BANCO DE DADOS II

Unidade 1 - SQL

1.1 Linguagem de programação Transact-SQL

O American National Standards Institute (ANSI) e a International Standards Organization (ISO) definiram padrões para o SQL. Usando o Transact-SQL, o Microsoft® SQL Server oferece suporte à implementação em nível de acesso do SQL-92, padrão SQL publicado pelo ANSI e pela ISO em 1992. Os elementos de linguagem do Transact-SQL compatíveis com ANSI-SQL podem ser executados em qualquer produto compatível com nível de acesso ANSI-SQL. O Transact-SQL também contém várias extensões para oferecer maior funcionalidade.

1.2 Elementos de sintaxe do Transact-SQL

As instruções de DML são construídas com o uso de vários elementos de sintaxe do Transact-SQL. Por exemplo:

Diretivas em lotes

Comentários

Identificadores

Tipos de dados

Variáveis

Funções do sistema

Operadores

Expressões

Elementos de linguagem de controle de fluxo

Palavras-chave reservadas

1.3 Diretivas em lotes

O SQL Server processa uma única ou várias instruções Transact-SQL em lotes. Uma diretiva em lotes instrui o SQL Server a analisar e executar todas as instruções contidas no lote. Existem dois métodos básicos de enviar os lotes para o SQL Server.

GO

Os utilitários do SQL Server interpretam GO como um sinal para enviar o lote atual de instruções Transact-SQL para o SQL Server. O comando GO define os lotes de instruções Transact-SQL para ferramentas e utilitários e finaliza o lote. Esse comando não é uma instrução Transact-SQL real.

Ao usar o comando GO, considere estes fatos:

O lote atual é composto de todas as instruções inseridas desde o último comando GO ou desde o início da sessão ad hoc (ou script, se esse for o primeiro GO).

Uma instrução Transact-SQL não pode ocupar o mesmo registro que um comando GO, embora o registro possa conter comentários.

Os usuários devem seguir as regras referentes a lotes. Por exemplo, algumas instruções de DDL devem ser executadas isoladamente de outras instruções Transact-SQL, separando-as com um comando GO.

O escopo das variáveis locais (definidas pelo usuário) é limitado a um lote e não é possível fazer referência a ele depois de um comando GO.

GO não é uma instrução Transact-SQL real; GO é usado para definir os lotes para ferramentas e utilitários.

EXEC

A diretiva EXEC é usada para executar uma função definida pelo usuário, um procedimento do sistema, um procedimento armazenado definido pelo usuário ou um procedimento armazenado estendido; ela também pode controlar a execução de uma seqüência de caracteres em um lote do Transact-SQL. Parâmetros podem ser passados como argumentos e um status de retorno pode ser atribuído.

1.4 Comentários

Comentários são seqüências de texto não executáveis inseridas em instruções para descrever a ação ou desativar uma ou mais linhas da instrução. Eles podem ser usados de duas maneiras. Em linha com uma instrução ou como um bloco.

Comentários em linha

É possível criar comentários em linha usando dois hifens (--) para isolá-los da instrução. O Transact-SQL ignora o texto posicionado à direita dos caracteres de comentário. Esses caracteres de comentário também podem ser usados para desativar linhas de uma instrução.

Este exemplo usa um comentário em linha para explicar um cálculo.

```
USE northwind
SELECT productname
  , (unitsinstock - unitsonorder) -- Calcula o inventário
 , supplierid
 FROM products
GO
```

Este exemplo usa um comentário em linha para impedir a execução de uma seção de uma instrução.

```
USE northwind
SELECT productname
  , (unitsinstock - unitsonorder) -- Calcula o inventário
-- , supplierid
 FROM products
GO
```

Comentários em bloco

É possível criar blocos de comentários de várias linhas, inserindo um caractere de comentário (/*) no início do texto do comentário, digitando o comentário e concluindo-o com um caractere de encerramento de comentário (*/). Use esse caractere indicativo para criar uma ou mais linhas de comentários ou cabeçalhos de comentários. Texto descritivo que documenta as instruções subseqüentes. Geralmente, os cabeçalhos incluem o nome do autor, a data de criação e da última modificação do script, informações sobre a versão e uma descrição da ação executada pela instrução.

Este exemplo apresenta um cabeçalho de comentário que ocupa várias linhas.

```
/*
 Este código recupera todas as linhas da tabela products e
 exibe o preço unitário, o preço unitário aumentado
 em 10% e o nome do produto.
*/
USE northwind
SELECT unitprice, (unitprice * 1.1), productname
 FROM products
GO
```

Insira comentários em um script inteiro para descrever as ações das instruções. Esse recurso é importante principalmente se outros usuários precisarem examinar ou implementar o script.

Esta seção de um script está comentada para impedir a sua execução. Esse é um recurso útil ao depurar ou solucionar problemas de arquivo script.

```
/*
DECLARE @v1 int
SET @v1 = 0
WHILE @v1 < 100
   BEGIN
    SELECT @v1 = (@v1 + 1)
    SELECT @v1
   END
*/
```

1.5 Identificadores

O SQL Server fornece uma série de regras de nomeação padrão para identificadores de objetos e um método de uso de delimitadores para identificadores que não sejam padrão. Sempre que possível, procure atribuir nomes aos objetos empregando os caracteres de identificadores padrão.

Identificadores padrão

Os identificadores padrão podem conter de 1 a 128 caracteres, inclusive letras, símbolos (_, @ ou #) e números. Não são permitidos espaços incorporados.

Examine a seguir as regras para usar identificadores:

- O primeiro caractere deve ser um caractere alfabético, de a-z ou A-Z.
- Após o primeiro caractere, os identificadores podem incluir letras, números ou os símbolos @, $, # ou _ .
- Os nomes de identificadores que começam com um símbolo têm utilizações especiais:
- Um identificador que inicie com o símbolo @ indica um parâmetro ou variável local.
- Um identificador que inicie com um sinal de tralha (#) indica um procedimento ou uma tabela temporária.
- Um identificador que inicie com um sinal de tralha duplo (##) indica um objeto temporário global.
- Os nomes de objetos temporários não devem ter mais de 116 caracteres, incluindo os sinais # ou ##, porque o SQL Server atribui um sufixo numérico interno aos objetos temporários.

Identificadores delimitados

Se um identificador atender a todas as regras de formato de identificadores, poderá ser utilizado com ou sem delimitadores. Se um identificador não atender a uma ou mais regras de formato de identificadores, deverá ser sempre delimitado.

Os identificadores delimitados podem ser usados nas seguintes situações:

- Quando os nomes contiverem espaços incorporados
- Quando forem utilizadas palavras reservadas como nomes de objetos ou partes de nomes de objetos
- Os identificadores delimitados devem ser colocados entre parênteses ou aspas duplas quando incluídos em instruções Transact-SQL.
- Os identificadores agrupados são delimitados por colchetes ([ ]):

```
SELECT * FROM [Espaços no nome da tabela]
```

É possível utilizar sempre os delimitadores agrupados, independentemente do status da opção SET QUOTED_IDENTIFIER.

Os identificadores entre aspas são delimitados por aspas duplas (""):

```
SELECT * FROM "Espaços no nome da tabela"
```

Só é possível usar identificadores entre aspas com a opção SET QUOTED_IDENTIFIER ativada.

Diretrizes de nomeação de identificadores

Ao nomear objetos do banco de dados:

Mantenha os nomes curtos.

Quando possível, empregue nomes significativos.

Use convenções de nomeação simples e fáceis. Descubra o que funcionará melhor na sua situação e seja coerente. Evite criar convenções de nomeação muito complexas, porque podem se tornar difíceis de controlar ou entender. Por exemplo, retire as vogais se o nome deum objeto precisar ficar parecido com uma palavra-chave (como um procedimento armazenado de backup chamado bckup).

Use um identificador que diferencie tipos de objeto, principalmente para views e procedimentos armazenados. Os administradores de sistemas costumam confundir views com tabelas, um descuido que pode gerar problemas imprevisíveis.

Mantenha a exclusividade dos nomes de objetos e usuários. Por exemplo, evite criar uma tabela sales (vendas) e um cargo sales dentro do mesmo banco de dados.

1.6 Tipos de dados

Os tipos de dados restringem os tipos de valores que podem ser armazenados em um banco de dados. Os tipos de dados são atributos que especificam o tipo de informação que pode ser armazenado em uma coluna, parâmetro ou variável.

A maioria das instruções Transact-SQL não faz referência explícita a tipos de dados, mas os resultados da maioria das instruções são influenciados pelas interações entre os tipos de dados dos objetos aos quais a instrução faz referência.

O SQL Server inclui tipos de dados fornecidos pelo sistema (base), mas você também pode criar tipos de dados. Alguns exemplos de tipos de dados base são:

Numbers

Este tipo de dados representa valores numéricos e inclui inteiros, como int, tinyint, smallint e bigint. Ele também inclui valores decimais precisos, como numeric, decimal, money e smallmoney, e valores de ponto flutuante, como float e real.

Dates

Este tipo de dados representa datas ou períodos de tempo. Os dois tipos de dados de data são datetime, cuja precisão é de 3,33 milissegundos, e smalldatetime, cuja precisão é de intervalos de 1 minuto.

Characters

Este tipo de dados é usado para representar seqüências ou dados de caractere e inclui tipos de dados de seqüência de tamanho fixo, como char e nchar, e de tamanho variável, como varchar e nvarchar.

Binary

Este tipo de dados é muito semelhante aos tipos de dados de caractere em termos de armazenamento e estrutura, porém o conteúdo dos dados é tratado como uma série de valores de bytes. Os tipos de dados binary incluem binary e varbinary. O tipo de dados bit indica um valor de bit único igual a zero ou um.

O tipo de dados rowversion indica um valor binário especial de 8 bytes que é exclusivo em um banco de dados.

Unique Identifiers

Este tipo especial de dados é um uniqueidentifier que representa um identificador global exclusivo (GUID), o qual é um valor hexadecimal de 16 bytes que deve ser sempre exclusivo.

SQL Variants

Este tipo de dados pode representar valores de vários tipos de dados para os quais há suporte no SQL Server, com exceção de text, ntext, image, timestamp e rowversion.

Image e Text

Estes tipos de dados são estruturas objeto binário extenso (BLOB) que representam tipos de dados de tamanho fixo e variável para o armazenamento de dados binários e de caractere Unicode e não-Unicode grandes, como image, text e ntext.

Tables

Este tipo de dados pode ser usado apenas para definir variáveis locais de tipo de tabela ou o valor de retorno de uma função definida pelo usuário.

Cursors

Este tipo de dados é usado para programação em procedimentos armazenados e com interfaces de cliente de baixo nível. Ele nunca é usado como parte de uma instrução de DDL.

Tipos de dados definidos pelo usuário

Este tipo de dados é criado pelo administrador do banco de dados e se baseia nos tipos de dados do sistema. Use os tipos de dados definidos pelo usuário quando várias tabelas devem armazenar o mesmo tipo de dados em uma coluna, e você deve garantir que as colunas tenham exatamente o mesmo tipo de dados, tamanho e nulidade.

1.7 Variáveis

Variáveis são elementos da linguagem com valores atribuídos. Você pode usar variáveis locais no Transact-SQL.

Uma variável local é definida pelo usuário na instrução DECLARE, recebe um valor inicial através da instrução SET ou SELECT e é usada na instrução, lote ou procedimento em que foi declarada. O escopo da variável local tem duração do lote em que é definido. A variável local é precedida de um símbolo @.

Os nomes de variáveis precedidos de dois símbolos @ representam um tipo de função.

Consulte Transact SQL Reference, Functions na guia Contents (Conteúdo) do Books Online (Livros on-line) do SQL Server para obter mais informações.

```
DECLARE {@variável_local tipo_de_dados} [,...n]
SET @nome_da_variável_local = expressão
```

Este exemplo cria as variáveis locais @EmpID e @vlname, atribui um valor a @vlname e atribui um valor a @EmpID consultando o banco de dados Northwind para selecionar o registro que contém o valor da variável @vlname.

```
USE northwind
```

```
DECLARE  @EmpID  varchar(11)
        ,@vlName char(20)
SET @vlname = 'Dodsworth'
SELECT @EmpID = employeeid
 FROM employees
 WHERE LastName = @vlname
SELECT @EmpID AS EmployeeID
GO
```

1.8 Funções do sistema

Você pode usar funções, inclusive Funções do sistema, em qualquer local onde uma expressão seja permitida em uma instrução de consulta do Transact-SQL.

O Transact-SQL dispõe de várias funções que retornam informações. Algumas funções recebem os parâmetros de entrada e retornam valores que podem ser utilizados em expressões. Outras apenas retornam valores sem exigir entrada. A linguagem de programação Transact-SQL fornece muitos tipos de funções. Estes são os três tipos de funções com os quais você deve estar familiarizado:

Agem sobre o conjunto de valores de uma coluna selecionada em um conjunto de resultados, mas retornam um valor individual e de resumo.

O exemplo a seguir calcula a média da coluna unitprice (preços unitários) para todos os produtos da tabela products.

```
USE northwind
SELECT AVG(unitprice) AS AvgPrice
 FROM products
GO
```

Retornam um valor individual que opera a partir de nenhum ou de diversos valores escalares individuais. É possível utilizar essas funções sempre que houver uma expressão. As funções escalares podem ser agrupadas nas seguintes categorias.

Configuração - Retorna informações sobre a configuração atual.

Cursor - Retorna informações sobre cursores.

Data e hora - Efetua uma operação sobre um valor inserido de data e hora e retorna um valor de seqüência, numérico ou de data e hora.

Matemática - Efetua um cálculo com base em valores inseridos como parâmetros para a função e retorna um valor numérico.

Metadados - Retorna informações sobre o banco de dados e os objetos do banco de dados.

Segurança - Retorna informações sobre usuários e cargos.

Seqüência de caracteres - Efetua uma operação sobre um valor inserido de seqüência (char ou varchar) e retorna um valor de seqüência de caracteres ou numérico.

Sistema - Efetua operações e retornam informações sobre valores, objetos e configurações no SQL Server.

Estatística do sistema - Retorna informações estatísticas sobre o sistema.

Texto e imagem - Efetua uma operação sobre um valor inserido de texto ou

imagem ou sobre uma coluna, e retorna informações sobre o valor.

Este exemplo de função de metadados retorna o nome do banco de dados em uso no momento.

```
USE northwind
SELECT DB_NAME() AS 'database'
GO
```

Podem ser usadas como referências de tabela em uma instrução Transact-SQL.

O exemplo a seguir executa uma consulta distribuída para recuperar informações da tabela titles (títulos). Observe que esse exemplo não será executado corretamente sem o acesso a um banco de dados Oracle. O SQL Server irá gerar uma mensagem para esse efeito.

```
SELECT *
FROM OPENQUERY(OracleSvr, 'SELECT name, id FROM owner.titles')
GO
```

Exemplos de funções do sistema

As funções do sistema são normalmente usadas para converter dados de datas do formato de um país no de outro.

Para alterar os formatos de data, use a função CONVERT com a opção de estilo para determinar o formato da data que será retornado.

Este exemplo demonstra como converter datas em diversos estilos.

```
SELECT 'ANSI:', CONVERT (varchar(30), GETDATE(), 102) AS Style
UNION
SELECT 'Japanese:', CONVERT(varchar(30), GETDATE(), 111)
UNION
SELECT 'European:', CONVERT(varchar(30), GETDATE(), 113)
GO
Style
ANSI: 1998.11.20
Japanese: 11/20/98
European:  20 Nov 1998 16:44:12:857
```

Este exemplo utiliza a opção DATEFORMAT da instrução SET para formatar datas para a duração de uma conexão. Essa configuração é usada somente na interpretação de seqüências de caracteres à medida que elas são convertidas em valores de data. Ela não afeta a exibição dos valores de data.

```
SET DATEFORMAT dmy
GO
DECLARE @vdate datetime
SET @vdate = '11/29/98'
SELECT @vdate
GO
```

Este exemplo retorna o nome do usuário atual e o aplicativo que o usuário está utilizando para a sessão ou conexão atual. Neste exemplo, o usuário é membro do cargo sysadmin.

```
USE library
SELECT user_name(), app_name()
GO
```

Este exemplo determina se a coluna firstname (nomes) da tabela member (membros) do banco de dados library permite valores nulos.

Um resultado igual a zero (falso) significa que não são permitidos valores nulos e um resultado igual a 1 (verdadeiro) significa que são permitidos valores nulos.

Observe que a função OBJECT_ID está incorporada à função COLUMNPROPERTY. Isso permite obter a object id da tabela member.

```
USE library
SELECT COLUMNPROPERTY(OBJECT_ID('member'), 'firstname', 'AllowsNull')
GO
```

1.9 Operadores

Operadores são símbolos que efetuam cálculos matemáticos, concatenações de seqüências de caracteres e comparações entre colunas, constantes e variáveis. É possível combinar e utilizar os operadores em condições de pesquisa. Ao combiná-los, a ordem em que os operadores serão processados baseia-se em uma precedência predefinida.

```
{constante | nome_da_coluna | função | (subconsulta)}
[{operador_aritmético | operador_de_seqüência_de_caracteres |
  AND | OR | NOT}
  {constante | nome_da_coluna | função | (subconsulta)}.]
```

Tipos de operadores

O SQL Server oferece suporte para quatro tipos de operadores: aritméticos, de comparação, de concatenação de seqüências de caracteres e lógicos.

Aritmético

Os operadores aritméticos efetuam cálculos com colunas ou constantes numéricas. O Transact-SQL dispõe de suporte para operadores multiplicativos, incluindo multiplicação (*), divisão (/) e módulo (%) . o resto inteiro de uma divisão de inteiros . e operadores aditivos, de adição (+) e subtração (-).

Comparação

Os operadores de comparação comparam duas expressões. É possível efetuar comparações entre variáveis, colunas e expressões semelhantes. Você encontrará os operadores a seguir.

= Igual a

> Maior que

< Menor que

>= Maior ou igual a

<= Menor ou igual a

<> Diferente de

Concatenação de seqüências de caracteres

O operador de concatenação de seqüências de caracteres (+) concatena valores de seqüências de caracteres. Todas as outras manipulações de seqüências de caracteres são tratadas através das funções de seqüências de caracteres. A seqüência vazia nunca é avaliada como um valor nulo.

Lógico

Os operadores lógicos AND, OR e NOT conectam condições de pesquisa em cláusulas WHERE.

Níveis de precedência dos operadores

Se você utilizar vários operadores (lógicos ou aritméticos) para combinar expressões, o SQL Server processará os operadores na respectiva ordem de precedência, o que pode afetar o valor resultante. Os operadores têm os seguintes níveis de precedência (do superior para o inferior).

Agrupamento Agrupamento primário ( )

Aritmético Multiplicativo * / %

Aritmético Aditivo - +

Outros Concatenação de

seqüências de caracteres

+

Lógico NOT NOT

Lógico AND AND

Lógico OR OR

O SQL Server trata primeiramente da expressão com o agrupamento mais interno. Além disso, se todos os operadores aritméticos em uma expressão compartilharem o mesmo nível de precedência, a ordem será da esquerda para a direita.

Os níveis de precedência dos operadores lógicos no SQL Server são diferentes dos de outras linguagens de programação.

1.10 Expressões

As expressões são uma combinação de símbolos e operadores que são avaliados como um valor de dados único.

Elas podem ser simples, como uma constante, variável, coluna ou valor escalar. Ou expressões complexas criadas conectando-se uma ou mais expressões simples com operadores.

O tipo de dados do resultado depende dos elementos contidos na expressão. As conversões implícitas de tipos de dados são freqüentemente executadas nos elementos da expressão durante a avaliação.

O exemplo a seguir calcula o valor total de um produto em um pedido multiplicando o preço unitário pela quantidade solicitada; depois, filtra os resultados para que os únicos registros retornados sejam os pedidos com produtos cujo valor total seja maior do que $ 10.000,00.

```
USE     northwind
SELECT  OrderID, ProductID
       ,(UnitPrice * Quantity) as ExtendedAmount
 FROM [Order Details]
```

```
 WHERE (UnitPrice * Quantity) > 10000
GO
```

1.11 Elementos de linguagem de controle de fluxo

O Transact-SQL contém vários elementos de linguagem que controlam o fluxo da lógica em uma instrução. Ele também contém a função CASE que permite usar uma lógica condicional em um único registro de cada vez, em uma instrução SELECT ou UPDATE.

Nível de instrução

Os elementos da linguagem a seguir permitem que você controle o fluxo da lógica em um script:

Estes elementos iniciam e encerram uma seqüência de instruções Transact-SQL para que sejam tratadas como uma unidade.

Estes elementos especificam que o SQL Server deve executar a primeira alternativa se determinada condição for verdadeira. Caso contrário, o SQL Server deverá executar a segunda alternativa.

Esses elementos executam uma instrução várias vezes, enquanto a condição especificada for verdadeira. As instruções BREAK e CONTINUE controlam a operação das instruções dentro de um loop WHILE.

- Recue as instruções Transact-SQL em um bloco de controle de fluxo para melhorar a legibilidade.

