. Digamos que de alguma forma o otimizador de consultas consiga identificar que apenas 5% dos alunos são do sexo feminino. Ao invés de validar se o Ramo de atividade é igual a "TI", poderíamos primeiro validar se o sexo do aluno é igual a "F". Com isso aumentamos a eficiência da execução do código já que eu não teria que perguntar se o ramo atividade do aluno é TI para 95% dos alunos lidos.

Vejamos mais um pseudo-código:

```
FOR EACH(Endereco)
{
  FOR EACH (Alunos_Classe)
    IF (Endereco.ID_End = Alunos_Classe.ID_End)
      IF (Alunos_Classe.Ramo_Atividade = 'TI')
        IF (Alunos_Classe.Sexo = 'F')
        {
          PRINT ('Aluno: ' + Alunos_Classe.Nome);
          PRINT ('Endereço: ' + Endereco. Endereco);
        }
}
```

Novamente, digamos que o SQL Server saiba que apenas dois alunos de um total de mil tenha o endereço cadastrado. Neste caso não seria mais eficiente, ao invés de varrer o aluno, varrer a tabela de endereços? E para cada endereço, localizar o aluno correspondente o endereço lido.

Poderiamos escrever vários outros pseudo-códigos para retornar os dados de nossa consulta. Na verdade existem ainda muitas variantes que podem e influenciam na decisão do otimizador de consultas, vejamos alguma delas:

- Avaliar possíveis Índices nas tabelas Alunos_Classe e Enderecos
- Ordenar as tabelas para realizar o join utilizando o algoritmo de Merge Join
- Aproveitar ordem dos índices para fazer o Merge Join
- Avaliar memória disponível
- Avaliar pressão de CPU
- Considerar processamento do join utilizando o algoritmo de Hash Join
- Considerar paralelismo
- Criar estatísticas
- Atualizar estatísticas
- Reutilizar plano do cache
- Considerar view Indexada
- Remover código redundante
- Detectar contradições
- Converter expressões
- Evitar acesso a tabelas

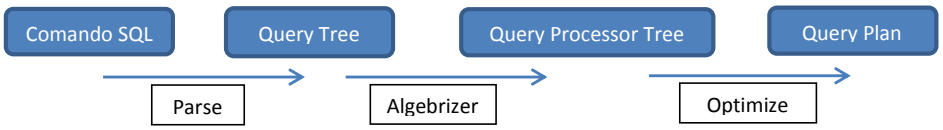
- Identificar correlação
- ...

Todos os itens listados acima podem ser considerados na otimização de uma consulta, existem ainda muitos outros pontos que são levados em conta. Vamos entender mais detalhes de como isso funciona e quais passos são executados no item etapas da otimização.

Etapas da otimização

O otimizador de consultas é executado em várias etapas e cada etapa é responsável por executar alguns passos necessários para a geração do plano.

Abaixo podemos visualizar uma simples ilustração com as etapas utilizadas pelo otimizador de consultas do SQL Server:

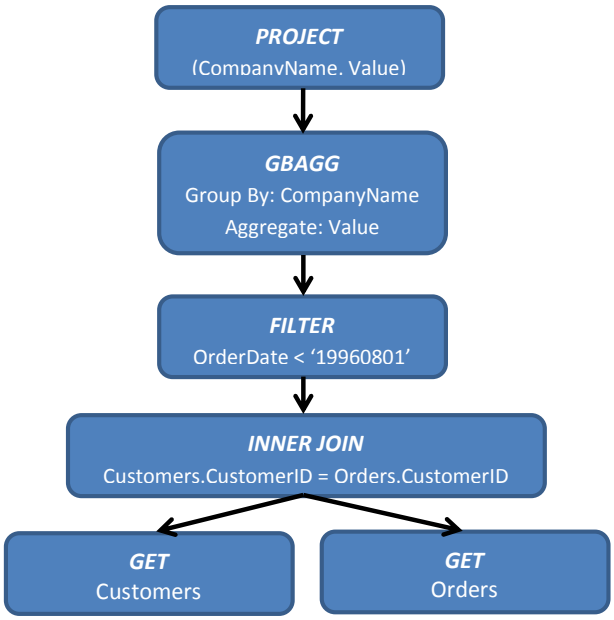


O primeiro passo na criação de um plano de execução é a validação da sintaxe do código SQL. Esta validação é efetuada pelo processo conhecido como Parse, nesta faze o SQL valida se o código digitado é válido.

