Tarefa Orientada S15

Triggers

Objectivos:

- Criar triggers AFTER
- Criar triggers INSTEAD OF
- Exemplos de utilização

Os *triggers* são um tipo especial de procedimento que são invocados, ou activados, de forma automática quando uma consulta de acção é executada sobre uma tabela ou vista. Contudo, não se pode invocar um *trigger* directamente, nem se podem passar parâmetros para um *trigger*. Um *trigger* não devolve nenhum valor.

Para criar um *trigger* utilize a seguinte sintaxe:

```
CREATE TRIGGER nome_do_trigger
ON {nome_da_tabela | nome_da_vista}
[WITH {ENCRYPTION | cláusula_EXECUTE_AS | ENCRYPTION , cláusula_EXECUTE_AS }]
{FOR | AFTER | INSTEAD OF} [INSERT] [,] [UPDATE] [,] [DELETE]
AS instruções_SQL
```

Um *trigger* é associado a uma única tabela ou vista, identificada na cláusula *ON*.

Um *trigger* pode ser definido para ser activado com uma instrução de *INSERT*, de *UPDATE*, de *DELETE*, ou com uma combinação destas instruções.

Um *trigger* pode ser definido para ser activado depois (*AFTER*) da execução da instrução DML (*Data Manipulation Language*) que o activa ser executada, ou em vez (*INSTEAD OF*) dessa instrução ser executada, isto é, a instrução DML que activa o *trigger* não chega a ser executada. Se a consulta de acção que activa um *trigger AFTER* causar um erro, então o *trigger* não será executado.

Uma tabela pode ter múltiplos triggers *AFTER*, mesmo que para a mesma acção. Por outro lado, uma vista não pode ter *triggers AFTER* associados.

Apenas se pode associar um *trigger INSTEAD OF* a uma tabela ou a uma vista por cada tipo de acção.

A opção *ENCRYPTION* impede os utilizadores de verem o código de um *trigger*.

A opção *EXECUTE AS* é uma novidade do SQL SERVER 2005. Pode utilizá-la para permitir que os utilizadores executem o procedimento armazenado com as permissões especificadas nesta cláusula.

Uma forma expedita de definir o nome de um *trigger* é utilizar o nome da tabela ou da vista a que está associado, seguido pela acção que faz executar o *trigger*.

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

1.1 *Script* que cria um *trigger AFTER* que é activado por instruções de *INSERT* ou *UPDATE* sobre a tabela *CópiaFornecedores*.

```
USE Pagamentos

IF OBJECT_ID('CópiaFornecedores_INSERT_UPDATE') IS NOT NULL
    DROP TRIGGER CópiaFornecedores_INSERT_UPDATE
GO

CREATE TRIGGER CópiaFornecedores_INSERT_UPDATE
    ON CópiaFornecedores
    AFTER INSERT,UPDATE
AS
    UPDATE CópiaFornecedores
    SET Localidade = UPPER(Localidade)
    WHERE IDFornecedor IN (SELECT IDFornecedor FROM INSERTED)
```

O *trigger* criado neste exemplo é activado após a execução de uma operação de *INSERT* ou *UPDATE* sobre a tabela *CópiaFornecedores*. O *trigger* actualiza o atributo *Localidade*, da tabela *CópiaFornecedores*, de modo a que o seu valor seja armazenado sempre em letras maiúsculas.

Note que é utilizada uma subconsulta que é baseada numa tabela especial designada *INSERTED*. Esta tabela contém os novos registos que estão a ser inseridos na tabela. Dado que esta tabela apenas existe enquanto o *trigger* está a ser executado, só é possível referenciá-la dentro do código do *trigger*.

Outra tabela que é criada pelo sistema, durante uma operação de eliminação de registos, e que pode ser referenciada a partir do código de um *trigger*, é a tabela *DELETED*. Esta tabela contém os registos originais que vão ser eliminados.

Para as instruções de *UPDATE*, a tabela *INSERTED* contém os registos com os dados actualizados e a tabela *DELETED* contém os registos originais que vão ser alvo das actualizações.

Para activar/executar o *trigger* criado no passo anterior, pode utilizar uma instrução do género da seguinte.

