



**CWI SOFTWARE**

módulo 2 | banco de dados

**Crescer 2015-1**

Feevale



## 6 - Comandos T-SQL

André Luís Nunes

abril/2015

**T-SQL**

**BLOCOS ANÔNIMOS**

**PROCEDURES**

**FUNCTIONS**

**TRIGGERS**

# TRANSACT-SQL

"O sistema que você procura talvez não exista,  
mas a empresa que o produz, sim".

CWI SOFTWARE - CMMI Nível 3  
<http://www.cwi.com.br>

UNIDADES CWI:  
Porto Alegre, São Paulo, São Leopoldo, Caxias do Sul.

## TRANSACTION-SQL (T-SQL)

Linguagem de programação utilizada no SQL Server, é uma extensão da SQL (SQL-92) e é proprietária da Microsoft.

Assim como a Oracle utiliza a PL/SQL (sua linguagem própria de SQL “avançada”).

- Variáveis locais;
- Controle de fluxo (IF, CASE, WHILE);
- Cursores;
- Tratamento de exceções.

# BLOCOS ANÔNIMOS

"O sistema que você procura talvez não exista,  
mas a empresa que o produz, sim".

CWI SOFTWARE - CMMI Nível 3  
<http://www.cwi.com.br>

UNIDADES CWI:  
Porto Alegre, São Paulo, São Leopoldo, Caxias do Sul.

# BLOCOS ANÔNIMOS

Podem ser definidos (opcionalmente) por um bloco BEGIN-END.

```
BEGIN
```

```
    DECLARE @<NomeVariavel> <DataType>  
    { blocos de instruções }
```

```
END
```

Sintaxe básica

```
BEGIN
```

```
    DECLARE @Nome VarChar(30)  
    SET @Nome = 'CWI Software'
```

```
    Print @Nome
```

```
END
```

Exemplo

## BLOCOS ANÔNIMOS: variáveis

As variáveis são definidas com o prefixo @ (arroba), veja o exemplo:

```
DECLARE @NomeCliente  VarChar(50) ,  
        @NomeCidade    VarChar(30) ,  
        @DataAdmissao  DateTime ,  
        @ValorVenda    Decimal(12,2)
```

Exemplos

Para definir as variáveis é necessário utilizar o comando SET ou utilizar o comando SQL:

```
SET @DataAdmissao = GetDate()  
SET @ValorVenda   = 578.50
```



## BLOCOS ANÔNIMOS: comandos DML

É possível realizar as operações do tipo DML dentro de um bloco. Abaixo um exemplo onde é retornado apenas 1 registro.

```
BEGIN
  DECLARE @vIDCliente int

  Select @vIDCliente = IDCliente
  From   Cliente
  Where  Nome = 'Mariana Ventura Che'

  Print 'Mariana possui o código ' +
        Cast(@vIDCliente AS VarChar(10))
END
```

Utilize a base de dados de exemplo (Crescer15-1) para executar os exemplos.

## BLOCOS ANÔNIMOS: executando consultas diretamente

É possível executar a consulta diretamente dentro do bloco, e seu resultado ser projetado para o client (caso de procedures que geram relatórios):

```
BEGIN
  DECLARE @vTamanho int

  -- Busca o maior nome (caracteres)
  Select @vTamanho = MAX(Len(Nome))
  From    Cliente

  -- Lista todos os clientes cfe tamanho do nome
  Select IDCliente, Nome
  From    Cliente
  Where   Len(Nome) = @vTamanho

END
```

» Deve ser utilizado **somente** quando a consulta retorna **1 registro**.

## BLOCOS ANÔNIMOS: nocount

O NOCOUNT habilita/desabilita as mensagens que são exibidas dentro de um bloco T-SQL.

```
BEGIN
  SET NOCOUNT ON
  DECLARE @vTamanho int

  -- Busca o maior nome (caracteres)
  Select @vTamanho = MAX(Len(Nome))
  From   Cliente

  -- Lista todos os clientes cfe tamanho do nome
  Select IDCliente, Nome
  From   Cliente
  Where  Len(Nome) = @vTamanho
  SET NOCOUNT OFF
END
```

- » Otimiza um procedimento com várias execuções, onde o número de resultado exibido é muito grande.
- » Boa prática utilizar.