O resultado do Parse é um Query Tree que nada mais é do que uma representação gráfica do comando em formato de uma árvore. Cada nó da arvore é organizado em operações que serão efetuadas. Por exemplo, um filtro aplicado na clausula WHERE contém seu próprio operador. Vamos considerar o seguinte comando:

```
USE Northwind
GO
SELECT Customers.CompanyName, SUM(Orders.Value) AS Value
  FROM Orders
 INNER JOIN Customers
    ON Orders.CustomerID = Customers.CustomerID
 WHERE Orders.OrderDate < '19960801'
 GROUP BY Customers.CompanyName
 ORDER BY Customers.CompanyName
```

Uma representação em forma de query tree do comando acima seria o seguinte:



A árvore acima representa as operações lógicas que o SQL Server irá executar, internamente o SQL utiliza outro formato de árvore com alterações das operações lógicas para operações físicas. Por exemplo, na árvore acima o processo lógico de INNER JOIN pode ser executado pelos operadores físicos de Merge Join, Nested Loop Join ou Hash Join.

O segundo passo é executar um processo conhecido como **algebrizer** onde as tabelas e campos na query tree são comparados com o metadata do banco de dados para validar se os objetos acessados realmente existem. Nesta fase um "*" será expandido pelo nome das colunas, os datatypes das colunas são carregados, views são expandidas, um sinônimo é interpretado. Após esta analise um binário chamado query processor tree é gerado e enviado para a fase de otimização.

Otimização

O otimizador de consultas do SQL Server trabalha por custo. Isso significa que a fase de otimização irá tentar várias alternativas de resolução da sua consulta e atribuir um custo a cada tentativa. A tentativa com menor custo tende a ser a melhor opção. Nem sempre o otimizador de consultas consegue gerar o melhor plano possível, pois a quantidade de opções a serem analisadas são muito grandes, e o processo de otimização é extremamente trabalhoso e faz alto consumo de CPU.

Atualmente é um dilema para o otimizador conseguir gerar um plano, que irá consumir menos tempo e recursos que a execução da consulta. Isso é um dilema, pois o SQL só pode estimar o tempo necessário para execução da consulta, e talvez esta estimativa não seja correta.

Os custos de CPU e IO são considerados em cada plano gerado. Uma leitura sequencial tem um custo mais baixo, uma leitura aleatória tem um custo mais alto, um ciclo de CPU tem um custo. Dependendo da forma de acesso aos dados, um recurso será mais utilizado que o outro.

Atualmente com a nova geração de discos SSDs (Solid State Discs) os custos considerados pelo SQL Server para leitura aleatória poderia ser revisto, mas isso é algo novo que ainda não influência o comportamento do otimizador de consultas. É possível influenciar no custo destas operações de leitura, mas isso requer vários testes e não é algo simples de ser controlado.

Good enough plan

Em alguns cenários é impossível para o otimizador validar todas as opções de planos disponíveis. Na verdade o otimizador tenta trabalhar até achar o que é conhecido como "*good enough plan*" ou seja, um plano que é bom o suficiente para execução da consulta.

Considere a seguinte consulta:

```
SELECT *
  FROM Tabela1
 INNER JOIN Tabela2
    ON Tabela1.Tab1ID = Tabela2.Tab2ID
 INNER JOIN Tabela3
    ON Tabela1.Tab1ID = Tabela3.Tab2ID
 INNER JOIN Tabela4
    ON Tabela1.Tab1ID = Tabela4.Tab2ID
 INNER JOIN Tabela5
    ON Tabela1.Tab1ID = Tabela5.Tab2ID
 INNER JOIN Tabela6
    ON Tabela1.Tab1ID = Tabela6.Tab2ID
```

A quantidade de possiblidades de resolução desta consulta é de N! [N x (N-1) x (N-2) x . . .] pois o otimizador pode considerar todas as ordens de acesso as tabelas (Tabela1, Tabela2, Tabela3... ou Tabela1, Tabela3, Tabela2...) e diferentes topologias para os joins (Tabela1 join Tabela2, Tabela1 join Tabela3, Tabela2 join Tabela3...). O espaço para armazenagem, tempo e recursos gastos nestas validações seria imenso.

