

Tarefa Orientada 17

Scripts

Objectivos:

- Declaração de variáveis escalares
- Declaração de variáveis do tipo tabela
- Tabelas temporárias
- Estruturas Condicionais
- Estruturas de Repetição
- Tratamento de erros

De um modo simplista, um *script* é um ficheiro que está dividido em um ou mais grupos que contêm uma série de instruções SQL. Cada grupo é executado como uma unidade.

Para definir o fim de um grupo, utilize o comando *GO*. Não é necessário utilizar este comando para definir o fim do último grupo de um *script*, nem no caso do *script* ser constituído por apenas um grupo.

1 Formule, analise e execute as instruções a seguir apresentadas.

1.1 *Script* constituído por dois grupos. O primeiro cria uma base de dados e o segundo cria três tabelas nessa base de dados.

```
CREATE DATABASE ClubeRicos
GO

USE ClubeRicos

CREATE TABLE Membros
(IDMembro int NOT NULL IDENTITY PRIMARY KEY,
ÚltimoNome varchar(75) NOT NULL,
PrimeiroNome varchar(50) NOT NULL)

CREATE TABLE Comitês
(IDComitê int NOT NULL IDENTITY PRIMARY KEY,
Nome varchar(50) NOT NULL)

CREATE TABLE MembrosComitês
(IDMembro int NOT NULL REFERENCES Membros(IDMembro),
IDComitê int NOT NULL REFERENCES Comitês(IDComitê))
```

Caso utilize num *script* as instruções *CREATE VIEW*, *CREATE PROCEDURE*, *CREATE FUNCTION* ou *CREATE TRIGGER*, deve criar um grupo para cada instrução.

Declaração de variáveis escalares

Para declarar uma variável, utilize a instrução *DECLARE*. O valor inicial de uma variável é sempre *NULL*. O nome da variável deve começar com o carácter @. A seguir apresenta-se a sintaxe para declarar uma variável.

```
DECLARE @nome_variável Tipo_de_dados [, @nome_variável Tipo_de_dados] ...
```

Uma variável à qual é associado um tipo de dados standard e que contém um único valor é comumente designada por variável escalar.

Para atribuir um valor a uma variável, utilize a instrução *SET*. A seguir apresenta-se a sintaxe para atribuir um valor a uma variável. Por exemplo, no *script* abaixo apresentado, usa-se esta técnica para atribuir valores às variáveis *@MaxFactura* e *@VarIDFornecedor*.

```
SET @nome_variável = expressão
```

Alternativamente, também se pode atribuir um valor a uma variável a partir da lista de colunas de uma instrução *SELECT*. Por exemplo, no *script* seguinte usa-se esta técnica para atribuir valores às variáveis *@MinFactura* e *@NúmFacturas*.

1.2 *Script* que ilustra a declaração e atribuição de valores a variáveis escalares.

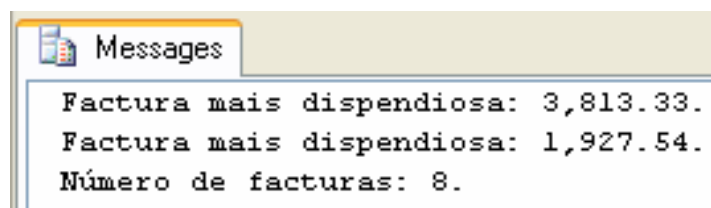
```
USE Pagamentos

DECLARE @MaxFactura money, @MinFactura money
DECLARE @VarIDFornecedor int, @NúmFacturas int

SET @VarIDFornecedor = 122
SET @MaxFactura = (SELECT MAX(TotalFactura) FROM Facturas
                  WHERE Fornecedor = @VarIDFornecedor)
SELECT @MinFactura = MIN(TotalFactura), @NúmFacturas = COUNT(*)
FROM Facturas
WHERE Fornecedor = @VarIDFornecedor

PRINT 'Factura mais dispendiosa: ' + CONVERT(varchar,@MaxFactura,1) +
      '.'
PRINT 'Factura mais dispendiosa: ' + CONVERT(varchar,@MinFactura,1) +
      '.'
PRINT 'Número de facturas: ' + CONVERT(varchar,@NúmFacturas) + '.'
```

O resultado da execução do *script* anterior é o seguinte.



```
Messages
Factura mais dispendiosa: 3,813.33.
Factura mais dispendiosa: 1,927.54.
Número de facturas: 8.
```

O âmbito de uma variável é o grupo em que foi declarada, isto é, a variável não pode ser referenciada fora do grupo em que foi declarada.