## BLOCOS ANÔNIMOS — exercício



- 1) Imprima qual produto é o mais vendido, considere a quantidade (PedidoItem).

## BLOCOS ANÔNIMOS: if-else

IF-ELSE: estrutura de comparação de expressões

```
BEGIN
  DECLARE @vCount int

  Select @vCount = COUNT(1)
  From   Cliente

  IF (@vCount=0)
    Print 'Nenhum cliente cadastrado.'
  ELSE IF (@vCount=1)
    Print 'Um registro encontrado.'
  ELSE
    Print 'Muitos registros encontrados.'
END
```

Referência: <http://msdn.microsoft.com/pt-br/library/ms182717.aspx>

## BLOCOS ANÔNIMOS: if-else

Se for necessário executar mais de 1 instrução dentro de 1 validação é preciso utilizar o BEGIN-END

```
BEGIN
  DECLARE @vCount int

  Select @vCount = COUNT(1)
  From   Cliente

  IF (@vCount=0) BEGIN
    Print 'Nenhum cliente cadastrado.'
    Print 'Execute o procedimento de carga'
  END ELSE IF (@vCount=1)
    Print 'Um registro encontrado.'
  ELSE
    Print 'Muitos registros encontrados.'
END
```

## BLOCOS ANÔNIMOS: while-loop

Repetição de um bloco de instruções, permite ainda o uso do *BREAK* e *CONTINUE*.

Se for necessário executar mais de 1 instrução dentro de 1 validação é preciso utilizar o BEGIN-END.

```
BEGIN
  DECLARE @vCount int
  SET @vCount = 0

  WHILE (@vCount<10) BEGIN
    SET @vCount = @vCount + 1
    Print 'Executou loop: ' + Cast(@vCount as Varchar(10) )
  END
END
```

Referência: <http://msdn.microsoft.com/pt-br/library/ms178642.aspx>

## BLOCOS ANÔNIMOS - exceções

Permitem o tratamento de exceções (a partir do 2005):

```
BEGIN

    BEGIN TRY
        {comando SQL | bloco de comandos}
    END TRY

    BEGIN CATCH
        {comando SQL | bloco de comandos}
    END CATCH

END
go
```

Referência: <http://msdn.microsoft.com/pt-br/library/ms175976.aspx>



## BLOCOS ANÔNIMOS - exceções

No escopo do Catch é possível utilizar as seguintes informações:

**ERROR\_NUMBER()** retorna o número do erro.

**ERROR\_SEVERITY()** retorna a severidade.

**ERROR\_STATE()** retorna o número do estado do erro.

**ERROR\_PROCEDURE()** retorna o nome do procedimento armazenado ou do gatilho no qual ocorreu o erro.

**ERROR\_LINE ()** retorna o número de linha dentro da rotina que causou o erro.

**ERROR\_MESSAGE ()** retorna o texto completo da mensagem de erro. O texto inclui os valores fornecidos para quaisquer parâmetros substituíveis, como complementos, nomes de objeto ou horas.

# BLOCOS ANÔNIMOS - exceções

Exemplo utilizado com transação.

```
BEGIN
  BEGIN TRY
    Begin Transaction

    Insert into Cidade (IDCidade, Nome, UF)
      Values (1, 'Morro da Pedra', 'RS');

    Commit
  END TRY

  BEGIN CATCH
    Rollback
    Print ERROR_MESSAGE()
  END CATCH

END
go
```

Referência: <http://msdn.microsoft.com/pt-br/library/ms175976.aspx>

# BLOCOS ANÔNIMOS - exceções

Exemplo utilizado com transação.

```
BEGIN
  BEGIN TRY
    Begin Transaction

    Set Identity_Insert Cidade ON;

    Insert into Cidade (IDCidade, Nome, UF)
      Values (1, 'Morro da Pedra', 'RS');

    Set Identity_Insert Cidade OFF;

    Commit
  END TRY

  BEGIN CATCH
    If @@TRANCOUNT > 0 Rollback; -- Se existir alguma transacao
    Print ERROR_MESSAGE()
  END CATCH
END
```