Explorando planos de execução

O otimizador do SQL Server é um ***search framework***. Este framework utiliza uma série de componentes que trabalham na fase de otimização.

Rules

Para explorar as alterativas de na criação de um plano de execução este search framework considera várias transformações no query processor tree. Estas transformações são efetuadas por ***rules***.

O conceito por trás das rules é similar aos teoremas matemáticos que dizem que se “a=b and b=1” então “b=1 and a=1” ambas as expressões são equivalentes e retornaram o mesmo resultado.

Vamos considerar a seguinte consulta:

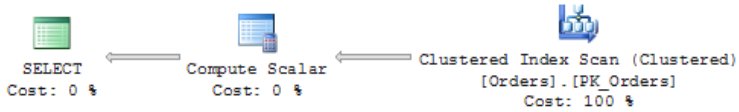
```
USE Northwind
GO
SELECT OrderID, MAX(Value) AS Max_Value
  FROM Orders
 GROUP BY OrderID
```

Existe um índice único por OrderID na tabela Orders que é a chave primária da tabela. Portanto podemos dizer que só existe um valor MAX para a coluna Value dado um OrderID.

Existe uma rule no SQL Server que é utilizada para efetuar esta validação. Ela é uma *role de exploração* e o nome dela é ***“GbAggToPrj”***, ela faz com que o plano abaixo seja otimizado para não gerar uma agregação.

Plano sem a rule:

Plano com a rule, não é necessário fazer a agregação:



Outro tipo de rule que existem são as *simplification rules*, como por exemplo a ***“TopOnEmpty”*** que remove o acesso ao objeto caso um TOP(0) seja especificado, por exemplo:

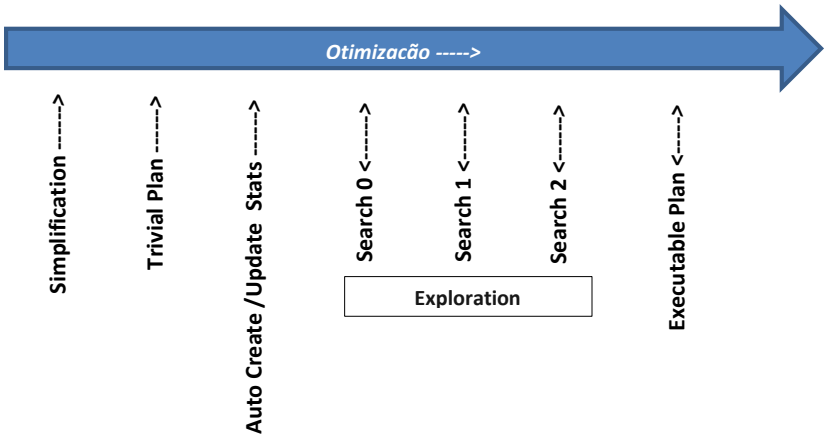
Plano sem a rule:

Plano com a rule onde não é necessário acessar a tabela nem obter locks para a leitura:

Outro caso similar é a rule ***"SelectOnEmpty"*** que consegue identificar que nenhuma linha será retornada, devido a uma contradição de WHERE 1=0 , ou devido a presença de uma check constraint na coluna filtrada.

Níveis de otimização

O fase de otimização pode chegar a vários níveis de exploração, cada fase executa diferentes métodos de otimização, e cada nível de exploração aplica determinadas rules. A sequência do processo de otimização e os níveis de exploração mais comuns podem ser visualizados abaixo.



A fase de ***simplification*** tenta alterar sua consulta para deixar a query tree mais simples para a otimização. Durante essa fase o SQL Server pode identificar contradições na sua consulta e remove-la reescrevendo a consulta de uma forma simplificada. Um simples exemplo de contradição é o seguinte:

```
USE NorthWind
GO
SELECT *
  FROM Orders
 WHERE CustomerID BETWEEN 10 AND 5
```

Plano de execução da consulta acima:

Como podemos observar a consulta contêm uma contradição, pois esta procurando dados "*between 10 e 5*" e isso é impossível, é o mesmo que pedir que CustomerID seja maior ou igual a 10 e menor ou igual a 5.

