Bases de Dados 2007/2008

Aula 8

Sumário

Referências

http://msdn2.microsoft.com/en-us/library/ms189826.aspx (linguagem t-sql)
http://www.di.ubi.pt/~pprata/bd/BD0405-Proc.sql (procedimentos)
http://www.di.ubi.pt/~hugomcp/bd2/t05.pdf (procedimentos)
http://www.di.ubi.pt/~hugomcp/bd2/t06.pdf (cursores)
SQL Server 2000 para Profissionais, Orlando Belo, FCA ISBN 972-722-505-5
SQL - Structured Query Language, Luís Manuel Dias Damas, FCA ISBN 972-722-443-1

1 Linguagem T- SQL

1.1 Mecanismos de controlo da linguagem T- SQL

A linguagem T-SQL possui alguns mecanismos de controlo: BEGIN e END, BREAK, CONTINUE, GOTO, IF e ELSE, RETURN, WAITFOR e WHILE. De seguida são apresentados vários exemplos da sua aplicação:

IF Exemplo de Execução Condicional	`IF @ano = 2007`
	` BEGIN`
	`  INSERT INTO ano_actual VALUES (@ano)`
	`  INSERT INTO ano_anterior`
	`  VALUES(@ano-1)`
	` END`
	`ELSE`
	` PRINT 'Ano Errado'`

WHILE	`DECLARE @contador int`
	`SET @contador = 0`
	`WHILE @contador < 100`
	`  BEGIN`
	`  PRINT 'contador = ' +`
	`CONVERT(VARCHAR(10), @contador)`
	`  SET @contador = @contador + 1`
	`  if @contador=10 BREAK`
	`END`

BREAK CONTINUE	`DECLARE @contador int`
	`SET @contador = 0`
	`WHILE @contador < 100`
	`  BEGIN`
	`  PRINT 'contador = ' +`
	`  CONVERT(VARCHAR(10), @contador)`
	`  SET @contador = @contador + 1`
	`  IF @contador > 10`
	`  BREAK`
	`  ELSE`
	`  CONTINUE`
	`  END`

GOTO	DECLARE @contador int; SET @contador = 1; WHILE @contador < 10 BEGIN PRINT @contador SET @contador = @contador + 1 IF @contador = 3 **GOTO posicao_1** IF @contador = 4 **GOTO posicao_0** END **posicao_0:** PRINT 'Nunca mostra esta mensagem!' **posicao_1:** PRINT 'Posicao_1!'
WAITFOR	DECLARE @contador int SET @contador = 0 while @contador < 10 BEGIN SET @contador = @contador + 1 PRINT @contador **WAITFOR DELAY '00:00:01'** END
RETURN	DECLARE @contador int SET @contador = 0 while @contador < 20 BEGIN IF @contador = 10 BEGIN PRINT 'Chegou ao ' + CONVERT(VARCHAR(10), @contador) **RETURN** END SET @contador = @contador + 1 PRINT @contador END

Todos os identificadores usados em T-SQL têm que começar com uma letra.

As variáveis começam sempre com @.

Todos os valores do tipo caracter, string ou data têm que ser indicados entre plicas.

Os comentários podem ser incluídos entre os símbolos /* e */ quando englobam várias linhas, ou então após -- quando o comentário é apenas uma linha.

2 Variáveis

Em T-SQL as variáveis são declaradas através da instrução DECLARE. Estas podem ser locais ou globais. As variáveis globais variáveis do sistema não podem ser criadas pelo utilizador. Estas distinguem-se das locais pois são iniciadas por dois símbolos @.

Sintaxe

```
DECLARE { @local_variable [AS] data_type, […n] }
```

Exemplo

```
Declare
@Data DATETIME,
@idade NUMERIC(2),
@nome VARCHAR(25),
@Controlador NUMERIC(2)
```

Exemplos variáveis globais

```
-- mostra data versão instalada e tipo de processador
SELECT @@version
```

```
/*@@ROWCOUNT mostra o número de registos afectados por uma instrução INSERT,
DELETE, UPDATE ou SELECT*/

UPDATE empregado
SET cargo = 'Director'
WHERE num = 1
IF @@ROWCOUNT = 0
PRINT 'Não foi actualizado';
GO
```

```
/* @@ERROR mostra o número do erro da última instrução
executada, ou zero se não existir */
UPDATE pessoa
SET nome = 1
WHERE morada = 'covilha'
IF @@ERROR <> 0
 PRINT 'Erro'
GO
```

Exemplo atribuição de valores

```
DECLARE @contador INT
SET @contador = 10
WHILE @contador > 0
BEGIN
   PRINT 'O contador está com ' + CONVERT(VARCHAR(10), @contador)
   SET @contador = @contador - 1
END
```

Exemplo atribuição de valores

```
DECLARE
@nome varchar(50),
@apelido varchar(50)
BEGIN
     SELECT @nome = nome, @apelido = apelido
     FROM empregado
     WHERE num = 1
END
```

Exemplo atribuição de valores

```
DECLARE
@nome varchar(50),
@apelido varchar(50)
BEGIN
     SELECT @nome = nome, @apelido = apelido
     FROM empregado
END
```

Na situação em que se obtém dados de mais do que um registo as variáveis ficam com o último valor.