Este exemplo determina se um cliente possui algum pedido, antes de excluí-lo da lista de clientes.

```
USE northwind
IF EXISTS (SELECT * FROM orders
           WHERE customerid = 'frank')
  PRINT '*** Cliente não pode ser excluído ***'
ELSE
  BEGIN
    DELETE customers WHERE customerid = 'frank'
    PRINT '*** Cliente foi excluído ***'
```

```
  END
GO
```

Nível de registro

Uma função CASE lista atributos, atribui um valor a cada atributo e testa cada um deles. Se a expressão na cláusula WHEN retornar um valor verdadeiro, a função CASE retornará a expressão na cláusula THEN. Se a expressão for falsa e você tiver especificado uma cláusula ELSE, o SQL Server retornará o valor na cláusula ELSE. Use uma função CASE em qualquer local válido para uma expressão.

```
CASE expressão
   {WHEN expressão THEN resultado}  [,.n]
[ELSE resultado]
END
```

O exemplo a seguir declara uma variável local, verifica se ela é igual a 4, 5 ou 6 e, em caso afirmativo, percorre um loop WHILE que determina se o valor atual é um número ímpar ou par.

```
DECLARE @n tinyint
SET  @n = 5
IF (@n BETWEEN 4 and 6)
 BEGIN
  WHILE (@n > 0)
   BEGIN
    SELECT  @n AS 'Número'
      ,CASE
        WHEN (@n % 2) = 1
          THEN 'ÍMPAR'
        ELSE 'PAR'
       END AS 'Tipo'
    SET @n = @n - 1
   END
```

```
 END
ELSE
 PRINT 'SEM ANÁLISE'
GO
```

Palavras-chave reservadas

O SQL Server reserva certas palavras-chave para seu uso exclusivo. Por exemplo, o uso da palavra-chave DUMP ou BACKUP em uma sessão do SQL Query Analyzer (Analisador de consultas do SQL) ou do osql instrui o SQL Server a fazer uma cópia de backup de todo ou parte de um banco de dados ou do log.

Você não pode incluir palavras-chave reservadas em nenhum local de uma instrução Transact-SQL, exceto onde definido pelo SQL Server. Você deve evitar nomear um objeto com uma palavra-chave reservada. Se o nome de um objeto coincidir com uma palavra-chave, você deverá colocá-lo entre identificadores de delimitação, como aspas ou colchetes [ ], sempre que fizer referência ao objeto.

Os cargos de administrador do banco de dados e de administrador de sistema, ou o criador do banco de dados, são normalmente responsáveis pela verificação das palavras-chave reservadas em nomes de bancos de dados e instruções Transact-SQL.

É possível construir instruções Transact-SQL sintaticamente corretas que podem ser compiladas e analisadas com êxito, mas que ainda retornam um erro de tempo de execução ao serem executadas. Recomenda-se não usar palavras-chave reservadas.

Laboratório

Para realizar os exercícios deste material, baixe gratuitamente o banco de dados de exemplo NorthWind do site da Microsoft.

1- Declare uma variável para receber um valor numérico. Crie uma consulta que mostre as vendas (order) que estão a média está acima desta variável

2- Calcule a soma das compras de cada cliente. Se metade do maior valor de um produto for menor que média exibir “maior’, se não exibir “menor”

3- Converta as datas das vendas para o formato dd-mm-aaaa e mostre a diferença entre as datas de pedido (RequiredDate) e data de envio (ShippedDate)

4- Mostre a maior venda de cada categoria de cada produto

5- Mostre a menor venda de cada cliente separado por anos e meses

6- Declare duas variáveis e realize as quatro operações matemáticas básicas

7- Exiba as Regiões dos empregados mostrando a média de vendas de cada categoria

8- Exiba a média de produtos vendidos por fornecedor

9- Crie um laço que inicie no menor valor de um produto e termine no menor. Em cada passo deste laço deve ser exibido o nome do produto com este valor ou exibir a mensagem “Sem produto neste valor”

10- Calcule a diferença entre as datas da ultima e penúltima compra de cada cliente

Unidade 2 - Implementando Views

2.1 O que são Views

As views permitem armazenar uma consulta predefinida como um objeto no banco de dados para uso posterior. As tabelas consultadas em uma view são chamadas tabelas base. Com algumas exceções, você pode nomear e armazenar qualquer instrução SELECT como uma view. Alguns exemplos comuns de views são:

- Um subconjunto de registros ou colunas de uma tabela base.
- Uma união de duas ou mais tabelas base.
- Uma associação de duas ou mais tabelas base.
- Um resumo estatístico de uma tabela base.
- Um subconjunto de outra view ou alguma combinação de views e tabelas base.

Este exemplo cria a view dbo.EmployeeView no banco de dados Northwind.

A view exibe duas colunas da tabela Employees (Funcionários).

```
USE Northwind
GO
CREATE VIEW dbo.EmployeeView
AS
SELECT LastName, Firstname
FROM Employees


SELECT * from EmployeeView
```

2.2 Vantagens das Views

As views oferecem diversas vantagens, incluindo focalizar os dados para os usuários, mascarar a complexidade dos dados, simplificar o gerenciamento de permissões e organizar dados para serem exportados para outros aplicativos.

Focalizar os dados para os usuários

As views criam um ambiente controlado que permite o acesso a dados específicos enquanto outros dados ficam ocultos. Dados desnecessários, confidenciais ou inadequados podem ser deixados fora de uma view.

Os usuários podem manipular a exibição dos dados em uma view da mesma forma que em uma tabela. Além disso, com as permissões apropriadas e algumas restrições, eles podem modificar os dados produzidos pela view.

Mascarar a complexidade do banco de dados

As views isolam do usuário a complexidade do design do banco de dados. Isso permite que os desenvolvedores alterem o design sem afetar a interação do usuário com o banco de dados. Além disso, os usuários podem ver uma versão mais amigável dos dados usando nomes mais fáceis de compreender do que os nomes abreviados geralmente usados nos bancos de dados.

Consultas complexas, incluindo consultas distribuídas a dados heterogêneos, também podem ser mascaradas através de views. O usuário consulta a view em vez de escrever a consulta ou executar um script.

Simplificar o gerenciamento de permissões de usuários

Em vez de conceder permissão para que os usuários consultem colunas específicas em tabelas base, os proprietários de bancos de dados podem conceder permissões para que os usuários consultem dados somente através de views. Isso também protege as alterações no design das tabelas base subjacentes. Os usuários podem continuar a consultar a view sem interrupção.

Melhorar o desempenho

As views permitem que você armazene os resultados de consultas complexas. Outras consultas podem usar esses resultados resumidos. As views também permitem o particionamento dos dados. Você pode colocar partições individuais em computadores separados.

Organizar dados para serem exportados para outros aplicativos

Você pode criar uma view com base em uma consulta complexa que associe duas ou mais tabelas e, depois, exportar os dados para outro aplicativo para análise adicional.

2.3 Criando views

Você pode criar views usando o Create View Wizard (Assistente para criação de views), o SQL Server Enterprise Manager (Gerenciador corporativo do SQL Server) ou o Transact-SQL. As views só podem ser criadas no banco de dados atual.

Criando uma view

Quando você cria uma view, o Microsoft® SQL Server verifica a existência de objetos aos quais a definição da view faz referência. O nome da view deve seguir as regras dos identificadores. A especificação do nome do proprietário da view é opcional.

Você deve desenvolver uma convenção de nomeação consistente para fazer a distinção entre views e tabelas. Por exemplo, você poderia adicionar a palavra view como um sufixo para cada objeto de view criado. Isso permite que você diferencie facilmente objetos semelhantes (tabelas e views) ao consultar a view INFORMATION_SCHEMA.TABLES.

```
CREATE VIEW proprietário.nome_da_view [(coluna [,n ])]
[WITH {ENCRYPTION | SCHEMABINDING | VIEW_METADATA} [,n ]]
AS
instrução_select
[WITH CHECK OPTION]
```

Para executar a instrução CREATE VIEW, você deverá ser participante do cargo de administradores do sistema (sysadmin), do cargo de proprietário do banco de dados (db_owner) ou do cargo de administrador de linguagem de definição de dados (db_ddladmin) ou deverá ter a permissão CREATE VIEW.

Você também deverá ter a permissão SELECT em todas as tabelas ou views às quais a view faz referência. Para evitar situações em que o proprietário de uma view e o proprietário das tabelas subjacentes sejam diferentes, recomenda-se que o usuário dbo possua todos os objetos de um banco de dados. Ao criar o objeto, especifique sempre o usuário dbo como o nome do proprietário; caso contrário, o objeto será criado com o seu nome de usuário como o proprietário do objeto.

Especifique o conteúdo de uma view usando uma instrução SELECT. Com algumas limitações, as views podem ser tão complexas quanto você desejar. Você deverá especificar nomes de colunas se:

Quaisquer colunas da view derivam de uma expressão aritmética, de uma função interna ou de uma constante.

Quaisquer colunas de tabelas que serão associadas compartilharem o mesmo nome.

Quando você cria views, é importante testar a instrução SELECT que define a view para garantir que o SQL Server retorne o conjunto de resultados esperado. Depois de escrever e testar a instrução SELECT e verificar os resultados, crie a view.

2.4 Restrições às definições de views

Ao criar views, considere as restrições a seguir:

- A instrução CREATE VIEW não pode incluir as cláusulas COMPUTE ou COMPUTE BY.
- A instrução CREATE VIEW não pode incluir a palavra chave INTO.
- A instrução CREATE VIEW só poderá incluir a cláusula ORDER BY se a palavra-chave TOP for usada.
- As views não podem fazer referência a tabelas temporárias.
- As views não podem fazer referência a mais de 1.024 colunas.
- A instrução CREATE VIEW não pode ser combinada com outras instruções Transact-SQL em um único lote.

Este é um exemplo de view que cria uma coluna (Subtotal) (Subtotal), que calcula os subtotais de um pedido com base nas colunas UnitPrice (Preço unitário), Quantity (Quantidade) e Discount (Desconto).

```
USE Northwind
GO
CREATE VIEW dbo.OrderSubtotalsView (OrderID, Subtotal)
AS
SELECT OD.OrderID,
SUM(CONVERT
(money,(OD.UnitPrice*Quantity*(1-Discount)/100))*100)
FROM [Order Details] OD
```

```
GROUP BY OD.OrderID
GO
```

Este exemplo consulta a view para ver os resultados.

```
SELECT * FROM OrderSubtotalsView
```

Freqüentemente, você cria views para fornecer um modo conveniente de examinar informações de duas ou mais tabelas associadas em um local central.

Neste exemplo, ShipStatusView associa as tabelas Customers (Clientes) e Orders (Pedidos).

```
USE Northwind
GO
CREATE VIEW dbo.ShipStatusView
AS
SELECT OrderID, ShippedDate, ContactName
FROM Customers c INNER JOIN Orders o
ON c.CustomerID = O.CustomerID
WHERE RequiredDate < ShippedDate

SELECT * FROM ShipStatusView
OrderID ShippedDate ContactName
```

2.5 Alterando e descartando views

Freqüentemente, você altera as views em resposta a solicitações de informações adicionais feitas pelos usuários ou a alterações na definição da tabela subjacente. Você pode alterar uma view descartando-a e criando-a novamente ou executando a instrução ALTER VIEW.

Alterando views

A instrução ALTER VIEW altera a definição de uma view, incluindo views indexadas, sem afetar os disparadores ou procedimentos armazenados dependentes. Isso permite manter as permissões da view. Essa instrução está sujeita às mesmas restrições que a instrução CREATE VIEW. Se descartar uma view e criá-la novamente, você deverá reatribuir permissões a ela.

```
ALTER VIEW proprietário.nome_da_view
[(coluna [,...n ])]
[WITH {ENCRYPTION | SCHEMABINDING | VIEW_METADATA} [,...n]]
AS
instrução_select
[WITH CHECK OPTION]
Se você usar a opção WITH CHECK OPTION, WITH
ENCRYPTION, WITH SCHEMABINDING ou WITH VIEW_METADATA
```

O exemplo a seguir altera EmployeeView para adicionar a coluna Extension (Extensão).

```
USE Northwind
GO
ALTER VIEW dbo.EmployeeView
AS
SELECT LastName, FirstName, Extension
FROM Employees
```

Se você definir uma view com uma instrução SELECT * e, depois, alterar a estrutura das tabelas subjacentes adicionando colunas, as novas colunas não aparecerão na view. Quando todas as colunas são selecionadas em uma instrução CREATE VIEW, a lista de colunas é interpretada apenas quando você cria a view pela primeira vez. Para ver as novas colunas na view, você deve alterá-la.

Descartando views

Se não precisar mais de uma view, você poderá remover sua definição do banco de dados executando a instrução DROP VIEW. Ao descartar uma view, você remove sua definição e todas as permissões atribuídas a ela. Além disso, se os usuários consultarem qualquer view que faça referência à view descartada, eles receberão uma mensagem de erro. No entanto, descartar uma tabela que faça referência a uma view não descarta automaticamente a view. Você deve descartá-la explicitamente.

A permissão para descartar uma view é atribuída ao proprietário da view e é intransferível. Esse é o padrão. No entanto, o administrador do sistema ou o proprietário do banco de dados pode descartar qualquer objeto especificando o nome do proprietário na instrução DROP VIEW.

2.6 Ocultando definições de views

Como os usuários podem exibir a definição de uma view usando o SQL Server Enterprise Manager, consultando INFORMATION_SCHEMA.VIEWS ou consultando a tabela do sistema syscomments, talvez você deseje ocultar certas definições de views.

Usar a opção WITH ENCRYPTION

Você pode criptografar as entradas da tabela syscomments que contêm o texto da instrução CREATE VIEW especificando a opção WITH ENCRYPTION na definição de view. Antes de criptografar uma view, certifique-se de que a sua definição (script) seja salva em um arquivo. Para descriptografar o texto de uma view, você deve descartá-la e criá-la novamente ou alterá-la e usar a sintaxe original.

Neste exemplo, dbo.[Order Subtotals View] é criada usando a opção WITH ENCRYPTION para ocultar a definição de view.

```
USE Northwind
GO
CREATE VIEW dbo.[Order Subtotals View]
WITH ENCRYPTION
AS
SELECT OrderID,
Sum(CONVERT(money,(UnitPrice*Quantity*(1-Discount)/100))*100)
AS Subtotal
FROM [Order Details]
```

```
GROUP BY OrderID
```

2.7 Modificando dados através de views

As views não mantêm uma cópia separada dos dados. Em vez disso, elas mostram o conjunto de resultados de uma consulta em uma ou mais tabelas base. Portanto, sempre que você modifica dados em uma view, a tabela base é que é realmente modificada.

Com algumas restrições, você poderá inserir, atualizar ou excluir livremente dados de tabelas através de uma view. Em geral, a view deve ser definida em uma única tabela e não deve incluir funções agregadas ou cláusulas GROUP BY na instrução SELECT.

Especificamente, as modificações feitas com o uso de views:

- Não podem afetar mais de uma tabela subjacente. Você pode modificar views derivadas de duas ou mais tabelas, mas cada atualização ou modificação pode afetar apenas uma tabela.
- Não podem ser feitas em certas colunas.
- O SQL Server não permite que você altere uma coluna que seja o resultado de um cálculo, como as colunas que contêm valores calculados, funções internas ou funções agregadas de registros.
- Poderão ocasionar erros se afetarem colunas às quais a view não faz referência.
- Por exemplo, você receberá uma mensagem de erro se inserir em uma view um registro que esteja definido em uma tabela com colunas às quais a view não faz referência e que não permitem NULLs nem contêm valores padrão.
- Serão verificadas se a opção WITH CHECK OPTION tiver sido especificada na definição da view.

A opção WITH CHECK OPTION força todas as instruções de modificação de dados que são executadas na view a obedecer a certos critérios. Esses critérios são especificados na instrução SELECT que define a view. Se os valores alterados estiverem fora do intervalo da definição da view, o SQL Server rejeitará as modificações.

2.8 Otimizando o desempenho com o uso de views

Esta seção descreve as considerações sobre o desempenho para o uso de views e como as views permitem aperfeiçoar o desempenho através do armazenamento dos resultados de consultas complexas e do particionamento dos dados.

Quando views que associam diversas tabelas e avaliam expressões complexas são aninhadas dentro de outra view, poderá ser difícil determinar a origem imediata de qualquer problema de desempenho. Portanto, convém considerar a criação de definições de views separadas, em vez de aninhar views.

No exemplo a seguir, TopSalesView consulta um subconjunto de registros de TotalPurchaseView.

```
USE Northwind
GO
CREATE VIEW dbo.TopSalesView
AS
SELECT *
FROM dbo.TotalPurchaseView
WHERE Subtotal > 50000
GO
```

A definição de view dbo.TopSalesView oculta a complexidade da consulta subjacente usada para criar TotalPurchaseView, que associa três tabelas base.

```
USE Northwind
GO
```

```
CREATE VIEW dbo.TotalPurchaseView
AS
SELECT CompanyName, Sum(CONVERT(money,
(UnitPrice*Quantity*(1-Discount)/100))*100) AS Subtotal
FROM Customers c INNER JOIN Orders o
ON c.CustomerID=o.CustomerID
INNER JOIN [Order Details] od
ON o.OrderID = od.OrderID
GROUP BY CompanyName
GO
```

Se os usuários tiverem problemas de desempenho ao executarem a consulta a seguir para listar os itens mais vendidos da empresa Ernst Handel, a origem do problema não estará imediatamente aparente.

```
SELECT *
FROM dbo.TopSalesView
WHERE CompanyName = 'Ernst Handel'
```

2.9 Usando views indexadas

Você pode criar índices em views. Uma view indexada armazena o conjunto de resultados de uma view no banco de dados. Devido ao tempo rápido de recuperação, é possível usar views indexadas para melhorar o desempenho de consultas.

Criando uma view indexada

Crie uma view indexada implementando um índice UNIQUE CLUSTERED em uma view. Os resultados da view são armazenados nas páginas do nível folha do índice de agrupamento.

Depois de criar o índice UNIQUE CLUSTERED, você poderá criar outros índices nessa view. Uma view indexada reflete automaticamente as modificações efetuadas nos dados das tabelas base. À medida que os dados são alterados, o índice UNIQUE CLUSTERED é atualizado.

Diretrizes para a criação de views indexadas

O otimizador de consultas determina automaticamente se uma consulta se beneficiará do uso de uma view indexada. Isso ocorre mesmo que a consulta não faça referência à view indexada. Como prática geral, deixe que o otimizador de consultas determine quando usar views indexadas.

Usando o Index Tuning Wizard (Assistente para ajuste de índice), você pode melhorar significativamente sua capacidade de determinar a melhor combinação de índices e views indexadas para otimizar o desempenho de consultas.

Crie views indexadas quando:

- A melhoria de desempenho em termos da maior velocidade na recuperação dos resultados compensa o maior custo de manutenção.
- Os dados subjacentes não são atualizados com freqüência.
- As consultas executam uma quantidade significativa de associações e agregações que processam vários registros ou são executadas freqüentemente por vários usuários.

Restrições à criação de views indexadas

Ao criar views indexadas, considere as restrições a seguir:

- O primeiro índice criado em uma view deve ser um índice de agrupamento exclusivo.
- Você deve criar a view com a opção SCHEMABINDING.
- A view pode fazer referência a tabelas base, mas não a outras views.
- Você deve usar nomes de duas partes para fazer referência a tabelas e funções definidas pelo usuário.
- As conexões subseqüentes devem ter as mesmas configurações de opções para usar a view indexada.
- Você deve usar a propriedade IsIndexable da função OBJECTPROPERTY para certificar-se de que possa indexar uma view.

2.10 Práticas recomendadas

As práticas recomendadas a seguir devem ajudá-lo a usar e gerenciar views em seus bancos de dados:

Você deve desenvolver uma convenção de nomeação consistente para fazer a distinção entre views e tabelas.

Especifique dbo como o proprietário quando criar views. O dbo deve ser proprietário de todos os objetos aos quais a definição de view faz referência. Isso torna desnecessário especificar o nome do proprietário quando você consulta a view porque o proprietário do banco de dados é o proprietário padrão. O proprietário do banco de dados também tem permissão em todos os objetos subjacentes do banco de dados, evitando, assim, possíveis cadeias de propriedades interrompidas.

Verifique as dependências dos objetos antes de descartá-los do banco de dados. Execute o procedimento armazenado do sistema sp_depends ou exiba as dependências no SQL Server Enterprise Manager para certificar-se de que não existam dependências de um objeto que você planeja descartar.

Nunca exclua entradas da tabela do sistema syscomments. Se o seu aplicativo exigir que a definição não esteja visível para outras pessoas, inclua a opção WITH ENCRYPTION com a instrução CREATE VIEW ou ALTER VIEW. Certifique-se de salvar a definição do script antes de criptografá-lo.

Avalie cuidadosamente se você deve criar views baseadas em views. Elas podem ocultar complexidades e poderiam ser a origem de problemas de desempenho.

Laboratório

Crie as seguintes visões:

1- Mostrando a média de dias entre as compras por cliente
2- Mostrando para quantos clientes cada vendedor atende por dia
3- Mostrando o produto mais vendido por mês
4- Mostrando a quantidade média de cada produto por cliente
5- Mostrando a média de produtos de cada fornecedor por cliente
6- Mostre o percentual de diferença entre a média e o maior valor de compra por cliente
7- Mostre a diferença entre o maior e menor valor por região
8- Criptografe a visão que mostre o maior valor de item de cada compra
9- Calcule a diferença de dias entre a primeira e a ultima compra por vendedor
10- Mostre a quantos dias cada produto não é vendido

Unidade 3 - Implementando Procedimentos Armazenados

3.1 Definindo procedimentos armazenados

Um procedimento armazenado é uma coleção nomeada de instruções Transact-SQL que é armazenada no servidor. Os procedimentos armazenados são um método de encapsulamento de tarefas repetitivas. Eles oferecem suporte para variáveis declaradas pelo usuário, execução condicional e outros recursos avançados de programação. O SQL Server oferece suporte a cinco tipos de procedimentos armazenados:

Armazenados no banco de dados master, os procedimentos armazenados do sistema (identificados pelo prefixo sp_) fornecem um método eficaz para recuperar informações das tabelas do sistema. Eles permitem que os administradores de sistema executem tarefas de administração de banco de dados que atualizam as tabelas do sistema, mesmo que eles não tenham permissão para atualizar as diretamente tabelas subjacentes. Os procedimentos armazenados do sistema podem ser executados em qualquer banco de dados.