1.2 Exemplo de instrução que activa o *trigger* criado no passo anterior.

```
INSERT CópiaFornecedores
VALUES (124, 'Elevadores Santa Justa', 'Rua da santa que foi justa,
        nº 1','lisboa', '1000-000', '21456321', NULL, NULL, 4,394)
```

1.3 Para ver o resultado da execução do *trigger*, pode utilizar a seguinte instrução.

```
SELECT *
FROM CópiaFornecedores
WHERE IDFornecedor = (SELECT MAX(IDFornecedor)
                      FROM CópiaFornecedores)
```

O resultado é o seguinte.

Results | Messages

	IDFornecedor	Nome	Endereço	Localidade	CódigoPostal	Telefone	PNomeContacto	ÚNomeContacto	CondiçãoPagamentoPredefinida	ContaBalançoPredefinida
1	124	Elevadores Santa Justa	Rua da santa que foi justa, nº1	LISBOA	1000-000	21456321	NULL	NULL	4	394

O *script* seguinte começa por eliminar as tabelas *CópiaFornecedores* e *CópiaFacturas*, às quais os *triggers* vão ser associados. Depois, volta a recriá-las através do comando *SELECT INTO* sobre as tabelas *Fornecedores* e *Facturas*. Dado que, quando se criam cópias de tabelas com esta técnica, as restrições de chave forasteira não são também copiadas, a integridade referencial entre as duas tabelas não vai ser forçada.

Embora seja possível adicionar uma restrição de chave forasteira à tabela *CópiaFacturas*, através do comando *ALTER TABLE*, vamos utilizar o *script* seguinte para impor a integridade referencial entre as tabelas *CópiaFornecedores* e *CópiaFacturas*.

1.4 *Script* que cria dois *triggers* para manter a integridade referencial entre duas tabelas.

```
USE Pagamentos
IF OBJECT_ID('CópiaFornecedores') IS NOT NULL
    DROP TABLE CópiaFornecedores

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

SELECT * INTO CópiaFornecedores FROM Fornecedores
SELECT * INTO CópiaFacturas FROM Facturas
GO

CREATE TRIGGER CópiaFornecedores_UPDATE_DELETE_IR
    ON CópiaFornecedores
    AFTER DELETE,UPDATE
AS
    IF EXISTS (SELECT * FROM Deleted JOIN CópiaFacturas
                   ON Deleted.IDFornecedor = CópiaFacturas.Fornecedor)
    BEGIN
        RAISERROR('Este fornedor tem facturas associadas. Logo, não
                   pode ser eliminado.', 11, 1)
        ROLLBACK TRAN
    END
GO

CREATE TRIGGER CópiaFacturas_INSERT_UPDATE_IR
    ON CópiaFacturas
    AFTER INSERT,UPDATE
AS
    IF NOT EXISTS (SELECT * FROM CópiaFornecedores
                   WHERE IDFornecedor IN (SELECT Fornecedor
                                          FROM Inserted))
    BEGIN
        RAISERROR('Não existe nenhum fornecedor com este
                   identificador. Escolha outro', 11, 1)
        ROLLBACK TRAN
    END
```

O primeiro *trigger* impede a eliminação de um fornecedor ou a actualização do seu identificador (*IDFornecedor*) na tabela *CópiaFornecedores*, caso esse fornecedor tenha uma ou mais facturas associadas na tabela *CópiaFacturas*.

Quando um registo é eliminado ou actualizado é copiado para a tabela DELETED. Depois, o *trigger* verifica se existem alguns registos na tabela *CópiaFacturas* que têm o mesmo identificador de fornecedor. Em caso afirmativo, o *trigger* gera uma mensagem de erro e recupera os registos eliminados ou alterados. Para tal é utilizada a instrução *ROLLBACK TRAN*. Esta instrução recupera os dados afectados pela instrução SQL que causou a activação do *trigger*.

O segundo *trigger* impede a inserção, na tabela *CópiaFacturas*, de uma factura cujo fornecedor não esteja registado na tabela *CópiaFornecedores*. Também impede que o atributo *Fornecedor* da tabela *CópiaFacturas* seja alterado para um valor que não existe como identificador de um fornecedor na tabela *CópiaFornecedores*.

Quando é executada uma instrução de *INSERT* ou *UPDATE* sobre a tabela *CópiaFacturas*, é adicionada, na tabela *INSERTED*, uma cópia do novo registo ou do registo modificado. Depois, o *trigger* verifica, na tabela *CópiaFornecedores*, se existe algum registo com um identificador igual aos identificadores de fornecedores que estão na tabela *INSERTED*. Em caso negativo, o *trigger* gera um erro e recupera os dados afectados pela instrução SQL que causou a activação do *trigger*.

