Complementos de Bases de Dados

2023/2024

Licenciatura em Engenharia Informática

IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Laboratório 2 – T-SQL: Funções, Stored Procedures,Triggers e Metadados

Objetivos:

- Familiarização com a sintaxe T-SQL
- Criação de funções, *stored procedures* e *triggers*;
- Familizarização e utilização de metadados

O enunciado está dividido em 2 partes:

I. Síntese da matéria T-SQL com exemplos de aplicação seguida de exercícios de prática.

II. Síntese da matéria Metadados com exemplos de aplicação seguida de exercícios de prática

Os exemplos e exercícios pressupõem o carregamento prévio da base de dados Adventure Works.

I. Resumo Matéria T-SQL

VARIAVEIS E TIPOS DE DADOS

Table 1 Declaração e atribuição

`DECLARE @nome_variável tipo_dado [, n]`	`SET @nome_variável = expressão`

Table 2 Data types

Exact numeric bigint numeric bit smallint decimal smallmoney int tinyint money	*Approximate numeric* float real	*Date and time* date datetime2 smalldatetime datetime time
Character strings char varchar text	*Unicode strings* nchar nvarchar ntext	*Binary strings* binary varbinary image

ESTRUTURAS DE CONTROLO DE FLUXO

```
IF Boolean_expression
     { sql_statement | statement_block
}
[ ELSE
     { sql_statement | statement_block
} ]
```

Blocos de instruções:

```
BEGIN
    { sql_statement | statement_block
}
END
```

i.

```
CASE input_expression
     WHEN when_expression THEN
result_expression [ ...n ]
     [ ELSE else_result_expression ]
END
```

ii.

```
CASE
     WHEN Boolean_expression THEN
result_expression [ ...n ]
     [ ELSE else_result_expression ]
END
```

CICLOS

```
WHILE Boolean_expression
     { sql_statement | statement_block | BREAK | CONTINUE }
```

EXEMPLO:

```
declare @avgPrice float;
declare @maxPrice float;
Select @avgPrice = avg(p.ListPrice) from SalesLT.Product p
Select @maxPrice = max(p.ListPrice) from SalesLT.Product p

WHILE @avgPrice < $5000
BEGIN
   set @avgPrice  = @avgPrice * 2
   print @avgPrice
      IF (@avgPrice > @maxPrice)
      BREAK
   ELSE
      CONTINUE
END
PRINT 'Já é demais!';
```

USER STORED PROCEDURES

```
CREATE PROCEDURE <name>
<parameter list>
AS
<instructions to execute>
```

Retorna sempre um inteiro com código de sucesso da execução (*default 0*)

Parametros:

INPUT (IN) (*default*) - permite a passagem de valores na evocação da sp para dentro do seu conjunto de instruções

OUTPUT (OUT) - permite a alteração de valores de variáveis passadas como

	parâmetro na sequência da execução do conjunto de instruções da *sp*

```
CREATE [ OR ALTER ] { PROC | PROCEDURE }
    [schema_name.] procedure_name [ ; number ]
    [ { @parameter_name [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ OUT | OUTPUT | [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]
```

Evocação/Execução da procedure:

```
EXEC procedure_name [{param_name}]
```

EXEMPLO:

```
CREATE PROCEDURE SalesLT.TopProducts @ProductCategoryID int
 AS
 SELECT TOP(10) name, listprice
      FROM SalesLT.Product
      WHERE ProductCategoryID = @ProductCategoryID
      GROUP BY name, listprice
      ORDER BY listprice DESC;
```

```
EXEC SalesLT.TopProducts 5
```

USER DEFINED FUNCTIONS

```
CREATE [ OR ALTER ] FUNCTION [ schema_name. ]
function_name
( [ { @parameter_name [ AS ][ type_schema_name. ]
parameter_data_type
 [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
[ ; ]
```

Parametros:

INPUT (IN) (*default*)- permite a passagem de valores na evocação da *function* para dentro do seu conjunto de instruções

SUMARIO STORED PRCEDURES VS USER FUNCTIONS

Stored Procedure - SP	User Defined Function - UDF
Devolve zero, 1 ou múltiplos valores	Devolve 1 valor
Pode utilizar transações	**Não** pode utilizar transações
Pode ter paramétrios de entrada (IN) e de saída (OUT)	Só tem parâmetros de entrada
Pode chamar funções	**Não** pode chamar SPs
Não pode ser evocada embebida em comandos SQL SELECT/WHERE/HAVING	Pode ser evocada embebida em comandos SQL SELECT/WHERE/HAVING
Pode considerar tratamento de exceções	**Não** pode considerar tratamento de exceções

EXEMPLO:

```
CREATE FUNCTION sales.udfNetSale(
    @quantity INT,
    @list_price DEC(10,2),
    @discount DEC(4,2)
)
RETURNS DEC(10,2)
AS
BEGIN
    RETURN @quantity * @list_price * (1 - @discount);
END;

SELECT sales.udfNetSale(10,100,0.1) net_sale;

SELECT TOP 1
       SalesLT.udfNetSale(od.OrderQty,od.UnitPrice,od.UnitPriceDiscount),
od.UnitPriceDiscount
FROM SalesLT.SalesOrderDetail od
```

TRIGGERS

```
CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
ON { table | view }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement  [ ; ] [ ,...n ]}
```

FOR | AFTER FOR or AFTER specifies that the DML trigger fires only when all operations specified in the triggering SQL statement have launched successfully. All referential cascade actions and constraint checks must also succeed before this trigger fires.	*INSTEAD OF* Specifies that the DML trigger launches *instead of* the triggering SQL statement, thus, overriding the actions of the triggering statements. You can't specify INSTEAD OF for DDL or logon triggers.