Declaração de variáveis do tipo tabela

Também é possível declarar variáveis que possam armazenar o conteúdo de uma tabela, através da instrução *DECLARE*.

```
DECLARE @nome_da_tabela TABLE
(coluna1 tipo_de_dados [restrições_coluna1]
[, coluna2 tipo_de_dados [restrições_coluna2]]...
[,restrições_tabela])
```

Note que, na declaração da variável, é utilizado o tipo de dados *TABLE* em vez de um tipo de dados standard. Depois, definem-se as colunas e as respectivas restrições de igual modo ao utilizado para criar uma tabela através da instrução *CREATE TABLE*.

Tal como no caso das variáveis escalares, o âmbito de uma variável do tipo tabela é o grupo em que foi declarada.

1.3 *Script* que ilustra a declaração e atribuição de valores a variáveis do tipo tabela.

```
USE Pagamentos

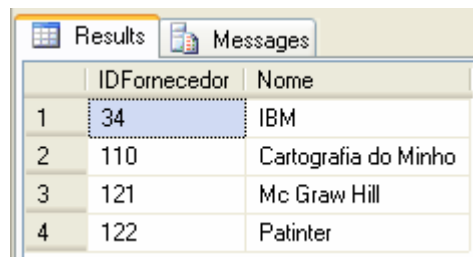
DECLARE @MelhoresFornecedores table
(IDFornecedor int,
 Nome varchar(50))

INSERT @MelhoresFornecedores
SELECT IDFornecedor, Nome
FROM Fornecedores
WHERE IDFornecedor IN (SELECT Fornecedor FROM Facturas WHERE
TotalFactura > 1000)

SELECT * FROM @MelhoresFornecedores
```

Neste exemplo, é declarada a variável *@MelhoresFornecedores* para que possa armazenar duas colunas: *IDFornecedor* e *Nome*. Depois é utilizada uma instrução *INSERT* para inserir, nessa variável, todos os fornecedores que têm facturas com montantes superiores a mil euros. Finalmente, é usada uma

instrução *SELECT* para devolver o conteúdo da variável *@MelhoresFornecedores*.



	IDFornecedor	Nome
1	34	IBM
2	110	Cartografia do Minho
3	121	Mc Graw Hill
4	122	Patinter

Note ainda que, neste exemplo, foi utilizada a variável do tipo tabela nas instruções *INSERT* e *SELECT*. Também pode utilizar uma variável do tipo tabela, em vez do nome de uma tabela, nas instruções *UPDATE* e *DELETE*. A única situação em que não pode usar uma variável do tipo tabela, no lugar do nome de uma tabela é na cláusula *INTO* de uma instrução *SELECT INTO*.

Tabelas temporárias

Uma tabela temporária existe apenas durante uma sessão da base de dados. Na aplicação *Management Studio*, isto significa que a tabela apenas está disponível até que feche a janela onde criou a tabela. As tabelas temporárias são úteis para testar consultas ou para armazenar dados temporariamente num *script* complexo.

As tabelas temporárias são armazenadas na base de dados do sistema *tempdb*. Se necessitar de eliminar uma tabela temporária antes de finalizar uma sessão, utilize o comando *DROP TABLE*.

Uma tabela temporária está visível apenas dentro da sessão corrente. Para identificar uma tabela temporária local, anteceda o nome da tabela com o carácter #.