# BLOCOS ANÔNIMOS - exceções - raise

Raise: gerando uma exceção dentro de um bloco.

```
BEGIN
  BEGIN TRY
    Declare @vRazaoSocial VarChar(50),
            @vNome          VarChar(50)
    Set      @vRazaoSocial = 'CWI Software Ltda'

    Select @vNome = Nome
    From    Cliente
    Where   RazaoSocial = @vRazaoSocial

    If @@ROWCOUNT = 0 -- Numero de linhas do Select anterior
      RAISERROR('Registro "%s" nao foi encontrado!', 16, 1, @vRazaoSocial)

    Print 'Nome encontrado! [' + @vNome + ']'

  END TRY
  BEGIN CATCH
    Print 'Error Code: ' + CAST(ERROR_NUMBER() AS VARCHAR(100))
    Print 'Error Text: ' + ERROR_MESSAGE()
  END CATCH
END
```

## BLOCOS ANÔNIMOS - cursores

Permite carregar uma consulta e processá-la em um laço (while), permitindo validações e instruções para cada linha da consulta.

O cursor deve ser declarado como uma variável, sua consulta será executada somente com a instrução para abrí-lo, e não na declaração.

```
DECLARE <Nome_Cursor> CURSOR
[ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
FOR <Comando_Select>
```

Referência: <http://msdn.microsoft.com/pt-br/library/ms180169.aspx>

## BLOCOS ANÔNIMOS - cursores - parâmetros

Local ou Global: permite definir o escopo do cursor;

Forward\_Only ou Scroll: define a rolagem dos registros.

Static, keyset, dynamic ou Fast\_Forward: define o tipo de cursor.

Read\_only, Scroll\_Locks ou Optimistic: indica o tipo de bloqueio.

```
BEGIN

    DECLARE ListaCidade CURSOR
        Local
        Fast_Forward
        FOR Select Nome, Uf
            From Cidade
            Group by Nome, Uf
            Having COUNT(1) > 1;

    ...
END
```

## BLOCOS ANÔNIMOS - cursores - open-fetch

Para listar os registros de um cursor e executar instruções sobre estes é necessário abrir o cursor e realizar um FETCH de TODAS AS COLUNAS em variáveis locais, e através de um laço processar todas as linhas.

```
...  
OPEN ListaCidade;  
FETCH NEXT FROM ListaCidade INTO @vNome, @vUF  
  
WHILE (@@FETCH_STATUS=0) BEGIN  
  
    <instruções>  
  
    FETCH NEXT FROM ListaCidade INTO @vNome, @vUF  
END  
  
CLOSE ListaCidade;  
DEALLOCATE ListaCidade;  
...
```

## BLOCOS ANÔNIMOS - cursores - exemplo

```
BEGIN
  DECLARE ListaCidade CURSOR
    Local
    Fast_Forward
    FOR Select Nome, Uf
        From Cidade
        Group by Nome, Uf
        Having COUNT(1) > 1;
  DECLARE @vNome varchar(50),
    @vUF varchar(2)

  OPEN ListaCidade;
  FETCH NEXT FROM ListaCidade INTO @vNome, @vUF

  WHILE (@@FETCH_STATUS=0) BEGIN
    Print @vNome + '/' + @vUF;
    FETCH NEXT FROM ListaCidade INTO @vNome, @vUF
  END

  CLOSE ListaCidade;
  DEALLOCATE ListaCidade;
END
```

» Exemplo que exibe todas as cidades com Nome e UF duplicados.



## BLOCOS ANÔNIMOS - exercícios



- 2) Liste as cidades com nome e UF duplicados que tenham clientes relacionados.
- 3) Identifique qual o material é utilizado por mais produtos e em seguida liste a quantidade de pedidos realizados, com produtos compostos por este material, liste também o valor total de vendas dos últimos 60 dias.

## BLOCOS ANÔNIMOS: arrays e/ou tabelas variáveis

É possível declarar uma *table-variable* com a estrutura igual a uma tabela permanente, porém não permite a criação de índices e estatísticas.

Não utiliza transação (mesmo após o rollback os registros permanecerão).

```
DECLARE @tabela TABLE
    (<ColumnName> <DataType> (<length>) <isnull> <constraint>
    , ...
)
```

» Exemplo de declaração:

```
DECLARE @vCidade TABLE
    (IDCidade      Int           not null PRIMARY KEY,
     Nome          Varchar(50)  not null,
     UF            Varchar(2)   not null)
```

## BLOCOS ANÔNIMOS: arrays e/ou tabelas variáveis

Permite qualquer operação (dml) sobre ela.