Os procedimentos armazenados locais são criados em bancos de dados de usuário individuais.

Os procedimentos armazenados temporários podem ser locais, com nomes que começam com um sinal de tralha único (#), ou global, com nomes que iniciam com um sinal de tralha duplo (##).

Os procedimentos armazenados temporários estão disponíveis em uma sessão de usuário único; procedimentos armazenados globais estão disponíveis para todas as sessões de usuários.

Os procedimentos armazenados remotos são um recurso anterior do SQL Server. Agora as consultas distribuídas dão suporte a essa funcionalidade.

Estes procedimentos são implementados como dynamic-link libraries (DLLs, bibliotecas de vínculos dinâmicos) executadas fora do ambiente do SQL Server. Os procedimentos armazenados estendidos costumam ser identificados pelo prefixo xp_. Eles são executados de modo semelhante aos procedimentos armazenados.

Os procedimentos armazenados do SQL Server assemelham-se aos procedimentos de outras linguagens de programação sob os seguintes aspectos:

Contêm instruções que executam operações no banco de dados, incluindo a capacidade de chamar outros procedimentos armazenados.

Aceitam parâmetros de entrada.

Retornam um valor de status para um lote ou procedimento armazenado de chamada para indicar um êxito ou uma falha (e a razão da falha).

Retornam vários valores para o lote ou o procedimento armazenado de chamada na forma de parâmetros de saída.

3.2 Processamento inicial de procedimentos armazenados

O processamento de um procedimento armazenado inclui a sua criação e, depois, sua execução pela primeira vez, que coloca o plano de consulta respectivo no cache. O cache de procedimentos é um conjunto de páginas que contém planos de execução para todas as instruções Transact-SQL executadas no momento. O tamanho desse cache varia dinamicamente, de acordo com os níveis de atividade. O cache de procedimentos está localizado no pool de memória, que é a unidade principal de memória do SQL Server. Ele contém a maioria das estruturas de dados que usa a memória no SQL Server.

Criação

Quando um procedimento armazenado é criado, as instruções que ele contém são analisadas para verificar sua precisão sintática. Depois, o SQL Server armazena o nome do procedimento armazenado na tabela do sistema sysobjects (objetos do sistema) e seu texto na tabela do sistema syscomments (comentários do sistema), dentro do banco de dados atual. Será retornado um erro se for encontrado um erro de sintaxe, e o procedimento armazenado não será criado.

Resolução de nomes com atraso

Um processo chamado resolução de nomes com atraso permite que os procedimentos armazenados façam referência a objetos que não existem quando o procedimento é criado. Esse processo proporciona flexibilidade, visto que os procedimentos armazenados e os objetos aos quais eles fazem referência não precisam ser criados em uma ordem específica. Os objetos deverão existir quando o procedimento armazenado for executado. A resolução de nomes com atraso é executada no momento em que o procedimento armazenado é executado.

Execução (primeira vez ou recompilação)

Na primeira vez que um procedimento armazenado é executado ou se ele precisar ser recompilado, o processador de consultas o lerá em um processo chamado resolução.

Certas alterações efetuadas em um banco de dados podem tornar um plano de execução ineficiente ou inválido. O SQL Server detecta essas alterações e recompila automaticamente o plano de execução quando ocorre uma das situações a seguir:

Uma alteração estrutural é efetuada em uma tabela ou view à qual a consulta (ALTER TABLE e ALTER VIEW) faça referência.

Novas estatísticas de distribuição são geradas explicitamente em uma instrução, como UPDATE STATISTICS, ou automaticamente.

Um índice usado pelo plano de execução é descartado.

Alterações significativas são efetuadas nas chaves (instrução INSERT ou DELETE) de uma tabela à qual a consulta faça referência.

Otimização

Quando um procedimento armazenado passa com êxito pelo estágio de resolução, o otimizador de consultas do SQL Server analisa as instruções

Transact-SQL do procedimento e cria um plano que contém o método mais rápido de acesso aos dados. Para fazer isso, ele leva em consideração:

- O volume de dados das tabelas.
- A presença e a natureza dos índices da tabela e a distribuição dos dados nas colunas indexadas.
- Os operadores e os valores de comparação usados nas condições da cláusula WHERE.
- A presença de associações e das cláusulas UNION, GROUP BY ou ORDER BY.

Compilação

A compilação diz respeito ao processo de análise do procedimento armazenado e de criação de um plano de execução que é armazenado no cache de procedimentos. Esse cache contém os planos de execução de procedimentos armazenados mais importantes. Alguns dos fatores que aumentam o valor de um plano de execução incluem:

- Tempo necessário para a recompilação (alto custo de compilação).
- Uso freqüente.

3.3 Processamento subseqüente de procedimentos armazenados

O processamento subseqüente de procedimentos armazenados é mais rápido que o processamento inicial, pois o SQL Server usa o plano de consulta otimizado armazenado no cache de procedimentos.

Se as condições a seguir se aplicarem, o SQL Server usará o plano armazenado na memória para executar a consulta posteriormente: _ O ambiente atual é igual àquele em que o plano foi compilado.

As configurações de servidor, banco de dados e conexão determinam o ambiente.

Os objetos aos quais o procedimento armazenado faz referência não precisam de resolução de nomes.

A resolução de nomes é necessária quando objetos pertencentes a usuários diferentes possuem nomes idênticos. Por exemplo, se o cargo sales for proprietário de uma tabela Product (Produto) e o cargo development for proprietário de uma tabela Product, o SQL Server deverá determinar qual tabela deverá ser consultada toda vez que uma tabela Product for referenciada.

Os planos de execução do SQL Server apresentam dois componentes principais:

Plano de consulta — a maior parte do plano de execução encontra-se nessa estrutura de dados reentrante, somente leitura, que pode ser usada por inúmeros usuários.

Contexto de execução — cada usuário que está executando a consulta no momento possui essa estrutura de dados reutilizável, que armazena os dados específicos de sua execução, como os valores de parâmetros. Se um usuário executar uma consulta, e uma das estruturas não estiver em uso, ela será reinicializada com o contexto do novo usuário. Haverá sempre, no máximo, um plano compilado no cache para cada combinação exclusiva de procedimento armazenado e ambiente. Poderão existir vários planos no cache para o mesmo procedimento armazenado se cada um se destinar a um ambiente diferente.

Os fatores a seguir resultam em diferentes ambientes que afetam as escolhas de compilação:

- Planos compilados paralelos versus seriais.
- Propriedade implícita de objetos.
- Diferentes opções SET.

3.4 Vantagens dos procedimentos armazenados

Os procedimentos armazenados oferecem inúmeras vantagens. Eles podem:

- Compartilhar a lógica do aplicativo com outros aplicativos, garantindo, dessa maneira, o acesso e a modificação consistente dos dados.
- Os procedimentos armazenados podem encapsular as regras de negócios. As regras ou políticas de negócios encapsulados nos procedimentos armazenados podem ser alteradas em um único local. Todos os clientes podem usar os mesmos procedimentos armazenados para garantir o acesso e a modificação consistentes dos dados.
- Proteger os usuários da exposição aos detalhes das tabelas do banco de dados. Se um conjunto de procedimentos armazenados der suporte para todas as funções de negócios que os usuários precisam executar, eles nunca precisarão acessar as tabelas diretamente.
- Fornecer mecanismos de segurança. Os usuários podem receber permissão para executar um procedimento armazenado mesmo que não tenham permissão para acessar as tabelas ou views às quais o procedimento armazenado faz referência.
- Melhorar o desempenho. Os procedimentos armazenados implementam várias tarefas como uma série de instruções Transact-SQL. A lógica condicional pode ser aplicada aos resultados das primeiras instruções Transact-SQL para determinar quais instruções Transact-SQL subseqüentes serão executadas. Todas essas instruções Transact-SQL e a lógica condicional tornam-se parte de um único plano de execução no servidor.
- Reduzir o tráfego de rede. Em vez de enviar centenas de instruções Transact-SQL através da rede, os usuários podem executar uma operação complexa enviando uma única instrução, o que reduz o número de solicitações transferidas entre o cliente e o servidor.

3.5 Criando procedimentos armazenados

Você poderá criar um procedimento armazenado somente no banco de dados atual — com exceção de procedimentos armazenados temporários, que são sempre criados no banco de dados tempdb. A criação de um procedimento armazenado é semelhante à criação de uma view. Em primeiro lugar, escreva e teste as instruções Transact-SQL que você deseja incluir no procedimento armazenado. Depois, se receber os resultados esperados, crie o procedimento armazenado.

Usando CREATE PROCEDURE

Os procedimentos armazenados são criados com a instrução CREATE PROCEDURE. Considere os fatos a seguir ao criá-los:

- Os procedimentos armazenados podem fazer referência a tabelas, views e procedimentos armazenados, bem como a tabelas temporárias.
- Se um procedimento armazenado criar uma tabela temporária local, essa tabela só existirá em função do procedimento e desaparecerá após ele ser executado.
- Uma instrução CREATE PROCEDURE não pode ser combinada com outras instruções SQL em um único lote.
- A definição de CREATE PROCEDURE pode incluir qualquer quantidade e tipo de instruções Transact-SQL, com exceção das instruções de criação de objetos a seguir: CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER e CREATE VIEW. Outros objetos de banco de dados podem ser criados em um procedimento armazenado e devem ser qualificados com o nome do proprietário do objeto.
- Para executar a instrução CREATE PROCEDURE, você deverá ser participante do cargo administradores do sistema (sysadmin), do cargo proprietário do banco de dados (db_owner) ou do cargo administrador da Data Definition Language (DDL, linguagem de definição de dados) (db_ddladmin), ou ter a permissão CREATE PROCEDURE.
- O tamanho máximo de um procedimento armazenado é 128 megabytes (MB), dependendo da memória disponível.

Sintaxe

```
CREATE PROC[EDURE] nome_do_procedimento [;número]
[{@parâmetro tipo de dados}
[VARYING] [= padrão] [OUTPUT]
][,...n]
[WITH
{RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}]
[FOR REPLICATION]
AS
```

```
instrução_sql [...n]
```

As instruções a seguir criam um procedimento armazenado que lista todos os pedidos vencidos no banco de dados Northwind.

```
USE Northwind
GO
CREATE PROC dbo.OverdueOrders
AS
SELECT *
FROM dbo.Orders
WHERE RequiredDate < GETDATE() AND ShippedDate IS Null
GO
```

3.6 Aninhando procedimentos armazenados

Os procedimentos armazenados podem ser aninhados (um procedimento armazenado chama outro). As características do aninhamento incluem:

- Os procedimentos armazenados podem ser aninhados em até 32 níveis. A tentativa de exceder esse limite fará com que toda a chamada em cadeia de procedimentos falhe.
- O nível de aninhamento atual é armazenado na função do sistema @@nestlevel.
- Se um procedimento armazenado chamar um segundo procedimento armazenado, esse poderá acessar todos os objetos criados pelo primeiro, incluindo as tabelas temporárias.
- Os procedimentos armazenados aninhados podem ser recursivos. Por exemplo, o Procedimento armazenado X pode chamar o Procedimento armazenado Y. Enquanto o Procedimento armazenado Y está sendo executado, ele pode chamar o Procedimento armazenado X.

3.7 Diretrizes para a criação de procedimentos armazenados

Considere as diretrizes a seguir ao criar procedimentos armazenados:

- Para evitar situações em que o proprietário de um procedimento armazenado e o proprietário das tabelas subjacentes sejam diferentes, é recomendado que o usuário dbo possua todos os objetos do banco de dados. Como um usuário pode ser participante de vários cargos, especifique sempre o usuário dbo como o nome do proprietário ao criar o objeto. Caso contrário, o objeto será criado tendo o seu nome de usuário como o proprietário:
- Você também deve ter as permissões apropriadas em todas as tabelas ou views às quais o procedimento armazenado faz referência.
- Evite situações em que o proprietário de um procedimento armazenado e o proprietário das tabelas subjacentes sejam diferentes. Se estiver criando um procedimento armazenado do sistema definido pelo usuário, você deverá ter efetuado login como um participante do cargo administradores do sistema (sysadmin) e usar o banco de dados master.
- Crie cada procedimento armazenado para realizar uma única tarefa.
- Crie, teste e depure seu procedimento armazenado no servidor; depois, teste-o no cliente.
- Para distinguir facilmente os procedimentos armazenados do sistema, evite usar o prefixo sp_ ao atribuir nomes a procedimentos armazenados locais.
- Todos os procedimentos armazenados devem assumir as mesmas configurações de conexão.
- Minimize o uso de procedimentos armazenados temporários para evitar a disputa por recursos nas tabelas do sistema de tempdb, o que poderá ocasionar um impacto negativo no desempenho.
- Use sp_executesql em vez de usar a instrução EXECUTE para executar dinamicamente uma seqüência em um procedimento armazenado. sp_executesql é mais eficiente, pois gera planos de execução que o SQL Server tem mais probabilidade de reutilizar. O SQL Server compila a instrução ou instruções Transact-SQL na seqüência de caracteres em um plano de execução separado do plano de execução do procedimento armazenado. Você pode usar o sp_executesql quando estiver executando uma instrução Transact-SQL várias vezes, se a única variação estiver nos valores de parâmetro fornecidos na instrução Transact-SQL.
- Nunca exclua entradas diretamente da tabela do sistema syscomments. Para que os usuários não possam exibir o texto de seus procedimentos armazenados, crie-os usando a opção WITH ENCRYPTION. Se você não usar essa opção, os usuários poderão usar o SQL Server Enterprise Manager (Gerenciador corporativo do SQL Server) ou executar o procedimento armazenado do sistema sp_helptext para exibir o texto dos procedimentos armazenados localizados na tabela do sistema syscomments.

3.8 Executando procedimentos armazenados

Você pode executar um procedimento armazenado isoladamente ou como parte de uma instrução INSERT. É necessário ter a permissão EXECUTE no procedimento armazenado.

Executando um procedimento armazenado isoladamente

Você pode executar um procedimento armazenado emitindo a instrução EXECUTE junto com o nome do procedimento e quaisquer parâmetros.

```
[[EXEC [UTE]]
{
[@status_do_retorno =]
{nome_do_procedimento [;número] |
@ var_nome_do_procedimento}
}
[[@parâmetro = {valor | @variável [OUTPUT] | [DEFAULT]]
[,...n]
[WITH RECOMPILE]
```

A instrução a seguir executa um procedimento armazenado que lista todos os pedidos vencidos no banco de dados Northwind.

```
EXEC OverdueOrders
```

Executando um procedimento armazenado em uma instrução INSERT

A instrução INSERT pode preencher uma tabela local com um conjunto de resultados que é retornado de um procedimento armazenado local ou remoto. O SQL Server carrega a tabela com dados que são retornados das instruções SELECT do procedimento armazenado. A tabela já deverá existir, e os tipos de dados deverão coincidir.

As instruções a seguir criam o procedimento armazenado EmployeeCustomer, que insere funcionários na tabela Customers (Clientes) do banco de dados Northwind.

```
USE northwind
GO
CREATE PROC dbo.EmployeeCustomer
AS
SELECT
UPPER(SUBSTRING(lastname, 1, 4)+SUBSTRING(FirstName, 1,1)),
'Northwind Traders', RTRIM(FirstName)+' '+LastName,
'Employee', Address, City, Region, PostalCode, Country,
('(206) 555-1234'+' x'+Extension), NULL
FROM Employees
WHERE HireDate = GETDATE ()
GO
```

As instruções a seguir executam o procedimento armazenado EmployeeCustomer.

```
INSERT INTO Customers
EXEC EmployeeCustomer
```

O número de funcionários contratados na data de hoje é adicionado à tabela

3.9 Alterando procedimentos armazenados

Para modificar um procedimento armazenado existente e manter as atribuições de permissão, use a instrução ALTER PROCEDURE. O SQL Server substitui a definição anterior do procedimento armazenado quando ele é alterado com ALTER PROCEDURE.

É altamente recomendável não modificar diretamente os procedimentos armazenados do sistema. Em vez disso, crie um procedimento armazenado do sistema definido pelo usuário copiando as instruções de um procedimento armazenado do sistema existente e, depois, modifique-o de acordo com suas necessidades. Considere os fatos a seguir ao usar a instrução ALTER PROCEDURE:

Para modificar um procedimento armazenado que tenha sido criado com quaisquer opções, como WITH ENCRYPTION, inclua a opção na instrução ALTER PROCEDURE para manter a funcionalidade fornecida pela opção.

ALTER PROCEDURE altera apenas um único procedimento. Se o seu procedimento chamar outros procedimentos armazenados, os procedimentos armazenados aninhados não serão afetados.

A permissão para executar essa instrução assume, por padrão, os criadores do procedimento armazenado inicial, os participantes do cargo do servidor sysadmin e os participantes dos cargos fixos de bancos de dados db_owner e db_ddladmin. Você não pode conceder permissão para executar ALTER PROCEDURE.

```
ALTER PROC[EDURE] nome_do_procedimento [;número]
[{@parâmetro tipo_de_dados}
[VARYING] [= padrão] [OUTPUT]
] [,...n]
[WITH
{RECOMPILE | ENCRYPTION
| RECOMPILE , ENCRYPTION
}
]
[FOR REPLICATION]
AS
instrução_sql [...n]
```

O exemplo a seguir modifica o procedimento armazenado OverdueBooks para selecionar somente nomes de colunas específicas em vez de todas as colunas da tabela Orders (Pedidos), bem como para classificar o conjunto de resultados.

```
USE Northwind
```

```
GO
ALTER PROC dbo.OverdueOrders
AS
SELECT CONVERT(char(8), RequiredDate, 1) RequiredDate,
CONVERT(char(8), OrderDate, 1) OrderDate,
OrderID, CustomerID, EmployeeID
FROM Orders
WHERE RequiredDate < GETDATE() AND ShippedDate IS Null
ORDER BY RequiredDate
GO
```

3.10 Descartando procedimentos armazenados

Use a instrução DROP PROCEDURE para remover os procedimentos armazenados definidos pelo usuário do banco de dados atual.

Antes de descartar um procedimento armazenado, execute o procedimento armazenado sp_depends para determinar se os objetos dependem do procedimento.

```
DROP PROCEDURE {procedimento} [,...n]
Este exemplo descarta o procedimento armazenado OverdueOrders.
USE Northwind
GO
DROP PROC dbo.OverdueOrders
GO
```

3.11 Usando parâmetros em procedimentos armazenados

Usando parâmetros de entrada

Os parâmetros de entrada permitem passar informações para um procedimento armazenado. Para definir um procedimento armazenado que aceite parâmetros de entrada, declare uma ou mais variáveis como parâmetros na instrução CREATE PROCEDURE. @parâmetro tipo_de_dados [= padrão] Considere os fatos e as diretrizes a seguir ao especificar parâmetros:

- Todos os valores de parâmetros de entrada devem ser verificados no início de um procedimento armazenado para interceptar primeiro valores ausentes e inválidos.
- Você deve fornecer valores padrão apropriados para um parâmetro. Se for definido um padrão, os usuários poderão executar o procedimento armazenado sem especificar um valor para esse parâmetro.
- Os padrões de parâmetros devem ser constantes ou NULL. Ao especificar NULL como um valor padrão de um parâmetro, você deve usar =Null; IS NULL não funcionará, pois a sintaxe não oferece suporte à designação ANSI NULL.
- O número máximo de parâmetros em um procedimento armazenado é 1.024.
- O número máximo de variáveis locais em um procedimento armazenado é limitado somente pela memória disponível.
- Os parâmetros são locais a um procedimento armazenado. Os mesmos nomes de parâmetros podem ser usados em outros procedimentos armazenados.
- As informações sobre parâmetros são armazenadas na tabela do sistema syscolumns (colunas do sistema).

O exemplo a seguir cria o procedimento armazenado Year to Year Sales, que retorna todas as vendas entre as datas específicas.

```
CREATE PROCEDURE dbo.[Year to Year Sales]
@BeginningDate DateTime, @EndingDate DateTime
AS
IF @BeginningDate IS NULL OR @EndingDate IS NULL
BEGIN
RAISERROR('NULL values are not allowed', 14, 1)
RETURN
END
```

```
SELECT O.ShippedDate,
O.OrderID,
OS.Subtotal,
DATENAME(yy,ShippedDate) AS Year
FROM ORDERS O INNER JOIN [Order Subtotals] OS
ON O.OrderID = OS.OrderID
WHERE O.ShippedDate BETWEEN @BeginningDate AND @EndingDate
GO
```

Executando procedimentos armazenados com parâmetros de entrada

Você pode definir o valor de um parâmetro passando o valor para o procedimento armazenado por nome de parâmetro ou por posição. Você não deve misturar formatos diferentes ao fornecer os valores.

A especificação de um parâmetro em uma instrução EXECUTE no formato @parâmetro = valor é denominada de passagem por nome do parâmetro.