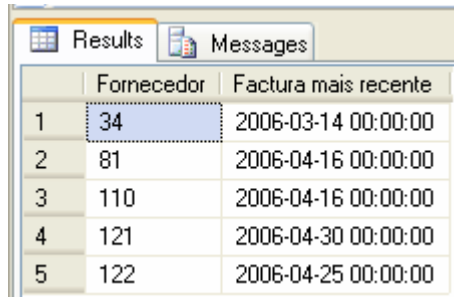
1.4 *Script* que ilustra a utilização de uma tabela temporária.

```
USE Pagamentos

SELECT TOP 5 Fornecedor,
           AVG(TotalFactura) AS [Montante médio das facturas]
INTO #MelhoresFornecedores
FROM Facturas
GROUP BY Fornecedor
ORDER BY [Montante médio das facturas] DESC

SELECT Facturas.Fornecedor,
       MAX(DataFactura) AS [Factura mais recente]
FROM Facturas JOIN #MelhoresFornecedores
ON Facturas.Fornecedor = #MelhoresFornecedores.Fornecedor
GROUP BY Facturas.Fornecedor
```

Neste exemplo, é criada uma tabela temporária com o nome *#MelhoresFornecedores* através da instrução *SELECT INTO*. Esta tabela temporária contém o identificador do fornecedor e a média do montante das respectivas facturas dos fornecedores que têm os cinco maiores montantes médios das facturas. Depois, a segunda instrução *SELECT* realiza uma junção interna entre a tabela temporária e a tabela *Fornecedores*, de modo a devolver a data das facturas mais recentes desses fornecedores.



	Fornecedor	Factura mais recente
1	34	2006-03-14 00:00:00
2	81	2006-04-16 00:00:00
3	110	2006-04-16 00:00:00
4	121	2006-04-30 00:00:00
5	122	2006-04-25 00:00:00

Note ainda que, apesar de a tabela temporária ser armazenada noutra base de dados (*tempdb*), não é necessário especificar o nome dessa base de dados, pois o nome da tabela identifica-a como sendo uma tabela temporária.

Estruturas Condicionais

Pode utilizar a instrução *IF ... ELSE* para testar uma expressão condicional. Se essa expressão for verdadeira, as instruções que seguem a palavra-chave *IF* são executadas. Caso contrário, as instruções seguintes à palavra-chave *ELSE* são executadas, se essa palavra-chave estiver incluída.

```
IF expressão_condicional
{Instrução | BEGIN ... END}
[ELSE
{Instrução | BEGIN ... END}]
```

1.5 Script que ilustra a utilização de uma instrução *IF ... ELSE*.

```
USE Pagamentos

DECLARE @MenorDívida money, @MaiorDívida money
DECLARE @MenorDataDívida smalldatetime
DECLARE @MaiorDataDívida smalldatetime

SELECT @MenorDívida = MIN(TotalFactura - Pagamento - Crédito),
       @MaiorDívida = MAX(TotalFactura - Pagamento - Crédito),
       @MenorDataDívida = MIN(DataVencimentoFactura),
       @MaiorDataDívida = MAX(DataVencimentoFactura)
FROM Facturas
WHERE TotalFactura - Pagamento - Crédito > 0

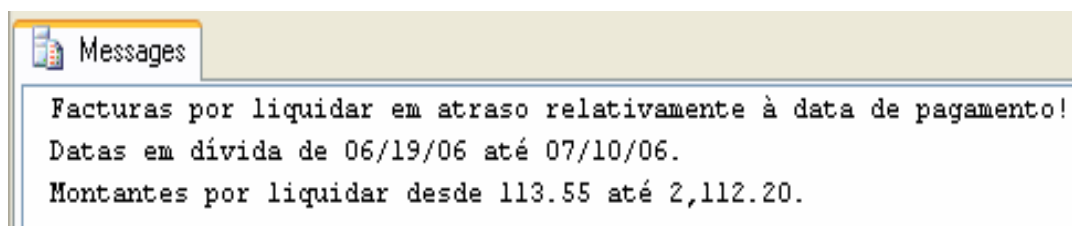
IF @MenorDataDívida < GETDATE()
BEGIN
    PRINT 'Facturas por liquidar em atraso relativamente à data
          de pagamento!'
    PRINT 'Datas em dívida de' + CONVERT(varchar,@MenorDataDívida,1) +
          ' até ' + CONVERT(varchar,@MaiorDataDívida,1) + '.'
    PRINT 'Montante em dívida desde ' + CONVERT(varchar,@MenorDívida,1)
          + ' a ' + CONVERT(varchar,@MaiorDívida,1) + '.'
END
ELSE --@MenorDataDívida >= GETDATE()
    PRINT 'Não existem facturas com atrasos de pagamento'
```