É recomendado para processos para armazenar um volume pequeno de registros/dados. Se o processo necessitar um volume muito grande de dados utilize tabelas temporárias (#).

```
...  
DECLARE @vCidade TABLE  
    (IDCidade      Int           not null PRIMARY KEY,  
     Nome          Varchar(50) not null,  
     UF            Varchar(2)  not null)  
  
Insert into @vCidade (IDCidade, Nome, UF)  
    Select IDCidade,  
           Nome,  
           UF  
    From    Cidade;  
...
```

## BLOCOS ANÔNIMOS: arrays e/ou tabelas variáveis

```
BEGIN TRANSACTION
SET NOCOUNT ON
DECLARE @AuxCidade table (IDCidade integer)

Insert into @AuxCidade values (999)
Insert into @AuxCidade values (457)
Insert into @AuxCidade values (558)

Update Cidade
Set     Nome = Nome + '*'
Where   IDCidade in (Select IDCidade from @AuxCidade)

Print 'Cidades alteradas: ' + cast(@@rowcount as varchar(10))

Delete @AuxCidade

SET NOCOUNT OFF

ROLLBACK TRANSACTION
```

## BLOCOS ANÔNIMOS: tabelas temporárias

Permite definir uma tabela que será armazenada na área temporária do SQL Server (TempDB).

Pode ser definido no escopo local (#) ou global (##).

Utiliza transações.

É recomendada para processos onde o volume de dados é maior (relatórios, etc).

» Exemplo tabela temporária **local**:

```
Create table #TempCidade
( IDCidade int not null primary key,
  Nome varchar(50) not null,
  UF varchar(2) not null
);
```

» Veja um exemplo no Wiki:

<http://10.0.100.9/cwiwiki/CWI%20-%20N%C3%BAcleo%20de%20Tecnologia%20-%20SQL%20Server%20Primary%20key%20em%20Temporarias.ashx>

# BLOCOS ANÔNIMOS - exercício



- 4) Faça um bloco T-SQL que liste todos os Clientes que estão relacionados com as cidades duplicadas (Nome e UF). Não considere as cidades duplicadas de menor IDCidade.

IDCliente	Nome Cliente	IDCidade	Nome Cidade
879	Juvenilsa Allebrandt Faller	4285	Araguari
882	Bernardete Rafaela Knevit	1964	Cachoeirinha
886	Juscelio Branquier Coutinho	1585	Chuí
887	Uiliane Toquetto Alcantra	4287	São Gabriel
891	Zemario Delvino Pietroski	4011	Jaguapita

» Exemplo.

## PROCEDURES - stored procedure

É um procedimento armazenado dentro do metadado da base de dados. Contém um bloco T-SQL e permite que tenha parâmetros de entrada e saída.

```
CREATE PROCEDURE [owner.]<procedure_name>  
    [@parameter data_type] [outuput]  
AS  
BEGIN  
    [comandos_sql]  
END
```

Referência: <http://msdn.microsoft.com/en-us/library/aa258259%28SQL.80%29.aspx>

# PROCEDURES - stored procedure

Exemplo de criação de um procedimento:

```
CREATE PROCEDURE Prc_Saudacao AS
BEGIN
    SET NOCOUNT ON

    DECLARE @vHora Int
    SET @vHora = DATEPART(HOUR, GETDATE())

    IF (@vHora) < 12
        Print 'Bom dia'
    ELSE IF (@vHora >= 12) AND (@vHora < 18)
        Print 'Boa tarde'
    ELSE
        Print 'Boa noite'

    SET NOCOUNT OFF
END
```

Procedure simples, sem uso de parâmetros.



## PROCEDURES - stored procedure

Executando uma procedure:

```
Execute Prc_Saudacao
```

Pode ser chamada diretamente, ou ainda dentro de outra procedure.

Para alterar o código de uma procedure é necessário apenas substituir o "Create" por "Alter".