Quando você passa valores por nome de parâmetro, os valores dos parâmetros podem ser especificados em qualquer ordem, e é possível omitir os parâmetros que permitem valores nulos ou que possuem um padrão.

O valor padrão de um parâmetro, se definido no procedimento armazenado, é usado quando:

- Nenhum valor é especificado para o parâmetro quando o procedimento armazenado é executado.
- A palavra-chave DEFAULT é especificada como o valor do parâmetro.

```
[.[ EXEC [ UTE ] ]
{
```

```
[@status_do_retorno =]
{nome_do_procedimento [;número] |
@ var_nome_do_procedimento}
}
[[@parâmetro = {valor | @variável [OUTPUT] | [DEFAULT]]
[,...n]
[WITH RECOMPILE]
```

O exemplo parcial a seguir cria o procedimento armazenado AddCustomer, que adiciona um novo cliente ao banco de dados Northwind. Observe que todas as variáveis, exceto CustomerID e CompanyName são especificadas para permitir um valor nulo.

```
USE Northwind
GO
CREATE PROCEDURE dbo.AddCustomer
@CustomerID nchar (5),
@CompanyName nvarchar (40),
@ContactName nvarchar (30) = NULL,
@ContactTitle nvarchar (30) = NULL,
@Address nvarchar (60) = NULL,
@City nvarchar (15) = NULL,
@Region nvarchar (15) = NULL,
@PostalCode nvarchar (10) = NULL,
@Country nvarchar (15) = NULL,
@Phone nvarchar (24) = NULL,
@Fax nvarchar (24) = NULL
AS
```

O exemplo a seguir passa valores por nome de parâmetro para o procedimento armazenado AddCustomer. Observe que a ordem dos valores é diferente da instrução CREATE PROCEDURE.

Observe, também, que os valores dos parâmetros @Region e @Fax não são especificados. Se as colunas Region (Região) e Fax na tabela permitirem valores nulos, o procedimento armazenado AddCustomer será executado com êxito. No entanto, se as colunas Region e Fax não permitirem valores nulos, você deve passar um valor para um parâmetro, independentemente de você ter definido o parâmetro para permitir um valor nulo.

```
EXEC AddCustomer
@CustomerID = 'ALFKI',
@ContactName = 'Maria Anders',
@CompanyName = 'Alfreds Futterkiste',
@ContactTitle = 'Sales Representative',
@Address = 'Obere Str. 57',
@City = 'Berlin',
@PostalCode = '12209',
@Country = 'Germany',
@Phone = '030-0074321'
```

A passagem somente de valores (sem uma referência aos parâmetros para os quais eles estão sendo passados) é denominada passagem de valores por posição. Quando você especifica somente um valor, os valores dos parâmetros devem ser listados na ordem em que estão definidos na instrução CREATE PROCEDURE.

Ao passar valores por posição, você poderá omitir os parâmetros quando existirem padrões, mas não poderá interromper a seqüência. Por exemplo, se um procedimento armazenado tiver cinco parâmetros, você poderá omitir o quarto e o quinto parâmetros, mas não poderá omitir o quarto parâmetro e especificar o quinto.

O script a seguir passa valores por posição para o procedimento armazenado AddCustomer. Observe que os parâmetros @Region e @Fax não têm valores.

No entanto, somente o parâmetro @Region é fornecido com NULL. O parâmetro @Fax é omitido porque ele é o último parâmetro.

```
EXEC AddCustomer 'ALFKI2', 'Alfreds Futterkiste', 'Maria
Anders', 'Sales Representative', 'Obere Str. 57', 'Berlin',
NULL, '12209', 'Germany', '030-0074321'
```

3.12 Retornando valores com parâmetros de saída

Os procedimentos armazenados podem retornar informações para o cliente ou o procedimento armazenado de chamada com parâmetros de saída (variáveis designadas com a palavra-chave OUTPUT). Com o uso de parâmetros de saída, quaisquer alterações no parâmetro que resultem da execução do procedimento armazenado poderão ser retidas, mesmo após sua execução ser concluída.

Para usar um parâmetro de saída, a palavra-chave OUTPUT deverá ser especificada nas instruções CREATE PROCEDURE e EXECUTE. Se essa palavra-chave for omitida quando o procedimento armazenado for executado, ele ainda será executado, mas não retornará um valor. Os parâmetros de saída apresentam as seguintes características:

A instrução de chamada deve conter um nome de variável para receber o valor de retorno. Não é possível passar constantes.

Você poderá usar a variável posteriormente em instruções Transact-SQL adicionais, no procedimento armazenado de chamada ou no lote.

O parâmetro pode ser qualquer tipo de dados, exceto text ou image.

Eles podem ser espaços reservados para o cursor.

Este exemplo cria um procedimento armazenado MathTutor que calcula o produto de dois números. Este exemplo usa a instrução SET. No entanto, você também pode usar a instrução SELECT para concatenar uma seqüência dinamicamente. Uma instrução SET requer que você declare uma variável para imprimir a seqüência “The result is:”

```
CREATE PROCEDURE dbo.MathTutor
@m1 smallint,
@m2 smallint,
@result smallint OUTPUT
AS
SET @result = @m1* @m2
GO
```

Este lote chama o procedimento armazenado MathTutor e passa os valores de 5 e 6. Esses valores se tornam variáveis, que são fornecidas na instrução SET.

```
DECLARE @answer smallint
```

```
EXECUTE MathTutor 5,6, @answer OUTPUT
SELECT 'The result is:', @answer
```

O parâmetro @result é designado com a palavra chave OUTPUT. O SQL Server imprime o conteúdo da variável @result quando você executa o procedimento armazenado MathTutor. A variável do resultado é definida como o produto de dois valores, 5 e 6.

3.13 Recompilando explicitamente procedimentos armazenados

Os procedimentos armazenados podem ser recompilados explicitamente, mas você não deve fazer isso com freqüência. Use esse procedimento somente quando:

- Os valores de parâmetros são passados para um procedimento armazenado que retorna conjuntos de resultados muito variados.
- Um novo índice é adicionado a uma tabela subjacente da qual um procedimento armazenado poderá se beneficiar.
- O valor do parâmetro fornecido é atípico. O SQL Server fornece três métodos para a recompilação explícita de um procedimento armazenado.

CREATE PROCEDURE...[WITH RECOMPILE]

A instrução CREATE PROCEDURE...[WITH RECOMPILE] indica que o SQL Server não armazena no cache um plano para esse procedimento armazenado. Em vez disso, a opção recompila o procedimento armazenado toda vez que ele é executado.

A instrução a seguir cria um procedimento armazenado chamado OrderCount que é recompilado toda vez que é executado.

```
USE Northwind
GO
```

```
CREATE PROC dbo.OrderCount
@CustomerID nchar (10)
WITH RECOMPILE
AS
SELECT count(*) FROM [Orders Qry]
WHERE CustomerID = @CustomerID
GO
```

EXECUTE...[WITH RECOMPILE]

A instrução EXECUTE...[WITH RECOMPILE] cria um novo plano de execução cada vez que o procedimento é executado, se você especificar WITH RECOMPILE. O novo plano de execução não é colocado no cache. Use essa opção se o parâmetro que você está passando for muito diferente dos que geralmente são passados para o procedimento armazenado. Como esse plano otimizado é uma exceção e não a regra, quando a execução for concluída, você deverá executar novamente o procedimento armazenado com um parâmetro que seja passado normalmente. Essa opção também será útil se os dados tiverem sido significativamente alterados desde que o procedimento armazenado foi compilado pela última vez. Este exemplo recompila o procedimento armazenado do sistema sp_help no momento em que ele é executado.

```
EXEC sp_help WITH RECOMPILE
```

sp_recompile

O procedimento armazenado do sistema sp_recompile recompila o procedimento armazenado do sistema ou disparador especificado na próxima vez que ele é executado. Se o parâmetro @objname especificar uma tabela ou view, todos os procedimentos armazenados que usam o objeto designado serão recompilados na próxima vez que forem executados.

Use o procedimento armazenado do sistema sp_recompile com a opção nome_da_tabela se tiver adicionado um novo índice a uma tabela subjacente à qual o procedimento armazenado faça referência e se acreditar que o desempenho do procedimento armazenado poderá melhorar com o novo índice.

Este exemplo recompila todos os procedimentos armazenados ou disparadores que fazem referência à tabela Customer do banco de dados Northwind.

```
EXEC sp_recompile Customers
```

Você pode usar DBCC FREEPROCCACHE para desmarcar todos os planos de procedimentos armazenados do cache.

3.14 Tratando mensagens de erro

Para tornar os procedimentos armazenados mais eficazes, você deve incluir mensagens de erro que comuniquem o status das transações (êxito ou falha) ao usuário. Você deverá executar a lógica de negócios e da tarefa, bem como a verificação de erro, antes de iniciar as transações, e mantê-las curtas. Você pode usar estratégias de codificação, como verificações de existência, para o reconhecimento de erros. Quando ocorrer um erro, forneça o máximo de informações ao cliente. Você pode verificar os elementos a seguir em sua lógica de tratamento de erros: códigos de retorno, erros do SQL Server e mensagens de erro personalizadas.

Instrução RETURN

A instrução RETURN sai de uma consulta ou procedimento armazenado de modo não condicional. Ela também pode retornar um valor inteiro de status (código de retorno). Um valor de retorno 0 indica êxito. Os valores de retorno de 0 a -14 estão em uso no momento e os valores de retorno de -15 a -99 estão reservados para uso futuro. Se um valor de retorno definido pelo usuário não for fornecido, o valor do SQL Server será usado. Os valores de retorno definidos pelo usuário sempre prevalecem sobre os fornecidos pelo SQL Server.

Este exemplo cria o procedimento armazenado GetOrders que recupera informações das tabelas Orders e Customers consultando a view Orders Qry. A instrução RETURN do procedimento armazenado GetOrders retorna o número total de registros da instrução SELECT para outro procedimento armazenado. Você também poderia aninhar o procedimento armazenado GetOrders em outro procedimento armazenado.

```
USE Northwind
GO
CREATE PROCEDURE dbo.GetOrders
@CustomerID nchar (10)
AS
SELECT OrderID, CustomerID, EmployeeID
FROM [Order Qry]
WHERE CustomerID = @CustomerID
RETURN (@@ROWCOUNT)
GO
```

sp_addmessage

Esse procedimento armazenado permite que os desenvolvedores criem mensagens de erro personalizadas. O SQL Server trata as mensagens de erro do sistema e personalizadas da mesma forma. Todas as mensagens são armazenadas na tabela sysmessages (mensagens do sistema) do banco de dados master. Essas mensagens de erro também podem ser gravadas automaticamente no log de aplicativos do Windows 2000. Este exemplo cria uma mensagem de erro definida pelo usuário que requer que a mensagem seja gravada no log de aplicativos do Windows 2000 quando ela ocorre.

```
EXEC sp_addmessage
@msgnum = 50010,
@lang='US_English',
@severity = 10,
@msgtext = 'Customer cannot be deleted.',
@with_log = 'true'
```

@@error

Esta função do sistema contém o número do erro referente à instrução Transact- SQL executada mais recentemente. Ela é limpa e redefinida toda vez que uma instrução é executada. Um valor igual a 0 será retornado se a instrução for executada com êxito. Você poderá usar a função do sistema @@error para detectar um número de erro específico ou para sair de um procedimento armazenado de modo condicional.

Este exemplo cria o procedimento armazenado AddSupplierProduct no banco de dados Northwind. Esse procedimento armazenado usa a função do sistema @@error para determinar se ocorre um erro toda vez que uma instrução INSERT é executada. Se o erro ocorrer, a transação será revertida.

```
USE Northwind
GO
CREATE PROCEDURE dbo.AddSupplierProduct
@CompanyName nvarchar (40) = NULL,
@ContactName nvarchar (40) = NULL,
@ContactTitle nvarchar (40)= NULL,
@Address nvarchar (60) = NULL,
@City nvarchar (15) = NULL,
@Region nvarchar (40) = NULL,
@PostalCode nvarchar (10) = NULL,
@Country nvarchar (15) = NULL,
```

```
@Phone nvarchar (24) = NULL,
@Fax nvarchar (24) = NULL,
@HomePage ntext = NULL,
@ProductName nvarchar (40) = NULL,
@CategoryID int = NULL,
@QuantityPerUnit nvarchar (20) = NULL,
@UnitPrice money = NULL,
@UnitsInStock smallint = NULL,
@UnitsOnOrder smallint = NULL,
@ReorderLevel smallint = NULL,
@Discontinued bit = NULL
AS
BEGIN TRANSACTION
INSERT Suppliers (
CompanyName,
ContactName,
Address,
City,
Region,
PostalCode,
Country,
Phone)
VALUES (
@CompanyName,
@ContactName,
@Address,
@City,
@Region,
@PostalCode,
@Country,
@Phone)
```

```
IF @@error <> 0
BEGIN
ROLLBACK TRAN
RETURN
END
DECLARE @InsertSupplierID int
SELECT @InsertSupplierID=@@identity
INSERT Products (
ProductName,
SupplierID,
CategoryID,
QuantityPerUnit,
Discontinued)
VALUES (
@ProductName,
@InsertSupplierID,
@CategoryID,
@QuantityPerUnit,
@Discontinued)
IF @@error <> 0
BEGIN
ROLLBACK TRAN
RETURN
END
COMMIT TRANSACTION
```

Instrução RAISERROR

A instrução RAISERROR retorna uma mensagem de erro definida pelo usuário e define um sinalizador do sistema para registrar a ocorrência de um erro. Você deve especificar um nível de gravidade do erro e o estado da mensagem quando estiver usando a instrução RAISERROR. RAISERROR permite que o aplicativo recupere uma entrada da tabela do sistema master..sysmessages (master..mensagens do sistema) ou crie dinamicamente uma mensagem com as informações de estado e gravidade especificadas pelo usuário. A instrução RAISERROR pode gravar mensagens de erro no log de erros do SQL Server e no log de aplicativos do Windows 2000.

Este exemplo gera uma mensagem de erro definida pelo usuário e grava-a no log de aplicativos do Windows 2000.

```
RAISERROR(50010, 16, 1) WITH LOG
```

A instrução PRINT retorna uma mensagem definida pelo usuário para o indicador de mensagem do cliente; no entanto, ao contrário da instrução RAISERROR, ela não armazena o número do erro na função do sistema @@error.

3.15Demonstração: Tratando mensagens de erro

Siga este script à medida que o instrutor demonstrar as técnicas de tratamento de erros contidas nele.

```
/* UpdateCustomerPhone
Atualiza o telefone de um cliente
A verificação de erros garante o fornecimento de um
número de identificação válido
*/
/*
A mensagem definida pelo usuário a seguir oferece suporte ao
procedimento armazenado UpdateCustomerPhone */
```

```
EXEC sp_addmessage 50010, 16, 'CustomerID not found.',
@replace='replace'
USE Northwind
GO
CREATE PROCEDURE UpdateCustomerPhone
@CustomerID nchar (5) = NULL,
@Phone nvarchar (24) = NULL
AS
IF @CustomerID IS NULL
BEGIN
PRINT 'You must supply a valid CustomerID'
RETURN
END
/* Certifique-se de que um cliente válido seja fornecido */
IF NOT EXISTS
(SELECT * FROM Customers WHERE CustomerID = @CustomerID)
BEGIN
RAISERROR (50010, 16, 1) --Cliente não encontrado.
RETURN
END

BEGIN TRANSACTION
UPDATE Customers
SET Phone = @Phone
WHERE CustomerID = @CustomerID
/* Exibir mensagem informando que o telefone de CompanyName
foi atualizado */
SELECT 'The phone number for' + @CustomerID + 'has been
updated to' + @Phone
COMMIT TRANSACTION
GO
```

3.16 Práticas recomendadas

Para escrever procedimentos armazenados mais eficazes e eficientes, siga estas práticas recomendadas:

- Verifique todos os parâmetros de entrada no início de cada procedimento armazenado para interceptar valores ausentes ou inválidos inicialmente.
- Crie cada procedimento armazenado para realizar uma única tarefa.
- Execute a lógica de negócios e da tarefa, bem como a verificação de erros, antes de iniciar as transações. Mantenha suas transações curtas.
- Use as mesmas configurações de conexão para todos os procedimentos armazenados.
- Para ocultar o texto de procedimentos armazenados, use a opção WITH ENCRYPTION. Nunca exclua entradas da tabela do sistema syscomments.

Laboratório

Crie os seguintes procedimentos:

1. Chamado ExcluirCliente que receba como parâmetro o código do cliente. Se ele comprou a menos de 3 meses ele não deve ser apagado. Se não, ele deve ser apagado e transferido os seus dados para uma tabela temporária.
2. Chamado VerificaEstoque que receba como parâmetro o código do produto e retorne em quantos meses o estoque do produto vai acabar baseado na média de vendas do mesmo.
3. Chamado CadastrarCliente que receba todos os dados dos clientes, incluído as tabelas que o mesmo depende
4. Chamado AumentarPreco que receba o percentual de aumento e categoria dos produtos e execute o referido aumento
5. Chamado CriaPromocao que receba o fabricante e o percentual de desconto, concedendo o este percentual aos produtos deste fabricante

Unidade 4 - Implementando Funções Definidas Pelo Usuário

4.1 O que é uma função definida pelo usuário?

Com o Microsoft® SQL Server, você pode criar suas próprias funções para complementar e estender as funções fornecidas pelo sistema (internas).

Uma função definida pelo usuário pode não ter nenhum ou ter vários parâmetros de entrada e retorna um valor escalar ou uma tabela. Os parâmetros de entrada podem ser qualquer tipo de dados, exceto timestamp, cursor ou table. As funções definidas pelo usuário não dão suporte para parâmetros de saída.

O SQL Server dá suporte para três tipos de funções definidas pelo usuário:

Funções escalares

As funções escalares assemelham-se às funções internas.

Funções com valor de tabela e várias instruções

As funções com valor de tabela e várias instruções retornam uma tabela criada por uma ou mais instruções Transact-SQL e assemelham-se a um procedimento armazenado. Diferentemente de um procedimento armazenado, essas funções podem ser referenciadas na cláusula FROM de uma instrução SELECT como se fossem uma view.

Funções com valor de tabela in-line

As funções com valor de tabela in-line retornam uma tabela que é o resultado de uma única instrução SELECT. Elas assemelham-se às views, mas oferecem maior flexibilidade do que elas quanto ao uso de parâmetros e estendem os recursos das views indexadas.

4.2 Criando uma função definida pelo usuário

Uma função definida pelo usuário é criada de maneira semelhante a uma view ou um procedimento armazenado.

Criando uma função

As funções definidas pelo usuário são criadas com a instrução CREATE FUNCTION. Cada função definida pelo usuário deve ter um nome totalmente qualificado exclusivo (nome_do_banco_de_dados.nome_do_proprietário.nome_da_função). A instrução especifica o tipo de dados para os parâmetros de entrada, as instruções de processamento e o valor retornado por cada tipo de dados.

```
CREATE FUNCTION [ nome_do_proprietário. ] nome_da_função
( [ { @nome_do_parâmetro tipo_de_dados_do_parâmetro_escalar
[ = padrão ] } [,...n ] ] )
RETURNS tipo_de_dados_de_retorno_escalar
[ WITH <opção_da_função> [,...n] ]
[ AS ]
BEGIN
corpo_da_função
RETURN expressão_escalar
END
```

Este exemplo cria uma função definida pelo usuário para substituir um valor nulo pelas palavras Not Applicable.

```
USE Northwind
GO
CREATE FUNCTION fn_NewRegion
(@myinput nvarchar(30))
RETURNS nvarchar(30)
BEGIN
IF @myinput IS NULL
SET @myinput = 'Not Applicable'
```

```
RETURN @myinput
END
```

Ao fazer referência a uma função escalar definida pelo usuário, especifique o proprietário e o nome da função na sintaxe de duas partes.

```
SELECT LastName, City, dbo.fn_NewRegion(Region) AS Region,
Country
FROM dbo.Employees
```

Restrições às funções

As funções não-determinísticas são funções, como GETDATE(), que podem retornar valores de resultado diferentes toda vez que são chamadas com o mesmo conjunto de valores de entrada. Não são permitidas funções não-determinísticas internas no corpo de funções definidas pelo usuário. As seguintes funções internas são não-determinísticas.

Definindo permissões para funções definidas pelo usuário

Você deve ter a permissão CREATE FUNCTION para criar, alterar ou descartar funções definidas pelo usuário.

Os usuários que não são o proprietário devem ter a permissão EXECUTE em uma função para usá-la em uma instrução Transact-SQL. Se a função for vinculada a esquema, você deverá ter a permissão REFERENCE nas tabelas, views e funções às quais ela faz referência. As permissões REFERENCE podem ser concedidas através da instrução GRANT para views e funções definidas pelo usuário, bem como para tabelas.

Se uma instrução CREATE TABLE ou ALTER TABLE fizer referência a uma função definida pelo usuário em uma restrição CHECK, uma cláusula DEFAULT ou uma coluna calculada, o proprietário da tabela também

4.3 Alterando e descartando funções definidas pelo usuário

Você pode alterar e descartar funções definidas pelo usuário usando a instrução ALTER FUNCTION.

A vantagem de alterar uma função em vez de descartá-la e recriá-la é a mesma das views e dos procedimentos. As permissões da função são mantidas e aplicadas imediatamente à função revisada.

Alterando funções

Modifique uma função definida pelo usuário usando a instrução ALTER FUNCTION.