Neste exemplo utiliza-se uma instrução *SELECT* para atribuir valores a quatro variáveis. As variáveis *@MenorDívida* e *@MaiorDívida* vão conter os valores das facturas não saldadas que têm a menor e a maior dívida, respectivamente. As variáveis *@MenorDataDívida* e *@MaiorDataDívida* vão

conter as datas das facturas não saldadas que têm a menor (mais antiga) e a maior (mais recente) data de vencimento, respectivamente.

Se a data de dívida mais antiga for inferior à data do sistema (devolvida pela função *GETDATE*), devem ser apresentadas três mensagens. Caso contrário, é apresentada a mensagem *Não existem facturas com atrasos de pagamento*.

A seguir, apresenta-se o resultado da execução do *script* anterior.



Pode aninhar instruções *IF ... ELSE* dentro de instruções *IF ... ELSE*.

Antes de trabalhar com um objecto da base de dados, deve-se assegurar que o objecto existe. Do mesmo modo, antes de criar um novo objecto, também se deve garantir que esse objecto ainda não existe. Para tal, utilize as funções *OBJECT_ID* e *DB_ID*.

A função *OBJECT_ID* permite verificar a existência de uma tabela, de uma vista, de um procedimento armazenado, de uma função definida pelo utilizador ou de um trigger. A seguir apresenta-se a sintaxe para a função *OBJECT_ID*.

```
OBJECT_ID('nome_do_objecto')
```

A função *DB_ID* permite verificar a existência de uma base de dados. A seguir apresenta-se a sintaxe para a função *DB_ID*.

```
DB_ID('nome_da_base_de_dados')
```

Ambas as funções devolvem um valor nulo, caso o objecto não exista. Caso contrário, devolvem o número de identificação desse objecto.

1.6 *Script* que verifica se existe uma determinada base de dados antes de a eliminar.

```
USE master

IF DB_ID('Teste') IS NOT NULL
    DROP DATABASE Teste

CREATE DATABASE Teste
```

1.7 *Script* que verifica se existe a tabela *CópiaFacturas* na base de dados corrente.

```
USE Pagamentos

IF OBJECT_ID('CópiaFacturas') IS NOT NULL
    DROP TABLE CópiaFacturas
```

1.8 Maneira alternativa de verificar a existência da tabela *CópiaFacturas* na base de dados corrente.

```
USE Pagamentos

IF EXISTS (SELECT * FROM sys.tables
           WHERE name = 'CópiaFacturas')
    DROP TABLE CópiaFacturas
```

1.9 Código que verifica a existência de uma tabela temporária.

```
USE Pagamentos

SELECT * FROM #MelhoresFornecedores

IF OBJECT_ID('#MelhoresFornecedores') IS NOT NULL
    DROP TABLE #MelhoresFornecedores

SELECT * FROM #MelhoresFornecedores
```

Estruturas de repetição

Para executar repetidamente uma instrução ou um conjunto de instruções, utilize o comando *WHILE*.

```
WHILE expressão_condicional
    {Instrução | BEGIN ... END}
    [BREAK]
    [CONTINUE]
```

A(s) intrução(ões) contida(s) num comando *WHILE* são repetidas enquanto a expressão condicional for verdadeira. Para sair de um ciclo *WHILE* sem testar a expressão condicional, utilize a instrução *BREAK*. Para voltar ao início de um ciclo *WHILE* sem executar quaisquer instruções adicionais no ciclo, utilize a instrução *CONTINUE*.