Dado que um *trigger AFTER* é activado depois da consulta de acção que o activou ser executada, o *trigger* não vai ser executado caso a consulta de acção que o activou provocar um erro. Por este motivo, não utilizaríamos estes *triggers* se a integridade referencial entre as duas tabelas tivesse sido forçada pela definição de uma restrição de chave forasteira. Nessa situação, podíamos utilizar *triggers INSTEAD OF* para verificar a integridade referencial antes de ocorrer um erro.

Um *trigger INSTEAD OF* é executado em vez da instrução DML que causa a sua activação. Dado que, deste modo, a instrução nunca é executada, é comum incluir no corpo de *trigger* código que realize essa instrução.

Os *triggers* INSTEAD OF são comummente associados a vistas de actualização. Também é comum utilizá-los na prevenção de erros, tais como violações de restrições, antes que eles ocorram.

Apenas pode ser associado um *trigger INSTEAD OF* a uma tabela ou a uma vista por cada tipo de operação *DML*. Contudo, se uma tabela for definida com uma restrição de chave forasteira que especifique a opção *UPDATE CASCADE* ou *DELETE CASCADE*, os triggers *INSTEAD OF UPDATE* ou *INSTEAD OF DELETE* não podem ser definidos para essa tabela.

1.5 Para analisarmos um exemplo de *triggers INSTEAD OF* associados a vistas, implemente a seguinte vista.

```
USE Pagamentos

IF OBJECT_ID('FacturasFornecedorIBM') IS NOT NULL
    DROP VIEW FacturasFornecedorIBM
GO

CREATE VIEW FacturasFornecedorIBM
AS
SELECT IDFactura, NúmeroFactura, DataFactura, TotalFactura
FROM Facturas
WHERE Fornecedor = (SELECT IDFornecedor
                    FROM Fornecedores
                    WHERE Nome = 'IBM')
```

Esta vista selecciona as facturas do fornecedor com o nome IBM.

Uma operação de *INSERT* do género da seguinte iria falhar, pois não são fornecidos valores para todas as colunas da tabela *Facturas* (subjacente à vista *FacturasFornecedorIBM*) que requerem valores.

```
INSERT INTO FacturasFornecedorIBM (IDFactura, NúmeroFactura,
DataFactura, TotalFactura)
VALUES (20, 'RA23999', ' 2006-07-25', 717.49)
```

A seguir, vamos criar um *trigger INSTEAD OF* para ser utilizado no controlo de operações de *INSERT* através da vista acabada de criar.

1.6 *Trigger INSTEAD OF INSERT* associado a uma vista.

```
USE Pagamentos

GO
IF OBJECT_ID('FacturasFornecedorIBM_INSERT') IS NOT NULL
   DROP TRIGGER FacturasFornecedorIBM_INSERT
GO
CREATE TRIGGER FacturasFornecedorIBM_INSERT
    ON FacturasFornecedorIBM
    INSTEAD OF INSERT
AS
DECLARE @DataFactura smalldatetime, @NúmeroFactura varchar(50),
        @TotalFactura money, @Fornecedor int,
        @DataVencimentofactura smalldatetime, @CondiçãoPagamento int,
        @CondiçãoPagamentoPredefinida smallint,
        @ContadorDeRegistosInserted int,
        @IDFactura int

SELECT @ContadorDeRegistosInserted = COUNT(*) FROM Inserted

IF @ContadorDeRegistosInserted = 1
    BEGIN

        SELECT @NúmeroFactura = NúmeroFactura,
               @DataFactura = DataFactura,
               @TotalFactura = TotalFactura
        FROM Inserted

        IF (@DataFactura IS NOT NULL AND @NúmeroFactura IS NOT NULL
            AND @TotalFactura IS NOT NULL)
            BEGIN

                SELECT @Fornecedor = IDFornecedor,
                      @CondiçãoPagamento=CondiçãoPagamentoPredefinida
                FROM Fornecedores
                WHERE Nome = 'IBM'

                SELECT @CondiçãoPagamentoPredefinida = PrazoPagamento
                FROM CondiçõesPagamento
                WHERE IDCondição = @CondiçãoPagamento

                SET @DataVencimentofactura = @DataFactura +
                                        @CondiçãoPagamentoPredefinida

                        SELECT @IDFactura = MAX(IDFactura)
                        FROM Facturas

                        SET @IDFactura = @IDFactura + 1

                INSERT Facturas(IDFactura, Fornecedor, NúmeroFactura,
                                DataFactura, TotalFactura,
                                CondiçãoPagamento,
                                DataVencimentofactura, DataPagamento)
                VALUES (@IDFactura, @Fornecedor, @NúmeroFactura,
                        @DataFactura,
                        @TotalFactura, @CondiçãoPagamento,
                        @DataVencimentofactura, NULL)
```

```
            END
    END
ELSE
    RAISERROR('Insira apenas um registo.', 1, 1)
```