Este exemplo mostra como alterar uma função.

```
ALTER FUNCTION dbo.fn_NewRegion
<Novo conteúdo de função>
```

Descartando funções

Descarte uma função definida pelo usuário usando a instrução DROP FUNCTION.

Este exemplo mostra como descartar uma função.

```
DROP FUNCTION dbo.fn_NewRegion
```

4.4 Exemplos de funções definidas pelo usuário

Usando uma função escalar definida pelo usuário

As funções escalares definidas pelo usuário assemelham-se às funções internas. Após criá-las, você poderá reutilizá-las.

Este exemplo cria uma função definida pelo usuário que recebe separadores de coluna e data como variáveis e reformata a data como uma seqüência de caracteres.

```
USE Northwind
GO
CREATE FUNCTION fn_DateFormat
(@indate datetime, @separator char(1))
RETURNS Nchar(20)
AS
BEGIN
RETURN
CONVERT(Nvarchar(20), datepart(mm,@indate))
+ @separator
+ CONVERT(Nvarchar(20), datepart(dd, @indate))
+ @separator
+ CONVERT(Nvarchar(20), datepart(yy, @indate))
END
```

Você pode chamar uma função escalar definida pelo usuário da mesma maneira que uma função interna.

```
SELECT dbo.fn_DateFormat(GETDATE(), ':')
```

Usando uma função com valor de tabela e várias instruções

Uma função com valor de tabela e várias instruções é uma combinação de uma view com um procedimento armazenado. Você pode usar funções definidas pelo usuário que retornam uma tabela para substituir procedimentos armazenados ou views.

Uma função com valor de tabela (como um procedimento armazenado) pode usar uma lógica complexa e várias instruções Transact-SQL para criar uma tabela. Você pode usar uma função com valor de tabela na cláusula FROM de uma instrução Transact-SQL da mesma maneira que usa uma view.

Ao usar uma função com valor de tabela e várias instruções, considere os fatos a seguir:

- BEGIN e END delimitam o corpo da função.
- A cláusula RETURNS especifica table como o tipo de dados retornado.
- A cláusula RETURNS define o nome e o formato da tabela. O escopo do nome da variável de retorno é local à função.

Exemplo de função com valor de tabela e várias instruções

Você pode criar funções usando várias instruções que executam operações complexas.

Este exemplo cria uma função com valor de tabela e várias instruções que retorna o sobrenome ou o nome e o sobrenome de um funcionário, dependendo do parâmetro fornecido.

```
USE Northwind
GO
CREATE FUNCTION fn_Employees
(@length nvarchar(9))
RETURNS @fn_Employees TABLE
(EmployeeID int PRIMARY KEY NOT NULL,
[Employee Name] Nvarchar(61) NOT NULL)
AS
BEGIN
IF @length = 'ShortName'
INSERT @fn_Employees SELECT EmployeeID, LastName
FROM Employees
ELSE IF @length = 'LongName'
INSERT @fn_Employees SELECT EmployeeID,
(FirstName + ' ' + LastName) FROM Employees
RETURN
END
```

Você pode chamar a função em vez de uma tabela ou view.

```
SELECT * FROM dbo.fn_Employees('LongName')
```

ou

```
SELECT * FROM dbo.fn_Employees('ShortName')
```

Usando uma função com valor de tabela in-line

As funções in-line definidas pelo usuário retornam uma tabela e são referenciadas na cláusula FROM, da mesma maneira que uma view. Ao usar funções in-line definidas pelo usuário, considere os fatos e diretrizes a seguir:

A cláusula RETURN contém uma única instrução SELECT entre parênteses. O conjunto de resultados da instrução SELECT constitui a tabela que a função retorna. A instrução SELECT usada em uma função in-line está sujeita às mesmas restrições que as instruções SELECT usadas em views. BEGIN e END não delimitam o corpo da função. RETURN especifica table como o tipo de dados retornado. Não é necessário definir o formato de uma variável de retorno porque ele é definido pelo formato do conjunto de resultados da instrução SELECT na cláusula RETURN.

Exemplo de uma função com valor de tabela in-line

Você pode usar funções in-line para obter a funcionalidade de views com parâmetros.

Não é possível incluir um parâmetro fornecido pelo usuário na view quando ela é criada. Normalmente você pode resolver essa questão fornecendo uma cláusula WHERE ao chamar a view. No entanto, isso poderá exigir a criação de uma seqüência de caracteres para execução dinâmica, o que poderá aumentar a complexidade do aplicativo. Você poderá obter a funcionalidade de uma view com parâmetros usando uma função com valor de tabela in-line.

Este exemplo cria uma função com valor de tabela in-line que aceita um valor de região como parâmetro.

```
USE Northwind
GO
CREATE FUNCTION fn_CustomerNamesInRegion
( @RegionParameter nvarchar(30) )
RETURNS table
AS
RETURN (
SELECT CustomerID, CompanyName
FROM Northwind.dbo.Customers
WHERE Region = @RegionParameter
```

)

4.5 Práticas recomendadas

As práticas recomendadas a seguir devem ajudá-lo a implementar funções definidas pelo usuário:

- Use funções escalares complexas em conjuntos de resultados pequenos. As funções definidas pelo usuário permitem encapsular um raciocínio complexo em uma consulta simples, mas, se todos os usuários da função não compreenderem a complexidade do cálculo subjacente, a função poderá resultar em cálculos demorados que não são visíveis para o usuário. Não aplique uma agregação complexa em cada participante de um conjunto grande de resultados.
- Use funções de várias instruções em vez de procedimentos armazenados que retornam tabelas. Escrever procedimentos armazenados que retornam tabelas como funções de várias instruções definidas pelo usuário poderá aumentar a eficiência.
- Use funções in-line para criar views usando parâmetros. O uso de parâmetros com funções in-line poderá simplificar as referências a tabelas e views.
- Use funções in-line para filtrar views. O uso de funções in-line com views indexadas poderá aumentar significativamente o desempenho.

Laboratório

Crie as seguintes funções;

1. Chamada MenorVenda que receba um código de cliente retorne a menor compra realizada por ele
2. Chamada MaisVendido que receba um mês e retorne o nome do produto mas vendido no referido mês
3. Chamada MelhorVendedor que receba um mês e retorne o código do melhor vendedor do referido
4. Chamada MaiorVenda que receba um código de produto e retorna a data da maior venda do mesmo
5. Chamada MelhorProduto que receba o código do fornecedor e exiba qual o seu produto mais vendido

Unidade 5 - Implementando Disparadores

5.1 O que são disparadores?

Um disparador é um tipo especial de procedimento armazenado que é executado sempre que há uma tentativa de modificar os dados de uma tabela protegida por ele. Os disparadores estão associados a tabelas específicas.

Associados a uma tabela

Os disparadores são definidos em uma tabela específica, que é denominada tabela de disparadores.

Chamados automaticamente

Quando há uma tentativa de inserir, atualizar ou excluir dados em uma tabela, e um disparador tiver sido definido na tabela para essa ação específica, ele será executado automaticamente. Ele não poderá ser ignorado.

Não podem ser chamados diretamente

Ao contrário dos procedimentos armazenados do sistema padrão, os disparadores não podem ser chamados diretamente e não passam nem aceitam parâmetros.

É parte de uma transação

O disparador e a instrução que o aciona são tratados como uma única transação que poderá ser revertida em qualquer ponto do procedimento. Quando estiver usando disparadores, considere estes fatos e diretrizes:

As definições de disparadores podem incluir uma instrução ROLLBACK TRANSACTION, mesmo que não exista uma instrução BEGIN TRANSACTION explícita.

Se uma instrução ROLLBACK TRANSACTION for encontrada, a transação inteira será revertida. Se uma instrução no script do disparador seguir a instrução ROLLBACK TRANSACTION, a instrução será executada. Portanto, poderá ser necessário usar uma cláusula RETURN em uma instrução IF para impedir o processamento de outras instruções.

_Se um disparador que contém uma instrução ROLLBACK TRANSACTION for acionado em uma transação definida pelo usuário, essa instrução reverterá a transação inteira. Um disparador executado em um lote que executa uma instrução ROLLBACK TRANSACTION cancela o lote; as instruções subseqüentes do lote não serão executadas.

Você deve minimizar ou evitar o uso de ROLLBACK TRANSACTION no código de seu disparador. A reversão de uma transação gera um trabalho adicional, pois todo o trabalho concluído até esse ponto na transação precisa ser desfeito. Isso terá um impacto negativo no desempenho. É recomendado que as informações sejam verificadas e validadas fora da transação. Inicie a transação depois que tudo for verificado.

O usuário que chamou o disparador também deve ter permissão para executar todas as instruções em todas as tabelas.

5.2 Usos de disparadores

Os disparadores são usados com maior eficiência para manter a integridade de dados de baixo nível, e não para retornar os resultados de consultas. A sua principal vantagem é que eles podem conter uma lógica de processamento complexa. Os disparadores podem colocar em cascata as alterações através de tabelas relacionadas em um banco de dados, impor a integridade dos dados mais complexos do que uma restrição CHECK, definir mensagens de erro personalizadas, manter os dados desnormalizados e comparar os estados anteriores e posteriores dos dados que sofreram modificações.

Alterações em cascata em tabelas relacionadas de um banco de dados

Você pode usar um disparador para atualizações e exclusões em cascata através de tabelas relacionadas em um banco de dados. Por exemplo, um disparador de exclusão na tabela Products (Produtos) do banco de dados Northwind pode excluir os registros correspondentes em outras tabelas que possuem registros com os mesmos valores de ProductID (Identificação do produto) excluídos. Um disparador faz isso usando a coluna de chave externa ProductID como uma forma de localizar registros na tabela Order Details (Detalhes do pedido).

Impor uma integridade de dados mais complexa do que uma restrição CHECK

Ao contrário das restrições CHECK, os disparadores podem fazer referência a colunas de outras tabelas. Por exemplo, você poderia colocar um disparador de inserção na tabela Order Details que verificasse a coluna UnitsInStock (Unidades em estoque) para esse item na tabela Products. O disparador determinaria que quando o valor UnitsInStock fosse menor do que 10, a quantidade máxima do pedido seria três itens. Este tipo de verificação faz referência a colunas em outras tabelas. Fazer referências a colunas em outras tabelas não é permitido com uma restrição CHECK. Você pode usar disparadores para impor uma integridade referencial complexa das seguintes maneiras: _ Executando uma ação ou efetuando atualizações ou exclusões em cascata. A integridade referencial pode ser definida através do uso das restrições FOREIGN KEY e REFERENCE com a instrução CREATE TABLE. Os disparadores são úteis para garantir ações apropriadas quando exclusões ou atualizações em cascata devem ser efetuadas. Se existirem restrições na tabela de disparadores, elas serão verificadas antes da execução do procedimento. Se as restrições forem violadas, o procedimento não será executado.

- Criando disparadores de vários registros.

Quando mais de um registro é inserido, atualizado ou excluído, você deve escrever um disparador para manipular vários registros.

- Impondo a integridade referencial entre bancos de dados.

Definir mensagens de erro personalizadas

Ocasionalmente, sua implementação poderá tirar vantagem de mensagens de erro personalizadas que indicam o status de uma ação. Usando disparadores, você poderá chamar mensagens de erro personalizadas predefinidas ou dinâmicas quando ocorrerem certas condições durante a execução de um procedimento.

Manter dados desnormalizados

Os disparadores podem ser usados para manter a integridade de dados de baixo nível em ambientes de bancos de dados desnormalizados. A manutenção de dados desnormalizados é diferente da manutenção em cascata, visto que essa geralmente se refere à manutenção de relacionamentos entre valores de chaves primárias e externas. Em geral, os dados desnormalizados são valores de dados projetados, derivados ou redundantes. Você deverá usar um disparador se:

A integridade referencial exigir algo diferente de uma correspondência exata, como, por exemplo, a manutenção de dados derivados (vendas do ano até a data) ou a sinalização de colunas (S ou N para não para indicar se um produto está disponível).

Você precisar de mensagens personalizadas e mensagens de erro complexas.

Os dados redundantes e derivados geralmente requerem o uso de disparadores.

Comparar os estados anteriores e posteriores dos dados que estão sendo modificados

Os disparadores permitem fazer referência às alterações efetuadas pela instrução INSERT, UPDATE ou DELETE nos dados. Dessa maneira, é possível fazer referência aos registros que estão sendo afetados pelas instruções de modificação contidas no procedimento.

As restrições, regras e padrões podem comunicar erros somente através de mensagens de erro padronizadas do sistema. Se o seu aplicativo exigir (ou puder tirar vantagem de) mensagens personalizadas e o tratamento de erros mais complexos, você deverá usar um disparador.

5.3 Considerações sobre o uso de disparadores

Considere os fatos e diretrizes a seguir ao trabalhar com disparadores:

A maioria dos disparadores é reativa; as restrições e o disparador INSTEAD OF são pró-ativos. Os disparadores são executados após a execução de uma instrução INSERT, UPDATE ou DELETE na tabela em que o procedimento é definido. Por exemplo, uma instrução UPDATE atualiza um registro de uma tabela e, depois, o disparador dessa tabela é executado automaticamente. As restrições são verificadas antes da execução de uma instrução INSERT, UPDATE ou DELETE.

As restrições são verificadas primeiro. Se existirem restrições na tabela de disparadores, elas serão verificadas antes da execução do procedimento. Se as restrições forem violadas, o disparador não será executado.

As tabelas podem conter vários disparadores para uma ação. O Microsoft® SQL Server permite o aninhamento de vários disparadores em uma única tabela. Uma tabela poderá conter vários disparadores definidos para ela. Cada procedimento poderá ser definido para uma única ou várias ações.

Os proprietários das tabelas podem designar o primeiro e o último disparador a ser acionado. Quando vários disparadores são colocados em uma tabela, o proprietário da tabela pode usar o procedimento armazenado do sistema sp_settriggerorder para especificar os primeiros disparadores a serem acionados. A ordem do acionamento dos disparadores restantes não pode ser definida.

Os proprietários das tabelas devem ter permissão para executar todas as instruções definidas pelo disparador. Somente o proprietário da tabela, os participantes do cargo fixo do servidor sysadmin e participantes dos cargos fixos do servidor db_owner e db_ddladmin podem criar e descartar disparadores para essa tabela. Essas permissões não podem ser transferidas. Além disso, o criador do disparador também deve ter permissão para executar todas as instruções em todas as tabelas afetadas. Se forem negadas permissões para qualquer parte das instruções Transact-SQL contidas no disparador, a transação inteira será revertida.

Os proprietários das tabelas não podem criar disparadores AFTER em views ou tabelas temporárias. No entanto, eles podem fazer referência a views e tabelas temporárias.

Os proprietários das tabelas podem criar disparadores INSTEAD OF em views e tabelas; dessa forma, os disparadores INSTEAD OF estendem, em grande parte, os tipos de atualizações para as quais uma view pode oferecer suporte.

Os disparadores não devem retornar conjuntos de resultados. Assim como os procedimentos armazenados, os disparadores contêm instruções Transact-SQL e podem conter instruções que retornam um conjunto de resultados. No entanto, a inclusão de instruções que retornem valores em disparadores não é recomendada, pois os usuários ou desenvolvedores não esperam ver nenhum conjunto de resultados quando uma instrução UPDATE, INSERT ou DELETE é executada.

Os disparadores podem manipular ações com vários registros. Uma ação INSERT, UPDATE ou DELETE que chama um disparador pode afetar vários registros. Você poderá:

Processar todos os registros em conjunto; nesse caso, todos os registros afetados devem satisfazer aos critérios do disparador para que qualquer ação ocorra.

Permitir ações condicionais. Por exemplo, se desejar excluir três clientes da tabela Customers (Clientes), você poderá definir um disparador para garantir que não existam pedidos ativos ou faturas pendentes para cada cliente excluído. Se um dos três clientes tiver uma fatura pendente, ele não será excluído, mas os clientes qualificados serão.

Para determinar se vários registros são afetados, use a função do sistema

@@ROWCOUNT.

5.4 Criando disparadores

Crie disparadores usando a instrução CREATE TRIGGER. A instrução especifica a tabela em que um disparador é definido, os eventos para os quais ele é executado e as suas instruções específicas.

```
CREATE TRIGGER [proprietário.] nome_do_disparador
ON [proprietário.] nome_da_tabela
[WITH ENCRYPTION]
{{FOR | AFTER | INSTEAD OF} {INSERT | UPDATE | DELETE}
AS
[IF UPDATE (nome_da_coluna)...]
[{AND | OR} UPDATE (nome_da_coluna)...]
instruções_sql}
```

Quando uma ação FOR UPDATE é especificada, a cláusula IF UPDATE (nome_da_coluna) pode ser usada para enfocar a ação em uma coluna específica que é atualizada. Tanto FOR quanto AFTER têm sintaxe igual criando o mesmo tipo de disparador, que é acionado depois da ação (INSERT, UPDATE ou DELETE).

Os disparadores INSTEAD OF cancelam a ação do disparador e executam uma nova ação. Quando você cria um disparador, as informações sobre o mesmo são inseridas nas tabelas do sistema sysobjects (objetos do sistema) e syscomments (comentários do sistema). Se um disparador for criado com o mesmo nome que outro existente, o novo procedimento sobrescreverá o original. O SQL Server não oferece suporte para a adição de disparadores definidos pelo usuário nas tabelas do sistema; portanto, não será possível criar disparadores nessas tabelas.

Permissões apropriadas

Os proprietários das tabelas, bem como os participantes dos cargos proprietário do banco de dados (db_owner) e administradores do sistema (sysadmin), têm permissão para criar um disparador. Para evitar situações em que o proprietário de uma view e o proprietário das tabelas subjacentes sejam diferentes, é recomendado que o usuário dbo seja o proprietário de todos os objetos de um banco de dados. Como um usuário pode ser participante de vários cargos, especifique sempre o usuário dbo como o nome do proprietário ao criar o objeto. Caso contrário, o objeto será criado tendo o seu nome de usuário como o proprietário.

Não pode conter certas instruções

O SQL Server não permite que as instruções a seguir sejam usadas na definição de um disparador:

_ ALTER DATABASE

_ CREATE DATABASE

_ DISK INIT

_ DISK RESIZE

_ DROP DATABASE

_ LOAD DATABASE

_ LOAD LOG

_ RECONFIGURE

_ RESTORE DATABASE

_ RESTORE LOG

Para determinar as tabelas com disparadores, execute o procedimento armazenado do sistema sp_depends <nome_da_tabela>. Para exibir a definição de um disparador, execute o procedimento armazenado do sistema sp_helptext <nome_do_disparador>. Para determinar os disparadores existentes em uma tabela específica e suas ações, execute o procedimento armazenado do sistema sp_helptrigger <nome_da_tabela>.

O exemplo a seguir cria um disparador na tabela Employees que impede que os usuários excluam mais de um funcionário por vez. O disparador é acionado toda vez que um registro ou grupo de registros é excluído da tabela. O disparador verifica o número de registros que está sendo excluído consultando a tabela Deleted. Se mais de um registro estiver sendo excluído, o disparador retorna uma mensagem de erro personalizada e reverte a transação.

```
Use Northwind
GO
CREATE TRIGGER Empl_Delete ON Employees
FOR DELETE
AS
IF (SELECT COUNT(*) FROM Deleted) > 1
BEGIN
RAISERROR(
'You cannot delete more than one employee at a time.',
16, 1)
ROLLBACK TRANSACTION
END
```

A instrução DELETE a seguir aciona o disparador e impede a transação.

```
DELETE FROM Employees WHERE EmployeeID > 6
```

A instrução DELETE a seguir aciona o disparador e permite a transação.

```
DELETE FROM Employees WHERE EmployeeID = 6
```

Alterando um disparador

Se for necessário alterar a definição de um disparador existente, você poderá alterá-lo sem precisar descartá-lo.

A definição alterada substitui a definição do disparador existente pela nova definição. A ação do procedimento também é alterada. Por exemplo, se você criar um disparador para INSERT e, depois, alterar a ação para UPDATE, o disparador alterado será executado sempre que a tabela for atualizada. Usando a resolução de nomes com atraso, você poderá fazer referência a tabelas e views em um disparador que ainda não existe. Se o objeto não existir quando um disparador for criado, você receberá uma mensagem de aviso, e o SQL Server atualizará a definição do procedimento imediatamente.

```
ALTER TRIGGER nome_do_disparador
ON tabela
[WITH ENCRYPTION]
{{FOR {[,] [DELETE] [,] [UPDATE] [,][INSERT]}
[NOT FOR REPLICATION]
AS
instrução_sql [...n] }
|
{FOR {[,] [INSERT] [,] [UPDATE]}
[NOT FOR REPLICATION]
AS
IF UPDATE (coluna)
[{AND | OR} UPDATE (coluna) [,...n]]
instrução_sql [...n] }
}
```

Este exemplo altera o disparador excluído criado no exemplo anterior. O conteúdo do novo disparador é fornecido, o que altera o limite de exclusão de um registro para seis registros.

```
Use Northwind
GO
ALTER TRIGGER Empl_Delete ON Employees
FOR DELETE
```

```
AS
IF (SELECT COUNT(*) FROM Deleted) > 6
BEGIN
RAISERROR(
'You cannot delete more than six employees at a time.',
16, 1)
ROLLBACK TRANSACTION
END
```

5.5 Desativando ou ativando um disparador

Você pode desativar ou ativar um disparador específico ou todos os disparadores de uma tabela. Quando um disparador é desativado, ele permanece definido para a tabela; no entanto, quando uma instrução INSERT, UPDATE ou DELETE é executada na tabela, as ações do procedimento não serão executadas até que ele seja reativado.