You can't define AFTER triggers on views.	At most, you can define one INSTEAD OF trigger per INSERT, UPDATE, or DELETE statement on a table or view.

EXEMPLO:

```
CREATE TRIGGER salesLT.tr_Customer_Update
ON SalesLT.Customer
FOR UPDATE
AS
Begin
	select FirstName from inserted;
	select FirstName from deleted;
End;

UPDATE SalesLT.Customer set FirstName='ORLANDO' where customerID=1;
```

Exercícios

a. Criar a função *fnTotalVendasProduto* que calcule o valor total monetário das vendas para um determinado produto (recebendoID).
b. Utilizando a função anterior, faça uma query que apresente o nome dos produtos e o respetivo total monetário de vendas;
c. Criar o procedimento *spClientesCidade* que recebe uma cidade (ex: Las Vegas) e lista os clientes residentes na respetiva cidade.
d.
   - Crie um schema Logs e nesse schema uma tabela CustomerLog.
   - Quando se altera ou se apaga um registo da tabela Customer, deve ser executada uma cópia do registo que sofreu as alterações para a tabela de CustomerLog, explicitando o tipo de operação e o *timestamp*
   - Crie o trigger que implemente a lógica descrita

II. Resumo Matéria de Metadados

SYS SCHEMA VIEWS	**INFORMATION_SCHEMA VIEWS**
+ Melhor desempenho + Informação mais pormenorizada • Orientado a objectos • Joins por objectID - Menos inteligível - Proprietário	+ "names friendly" + Joins através de *names* + Standard/potencialemente mais interoperável - Informação mais limitada - Desempenho pode ser inferior

EXEMPLO:

`select s.name as 'SchemaName',_o.name as 'TableName',_c.name as 'ColumnName'`	`SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME`

```
from sys.schemas as s
    inner join sys.all_objects as o
    on s.schema_id = o.schema_id
   inner join sys.all_columns as c
   on c.object_id = o.object_id
where o.name like N'Product' and o.type = 'U'
order by SchemaName,TableName,ColumnName;
```

```
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = N'Product' AND
TABLE_SCHEMA='SalesLT';
```

SYS SCHEMA FUNCTIONS

EXEMPLO:

```
DECLARE @ObId int;
SET @ObId = (SELECT OBJECT_ID('salesLT.Customer', 'U'));
SELECT name AS Name, object_id AS ObjectID, type_desc AS 'Desc' FROM sys.tables
WHERE name = OBJECT_NAME(@ObId)


SELECT COL_NAME(OBJECT_ID('SalesLT.Customer'), 6) AS ColLastName,
COL_NAME(OBJECT_ID('SalesLT.Customer'),10) AS ColEmail



SELECT c.name
       ,COLUMNPROPERTY( OBJECT_ID('SalesLT.Customer'),c.name,'AllowsNull') AS 'Allow
Null'
       ,COLUMNPROPERTY( OBJECT_ID('SalesLT.Customer'),c.name,'IsIdentity') AS
'IsIdentity'
FROM sys.all_objects as o
       join sys.columns as c on o.object_id=c.object_id
       join sys.tables as t on t.object_id=o.object_id
WHERE t.name = 'Customer'
```