## PROCEDURES - stored procedure - parâmetro de entrada

Procedure que listará todas as cidades do estado informado:

```
CREATE PROCEDURE Prc_RelatorioCidade
    @UF Varchar(2) AS
BEGIN
    SET NOCOUNT ON

    Select IDCidade, Nome
    From    Cidade WITH (NOLOCK)
    Where   UF = @UF

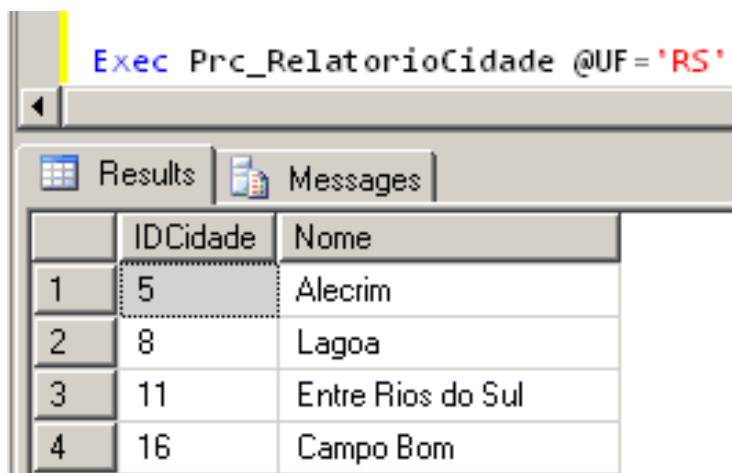
    SET NOCOUNT OFF
END
```

Utilização de um parâmetro de entrada.

A opção “with (nolock)” é recomendado em consultas de relatórios.

## PROCEDURES - stored procedure

Executando procedimento que resulta na lista de cidades



The screenshot shows a SQL client window with a command bar containing the text `Exec Prc_RelatorioCidade @UF='RS'`. Below the command bar, there are two tabs: "Results" and "Messages". The "Results" tab is active, displaying a table with the following data:

	IDCidade	Nome
1	5	Alecrim
2	8	Lagoa
3	11	Entre Rios do Sul
4	16	Campo Bom

O resultado da execução desta procedure será o mesmo que executar diretamente a consulta SQL.

```
Exec Prc_RelatorioCidade @UF='RS'
```

## PROCEDURES - stored procedure - parâmetro de saída

Procedure que retornará o nome e preço de custo do produto:

```
Create procedure Prc_BuscaProduto
    (@pIDProduto int,
    @pNome        varchar(50)  OUTPUT,
    @pPrecoCusto  decimal(18,2) OUTPUT) as
BEGIN
    SET NOCOUNT ON
    Select @pNome        = Nome
           ,@pPrecoCusto = PrecoCusto
    From    Produto
    Where   IDProduto = @pIDProduto

    IF (@@RowCount = 0) BEGIN
        Set @pNome = '**Nao Encontrado!'
        Set @pPrecoCusto = -1
    END
    SET NOCOUNT OFF
END;
```

Utilização de um parâmetro de entrada e saída (output).

Foi feito uma validação para verificar caso não exista o registro.

## PROCEDURES - stored procedure - parâmetro de saída

Executando o procedimento:

```
BEGIN
  DECLARE @nome varchar(50),
          @preco decimal(18,2)

  Exec Prc_BuscaProduto @pIDProduto=1934,
                      @pNome=@nome OutPut,
                      @pPrecoCusto=@preco OutPut

  Print @nome
END
```

Neste exemplo o retorno do parametro PrecoCusto não foi utilizado.

Exemplo de chamada em ASP.NET:

Fonte: <http://www.sqlteam.com/article/stored-procedures-returning-data>

## PROCEDURES - exercícios



- 5) Faça uma (ou várias) procedure que elimine as cidades duplicadas. Deve permanecer o registro duplicado de menor IDCidade, os clientes devem ser acertados (update) antes dessa exclusão (delete). Se necessário utilizem tabelas temporárias, variáveis.
- 6) Faça um procedimento que liste um relatório (select) dos pedidos que tenham produto que precisa de determinado Material. Parâmetros de entrada: IDMaterial (int), pAnoMes (int) Através do material informado, identificar os produtos que o utilizam e depois buscar os pedidos (PedidoItem) desses produtos...