Você poderá ativar ou desativar disparadores na instrução ALTER TABLE.

```
ALTER TABLE tabela
{ENABLE | DISABLE} TRIGGER
{ALL | nome_do_disparador [,…n]}
```

Descartando um disparador

Você pode remover um disparador descartando-o. Os disparadores são descartados automaticamente sempre que as tabelas associadas são descartadas. A permissão para descartar um disparador assume, por padrão, o proprietário da tabela e é intransferível. No entanto, os participantes dos cargos administradores do sistema (sysadmin) e proprietário do banco de dados (db_owner) podem descartar qualquer objeto especificando o proprietário na instrução DROP TRIGGER.

```
DROP TRIGGER nome_do_disparador
```

5.6 Como funcionam os disparadores

Ao criar disparadores, é importante compreender como eles funcionam. Esta seção aborda os disparadores INSERT, DELETE, UPDATE, INSTEAD OF, aninhados e recursivos.

Como funciona um disparador INSERT

Você pode definir um disparador para que seja executado sempre que uma instrução INSERT inserir dados em uma tabela. Quando um disparador INSERT é acionado, novos registros são adicionados à tabela de disparadores e à tabela inserted (inserido). A tabela inserted é uma tabela lógica que armazena uma cópia dos registros que foram inseridos. Essa tabela contém a atividade de inserção registrada no log da instrução INSERT. Ela permite que você faça referência aos dados registrados no log da instrução INSERT que iniciou a operação. O disparador pode examinar a tabela inserted para determinar se ou como as suas ações devem ser executadas. Os registros da tabela inserted sempre são cópias de um ou mais registros da tabela de procedimentos armazenados. Toda a atividade de modificação de dados (instruções INSERT, UPDATE e DELETE) é registrada no log, mas as informações do log de transações não podem ser lidas. No entanto, a tabela inserted permite que você faça referência às alterações registradas no log que foram ocasionadas pela instrução INSERT. Depois, você poderá comparar as alterações com os dados inseridos para verificá-las ou tomar outra ação. Também é possível fazer referência aos dados inseridos sem precisar armazenar as informações em variáveis.

O disparador neste exemplo foi criado para atualizar uma coluna (UnitsInStock) na tabela Products sempre que um pedido é solicitado (sempre que um registro é inserido na tabela Order Details). O novo valor é definido como o valor anterior menos a quantidade solicitada.

```
USE Northwind
GO
CREATE TRIGGER OrdDet_Insert
ON [Order Details]
FOR INSERT
AS
UPDATE P SET
UnitsInStock = (P.UnitsInStock - I.Quantity)
FROM Products AS P INNER JOIN Inserted AS I
ON P.ProductID = I.ProductID
```

Como funciona um disparador DELETE

Quando um disparador DELETE é acionado, os registros excluídos da tabela afetada são colocados em uma tabela especial chamada deleted (excluído). Essa é uma tabela lógica que armazena uma cópia dos registros que foram excluídos. Ela permite que você faça referência aos dados registrados no log da instrução DELETE que iniciou a operação. Considere os fatos a seguir ao usar o disparador DELETE:

Quando um registro é acrescentado à tabela deleted, ele deixa de existir na tabela do banco de dados; portanto, a tabela deleted e as tabelas do banco de

dados não apresentam registros em comum.

É alocado espaço da memória para criar a tabela deleted. Essa tabela está sempre no cache.

Um disparador definido para uma ação DELETE não é executado para a instrução TRUNCATE TABLE porque TRUNCATE TABLE não é registrada no log.

O disparador neste exemplo foi criado para atualizar uma coluna Discontinued na tabela Products sempre que uma categoria é excluída (sempre que um registro é excluído da tabela Categories). Todos os produtos afetados são marcados com 1, indicando que eles são descontinuados.

```
USE Northwind
GO
CREATE TRIGGER Category_Delete
ON Categories
FOR DELETE
AS
UPDATE P SET Discontinued = 1
FROM Products AS P INNER JOIN deleted AS d
ON P.CategoryID = d.CategoryID
```

Como funciona um disparador UPDATE

Uma instrução UPDATE pode ser analisada em duas etapas: a etapa DELETE que captura a imagem anterior dos dados e a etapa INSERT que captura a imagem posterior dos dados. Quando uma instrução UPDATE é executada em uma tabela que contém um disparador definido, os registros originais (imagem anterior) são movidos para a tabela deleted e os registros atualizados (imagem posterior) são inseridos na tabela inserted. O disparador pode examinar as tabelas deleted e inserted, bem como a tabela atualizada, para determinar se vários registros foram atualizados e como as ações do procedimento devem ser executadas. Você pode definir um disparador para monitorar as atualizações de dados em uma coluna específica usando a instrução IF UPDATE. Isso permite que o disparador isole facilmente a atividade relativa a uma coluna específica. Ao detectar que a coluna foi atualizada, ele poderá tomar a ação adequada, como, or exemplo, gerar uma mensagem de erro inormando que a coluna não pode er atualizada ou processar uma série de instruções com base no valor da coluna atualizada recentemente.

Este exemplo impede que um usuário modifique a coluna a EmployeeID

(Identificação do funcionário) na tabela Employees.

```
USE Northwind
GO
CREATE PROCEDURE Employee_Update
ON Employees
FOR UPDATE
AS
IF UPDATE (EmployeeID)
BEGIN TRANSACTION
RAISERROR ('Transaction cannot be processed.\
***** Employee ID number cannot be modified.', 10, 1)
ROLLBACK TRANSACTION
```

Como funciona um disparador INSTEAD OF

Você pode especificar um disparador INSTEAD OF em tabelas e views. Esse disparador é executado em vez da ação do disparador original. Os disparadores INSTEAD OF aumentam a variedade de tipos de atualizações que você pode executar em uma view. Cada tabela ou view é limitada a um disparador INSTEAD OF para cada ação de disparo (INSERT, UPDATE ou DELETE). Você não pode criar um disparador INSTEAD OF em views que têm WITH CHECK OPTION definido.

Este exemplo cria uma tabela com clientes na Alemanha e uma tabela com clientes no México. Um disparador INSTEAD OF colocado na view redireciona as atualizações para a tabela subjacente apropriada. A inserção na tabela CustomersGer (Clientes da Alemanha) ocorre em vez de (instead of) inserção na view.

Crie duas tabelas com dados do cliente

```
USE ClassNorthwind
SELECT * INTO CustomersGer FROM Customers WHERE
Customers.Country = 'Germany'
SELECT * INTO CustomersMex FROM Customers WHERE
Customers.Country = 'Mexico'
GO
```

Crie uma view sobre esses dados

```
CREATE VIEW CustomersView AS
SELECT * FROM CustomersGer
UNION
SELECT * FROM CustomersMex
GO
```

Crie um disparador INSTEAD OF sobre a view

```
CREATE TRIGGER Customers_Update2
ON CustomersView
INSTEAD OF UPDATE AS
DECLARE @Country nvarchar(15)
SET @Country = (SELECT Country FROM Inserted)
IF @Country = 'Germany'
BEGIN
UPDATE CustomersGer
SET CustomersGer.Phone = Inserted.Phone
FROM CustomersGer JOIN Inserted
ON CustomersGer.CustomerID = Inserted.CustomerID
END
ELSE
IF @Country = 'Mexico'
BEGIN
UPDATE CustomersMex
SET CustomersMex.Phone = Inserted.Phone
FROM CustomersMex JOIN Inserted
ON CustomersMex.CustomerID = Inserted.CustomerID
END
```

Teste o disparador atualizando a view

```
UPDATE CustomersView SET Phone = '030-007xxxx'
WHERE CustomerID = 'ALFKI'
SELECT CustomerID, Phone FROM CustomersView
WHERE CustomerID = 'ALFKI'
```

```
SELECT CustomerID, Phone FROM CustomersGer
WHERE CustomerID = 'ALFKI'
```

5.7 Como funcionam os disparadores aninhados

Qualquer disparador pode conter uma instrução UPDATE, INSERT ou DELETE que afete outra tabela. Quando o aninhamento está ativado, um disparador que altera uma tabela poderá ativar um segundo procedimento que, por sua vez, poderá ativar um terceiro e assim por diante. O aninhamento é ativado durante a instalação, mas você poderá desativá-lo e reativá-lo usando o rocedimento armazenado do sistema sp_configure

Os disparadores podem ser aninhados em até 32 níveis. Se um disparador em ma corrente aninhada inicia um loop infinito, o nível de aninhamento é ultrapassado. Dessa forma, o disparador termina e reverte a transação. Os disparadores aninhados podem ser usados para executar funções, como o armazenamento de uma cópia de backup dos registros que foram afetados por um disparador anterior. Considere os fatos a seguir ao usar disparadores aninhados:

Por padrão, a opção de configuração de disparadores aninhados está ativada.

Um disparador aninhado não será acionado duas vezes na mesma transação do disparador; um disparador não se chama em resposta a uma segunda atualização na mesma tabela no disparador. Por exemplo, se um disparador modificar uma tabela que, por sua vez, modifique a tabela do disparador original, o disparador não é acionado novamente.

Como um disparador é uma transação, uma falha em qualquer nível de um conjunto de disparadores aninhados cancelará a transação inteira, e todas as modificações de dados serão revertidas.

Verificando o nível de aninhamento

Toda vez que um disparador aninhado é acionado, o nível de aninhamento é incrementado. O SQL Server oferece suporte para até 32 níveis de aninhamento, mas você poderá limitar os níveis para evitar que o nível máximo de aninhamento seja ultrapassado. Você pode usar a função @@NESTLEVEL para ver os níveis atuais de aninhamento.

Determinando se o aninhamento deve ou não ser usado

O aninhamento é um recurso poderoso que pode ser usado para manter a integridade dos dados de um banco de dados. No entanto, ocasionalmente, talvez você deseje desativá-lo. Se o aninhamento for desativado, um disparador que modifica outra tabela não chamará nenhum dos disparadores da segunda tabela. Use a instrução a seguir para desativar o aninhamento: sp_configure ‘nested triggers’, 0 Você pode decidir desativar o aninhamento porque:

Os disparadores aninhados requerem um projeto complexo e bem planejado. As alterações em cascata podem modificar dados que você não tinha intenção de alterar.

Uma modificação de dados efetuada em qualquer ponto de uma série de disparadores aninhados aciona toda a série de procedimentos. Embora esse recurso ofereça uma poderosa proteção para seus dados, ele poderá ser um problema caso as suas tabelas devam ser atualizadas em uma ordem específica. Você pode criar a mesma funcionalidade com ou sem o recurso de aninhamento; no entanto, o projeto de seus disparadores será bastante diferente.

Ao criar disparadores aninhados, cada procedimento deverá iniciar somente a próxima modificação de dados . o projeto deverá ser modular. Ao criar disparadores não aninhados, cada procedimento deverá iniciar todas as modificações de dados que você deseja que ele faça.

Este exemplo mostra como a colocação de um pedido faz com que o disparador OrDe_Update seja executado. Este disparador executa uma instrução UPDATE na coluna UnitsInStock da tabela Products. Quando a atualização ocorre, ela aciona o disparador Products_Update e compara o novo valor da ação no inventário, mais a ação no pedido, com o nível de reordenação. Se a ação no inventário mais a ação no pedido cair abaixo do nível de reordenação, uma mensagem é enviada alertando ao comprador que compre mais ações.

```
USE Northwind
GO
CREATE TRIGGER Products_Update
ON Products
FOR UPDATE
AS
IF UPDATE (UnitsInStock)
IF (Products.UnitsInStock + Products.UnitsOnOrder) <
Products.ReorderLevel
BEGIN
-- Envie uma mensagem ao departamento de compras
END
```

Disparadores recursivos

Qualquer disparador pode conter uma instrução UPDATE, INSERT ou DELETE que afete a mesma ou outra tabela. Com a opção de disparador recursivo ativada, um procedimento que altere dados em uma tabela poderá ativar a si mesmo novamente, em uma execução recursiva. A opção de disparador recursivo está desativada por padrão quando um banco de dados é criado, mas você pode ativá-la usando a instrução para alterar o banco de dados.

Ativando um disparador de modo recursivo

Use a instrução a seguir para ativar disparadores recursivos: ALTER DATABASE ClassNorthwind SET RECURSIVE_TRIGGERS ON sp_dboption nome_do_banco_de_dados, 'recursive triggers', True Use o procedimento armazenado do sistema sp_settriggerorder para especificar um disparador que seja acionado como o primeiro disparador AFTER, ou o último disparador AFTER. Não há uma ordem fixa para a execução de vários disparadores definidos para um determinado evento. Cada disparador deve ser independente. Se a opção de disparador aninhado estiver desativada, a opção de disparador recursivo também estará, independente da configuração de disparador recursivo do banco de dados.

As tabelas inserted e deleted referentes a um determinado disparador contêm os registros que correspondem somente à última instrução UPDATE, INSERT ou DELETE que chamou o disparador. Até 32 níveis de recursão de disparadores são permitidos. Se qualquer disparador de um loop recursivo provocar um loop infinito, o nível de aninhamento será ultrapassado, o procedimento será encerrado e a transação será revertida.

Tipos de disparadores recursivos

Há dois tipos diferentes de recursão:

- Recursão direta, que ocorre quando um disparador aciona e executa uma ação que faz com que o mesmo disparador seja acionado novamente.
  Por exemplo, um aplicativo atualiza a tabela T1, fazendo com que o disparador Trig1 seja acionado. Trig1 atualiza a tabela T1 novamente, fazendo com que o disparador Trig1 seja acionado novamente.
- A recursão indireta, que ocorre quando um disparador aciona e executa uma ação que faz com que um disparador em outra tabela seja acionado, ocasionando subseqüentemente uma atualização da tabela original. Isso, por sua vez, faz com que o disparador original seja acionado novamente.
  Por exemplo, um aplicativo atualiza a tabela T2, fazendo com que o disparador Trig2 seja acionado. Trig2 atualiza a tabela T3 novamente, fazendo com que o disparador Trig3 seja acionado novamente. Trig3, por sua vez, atualiza a tabela T2, fazendo com que Trig2 seja acionado novamente.

Determinando se disparadores recursivos devem ou não ser usados

Os disparadores recursivos são um recurso complexo que você pode usar para solucionar relacionamentos complexos, como relacionamentos de autoreferência (também conhecidos como fechamentos transitivos). Nessas situações especiais, você poderá desejar ativar disparadores recursivos. Os disparadores recursivos podem ser úteis quando você tiver que manter:

O número de colunas de relatórios na tabela employee quando a tabela contém uma coluna employee ID (identificação do funcionário) e outra manager ID (identificação do gerente).

Por exemplo, suponha que dois disparadores de atualização, tr_update_employee e tr_update_manager, sejam definidos na tabela employee. O procedimento tr_update_employee atualiza a tabela employee.

Uma instrução UPDATE aciona os disparadores tr_update_employee e tr_update_manager uma vez. Além disso, a execução de tr_update_employee aciona a execução de tr_update_employee novamente (de modo recursivo) e de tr_update_manager.

Um gráfico para dados de planejamento de produção em que exista uma hierarquia implícita de planejamento.

Um sistema de controle de montagem no qual partes menores sejam controladas por suas partes principais.

Considere as diretrizes a seguir antes de usar disparadores recursivos:

Os disparadores recursivos são complexos e precisam ser bem projetados e completamente testados. Eles requerem um código de lógica de loop controlado (verificação de término). Caso contrário, o limite de aninhamento de 32 níveis será excedido.

Uma modificação de dados efetuada em qualquer ponto poderá acionar a série de disparadores. Embora permita o processamento de relacionamentos complexos, isso poderá ser um problema se suas tabelas tiverem de ser atualizadas em uma ordem específica.

Você poderá criar uma funcionalidade semelhante sem o recurso de disparadores recursivos; no entanto, o projeto de seus disparadores será bastante diferente. Ao criar disparadores recursivos, cada procedimento deverá conter uma verificação condicional para interromper o processamento recursivo quando a condição se tornar falsa. Ao criar disparadores não recursivos, cada procedimento deverá conter as verificações e estruturas completas de loop de programação.

5.8 Exemplos de disparadores

Os disparadores forçam a integridade dos dados e as regras de negócios. Algumas das ações que os disparadores executam podem ser realizadas através do uso de restrições e, no caso de algumas ações, você deverá considerar as restrições primeiro. No entanto, os disparadores são necessários para impor os vários graus de desnormalização e para impor regras de negócios complexas.

Impondo a integridade dos dados

Os disparadores podem ser usados para manter a integridade dos dados efetuando alterações em cascata nas tabelas relacionadas em todo o banco de dados.

O exemplo a seguir mostrar como um disparador mantém a integridade dos dados em uma tabela BackOrders (Pedidos retroativos). O disparador BackOrderList_delete mantém a lista de produtos na tabela BackOrders.

Quando os produtos são recebidos, o disparador UPDATE na tabela Products exclui registros de uma tabela BackOrders.

```
CREATE TRIGGER BackOrderList_Delete
ON Products FOR UPDATE
AS
IF (SELECT BO.ProductID FROM BackOrders AS BO JOIN
Inserted AS I ON BO.ProductID = I.Product_ID
) > 0
BEGIN
DELETE BO FROM BackOrders AS BO
INNER JOIN Inserted AS I
ON BO.ProductID = I.ProductID
END
```

Impondo regras de negócios

Você pode usar disparadores para impor regras de negócios que sejam muito complexas para a restrição CHECK. Isso inclui verificar o status dos registros em outras tabelas.

Por exemplo, você poderá desejar garantir que as multas pendentes dos participantes sejam pagas antes que eles possam acabar com a participação. Este exemplo cria um disparador que determina se um produto tem um histórico de pedidos. Se ele tiver, DELETE é revertido e o disparador retorna uma mensagem de erro personalizada.

```
Use Northwind
GO
CREATE TRIGGER Product_Delete
ON Products FOR DELETE
```

```
AS
IF (Select Count (*)
FROM [Order Details] INNER JOIN deleted
ON [Order Details].ProductID = Deleted.ProductID ) > 0
BEGIN
RAISERROR('Transaction cannot be processed. \
This product has order history.', 16, 1)
ROLLBACK TRANSACTION
END
```

5.9 Considerações sobre o desempenho

Considere estas questões relacionadas ao desempenho ao usar disparadores.

Os disparadores funcionam com rapidez, pois as tabelas Inserted e Deleted encontram-se no cache. As tabelas Inserted e Deleted estão sempre na memória, em vez de no disco, pois são tabelas lógicas e geralmente são muito pequenas.

O número de tabelas referenciadas e o número de registros afetados determinam o tempo de execução.

O tempo gasto para chamar um disparador é mínimo. A maior parte do tempo de execução resulta da referência a outras tabelas (que podem estar na memória ou em disco) e da modificação dos dados, se a definição do disparador determinar isso.

As ações contidas nos disparadores consistem em uma parte implícita de uma transação.

Depois que um disparador é definido, a ação do usuário (instrução INSERT, UPDATE ou DELETE) na tabela que executa o procedimento é sempre uma parte implícita de uma transação, junto com o disparador propriamente dito. Se uma instrução ROLLBACK TRANSACTION for encontrada, a

transação inteira será revertida. Se existirem quaisquer instruções no script do disparador após a instrução ROLLBACK TRANSACTION, elas serão executadas. Portanto, poderá ser necessário usar uma cláusula RETURN em uma instrução IF para evitar o processamento de outras instruções.

Laboratório

Crie os seguintes disparadores:

1. Ao incluir um novo produto na compra verifique se existe estoque suficiente para realizar, caso contrário emita um aviso
2. Ao incluir um novo cliente emita um aviso que mostre o vendedor que realizou mais vendas para a região do mesmo
3. Ao atualizar o valor de algum produto verifique a diferença entre os valores e exiba a diferença entre as vendas realizadas com o antigo valor e o novo valor
4. Ao deletar algum cliente verifique se o mesmo tem vendas no ultimo mês e em caso positivo crie uma tabela idêntica a dos clientes com o mesmo ou se ela já existir apenas insira o referido
5. Ao deletar um fornecedor verificar quando foi a ultima venda o mesmo e exibir a soma de todas as vendas realizadas nesta data

Unidade 6 - Criando Índices

6.1 Criando e descartando índices

Use a instrução CREATE INDEX para criar índices e a instrução DROP INDEX para removê-los.

Você deverá ser o proprietário da tabela para executar qualquer uma das duas instruções em um banco de dados.

Usando a instrução CREATE INDEX

Use a instrução CREATE INDEX para criar índices. Você também pode usar o Create Index Wizard (Assistente para criação de índices) no SQL Server Enterprise Manager (Gerenciador corporativo do SQL Server). Ao criar um índice em uma ou mais colunas de uma tabela, considere os fatos e as diretrizes a seguir:

O SQL Server cria índices automaticamente quando uma restrição PRIMARY KEY ou UNIQUE é criada em uma tabela. É preferível definir uma restrição PRIMARY KEY ou UNIQUE a criar índices padrão.

Você deve ser o proprietário da tabela para executar a instrução CREATE INDEX.

É possível criar índices em views.

O SQL Server armazena as informações sobre índices na tabela do sistema sysindexes.

Antes de criar um índice em uma coluna, verifique se ela já contém índices.

Mantenha seus índices pequenos definindo-os em colunas pequenas. Em geral, índices menores são mais eficientes do que os que apresentam valores de chaves maiores.