Na fase de simplificação o SQL Server gera um plano que não precisa acessar a tabela Orders.

Um ***Trivial Plan*** é gerado quando o SQL Server consegue identificar que não é necessário explorar várias fases de otimização tentando localizar um plano de execução bom o suficiente para uma consulta.

Um exemplo de um trivial plan pode ser visualizado considerando a seguinte consulta:

```
DECLARE @t1 TABLE (id Integer)

SELECT * FROM @t1
```

O SQL Server sabe que só a uma forma de acesso a tabela @t1, pois não a nenhum índice na tabela, portanto ele nem perde tempo tentando otimizar o que não é otimizável.

Após o passo de identificação de um trivial plan o SQL Server inicia a fase de exploração onde os níveis de exploração Search 0, Search 1 e Search 2 são executados.

Por fim o processo de otimização gera um código que será interpretado pelo query processor que irá executar a consulta.

Esta é uma visão superficial de como o otimizador de consultas do SQL Server trabalha, ainda a muito a discutir e abordar. Irei abordar mais detalhes sobre o processo de otimização de consultas do SQL Server em uma futura edição deste livro.

Overview sobre cache de planos

Conforme podemos perceber analisando o que apresentei acima, o processo de criação de um plano de execução é extremamente pesado e consumidor de recursos do servidor, principalmente CPU. Por conta disso, eu gostaria de comentar um pouco sobre um mecanismo que o SQL Server utiliza para tentar minimizar este impacto no uso dos recursos utilizados para compilação de um plano de execução.

Voltando a analogia que usei para pegar o metro em Londres. Depois que aprendemos o percurso para chegar no destino desejado, nós simplesmente guardamos esta informação em nossa memória, e quando precisamos refazer o percurso, simplesmente já sabemos qual percurso fazer, não precisamos parar para pensar de novo, nem olhar em mapa, nada, já que temos uma memória boa (não eu!) conseguimos lembrar como fazer para chegar na estação final.

Não seria interessante se o SQL Server também fosse esperto assim? E ao invés de ter todo o trabalho para pensar nas variáveis em relação ao percurso que queremos pegar, ele simplesmente "guarde" esta informação em memória, e caso a mesma requisição seja solicitada (pegar o metrô do ponto "a" para o ponto "b") ele já saiba como fazer isso.

De fato o SQL Server tem uma área conhecida como Plan Cache que armazena todos os planos de execução criados para que futuramente eles sejam reutilizados. Isso tem grande impacto em um servidor com milhares de requisições de uma mesma consulta, ao invés de gerar um novo plano, o SQL Server simplesmente reutiliza o plano armazenado em cache.

O que estou dizendo, é que antes de tentar otimizar uma consulta (gerar um plano de execução) o SQL Server analisa o plan cache para verificar se já existe um plano de execução para a consulta a ser executada. Vejamos isso na prática:

Vamos supor a seguinte consulta:

```
CREATE INDEX ix_ContactName ON Customers(ContactName)
GO
-- Limpa o PlanCache
DBCC FREEPROCCACHE
GO
SELECT * FROM Customers
 WHERE ContactName = 'Antonio Moreno'
GO
```

O comando DBCC FREEPROCCACHE é utilizado para limpar a área de plan cache, ou seja, quando a consulta na tabela Customers for executada, um plano de execução para a consulta será criado já que a área de plan cache está vazia.