1.10 Apesar de este *Script* representar uma situação não muito realista, serve para ilustrar a utilização de um ciclo *WHILE*. Um exemplo mais realista seria usar um ciclo *WHILE* para processar cursores. Voltaremos a este assunto mais tarde.

```
USE Pagamentos

IF OBJECT_ID('tempdb..#CópiaFacturas') IS NOT NULL
DROP TABLE #CópiaFacturas

SELECT * INTO #CópiaFacturas FROM Facturas
WHERE TotalFactura - Pagamento - Crédito > 0

WHILE (SELECT SUM(TotalFactura - Pagamento - Crédito)
FROM #CópiaFacturas) >= 50000
BEGIN
    UPDATE #CópiaFacturas
    SET Crédito = Crédito + .01
    WHERE TotalFactura - Pagamento - Crédito > 0

    IF (SELECT MAX(Crédito) FROM #CópiaFacturas) > 3000
        BREAK
    ELSE --(SELECT MAX(Crédito) FROM #CópiaFacturas) <= 3000
        CONTINUE
END

SELECT DataFactura, TotalFactura, Crédito
FROM #CópiaFacturas
```

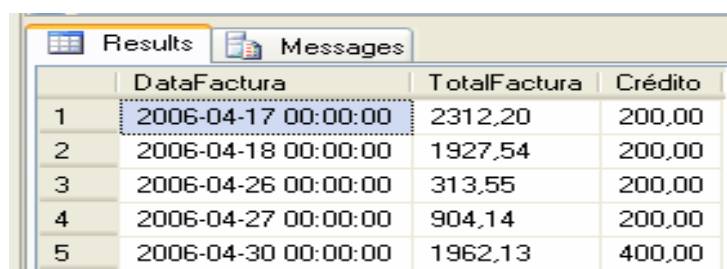
Neste *script* começasse por criar uma tabela temporária com o nome *#CópiaFacturas* que vai conter as facturas ainda por saldar da tabela *Facturas*. Depois, a expressão de teste do ciclo *WHILE* inclui uma instrução *SELECT* para devolver o montante total em dívida das facturas por saldar. Caso esse montante for superior ou igual a 50000 euros, as restantes instruções são executadas. Caso contrário, o ciclo acaba.

A instrução *UPDATE* englobada no ciclo actualiza o valor do campo crédito da tabela temporária em 1%, para as facturas ainda por saldar. Note que apesar da tabela inicialmente apenas conter facturas por saldar, essa situação pode alterar-se à medida que os créditos vão sendo aplicados dentro do ciclo.

Depois, é utilizada uma instrução *IF...ELSE* para verificar se o montante máximo do crédito é superior a 3000 euros. Em caso afirmativo, Usa-se uma instrução *BREAK* para terminar abruptamente o ciclo. Caso contrário, é utilizada uma instrução *CONTINUE*, de modo a que o controlo volte automaticamente ao início do ciclo. Depois, a condição do teste é avaliada novamente e repetem-se os passos descritos anteriormente.

Note que, neste caso a instrução *CONTINUE* não é necessária, pois é a última instrução do ciclo; o que significa que o controlo voltaria, na mesma, ao início do ciclo. A instrução *CONTINUE* pode ser utilizada numa cláusula *IF* para desviar as restantes instruções do ciclo. Contudo, tal como no caso da utilização da instrução *BREAK*, isto torna a leitura do código mais difícil.

A seguir, apresenta-se o resultado da execução do *script* anterior.



	DataFactura	TotalFactura	Crédito
1	2006-04-17 00:00:00	2312,20	200,00
2	2006-04-18 00:00:00	1927,54	200,00
3	2006-04-26 00:00:00	313,55	200,00
4	2006-04-27 00:00:00	904,14	200,00
5	2006-04-30 00:00:00	1962,13	400,00

Tratamento de erros

Para efectuar o tratamento de erros pode utilizar a instrução *TRY ... CATCH*. Este procedimento também se pode designar por tratamento de excepções.

```
BEGIN TRY
    {Instrução_SQL | bloco_de_instruções}
END TRY
BEGIN CATCH
    {Instrução_SQL | bloco_de_instruções}
END CATCH
```

Quando ocorre um erro numa instrução incluída num bloco *TRY*, o controlo é passado para o bloco *CATCH* onde o erro pode ser processado. Se não ocorrer nenhum erro nas instruções do bloco *TRY*, o bloco *CATCH* é ignorado.