Selecione as colunas com base na exclusividade, de modo que cada valor de chave identifique um pequeno número de registros.

Quando você cria um índice de agrupamento, todos os índices sem agrupamento existentes são recriados.

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
INDEX nome_do_índice ON { tabela | view } ( coluna [ ASC | DESC ]
[ ,...n ] )
[WITH
[PAD_INDEX ]
[[,] FILLFACTOR = fator_de_preenchimento ]
[[,] IGNORE_DUP_KEY ]
[[,] DROP_EXISTING ]
[[,] STATISTICS_NORECOMPUTE ]
[[,] SORT_IN_TEMPDB ]
]
[ON grupo_de_arquivos ]
```

Este exemplo cria um índice de agrupamento na coluna LastName (Sobrenome) da tabela Employees (Funcionários).

```
USE Northwind
CREATE CLUSTERED INDEX CL_lastname
ON employees(lastname)
```

Usando a instrução DROP INDEX

Use a instrução DROP INDEX para remover um índice de uma tabela. Ao descartar um índice, considere os fatos a seguir:

O SQL Server recupera o espaço em disco ocupado pelo índice quando você executa a instrução DROP INDEX.

Você não pode usar a instrução DROP INDEX nos índices criados pelas restrições PRIMARY KEY ou UNIQUE. Será necessário descartar a restrição para descartar esses índices.

Quando você descartar uma tabela, todos os índices da tabela também serão descartados.

Quando você descartar um índice de agrupamento, todos os índices sem agrupamento da tabela serão recriados automaticamente.

Você deverá estar no banco de dados em que o índice reside para descartá-lo.

Não é possível usar a restrição DROP INDEX nas tabelas do sistema. DROP INDEX ‘índice.tabela | índice.view’ [, ...n]

Este exemplo descarta o índice cl_lastname da tabela Employees (Funcionários).

```
USE Northwind
DROP INDEX employees.CL_lastname
```

6.2 Criando índices exclusivos

Os índices exclusivos garantem que todos os dados da coluna indexada sejam exclusivos e não contenham valores duplicados. Os índices exclusivos garantem que os dados das colunas indexadas sejam exclusivos. Se a tabela tiver uma restrição PRIMARY KEY ou UNIQUE, o SQL Server criará um índice exclusivo automaticamente quando você executar a instrução CREATE TABLE ou ALTER TABLE.

Garantindo que os dados das colunas indexadas sejam exclusivos

Crie um índice exclusivo para índices de agrupamento ou sem agrupamento quando os próprios dados forem inerentemente exclusivos. No entanto, se for necessário impor a exclusividade, crie restrições PRIMARY KEY ou UNIQUE na coluna em vez de criar um índice exclusivo. Ao criar um índice exclusivo, considere os fatos e as diretrizes a seguir:

O SQL Server cria automaticamente índices exclusivos nas colunas de uma tabela definida com restrições PRIMARY KEY ou UNIQUE.

Se a tabela contiver dados, o SQL Server verificará se existem valores duplicados quando você criar o índice.

O SQL Server verificará se existem valores duplicados toda vez que você usar a instrução INSERT ou UPDATE. Se existirem valores de chaves duplicados, ele cancelará sua instrução e retornará uma mensagem de erro com o primeiro valor duplicado.

Certifique-se de que cada registro tenha um valor exclusivo — dois registros não podem ter o mesmo número de identificação se um índice exclusivo for criado nessa coluna. Essa regra garante que cada entidade seja identificada de maneira exclusiva.

Crie índices exclusivos somente nas colunas em que a integridade da entidade possa ser imposta. Por exemplo, você não criaria um índice exclusivo na coluna LastName da tabela Employees porque alguns funcionários podem ter o mesmo sobrenome.

Este exemplo cria um índice sem agrupamento exclusivo chamado U-CustID na tabela Customers (Clientes). O índice é criado na coluna CustomerID (Identificação do cliente). O valor dessa coluna deve ser exclusivo para cada registro da tabela.

```
USE Northwind
CREATE UNIQUE NONCLUSTERED INDEX U_CustID ON customers(CustomerID)
```

6.3 Localizando todos os valores duplicados em uma coluna

Se existirem valores de chaves duplicados quando você criar um índice exclusivo, a instrução CREATE INDEX falhará. O SQL Server retornará uma mensagem de erro com o primeiro valor duplicado, mas também poderão existir outros valores duplicados. Use o script de exemplo a seguir em qualquer tabela para localizar todos os valores duplicados em uma coluna. Substitua o texto em itálico pelas informações específicas à sua consulta.

```
SELECT coluna_de_índice, COUNT (coluna_de_índice)
FROM nome_da_tabela
GROUP BY coluna_de_índice
HAVING COUNT(coluna_de_índice)>1 ORDER BY coluna_de_índice
```

Este exemplo determina se existe uma identificação de cliente duplicada na coluna CustomerID da tabela Customers. Se existir, o SQL Server retornará a identificação do cliente e o número de entradas duplicadas no conjunto de resultados.

```
SELECT CustomerID, COUNT(CustomerID) AS '# of Duplicates'
FROM Northwind.dbo.Customers
GROUP BY CustomerID
HAVING COUNT(CustomerID)>1
ORDER BY CustomerID
```

6.4 Criando índices compostos

Os índices compostos especificam mais de uma coluna como o valor de chave.

Crie índices compostos:

Quando for mais eficiente pesquisar duas ou mais colunas como uma chave.

Se as consultas fizerem referência somente às colunas do índice.

Por exemplo, um catálogo de telefones é um bom exemplo em que seria útil usar um índice composto. O catálogo é organizado por sobrenomes. Dentro dos sobrenomes, ele é organizado por nomes, pois geralmente existem entradas com o mesmo sobrenome.

Ao criar um índice composto, considere os fatos e as diretrizes a seguir:

Você pode combinar até 16 colunas em um único índice composto. O somatório dos comprimentos dos dados das colunas que constituem o índice composto não pode ultrapassar 900 bytes.

Todas as colunas de um índice composto devem fazer parte da mesma tabela, exceto quando o índice é criado em uma view.

Defina a coluna mais exclusiva primeiro. A primeira coluna definida na instrução CREATE INDEX é considerada a de ordem mais alta.

A cláusula WHERE de uma consulta deve fazer referência à primeira coluna do índice composto para que o otimizador de consultas use esse índice.

Um índice composto em (coluna1, coluna2) não é igual a um índice composto em (coluna2, coluna1) — cada um apresenta uma ordem diferente de colunas. A coluna que contém dados mais seletivos ou que retornaria a porcentagem mais baixa de registros geralmente determina a ordem das colunas.

Os índices compostos são úteis para tabelas com várias chaves de colunas.

Use índices compostos para aumentar o desempenho das consultas e reduzir o número de índices criados em uma tabela. Em geral, vários índices nas mesmas colunas não são úteis.

Este exemplo cria um índice sem agrupamento composto na tabela Order Details (Detalhes do pedido). As colunas OrderID (Identificação do pedido) e ProductID (Identificação do produto) são os valores de chaves compostas. Observe que a coluna OrderID é listada primeiro, pois é mais seletiva do que a coluna ProductID.

```
USE Northwind
CREATE UNIQUE NONCLUSTERED INDEX U_OrdID_ProdID
ON [Order Details] (OrderID, ProductID)
```

6.5 Obtendo informações sobre os índices existentes

Você poderá precisar de informações sobre os índices existentes antes de criar, modificar ou remover um índice.

Usando o procedimento armazenado do sistema sp_helpindex

Você pode usar o SQL Server Enterprise Manager ou executar o procedimento armazenado do sistema sp_helpindex para obter informações sobre índices, como nome, tipo e opções de índice para uma tabela específica.

Este exemplo lista os índices da tabela Customers.

```
USE Northwind
EXEC sp_helpindex Customers
index_name
index_description
index_keys
PK_Customers clustered, unique, Primary Key located on PRIMARY CustomerID
PostalCode nonclustered located on PRIMARY PostalCode
City nonclustered located on PRIMARY City
```

Usando o procedimento armazenado do sistema sp_help nome_da_tabela

Você também pode executar o procedimento armazenado do sistema sp_help nome_da_tabela para obter informações sobre índices, bem como outras informações sobre a tabela.

```
USE Northwind
EXEC sp_help Customers
```

Laboratório

1. Localize os produtos que estiveram em mais de uma venda no mesmo mês.

2. Exiba estrutura de cada tabela do sistema
3. Entre os produtos que foram vendidos em quantidades maiores a 10 itens em mais de 3 vendas
4. Localize produtos em que a média itens vendidos seja maior que a quantidade de produtos em estoque
5. Monte uma consulta que retorne a quantidade de vezes em que cada produto é utilizado em uma venda, para os produtos vendidos em quantidades superiores a 10

Unidade 7 - Otimizando o desempenho de consultas

7.1 Introdução ao otimizador de consultas

O conhecimento sobre a função do otimizador de consultas na otimização de consultas prepara você para criar índices úteis, escrever consultas eficientes e ajustar as consultas de baixo desempenho.

Função do otimizador de consultas

O otimizador de consultas é o componente responsável pela geração do melhor plano de execução para uma consulta.

Determina o plano de execução mais eficiente

O otimizador de consultas avalia cada instrução Transact-SQL e determina o plano de execução mais eficiente.

O otimizador de consultas calcula a E/S necessária para processar uma consulta:

Determinando se os índices existem e calculando sua utilidade para uma consulta.

Determinando os índices ou colunas que podem ser usados para reduzir o número de registros examinados pela consulta. Com a redução do número de registros examinados, a quantidade de E/S é reduzida, que é o objetivo do desempenho da consulta.

Determinando a estratégia mais efetiva de processamento de operações de associação, como a ordem em que as tabelas devem ser associadas e a estratégia de associação a ser usada.

Usando a avaliação baseada em custos de alternativas para selecionar o plano mais eficiente para determinada consulta.

Criando estatísticas de colunas para melhorar o desempenho da consulta.

Usa informações adicionais

O otimizador de consultas usa informações adicionais sobre os dados subjacentes e estruturas de armazenamento, tamanho do arquivo e tipos de estruturas de arquivos. Ele também usa suas próprias operações internas, como a criação de índices ou tabelas temporárias na memória, para melhorar o desempenho de consultas.

Produz um plano de execução

O otimizador de consultas produz um plano de execução que descreve a seqüência de etapas necessária para executar uma consulta. Além disso, otimiza o processo de localização, associação, agrupamento e ordenamento de registros.

7.2 Como o otimizador de consultas usa a otimização baseada em custos

O otimizador de consultas é baseado em custos, o que significa que avalia cada plano de execução estimando o custo de execução. As estimativas de custo só podem ter o mesmo grau de exatidão que os dados estatísticos disponíveis sobre as colunas, índices e tabelas.

Limita o número de planos de otimização

Para ser executado em um período de tempo razoável, o otimizador de consultas limita o número de planos de otimização por ele considerado. Ao avaliar as seqüências das operações relacionais necessárias para produzir o conjunto de resultados, o otimizador de consultas alcança um plano de execução que tem o custo estimado mais baixo em termos perda de recursos de E/S e CPU.

Determina o tempo de processamento da consulta

O desempenho da consulta é determinado pelos operadores físicos usados pelo otimizador de consultas e a seqüência em que as operações são processadas. O objetivo é reduzir:

- O número de registros retornados.
- O número de páginas lidas.
- O tempo total de processamento minimizando os recursos de E/S e CPU usados para um plano de execução.

Quando o otimizador de consultas otimiza consultas, ele não inicia o plano de execução com a menor perda de recursos. Em vez disso, escolhe o plano de execução que retorna os resultados da maneira mais rápida para o usuário, com uma redução razoável de recursos.

Se o Microsoft® SQL Server tiver mais de um processador disponível, o otimizador de consultas poderá dividir a consulta entre eles. Em geral, as consultas de longa duração tiram proveito dos planos de execução, mas uma consulta paralela pode usar mais recursos gerais que o processamento serial de uma consulta.

7.3 Como o otimizador de consultas funciona

Depois que uma consulta é submetida, ocorrem várias etapas que transformam a consulta original em um formato que o otimizador de consultas pode interpretar.

Processo de análise

O processo de análise verifica se a consulta recebida possui a sintaxe correta e divide essa sintaxe em partes de componentes às quais o mecanismo de banco de dados relacional pode responder. A saída dessa etapa é uma árvore de consultas analisada.

Processo de padronização

O processo de padronização transforma uma consulta em um formato útil de otimização. Qualquer cláusula de sintaxe redundante detectada é removida. As subconsultas são padronizadas. A saída dessa etapa é uma árvore de consulta padronizada.

Otimização de consultas

O processo de selecionar um plano de execução a partir de vários planos possíveis chama-se otimização. Várias etapas são envolvidas nessa fase. No entanto, as etapas a seguir têm o efeito mais significativo no custo do plano de execução: análise de consultas, seleção de índices e seleção de associações.

Compilação

A consulta é compilada em um código executável.

Rotinas de acesso a bancos de dados

O otimizador de consultas determina o melhor método de acesso a dados, executando um exame de tabela ou usando um índice disponível. O melhor método é então aplicado.

7.4 Fases de otimização de consultas

O processo de otimização de consultas é constituído de três fases. Essas fases não são etapas de processamento distintas e são usadas apenas para representar conceitualmente a atividade interna do otimizador de consultas.

Análise de consultas

A primeira fase da otimização de consultas chama-se análise de consultas. Nessa fase, o otimizador de consultas identifica a pesquisa e os critérios de associação da consulta. Ao limitar a pesquisa, o otimizador minimiza o número de registros processados. A redução do número de registros processados reduz o número de páginas de índice e de dados lidas.

Seleção de índices

A seleção de índices é a segunda fase da otimização de consultas. Durante essa fase, o otimizador de consultas detecta se um índice existe para as cláusulas identificadas. Em seguida, há uma avaliação da utilidade do(s) índice(s). A utilidade de um índice é determinada pela maneira como vários registros são retornados. Essas informações são reunidas a partir de estatísticas de índice ou de coluna. Uma estimativa do custo de vários métodos de acesso ocorre por meio da estimativa das leituras de páginas lógicas e físicas necessárias para localizar os registros qualificados.

Seleção de associações

A seleção de associações é a terceira fase da otimização de consultas. Se houver uma consulta de várias tabelas ou auto-associação, ocorrerá uma avaliação de qual estratégia de associação será usada. A determinação da estratégia de associação a ser usada envolve a consideração de várias fatores: seletividade, densidade e memória necessárias para processar a consulta.

7.5 Armazenando em cache o plano de execução

O SQL Server possui um pool de memória usado para armazenar os planos de execução e buffers de dados. A porcentagem alocada pelo pool para os planos de execução ou buffers de dados oscila dinamicamente, de acordo com o estado do sistema. A parte do pool de memória usada para armazenar os planos de execução é chamada cache de procedimento.

Armazenando um plano de execução na memória

O plano de execução em massa é uma estrutura de dados somente leitura reutilizável, que pode ser usada por inúmeros usuários. Nenhum contexto de usuário é armazenado no plano de execução. Nunca há mais de duas cópias do plano de execução na memória:

Uma cópia de todas as execuções seriais.

Outra cópia de todas as execuções paralelas.

A cópia paralela inclui todas as execuções paralelas, independentemente de seu grau de paralelismo.

Usando um contexto de execução

Cada usuário que executa uma consulta possui uma estrutura de dados que contém os dados específicos a uma execução, como valores de parâmetro. Essa estrutura de dados chama-se contexto de execução. Quando uma instrução Transact-SQL é executada, o SQL Server examina o cache de procedimentos

para determinar se existe um plano de execução para a mesma instrução Transact-SQL.

Se existir algum plano de execução, o SQL Server o reutilizará. Esse procedimento economiza a sobrecarga de recompilar a instrução Transact-SQL.

Se não existir um plano de execução, o SQL Server gerará um novo para a consulta.

7.6 Recompilando os planos de execução

Certas alterações efetuadas em um banco de dados podem tornar um plano de execução ineficiente ou impreciso. Quando o SQL Server detecta alterações que invalidam um plano de execução, marca-o como inválido. Um novo plano de execução é compilado para a próxima conexão que executa a consulta. O desempenho pode ser melhorado através da redução do número de vezes que um plano é recompilado. As condições que invalidam um plano de execução incluem:

Qualquer alteração estrutural efetuada em uma tabela ou view à qual a consulta (instruções ALTER TABLE e ALTER VIEW) faça referência.

Novas estatísticas de distribuição geradas explicitamente em uma instrução, como UPDATE STATISTICS, ou de forma automática.

Descartar um índice usado pelo plano de execução.

Uma chamada explícita para o procedimento armazenado do sistema sp_recompile.

Diversas alterações são efetuadas nas chaves (instrução INSERT ou DELETE) de uma tabela à qual a consulta faça referência.

Para tabelas com disparadores, se o número de registros nas tabelas inseridas ou excluídas crescer de modo significativo.

O SQL Server usa um algoritmo de contagem de tempo para gerenciar eficientemente os planos de execução no cache, além de avaliar o custo e uso do plano de execução.

7.7 Definindo um limite de custo

Convém controlar o custo de execução de uma consulta definindo um limite de custo. O termo custo da consulta refere-se ao tempo decorrido estimado, em segundos, necessário para executar uma consulta em uma configuração de hardware específica.

Especificando um limite superior

Você pode usar a opção query governor cost limit (limite de custos do regulador de consultas) para impedir que consultas de longa duração sejam executadas e consumam os recursos do sistema. Por padrão, é possível executar as consultas, seja qual for sua duração. O regulador de consultas usa um custo estimado para impedir que as consultas com um alto custo sejam executadas. Embora o valor da configuração seja especificado em segundos, não se correlaciona verdadeiramente ao tempo, mas ao custo estimado real da consulta.

É possível especificar um limite superior do custo da consulta a ser executada. Como o regulador de consultas é baseado no custo estimado da consulta, em vez de no tempo decorrido real, não possui sobrecarga de tempo de execução. Se o custo estimado de uma consulta for maior que o limite de custo especificado, a instrução do regulador de consultas impedirá que a consulta seja executada. Essa ação será mais eficiente que permitir que uma consulta seja executada, até que algum limite predefinido seja alcançado, e depois interromper sua execução.

Especificando limites de conexão

É possível especificar limites para todas as conexões ou para apenas as consultas referentes a uma conexão específica. Para aplicar limites de custo ao regulador de consultas, você pode:

Usar o procedimento armazenado sp_configure para aplicar limites a todas as conexões. Só será possível alterar o limite de custo do regulador de consultas quando show advanced options for definida como 1. A configuração terá efeito imediatamente. Não será necessário pará-la e reiniciar o servidor.

Execute a instrução SET QUERY_GOVERNOR_COST_LIMIT para aplicar limites para uma conexão específica.

Especifique 0 (o padrão) para desativar o regulador de consultas. Nesse caso, todas as consultas serão executadas sem limites.

7.8 Obtendo informações sobre planos de execução

O otimizador de consultas responde às informações disponíveis durante a determinação do melhor plano de execução. Você pode obter informações sobre o plano de execução consultando a tabela sysindexes. Também é possível obter informações usando as instruções STATISTICS e SHOWPLAN e exibindo graficamente o plano de execução.

Exibindo a saída de instruções STATISTICS

Você pode usar as instruções STATISTICS IO, STATISTICS TIME e STATISTICS PROFILE para obter informações que podem ajudá-lo a diagnosticar consultas de longa duração. A saída das instruções STATISTICS fornece informações sobre o plano de execução real. STATISTICS TIME obtém informações sobre o número de milissegundos necessários para analisar, compilar e executar cada instrução.

STATISTICS PROFILE exibe as informações de perfil referentes a uma instrução. Quando você executa uma consulta, a saída da instrução SHOWPLAN_ALL e as duas colunas adicionais são incluídas no conjunto de resultados

STATISTICS IO obtém informações sobre a quantidade de leituras de página geradas por consultas.

Exibindo a saída de SHOWPLAN_ALL e de SHOWPLAN_TEXT

Você pode usar as instruções SET SHOWPLAN_TEXT e SET SHOWPLAN_ALL para obter informações detalhadas sobre como as consultas são executadas e quantos recursos são necessários para processar a consulta.

Estrutura da saída da instrução SHOWPLAN

A saída da instrução SHOWPLAN:

Retorna as informações como um conjunto de registros.

Forma uma árvore hierárquica.

Representa as etapas usadas pelo otimizador de consultas para executar cada instrução.

Mostra os valores estimados de como uma consulta foi otimizada, e não o plano de execução real. Os valores estimados são baseados em estatísticas existentes.

Detalhes das etapas de execução

Cada instrução refletida na saída contém um único registro com o texto da instrução, seguido de vários registros com os detalhes das etapas de execução.

Os detalhes das etapas de execução incluem:

Os índices que são usados com determinadas tabelas.

A ordem de associação das tabelas.

O modo de atualização escolhido.

As tabelas de trabalho e outras estratégias.

Diferença entre a saída de SHOWPLAN_TEXT e de SHOWPLAN_ALL

A diferença entre a saída SHOWPLAN_TEXT e SHOWPLAN_ALL é que a última retorna informações adicionais, como os registros estimados, E/S, CPU e o tamanho de registro médio da consulta. As instruções SET permanecem ativas em relação à sessão até que você especifique a opção OFF, ou até que finalize a sessão.

7.9 Exibindo graficamente o plano de execução

Você pode usar o SQL Query Analyzer (Analisador de consultas do SQL) para exibir graficamente um plano de execução codificado por cores.

Elementos do plano de execução gráfico

O plano de execução gráfico, que contém os elementos a seguir, usa ícones para representar a execução de partes específicas das instruções e consultas: Etapas são unidades de trabalho usadas para processar uma consulta. Seqüência de etapas é a ordem em que as etapas são processadas. Os operadores lógicos descrevem a operação algébrica relacional usada para processar uma instrução; por exemplo, a execução de uma agregação. Em geral, o operador lógico corresponde ao operador físico. Nem todas as etapas usadas para processar uma consulta ou atualizar operações envolvem operações lógicas. Os operadores físicos descrevem o algoritmo de implementação física usado para processar uma instrução; por exemplo, o exame de um índice de agrupamento. Cada etapa da execução de uma consulta ou operação de atualização envolve um operador físico.

Lendo a saída do plano de execução gráfico

A saída do plano de execução gráfico é lida da direita para a esquerda e de cima para baixo. Cada consulta no lote analisado é exibida, inclusive o custo de cada consulta como uma porcentagem do custo total do lote.

Cada etapa pode ter um ou vários nós a serem processados. O termo nó referese a uma operação usada pelo otimizador de consultas, que é representado por um ícone.

O plano de execução pode ter vários nós para uma etapa específica.

Cada nó é relacionado a um nó pai.

Todos os nós com o mesmo pai são dispostos na mesma coluna.

As pontas de seta conectam-se ao nó e a seu pai.

As operações recursivas são mostradas com um símbolo de iteração.

Os operadores são mostrados como símbolos relacionados a um pai específico.

Quando o lote contém várias instruções, vários planos de execução são desenhados.

Exibindo informações adicionais

Ao posicionar o ponteiro sobre cada nó (representado por um ícone), você pode visualizar informações detalhadas sobre os operadores físicos e lógicos.

Unidade 8 - Criando Cursores

8.1 Introdução sobre cursores

Os Cursores são utilizados quando precisamos executar uma ou mais ações em cada registro individualmente retornado por determinada consulta.

Antes de utilizamos os bancos de dados relacionais, em que o armazenamento ocorria nos arquivos textos e flat files, como o dBase, o acesso aos dados era realizado seqüencialmente.

Se houvesse a necessitasse acessar vários elementos de uma só vez era necessária a utilização de um laço (comando While, por exemplo) para acessar os registros necessários.

No SQL Server, assim como em qualquer SGBD isso não é mais necessário, pois com o modelo relacional de banco de dados, são acessados vários registros de uma só vez através dos comandos SELECT, UPDATE e DELETE.

A quantidade de linhas retornadas depende do tamanho da tabela e da forma com que são buscados em conjunto com a cláusula WHERE, que realiza uma filtragem nos dados selecionados.

Contudo, existem situações em que trazer os registros de uma só vez não é conveniente ou possível para realizar certos tipos de operações, onde é necessário obter resultado de cada linha uma a uma. Nestes casos os SGBD's atuais fornecem um recurso bastante interessante, chamados cursores.

O cursor é uma instrução SELECT que será acessada linha a linha através de um laço While e alguns comandos específicos para cursores e é utilizado normalmente em procedimentos armazenados (stored procedures).

8.2 Declarando um cursor

Para criarmos um curso deveremos em primeiro lugar definir a instrução SELECT que ele acessará. Vamos utilizar uma consulta que nos mostre quantidade de produtos em vendidos em uma determinada venda, para isso vamos usar uma variável de controle @orderid que virá de um procedimento.