3 Cursores

Para permitir aceder e manipular todos os registos devolvidos por uma instrução SQL, recorremos a cursores. Estes permitem percorrer os registos obtidos de um modo semelhante a uma lista possibilitando posicionar o cursor em qualquer um dos registos. Para utilizar esta funcionalidade são necessários os seguintes passos:

1. Declarar variáveis para guardar os valores devolvidos pelo cursor.
2. Declarar um cursor com o comando `DECLARE` e associa-lo à instrução `SELECT` pretendida
3. Activar o cursor através do comando `OPEN` para executar o `SELECT` e abrir o cursor
4. Utilizar as instruções de posicionamento do cursor com `FETCH`

5. Fechar o cursor com a instrução `CLOSE`.
6. Para libertar espaço na memória utilizar a instrução `DEALLOCATE` caso não pretenda usar novamente o cursor.

A variável `@@Fetch_Status` permite testar o estado do cursor (se for diferente de zero, ocorreu um erro).

As instruções de posicionamento do cursor são:

- `FETCH FIRST` avança para o primeiro registo do cursor.
- `FETCH NEXT` avança para o registo seguinte.
- `FETCH PRIOR` avança para o registo anterior.
- `FETCH LAST` avança para o último registo do cursor.
- `FETCH ABSOLUTE n` avança para posição n do cursor.
- `FETCH RELATIVE n` avança n registos a partir do registo actual do cursor.

Para utilizar os comandos de fetch (com excepção do `NEXT`) é necessário declarar um cursor do tipo `SCROLL`.

Sintaxe

```
DECLARE cursor_name CURSOR
[FORWARD_ONLY | SCROLL]
[LOCAL | GLOBAL]
FOR select_statement
[FOR UPDATE [OF column_name [,...n]]]
```

Exemplo

```
--declarar cursor
DECLARE novo_cursor CURSOR FOR
SELECT num, nome
FROM pessoa
-- declarar variáveis locais
Declare @num int, @nome varchar(50)

-- abrir um cursor
OPEN novo_cursor

-- retirar valores para uma lista de variáveis
FETCH NEXT FROM novo_cursor INTO @num, @nome

WHILE @@FETCH_STATUS = 0
BEGIN
PRINT @num
PRINT @nome
FETCH NEXT FROM novo_cursor INTO @num, @nome
END
-- fechar o cursor
CLOSE novo_cursor

-- terminar utilização do cursor
DEALLOCATE novo_cursor
```

4 Procedimentos

Em T-SQL os procedimentos armazenados são criados

Sintaxe

```
CREATE PROC[EDURE] nome_do_procedimento [;versão]
     [
            {@parâmetro tipo_de_dados} [= default] [OUTPUT]
     ]
AS
     instruções_sql [...n]
```

Exemplo

```
CREATE PROC novo_select
     AS
             SELECT * from pessoa;
```

Nesta situação temos um procedimento muito simples para mostrar o resultado do select.

Exemplo com passagem de parâmetros

```
CREATE PROC novo_utilizador @num int, @nome VARCHAR(20), @funcao VARCHAR(20)
AS
INSERT INTO utilizadores(num,nome,cargo) VALUES(@num,@nome,@funcao);
```

Nesta situação temos três variáveis que são usadas para inserir dados na tabela pessoa.

Exemplo com devolução de parâmetros

```
CREATE PROC numero_inscritos @num int OUTPUT
AS
     SELECT @num = (SELECT COUNT(*) FROM utilizadores)
     RETURN
```

Nesta situação utilizamos apenas um parâmetro para devolver um valor.

Exemplo de execução do procedimento anterior

```
DECLARE @num_insc int
EXECUTE numero_inscritos @num = @num_insc OUTPUT
SELECT @num_insc
```

Outro exemplo com passagem e devolução de parâmetros

```
CREATE PROC mostra_salario  @numfunc int, @salario_empregado int OUTPUT
     AS
             SELECT @salario_empregado = salario
             FROM empregado
             WHERE numfunc = @numfunc
             IF @@ROWCOUNT=1
                     RETURN
             SET @salario_empregado=0
             RETURN 1
GO
```

Nesta situação temos duas variáveis numfunc e salario_empregado. Estas são usadas para indicar o salário do empregado com o numfunc fornecido (INPUT), o salário_empregado (OUTPUT) é usado para passar para fora o resultado correspondente ao funcionário indicado.
Se for encontrado um registo devolve o resultado senão devolve 0 com um erro através do RETURN 1

Exemplos de execução do procedimento anterior

```
DECLARE @salario INT
EXEC mostra_salario 123, @salario_empregado = @salario OUTPUT
PRINT @salario
```

5 Exercícios

5.1 Criar uma base de dados para armazenar informação sobre notas de alunos com os seguintes campos numaluno, nomealuno, moradaaluno, coddisc, nomedisc, notadisc, não se esqueça de incluir todas as restrições que achar necessárias.

5.2 Criar o procedimento insere_disciplina para inserir novas disciplinas.

5.3 Criar o procedimento insere_aluno para inserir novos alunos.

5.4 Criar o procedimento insere_nota para inserir as notas dos alunos.

5.5 Criar o procedimento melhor_nota para mostrar a melhor nota de uma disciplina

5.6 Criar o procedimento melhor_disciplina que deve devolver o código da disciplina com a melhor média.

5.7 Criar o procedimento alterar_nota para alterar a nota de um aluno a uma disciplina.

5.8 Criar um procedimento media_disciplina para mostrar a média das notas a todas as disciplinas.

5.9 Criar um procedimento curriculum_aluno para mostrar as notas obtidas pelo aluno sendo o resultado visualizado do seguinte modo

Ex:

```
Curriculum
========
aluno nº 1234 Carlos Patrício
Cód – Disciplina – Nota
1 – Matemática – 10
2 – Física – 18
3 – Português – 10
========
```