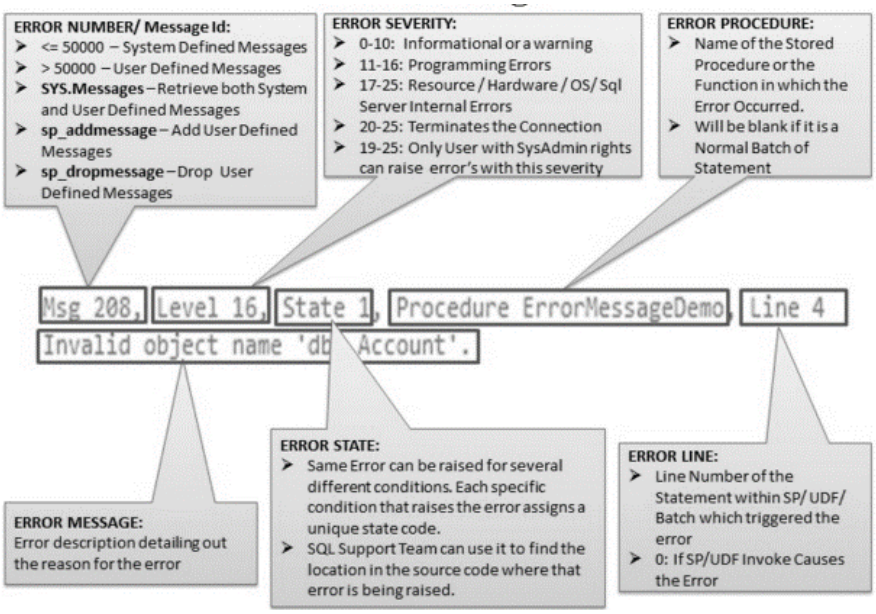
Exercícios

a. Crie um conjunto de queries para uma determinada tabela (e.g. Customer), que:
   i. Visualize a(s) coluna(s) que constituem a chave primária
   ii. Visualize para as chaves estrangeiras, o nome da coluna e a tabela/coluna que é referenciada

b. Crie o stored procedure sp_disable_FK que recebe como argumento o nome de uma tabela (@p_table_name), e gera como saída um script (lista de comandos sql) que permite fazer disable a todas as chaves estrangeiras que fazem referência à tabela.
   [Poderá ter de consultar o **anexo B** sobre cursores]

ANEXOS:

A. Error Handling



EXEMPLO:

```
SELECT Size/Weight
FROM SalesLT.Product
WHERE ProductID = 849;


BEGIN TRY
  SELECT Size/Weight
  FROM SalesLT.Product
  WHERE ProductID = 849;
END TRY
BEGIN CATCH
  SELECT
    ERROR_NUMBER() AS ErrorNumber,
    ERROR_STATE() AS ErrorState,
    ERROR_SEVERITY() AS ErrorSeverity,
    ERROR_PROCEDURE() AS ErrorProcedure,
    ERROR_LINE() AS ErrorLine,
    ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
```

B. Cursores

Apoio ao processamento orientado linha/registo a linha/registo:

- Declaração
  - Declare @Cursor Cursor
- Atribuição
  - Set @Cursor = Cursor for (*select statement*)
- Abertura
  - Open @Cursor
- Utilização (*dentro de ciclo até exceção*)
  - Fetch Next From @Cursor into (variáveis correspondentes aos campos do select)
- Fecho
  - Close @Cursor

EXEMPLO:

Crie um procedimento *spListaCompra*, que liste para uma dada compra (SalesOrderId) a lista de produtos que a compõe. Utilize cursores e a instrução print para gerar o output (ver exemplo).

```
CREATE OR ALTER PROCEDURE spListaCompra
	@SalesOrderId int
AS
BEGIN
	DECLARE c CURSOR FOR
	SELECT
		p.Name, s.OrderQty, s.UnitPrice, s.UnitPriceDiscount, s.LineTotal
		from SalesLT.SalesOrderDetail s
		join SalesLT.Product p on p.ProductID = s.ProductID
		where s.SalesOrderID = @SalesOrderId
		order by s.SalesOrderDetailID

	DECLARE
		@EmailAddress varchar(50)
		,@SalesOrderNumber varchar(25)
		,@OrderDate datetime
		,@TotalDue money
		,@OrderQty smallint
		,@UnitPrice money
		,@UnitPriceDiscount money
		,@LineTotal money
		,@Name varchar(50)

		select @EmailAddress = c.EmailAddress, @SalesOrderNumber =
s.SalesOrderNumber, @OrderDate = s.OrderDate, @TotalDue = s.TotalDue
		from SalesLT.SalesOrderHeader s
		join SalesLT.Customer c on c.CustomerID = s.CustomerID
		where SalesOrderID = @SalesOrderId;

		OPEN c
		FETCH NEXT FROM c INTO
		@Name
```

```
		,@OrderQty
		,@UnitPrice
		,@UnitPriceDiscount
		,@LineTotal

		PRINT '-------------------------------------------------------------'
		PRINT 'Customer: ' + @EmailAddress
		PRINT 'Order: ' + @SalesOrderNumber
		PRINT 'Date: ' + CAST(@OrderDate as varchar)
		PRINT 'Total: ' + CAST(@TotalDue as varchar)
		PRINT '-------------------------------------------------------------'


		WHILE @@FETCH_STATUS = 0
		BEGIN

			PRINT 'Product: ' + @Name +
				' / OrderQty: ' + CAST(@OrderQty as varchar) +
				' / UnitPrice: ' + CAST(@UnitPrice as varchar) +
				' / UnitPriceDiscount: ' + CAST(@UnitPriceDiscount as varchar)
+
				' / LineTotal: ' + CAST(@LineTotal as varchar)

			FETCH NEXT FROM c INTO
			@Name
			,@OrderQty
			,@UnitPrice
			,@UnitPriceDiscount
			,@LineTotal
		END
		CLOSE c
		DEALLOCATE c
END
GO


Exec spListaCompra 71863;
```

- Fim de enunciado -