```
--Select utilizado para o cursor

SELECT productid, quantity FROM [order details] WHERE orderid = @orderid
```

É recomendado que execute a instrução de SELECT para verificar se o resultado é o esperado. Após isso, deve ser utilizado o comando DECLARE, que serve para declarar variáveis e o cursor.

A variável @orderid será um parâmetro do procedimento, logo não necessita declaração. Abaixo está a declaração do cursor.

```
--Declarando cursor

DECLARE CursorDosItens --Nome do cursor

CURSOR FOR

-- Select utilizado para o cursor

SELECT productid, quantity FROM [order details] WHERE orderid = @orderid
```

8.3 Abrindo um cursor

Realizada a declaração do cursor é necessário realizar a abertura dele buscando o primeiro registro.

Para isto serão declaradas variáveis que receberão o código do produto e a quantidade vendida, através do comando FETCH.

O comando FETCH NEXT traz a próxima linha do SELECT, contudo o comando FETCH pode ser usado em conjunto com outras cláusulas para outros comportamentos, como o FETCH PRIOR (Anterior), FETCH FIRST (Primeiro), FETCH LAST (Último), entre outros.

```
--Declarando variáveis

DECLARE @productid INTEGER, @quantity DOUBLE

--Abrindo cursor

OPEN CursorDosItens
```

```
--Atribuindo valores do select nas variáveis

FETCH NEXT FROM CursorDosItens INTO @productid, @quantity
```

8.4 Percorrendo um cursor

Após definirmos os valores da primeira linha de retorno, é necessário acessar as demais linhas, pois está é a finalidade do mesmo.

Para isso será utilizado um laço (WHILE) em conjunto com a variável global do SQL Server @@FETCH_STATUS.

Esta variável retorna 0 (zero) caso o último comando de FETCH tenha sido executado com sucesso e tenha retornado dados e –1 caso não haja mais dados (EOF – fim de arquivo).

Por fim utilizamos os comandos CLOSE , para fechar o cursor e DEALLOCATE, para eliminá-lo da memória, pois caso o procedimento seja executado novamente, pode apresentar erro na declaração do cursor, caso o cursor ainda exista.

O laço é descrito abaixo:

```
--Iniciando laço

WHILE @@FETCH_STATUS = 0

BEGIN
--Próxima linha do cursor

FETCH NEXT FROM CursorDosItens INTO @productid, @quantity

END
--Fechando e desalocando cursor

CLOSE CursorDosItens

DEALLOCATE CursorDosItens
```

8.5 Executando comandos em um cursor

De posse do código do produto e da quantidade em de itens comprados na venda atual podemos baixar o estoque na tabela de produtos(products) com o cuidado de não deixarmos os estoque negativo, o que poderia ser feito através de uma restrição de domínio (check) na tabela de estoque, entretanto neste exemplo utilizaremos uma verificação com um SELECT na tabela de estoque.

Caso o estoque não fique negativo, o comando para baixar o estoque é realizado, caso contrário será levantado um erro com o comando RAISERROR.

```
--Iniciando laço

WHILE @@FETCH_STATUS = 0

BEGIN
IF (SELECT unitinstock - @quantity FROM products WHERE productid = @productid) >= 0

UPDATE products

SET unitinstock = unitinstock - @quantity

WHERE productid = @productid

ELSE
RAISERROR('Abaixo do estoque mínimo', 15, 1)

--Próxima linha do cursor

FETCH NEXT FROM CursorDosItens INTO @productid, @quantity

END
--Fechando e desalocando cursor

CLOSE CursorDosItens

DEALLOCATE CursorDosItens
```

8.6 Confirmando ou retornando um cursor

Para finalizar, caso tudo tenha ocorrido com sucesso, devemos finalizar a venda propriamente dita, mudando o campo shipvia de 0 para 1.

Uma prática muito recomendada é trabalhar com transação, pois caso um item dê problemas, os demais que já teriam sido baixados devem ser retornados.

Desta forma o procedimento completo ficaria como descrito:

```
--Procedimento para finalização de uma venda

CREATE PROCEDURE realiza_venda (@orderid INTEGER) AS

--Declarando cursor

DECLARE CursorDosItens --Nome do cursor

CURSOR FOR

-- Select utilizado para o cursor

SELECT productid, quantity FROM [order details] WHERE orderid = @orderid

--Declarando variáveis

DECLARE @productid INTEGER, @quantity DOUBLE

--Iniciando transação

BEGIN TRANSACTION

--Abrindo cursor

OPEN CursorDosItens

--Atribuindo valores do select nas variáveis

FETCH NEXT FROM CursorDosItens INTO @productid, @quantity
```

```
--Iniciando laço

WHILE @@FETCH_STATUS = 0

BEGIN
IF (SELECT unitinstock - @quantity FROM products WHERE productid =
@productid) >= 0

UPDATE products

SET unitinstock = unitinstock - @quantity

WHERE productid = @productid

ELSE
BEGIN
--Desfazendo o que foi realizado anteriormente

ROLLBACK TRANSACTION

--Levantando erro

RAISERROR('Abaixo do estoque mínimo', 15, 1)

--Fechando e desalocando cursor aqui também, pois o return sairá do
procedimento

CLOSE CursorDosItens

DEALLOCATE CursorDosItens

--Saindo do procedimento

RETURN

END
--Próxima linha do cursor
```

```
FETCH NEXT FROM CursorDosItens INTO @productid, @quantity

END
--Fechando e desalocando cursor

CLOSE CursorDosItens

DEALLOCATE CursorDosItens

--Caso tudo tenha ocorrido OK, alterando a situação da venda

UPDATE Orders

SET shipvia = 1

WHERE orderid = @orderid

--Confirmando transação

COMMIT TRANSACTION
```

Laboratório

Crie os seguintes cursores:

1. Verifique as vendas de cada cliente. Se o valor total da venda for maior que 10.000 então atualize o campo contacttitle para o valor atual mais a string “vip”
2. Verifique cada produto, e se o mesmo não é comercializado a mais de 3 meses atualize o parâmetro discotinued para 1, do contrário exiba o nome do fabricante do mesmo
3. Verifique a quantidade de produtos vendidos em cada venda e exiba se a quantidade for de itens da venda anterior é maior ou menor que a atual
4. Verifique em cada vendedor por ordem alfabética e se região a região vendedor é a mesma ou não
5. Verifique se cada venda para ver se cada a região do cliente é a mesma do vendedor, se positivo conceda 10% de desconto do contrário 3%

Unidade 9 - Esquemas XML

9.1 Introdução sobre XSD

Com é de conhecimento de todos, o SQL Server fornece armazenamento nativo de dados XML por meio do tipo de dados XML.

Podemos, opcionalmente, associar esquemas XML a uma variável ou a uma coluna de tipo XML por meio de uma coleção de esquema XML. A coleção de esquema XML armazena os esquemas XML importados e, em seguida, é usada para fazer o seguinte:

- Validar instâncias XML
- Definir o tipo dos dados XML conforme eles são armazenados no banco de dados

Atentamos ao fato de que a coleção de esquema XML é uma entidade de metadados como uma tabela no banco de dados, portanto é possível criar, modificar e deletá-la.

Os esquemas criados para as exibições XML de dados relacionais são montados usando a linguagem XSD.

Dessa forma, essas exibições podem ser consultadas por meio de consultas em linguagem XPath. Isso é semelhante à criação de exibições usando instruções CREATE VIEW e especificando consultas SQL com base na exibição.

Um esquema XML descreve a estrutura de um documento XML, além das várias restrições referentes aos dados do documento.

Também é possível usar a coleção de esquema XML para digitar variáveis, parâmetros e colunas XML.

9.2 Criando um esquema XSD

Para criarmos um esquema XML (XSD) utilizamos uma instrução CREATE XML SCHEMA COLLECTION (Transact-SQL)

```
CREATE XML SCHEMA COLLECTION MeuEsquema AS N'
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Funcionario" sql:relation="dbo.Employees" >
   <xsd:complexType>
     <xsd:sequence>
        <xsd:element name="FName"
                     sql:field="FirstName"
                     type="xsd:string" />
        <xsd:element name="LName"
                     sql:field="LastName"
                     type="xsd:string" />
     </xsd:sequence>
        <xsd:attribute name="FId"
                       sql:field="employeedid"
                       type="xsd:integer" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>'
```

9.3 Associando um esquema XSD

No momento da criação de uma tabela podemos associar um campo do tipo de dado XML a um esquema previamente criado:

```
[campo] [xml](CONTENT [dbo].[MeuEsquema])
```

Os dados em XML armazenados em uma coluna ou variável à qual um esquema está associado são referidos como **XML com tipo** porque o esquema fornece as informações necessárias do tipo de dados para os dados da instância.

O SQL Server usa essas informações de tipo para otimizar o armazenamento de dados. No processamento de consultas também é utilizado o esquema para verificação de tipo e otimizar modificação de dados e consultas.

Além disso, o SQL Server usa a coleção de esquema XML associada, no caso de XML com tipo de dados de uma coluna para validar a instância XML. Se a instância XML estiver de acordo com o esquema, o banco de dados permitirá que a instância seja armazenada no sistema com suas informações de tipo. Caso contrário, a instância será rejeitada.

O SQL Server fornece várias instruções DDL para gerenciar os esquemas no banco de dados. No entanto, primeiro a coleção de esquema XML precisa ser criada para que seja possível usá-la..

É possível usar a função intrínseca XML_SCHEMA_NAMESPACE para obter dados sobre a coleção de esquema que está armazenada no banco de dados.

9.4 Alterando um esquema XSD

É possível alterar componentes do esquema em um objeto de coleção existente no banco de dados usando a instrução ALTER XML SCHEMA COLLECTION (Transact-SQL).

```
ALTER XML SCHEMA COLLECTION MeuEsquema ADD '
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
         targetNamespace="http://MySchema/test_xml_schema">
     <element name="outroElemento" type="byte"/>
 </schema>'
```

9.5 Deletando um esquema XSD

Para deletar um esquema XSD de nosso banco de dados basta utilizarmos a intrução DROP XML SCHEMA COLLECTION.

Veja o script abaixo

```
DROP XML SCHEMA COLLECTION MeuEsquema
GO
```

Laboratório

Para realizar o exercício crie a seguinte tabela:

1. Crie um esquema XML para a estrutura de clientes e insira 5 registros
2. Crie um esquema XML para a estrutura de fornecedores e insira 5 registros
3. Crie um esquema XML para a estrutura de produtos e insira 5 registros
4. Crie um esquema XML para a estrutura de categorias e insira 5 registros
5. Crie um esquema XML para a estrutura de territórios e insira 5 registros

Unidade 10 - Agente de Serviços

10.1 Introdução sobre Agente de Serviços

O SQL Server Agent é um serviço do Windows que executa tarefas administrativas agendadas, que são chamadas de *trabalhos*. O SQL Server Agent usa o SQL Server para armazenar informações de trabalhos.

Estes trabalhos contêm uma ou mais etapas de trabalho, e cada etapa contém sua própria tarefa; por exemplo, fazer o backup de um banco de dados.

O SQL Server Agent pode executar um trabalho de uma agenda, em resposta a um evento específico ou sob demanda. Por exemplo, se desejar fazer o backup de todos os servidores da empresa todo dia após o expediente, você pode automatizar essa tarefa. Agende o backup para execução após as 22:00, de segunda a sexta; se o backup encontrar um problema, o SQL Server Agent poderá registrar o evento e notificá-lo.

Caso ocorra algum erro na execução de um trabalho agendado, os trabalhos que estão vinculados a este trabalho que gerou erro são executados até que um novo erro ocorra novamente.

10.2 Iniciando o Agente de Serviços

Por padrão, o serviço do SQL Server Agent encontra-se desabilitado quando o SQL Server 2005 ou posterior é instalado, a menos que o usuário escolha, de maneira explícita, iniciar automaticamente o serviço.

Inicie uma instância de serviço do SQL Server ou do SQL Server Agent a partir de um prompt de comando com o comando **net start** ou executando o **sqlservr.exe**.

Ou siga os seguintes passos:

1. No menu **Iniciar**, aponte para **Todos os Programas**, aponte para **Microsoft SQL Server,** aponte para **Ferramentas de Configuração** e clique em **SQL Server Configuration Manager**.
2. No **SQL Server Configuration Manager**, expanda **Serviços** e clique em **SQL Agent**.
3. No painel de resultados, clique com o botão direito do mouse em qualquer instância e depois clique em **Iniciar**.

Uma seta verde no ícone próximo ao SQL Server Agent e na barra de ferramentas indica que o SQL Server Agent foi iniciado com êxito.

4. Clique em **OK**.

10.3 Criando um trabalho

Para que uma determinada ação ocorra momentaneamente ou em um tempo determinando é necessário criar o que chamamos de trabalho.

Para tal execute os seguintes passos:

1. No **Pesquisador de Objetos**, conecte-se a uma instância do Mecanismo de banco de dados do SQL Server e expanda-a.
2. Expanda o **SQL Server Agent**.
3. Clique com o botão direito do mouse em **Trabalhos** e clique em **Novo Trabalho**.
4. Na página **Geral**, na caixa **Nome**, digite um nome para o trabalho.
5. Desmarque a caixa de seleção **Habilitado** se não quiser executar o trabalho imediatamente após a sua criação. Por exemplo, se você quiser testar um trabalho antes agendá-lo para execução, desabilite-o.
6. Na caixa **Descrição**, insira uma descrição do que o trabalho faz. O número máximo de caracteres é 512.

10.4 Para criar uma etapa de trabalho Transact-SQL

Um trabalho é formado por uma ou mais etapas. Estas etapas são as ações propriamente ditas que o trabalho irá executar.

Siga os seguintes passos;

1. No **Pesquisador de Objetos**, conecte-se a uma instância do Mecanismo de banco de dados do SQL Server e expanda-a.
2. Expanda **SQL Server Agent**, crie um novo trabalho ou clique com o botão direito do mouse em um trabalho existente e, em seguida, clique em **Propriedades**.

Para obter mais informações sobre como criar um trabalho, consulte Criando trabalhos.

3. Na caixa de diálogo **Propriedades do Trabalho**, clique na página **Etapas** e, em seguida, em **Nova**.
4. Na caixa de diálogo **Nova Etapa de Trabalho**, digite o **Nome da etapa** de trabalho.
5. Na lista **Tipo**, clique em **Script Transact-SQL (TSQL)**.
6. Na caixa **Comando**, digite os lotes de comandos Transact-SQL ou clique em **Abrir** para selecionar um arquivo Transact-SQL a ser usado como comando.
7. Clique em **Analisar** para verificar a sintaxe.
8. A mensagem "Êxito da análise" será exibida se a sintaxe estiver correta. Se um erro for encontrado, corrija a sintaxe antes de continuar.
9. Clique na página **Avançado** para definir opções para a etapa de trabalho, tais como: que ação deve ser adotada em caso de êxito ou falha da etapa, quantas vezes o SQL Server Agent deve tentar executar a etapa e em que arquivo ou tabela o SQL Server Agent deve gravar a saída da etapa de trabalho. Só membros da função de servidor fixa **sysadmin** podem gravar a saída de etapas de trabalho em um arquivo do sistema operacional. Todos os usuários do SQL Server Agent podem registrar a saída em uma tabela.
10. Se você for membro da função de servidor fixa **sysadmin** e desejar executar a etapa de trabalho como um logon SQL diferente, selecione esse logon na lista **Executar como usuário**.

10.5 Para criar e anexar uma agenda a um trabalho

Após ter criado um trabalho e gerado tarefas para o mesmo é necessário criar e anexar uma agenda a ele.

Para isso siga os seguintes passos:

1. No **Pesquisador de Objetos**, conecte-se à instância do Mecanismo de banco de dados do SQL Server e expanda-a.
2. Expanda **SQL Server Agent**, expanda **Trabalhos**, clique com o botão direito do mouse no trabalho que deseja agendar e clique em **Propriedades**.
3. Selecione a página **Agendas** e clique em **Nova**.

4. Na caixa **Nome**, digite um nome para a nova agenda.
5. Desmarque a caixa de seleção **Habilitado** se não quiser que a agenda entre em vigor imediatamente após a sua criação.
6. Para **Tipo de Agenda**, siga um destes procedimentos:
    - Clique em **Iniciar automaticamente quando o SQL Server Agent for iniciado** para iniciar o trabalho quando o serviço do SQL Server Agent for iniciado.
    - Clique em **Iniciar quando as CPUs estiverem ociosas** para iniciar o trabalho quando as CPUs atingirem uma condição de ociosidade.
    - Clique em **Recorrente** se desejar que a agenda seja executada seguidamente. Para definir a agenda recorrente, complete os grupos **Freqüência**, **Freqüência Diária** e **Duração** na caixa de diálogo.
    - Clique em **Uma vez** se quiser que a agenda seja executada apenas uma vez. Para definir uma agenda executada apenas **Uma vez**, complete o grupo**Ocorrência única** na caixa de diálogo.

Laboratório:

Crie os seguintes Jobs:

1. Backups periódicos do log de transações
2. Checagem de tamanho de banco de dados
3. Exportação/importação de dados em um determinado horário
4. Envio de e-mail com o resultado de um SELECT a cada 30 minutos
5. Atualização de estatísticas quaisquer

Unidade 11 - Projeto Final

11.1 Situação Problema

Construa o seguinte bando de dados para uma empresa de vendas de ingressos on-line:

O sistema deve possuir controle de usuário e senha.

A empresa controla a venda de ingressos para determinados eventos, que pode ser de tipos diferentes

Os ingressos são de tipos e valores diferentes

Cada evento tem um ou mais contratantes diferentes

Todo evento possui uma ou mais atrações que por sua vez são classificadas por categorias

Deve existir um controle de lotação do evento por tipo de igresso.

È possível realizar um controle de “over booking” (excesso de vendas)

Relatório de vendas diário

Permite várias formas de pagamento

Venda com marcação de lugar

Log de atividades dos usuários

Fechamento de caixa por caixa/usuário.

Histórico detalhado de todas as vendas.

Permite desfazer vendas indevidas.

Permite emissão de cupons de venda de produtos. (apenas venda)

Vendas e renovações on-line de ingressos para temporada