Após rodar a consulta acima, conseguimos utilizar alguns objetos de sistema do SQL Server para visualizar as informações armazenadas no plan cache, a consulta abaixo lê um objeto chamado "sys.dm_exec_cached_plans" e faz join com outros objetos do sistema para retornar informações sobre o plano armazenado em cache:

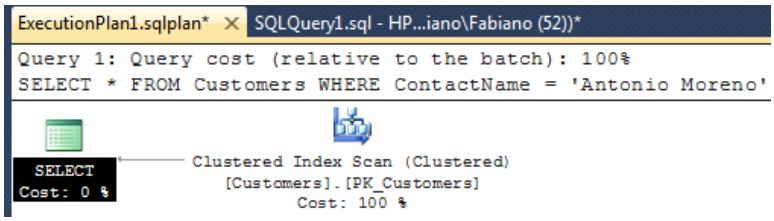
```
-- Consulta o plano de execução em cache
SELECT a.usecounts,
       b.text AS Comando_SQL,
       c.query_plan
  FROM sys.dm_exec_cached_plans a
 CROSS APPLY sys.dm_exec_sql_text (a.plan_handle) b
 CROSS APPLY sys.dm_exec_query_plan (a.plan_handle) c
 WHERE "text" NOT LIKE '%sys.%'
   AND "text" LIKE '%SELECT * FROM Customers%'
```

Resultado da consulta acima:

	usecounts	Comando_SQL	query_plan
1	1	SELECT * FROM Customers WHERE ContactName = 'Antonio Moreno'	<ShowPlanXML xmlns="http://schemas.microsoft.com...

Conforme podemos visualizar na imagem acima, existe um registro para o select na customers contendo o plano de execução (coluna query_plan) em XML para a consulta.

A coluna usecounts retorna a quantas vezes o plano acima foi utilizado e ao clicar na coluna com o resultado do plano em XML, o plano em modo gráfico é exibido.



Se você rodar a mesma consulta procurando por ContactName = “Antonio Moreno”, ao consultar o plan cache teremos um usecounts igual a 2 conforme podemos ver a imagem abaixo:

	usecounts	Comando_SQL	query_plan
1	2	SELECT * FROM Customers WHERE ContactName = 'A...	<ShowPlanXML xmlns="http://schemas.microsoft.com...

Conforme podemos visualizar na imagem acima, ao invés de criar um novo plano de execução, antes de iniciar a faze de otimização da consulta o SQL Server consultou a área de cache plan e identificou que já existia um plano para a consulta solicitada e reutilizou este plano.

Parametrização

Existe um problema na consulta que executamos acima, o problema é que a consulta não está parametrizada, e por isso o SQL Server não consegue reutilizar o plano de execução gerado para ela. Ao procurar um plano de execução no plan cache, o SQL Server gera um Hash do texto (comando T-SQL) enviado para o servidor e vai no plan cache procurar por um plano que “bate” com o hash gerado. O problema com isso é que se eu mudar o texto da clausula WHERE o SQL Server vai gerar um novo hash e consequentemente não vai conseguir encontrar o plano em cache.

Vamos ver isso na prática para entendermos melhor, suponha que agora eu consulte um cliente chamado Pedro Afonso:

```
SELECT * FROM Customers
 WHERE ContactName = 'Pedro Afonso'
```

Após executar a consulta acima, vamos consultar o plan cache para verificar se conseguimos utilizar o plano gerado anteriormente para o cliente Antonio Moreno:

Capítulo 3 – Estatísticas

Commented [F6]: Estimativa de 40 páginas

Introdução a estatísticas

Entendendo como estatísticas funcionam e sua importância
Lendo um histograma

Estatísticas avançado

Manutenção em estatísticas
Auto Update, Auto Created, Auto update async e norecompute
Estatísticas filtradas
Estatísticas cross-table
Variáveis do tipo table versus tabelas temporárias
Estatísticas correlatas e densidade
Atualizando estatísticas com valores falsos
TraceFlags – 2388, 2389, 2390, 2371 (SQL2008SP1), 9292, 9204, 4137 e 8666.
Identificando colunas ascendentes
Date correlation optimization
Estatísticas faltando
Gap nas estatísticas
Estatísticas em várias colunas
Regra diferenciada para ler um histograma
Tried trees para estimar strings
AutoCreated – Computed columns
Estatísticas em views
Estatísticas em functions
Impacto das estatísticas em operações de rebuild

Capítulo 4 – Termos e pontos chave na otimização

Commented [F7]: Estimativa de 20 páginas

Seletividade
Densidade
Cardinalidade
Magic Density/Guess
Foldable expressions
SARGs
Simplifications

Eliminando joins (FKs)
Detectando contradição (Check constraints)
Índices únicos (eliminando Asserts)