# PROCEDURES - exercícios



- 6) (continuação) Para o período informado deve ser considerado a data de entrega do pedido. Abaixo um exemplo do resultado esperado:

IDPedido	DataPedido	DataEntrega	IDCliente	NomeCliente	Cidade	UF	Valor Pedido
1293	19/8/2010	27/8/2010	123	Joao da Silva	Sao Paulo	SP	1.293,40
1405	19/8/2010	30/8/2010	345	Maria Betanha	Salvador	BA	3.491,90

## FUNCTIONS - funções de usuário

- As funções permitem receber um ou mais parâmetros, não permitem parâmetros do tipo OUT, e sempre devem retornar um valor.
- Não permitem instruções de manipulação de dados (Insert, Update ou Delete).
- Três tipos de funções:
  - o Scalar;
  - o Inline Table-valued;
  - o Multi-statement Table-valued;

Referência: <http://msdn.microsoft.com/pt-br/library/ms186755.aspx>



## FUNCTIONS - funções de usuário - *scalar*

- Permite retornar um valor apenas, conforme o exemplo abaixo:

```
CREATE FUNCTION FctBuscaCliente (@pIDCliente INT)
    Returns Varchar(50) AS
Begin
    -- VARIÁVEL QUE RECEBERÁ O NOME DA CONSULTA (SQL)
    Declare @NomeRetorno varchar(50)

    -- BUSCA O NOME E ATRIBUI O VALOR PARA A VARIÁVEL
    Select @NomeRetorno = Nome
    From    Cliente
    Where   IDCliente = @pIDCliente

    -- VERIFICA SE ENCONTROU ALGUM REGISTRO --
    if (@@RowCount=0)
        Set @NomeRetorno = '***Cliente Inexistente!'

    -- RETORNA VARIÁVEL LOCAL
    Return @NomeRetorno
End
```

Função que retorna o nome do cliente, conforme o IDCliente informado através de parâmetro.

## FUNCTIONS - funções de usuário - *scalar*

- Executando uma função escalar:

```
Select dbo.FctBuscaCliente(35) As NomeCliente
```

Results		Messages	
		NomeCliente	
1		Aurenilza Flademir Meneghel	

```
Select ped.IDPedido,
       ped.IDCliente,
       dbo.FctBuscaCliente(ped.IDCliente) as NomeCliente
From Pedido ped
Where ped.IDPedido in (1,2,3,4,5)
```

Results		Messages	
	IDPedido	IDCliente	NomeCliente
1	1	11674	Persivaldo Link Rodrigues
2	2	11768	Ondina Olimpio Gottfried
3	3	2942	Elecario Caio Prudente
4	4	4587	Deriomar Cleni Caurio
5	5	5368	Linsmar Jussiane Cremonini

## FUNCTIONS - funções de usuário - *inline-table-valued*

- Permite executar uma consulta utilizando parâmetros de entrada.

```
Create function FctRelatorioPedido (@pIDPedido int)
Returns Table as
--
Return Select ped.DataPedido,
              ped.DataEntrega,
              ped.IDCliente,
              dbo.FctBuscaCliente(ped.IDCliente) as NomeCliente,
              ped.ValorPedido,
              ped.Situacao
From Pedido ped
Where ped.IDPedido = @pIDPedido;
```

Função que retorna a consulta de informações referente ao Pedido, conforme o parâmetro de entrada “pIDPedido”.

## FUNCTIONS - funções de usuário - *inline-table-valued*

- Executando uma função Inline table-value

```
Select *
  From dbo.FctRelatorioPedido(3500) consulta
```

>> Consultando todas as colunas

Results		Messages				
	DataPedido	DataEntrega	IDCliente	NomeCliente	ValorPedido	Situacao
1	2003-09-12 01:53:45.000	2003-09-22 01:53:45.000	9857	Cleia Casarin Feminio	44319.24	Q

```
Select DataPedido, NomeCliente, ValorPedido
  From dbo.FctRelatorioPedido(12345) consulta
```

>> Consultando colunas específicas.