Para dar uma informação mais personalizada acerca do erro, pode utilizar as seguintes funções dentro de um bloco *CATCH*.

ERROR_NUMBER() - devolve o número do erro

ERROR_MESSAGE() - devolve a mensagem associada ao erro

ERROR_SEVERITY() - devolve a severidade do erro

ERROR_STATE() - devolve o estado do erro

Os erros com uma severidade igual ou inferior a 10 são considerados *Warnings* e não são tratados pela instrução *TRY ... CATCH*. Por outro lado, erros que tenham uma severidade igual ou superior a 20 e que causem o fecho da ligação à base de dados também não são tratados pela instrução *TRY ... CATCH*.

1.11 Script que ilustra a utilização da instrução *TRY ... CATCH*.

```

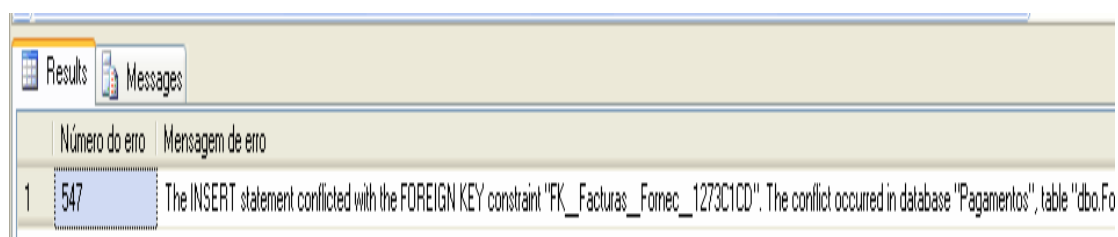
USE Pagamentos

GO

BEGIN TRY
    INSERT Facturas
    VALUES (20, 799, 'ZXX-799', '2006-07-01', 299.95, 0, 0,
            1, '2006-08-01', NULL)
    PRINT 'SUCESSO: O registo foi inserido.'
END TRY
BEGIN CATCH
    PRINT 'FALHA: O registo não foi inserido.'
    -- Pode utilizar uma instrução SELECT para devolver
    -- um resultado que contenha dados sobre o erro
    SELECT
        ERROR_NUMBER() AS [Número do erro],
        ERROR_MESSAGE() AS [Mensagem de erro]
END CATCH

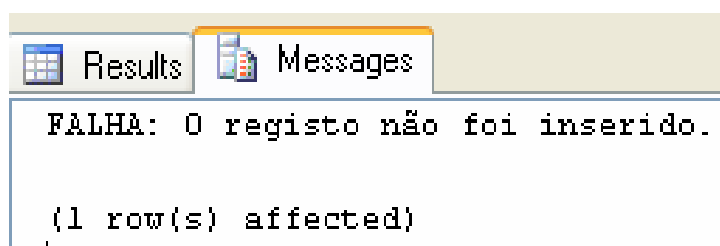
```

Na instrução *INSERT* incluída no bloco *TRY* foi utilizado um identificador de fornecedor que não existe na tabela de fornecedores, o que implica que seja gerado um erro de integridade referencial. O erro vai ser apanhado pelo bloco *CATCH* e as instruções incluídas nesse bloco vão ser executadas. A seguir, apresenta-se o resultado da instrução *SELECT* interior ao bloco *CATCH*.



Número do erro	Mensagem de erro
547	The INSERT statement conflicted with the FOREIGN KEY constraint "FK_Facturas_Fornec_1273C1CD". The conflict occurred in database "Pagamentos", table "dbo.Fo

A seguir, apresenta-se o resultado da instrução *PRINT* interior ao bloco *CATCH*.



```

FALHA: O registo não foi inserido.

(1 row(s) affected)

```

A seguir, apresentam-se alguns comandos que são comumente utilizados na implementação de *scripts*.

USE - passa a utilizar a base de dados especificada.

PRINT - devolve uma mensagem para o cliente. Se o cliente for o *Management Studio*, por exemplo, a mensagem è mostrada no tabulador *Messages* do editor de consultas.

EXEC - executa uma instrução de SQL dinâmica ou um procedimento armazenado.