NonUpdating updates

Capítulo 5 – Algoritmos de Join

Commented [F8]: Estimativa de 20 páginas

Nested Loop Join

Entendendo o algoritmo de loop join

Merge Join

Entendendo o algoritmo de Merge Join

Evitando Sort Merge Join

Otimizando Merge Joins em disco

Cenários propensos a otimização do Merge Join

Hash Join

Entendendo o algoritmo de Merge Join

Hash Warning Event

Capítulo 6 – Operadores

Commented [F9]: Estimativa de 60 páginas

Index scan e table scan

Allocation order scan

Nolock – Uma bomba relógio

Inconsistências, leitura repetida e pulando linhas

Index order scan

Inconsistências, leitura repetida e pulando linhas

Advanced scan (merry-go-round scanning)

Scan direction e paralelismo

Scan sempre faz scan?

Index Seek

Seek predicate

Predicate

Range scan

Quando um seek é na verdade um Scan

Seek é sempre melhor que scan?

Hints – "Ajudando" otimizador de consultas com ForceSeek

Key Lookup e RID Lookup

- O que é?
- Como melhorar consultas com Key Lookup utilizando clausula Include
- Diferença entre Key Lookup e Rid Lookup
- Cuidados com “missing indexes” (dicas do SQL Server)
- Até quando vale a pena fazer um lookup?
- O que é prefetch? Devo me preocupar com ele?
- Nested Loops Optimized, o que é isso?

Melhorando consultas com o operador Sort

- Otimizando consultas com operadores de SORT
- Ordenação na aplicação ou no banco de dados?
- Monitorando Sort Warnings
- Entendendo memory grant
- xEvent – sort_memory_grant_adjustment
- In-Memory sort versus regular-sort
  - Single pass spill e Multiple Pass spill
- Analisando opções para evitar sort_warnings

- Melhorando consultas com operador Merge Join
  - Entendendo o algoritmo de Merge Join
  - Evitando Sort Merge Join
  - Otimizando Merge Joins em disco
  - Cenários propensos a otimização do Merge Join

Spool

- Table Spool – Lazzy e Eager
  - Halloween problem
  - Entendendo rebind e rewind
- Index Spool
  - Regra diferenciada para rebind e rewind
- Otimizando planos com operações de spool
- RowCount Spool
- Evitando spools em operações de insert com scalar functions

Scan

FullScan

Clustered Index Scan

Non-Clustered Index Scan

Index Seek

Bookmark Lookup

RID Lookup

Operadores de Spool

Eager Spool

Lazy Spool

Index Spool

RowCount Spool

Stream Aggregate

Sort

Merge Interval

Split/Sort/Collapse

Assert

Concatenation

Compute Scalar

Capítulo 7 – Tópicos avançados

Commented [F10]: Estimativa de 30 páginas

Influenciando o otimizador de consultas

Comandos avançados

- DBCC OPTIMIZER_WHATIF
- Rules (DBCC RULEON/OFF, QueryRuleOff)

- Índices hipotéticos (DBCC AUTOPILOT, SET AUTOPILOT)
- Discos SSDs VS peso do custo de IO/CPU (DBCC SETIOWEIGHT, SETCPUWEIGHT)
- xEvent – inaccurate_cardinality_estimate

Hints – "ajudando" otimizador com force order

- Criando bushy plans

- Hints – "ajudando" otimizador forçando um algoritmo de join

Aprendendo mais com analise de bugs e "gaps na funcionalidade" do Otimizador de consultas

- IS NOT NULL
- Comando Merge
- Expression in queries
- Operador de Filter
- Produto cartesiano
- Stream Aggregate
- Filter vs Aggregation
- CTE e colunas duplicadas

Quebrando mitos

- COUNT(1) versus COUNT(*)
- JOIN versus EXISTS
- DISTINCT versus GROUP BY
- SET versus SELECT
- TOP 1 ORDER BY DESC versus MAX
- UNION versus UNION ALL
- NOT IN versus NOT EXISTS
- CURSOR versus WHILE
- Ordem das tabelas no JOIN

Operadores

- Merge interval
- Assert
- Concatenation
- Split, Sort e Collapse