Results		Messages	
	DataPedido	NomeCliente	ValorPedido
1	2003-09-18 15:08:46.000	Aleude Alain Habib	7386.54

## FUNCTIONS - funções de usuário - *multi-statement-table-valued*

- Permite executar mais de uma consulta, conforme parametrização.
- Execute o código abaixo para auxiliar na execução do próximo exemplo:

```
Use Crescer13i
Go

Alter table Cliente Add TipoCliente Int;

Update Cliente
Set TipoCliente = 1
Where IDCliente < 1000;

Update Cliente
Set TipoCliente = 2
Where IDCliente >= 1000
And IDCliente < 2000;
```

## FUNCTIONS - funções de usuário - *multi-statement-table-valued*

- Exemplo com opção para execução de 2 consultas:

```
CREATE Function FctListaCliente (@pTipo int=1, @pSituacao
Varchar(1)=NULL)
  RETURNS @TabelaCliente table(Codigo INT PRIMARY KEY,
                                Nome varchar(50),
                                Cidade varchar(50),
                                Situacao varchar(10)) AS

BEGIN
  IF (@pTipo=1) -- Fisica
    INSERT INTO @TabelaCliente
      SELECT cli.IDCliente,
             cli.Nome,
             cid.Nome as Cidade,
             case when Situacao = 'A' then 'Ativo'
                  else 'Inativo'
             end Situacao
    FROM Cliente cli
      LEFT JOIN Cidade cid ON cid.IDCidade = cli.IDCliente
    WHERE cli.TipoCliente = 1
    AND   (@pSituacao IS NULL or Situacao = @pSituacao)
```

Primeira parte do código.

## FUNCTIONS - funções de usuário - *multi-statement-table-valued*

- Permite executar uma consulta utilizando parâmetros de entrada.

```
ELSE IF (@pTipo=2) -- Juridica
    INSERT INTO @TabelaCliente
        SELECT cli.IDCliente,
               cli.RazaoSocial,
               cid.Nome as Cidade,
               case when Situacao = 'A' then 'Ativo'
                    else 'Inativo'
               end Situacao
    FROM Cliente cli
        LEFT JOIN Cidade cid ON cid.IDCidade = cli.IDCliente
    WHERE cli.TipoCliente = 2
        AND (@pSituacao IS NULL or Situacao = @pSituacao)

RETURN
END
```

Segunda parte do código.

## FUNCTIONS - funções de usuário - *multi-statement-table-valued*

- Executando uma função que retorna uma tabela (variável):

```
Select * From dbo.FctListaCliente (2, null)
```

	Codigo	Nome	Cidade	Situacao
1	1000	Gradual Pinturas Sc Ltda	Virginopolis	Ativo
2	1001	Uniao Agro Ara Ind Com Alims L	Salete	Inativo
3	1002	Silvana Villela	Malhada de Pedras	Ativo
4	1003	Restaurante Artes Embu Ltda	Jacarei	Ativo
5	1004	Garutti & Severino Ltda	Alto Alegre do Pindare	Inativo
6	1005	Ind Grafica Intergrafica Ltda	Araruama	Ativo
7	1006	Lidiane Judite Cabrioli Epp	Mundo Novo	Ativo



## TRIGGERS - gatilhos de tabelas

É um tipo de stored procedure que é chamado/executado através de determinado evento ([insert](#), [update](#) ou [delete](#)) de uma tabela.

É possível definir quando será executada a trigger:

AFTER: após a operação (*dml*);

INSTEAD OF: substitui a operação (ao invés de).

Existe a possibilidade de criar gatilhos (trigger) a nível de database, para monitorar comandos DDL e logon/logout em cada database.

Referência: <http://msdn.microsoft.com/pt-br/library/ms189799%28v=SQL.90%29.aspx>

## TRIGGERS - gatilhos de tabelas: *exemplo 1*

Abaixo um exemplo onde o bloco T-SQL do gatilho irá substituir os comandos de INSERT, UPDATE e DELETE quando executados.

```
Create trigger Trg_LockCidade On Cidade
  INSTEAD OF Insert, Update, Delete
  AS
  BEGIN

    RAISERROR('Nao e possivel alterar o cadastro de cidades!!!', 16, 1);

  END
```

Este código serve para garantir que nenhuma alteração seja realizada sobre a tabela Cidade.

Referência: <http://msdn.microsoft.com/pt-br/library/ms189799%28v=SQL.90%29.aspx>

## TRIGGERS - gatilhos de tabelas: *exemplo 2*

Abaixo um exemplo onde todo novo registro será salvo em outra tabela

```
Create trigger Trg_AuditCliente_Novo On Cliente  FOR Insert  AS
BEGIN

    Declare @vUsuario varchar(128);
    Set NoCount On

    Set @vUsuario = CAST(CONTEXT_INFO() as varchar(128));

    INSERT INTO LogCliente (Operacao, Usuario, IDCliente, Nome, RazaoSocial)
        SELECT 'I' as Operacao,
               @vUsuario,
               ins.IDCliente,
               ins.Nome,
               ins.RazaoSocial
        FROM    INSERTED ins;

    Set NoCount Off
END;
```

Slide 51

Slide 52

Slide 54

## TRANSACT-SQL: context\_info()

Permite que seja definida uma informação na sessão do usuário. Que informe o contexto da operação ou identificação do usuário dentro da aplicação.

Para definir o contexto dentro de uma sessão (conexão) do banco é necessário definir o CONTEXT\_INFO, conforme o exemplo abaixo:

```
DECLARE @BinVar varbinary(128)
SET @BinVar = CAST('AndreNunes' AS varbinary(128) )
SET CONTEXT_INFO @BinVar
```

Para consultar o valor atual do CONTEXT\_INFO() basta atribuir o valor para uma variável.

```
SELECT CAST(CONTEXT_INFO() as varchar(128)) AS ContextoAtual
```

Referência: <http://msdn.microsoft.com/pt-br/library/ms187768.aspx>

## TRIGGERS - gatilhos de tabelas: *executando*

Estrutura da tabela "LogCliente", utilizada no exemplo do slide 41:

```
Create table LogCliente (
  IDLogCliente int identity          not null,
  Data          datetime default getdate() not null,
  Operacao      char(1)              not null,
  Usuario       varchar(128),
  IDCliente     int,
  Nome          varchar(50),
  RazaoSocial   varchar(50),
  constraint PK_LogCliente primary key (IDLogCliente)
)
```

Inserindo um registro na tabela de cliente (fazendo que a trigger seja executada):

```
Insert into Cliente
  (nome, razaoSocial, situacao)
Values
  ('Coca-Cola do Brasil', 'Vonpar Distribuidora Ltda', 'A')
```

## TRIGGERS - gatilhos de tabelas: *consultando o log*

Consultando a tabela de logs, após inserir um registro:

```
Select * From LogCliente
```

Resultado da consulta:

Results		Messages					
	IDLogCliente	Data	Operacao	Usuario	IDCliente	Nome	RazaoSocial
1	1	2010-10-18 14:35:47.927	I	AndreNunes	12502	Coca-Cola do Brasil	Vonpar Distribuidora Ltda

## TRIGGERS - gatilhos de tabelas: *valores antigos e novos*

Para identificar os valores novos e antigos dentro de um gatilho é preciso utilizar uma estrutura de memória oferecida pelo SQL Server, chamadas de INSERTED e DELETED, onde:

→ **INSERTED**: contém os novos valores, registro de uma inserção (insert) ou colunas definidas por uma alteração (update).

→ **DELETED**: contém os antigos valores, registro excluído e/ou valores originais de uma alteração.

Está estrutura possui as mesmas colunas da tabela definida na trigger.

Para identificar os valores antigos e novos de um UPDATE é necessário relacionar as visões (inserted e deleted) pelo ID da tabela.

## TRIGGERS - gatilhos de tabelas: *tipo de operação*

Para identificar o tipo de operação que está sendo executando em um gatilho é preciso verificar as visões INSERTED e DELETED, conforme o exemplo abaixo:

```
Create trigger Trg_AuditMaterial On Material
  AFTER Insert, Update, Delete AS
BEGIN

  IF EXISTS(Select top(1) 1 From Inserted)
    IF EXISTS(Select top(1) 1 From Deleted)
      Print 'Operacao de Update'
    ELSE
      Print 'Operacao de Insert'
  ELSE
    Print 'Operacao de Delete'

END ;
```



## TRIGGERS - gatilhos de tabelas: *testando colunas*

É possível verificar se determinadas colunas foram acionadas pelo comando de INSERT ou UPDATE. Para isso deve ser utilizada a função UPDATE().

Verifique um exemplo abaixo:

```
Create trigger Trg_AuditProduto On Produto
  AFTER Update AS
BEGIN

  IF UPDATE(PrecoCusto)
    Print 'Preco de custo foi alterado'

END;
```

Esta função não valida se o valor antigo e novo são diferentes, apenas verifica se a coluna foi definida na cláusula SET.

# André Nunes

## Núcleo de Tecnologia

andrenunes@cwí.com.br

(51) 3081.3622