1.7 Elabore uma instrução *INSERT* que insira um registo na tabela *Facturas*, através da vista *FacturasFornecedorIBM*.

```
Select * from facturas

INSERT FacturasFornecedorIBM (NúmeroFactura, DataFactura,
TotalFactura)
VALUES ('RA23999', '2006-07-25', 617.34)

Select * from facturas
```

A seguir, apresenta-se o resultado da instrução `Select * from facturas`, executada após a inserção do registo na tabela *Facturas*, através da vista *FacturasFornecedorIBM*.

Results | Messages

	IDFactura	Fornecedor	NúmeroFactura	DataFactura	TotalFactura	Pagamento	Crédito	CondiçãoPagamento	DataVencimentoFactura	DataPagamento
1	1	34	QP58872	2006-02-25 00:00:00	116,54	116,54	0,00	4	2006-04-22 00:00:00	2006-04-11 00:00:00
2	2	34	Q545443	2006-03-14 00:00:00	1083,58	1083,58	0,00	4	2006-05-23 00:00:00	2006-05-14 00:00:00
3	3	110	P-0608	2006-04-11 00:00:00	20551,18	19351,18	1400,00	5	2006-06-30 00:00:00	2006-08-01 00:00:00
4	4	110	P-0259	2006-04-16 00:00:00	26881,40	26881,40	0,00	3	2006-05-16 00:00:00	2006-05-12 00:00:00
5	5	81	MABO1489	2006-04-16 00:00:00	936,93	936,93	0,00	3	2006-05-16 00:00:00	2006-05-13 00:00:00
6	6	122	989319-497	2006-04-17 00:00:00	2312,20	0,00	200,00	4	2006-06-26 00:00:00	NULL
7	7	82	C73-24	2006-04-17 00:00:00	600,00	600,00	0,00	2	2006-05-10 00:00:00	2006-05-05 00:00:00
8	8	122	989319-487	2006-04-18 00:00:00	1927,54	0,00	200,00	4	2006-06-19 00:00:00	NULL
9	9	122	989319-477	2006-04-19 00:00:00	2184,11	2184,11	0,00	4	2006-06-12 00:00:00	2006-06-07 00:00:00
10	10	122	989319-467	2006-04-24 00:00:00	2318,03	2318,03	0,00	4	2006-06-05 00:00:00	2006-05-29 00:00:00
11	11	122	989319-457	2006-04-24 00:00:00	3813,33	3813,33	0,00	3	2006-05-29 00:00:00	2006-05-20 00:00:00
12	12	122	989319-447	2006-04-24 00:00:00	3689,99	3689,99	0,00	3	2006-05-22 00:00:00	2006-05-12 00:00:00
13	13	122	989319-437	2006-04-24 00:00:00	2765,36	2765,36	0,00	2	2006-05-15 00:00:00	2006-05-03 00:00:00
14	14	122	989319-427	2006-04-25 00:00:00	2115,81	2115,81	0,00	1	2006-05-08 00:00:00	2006-05-01 00:00:00
15	15	121	97/553B	2006-04-26 00:00:00	313,55	0,00	200,00	4	2006-07-09 00:00:00	NULL
16	18	121	97/553	2006-04-27 00:00:00	904,14	0,00	200,00	4	2006-07-09 00:00:00	NULL
17	19	121	97/522	2006-04-30 00:00:00	1962,13	0,00	400,00	4	2006-07-10 00:00:00	NULL
18	20	34	RA23999	2006-07-25 00:00:00	617,34	0,00	0,00	1	2006-08-04 00:00:00	NULL

O *trigger* criado no passo 1.6 atribui valores às colunas em falta na instrução de *INSERT* do passo 1.7 e cujo preenchimento é necessário. Para tal, baseia-se em quatro assumpções lógicas. Primeira, o identificador da factura pode ser considerado, por exemplo, como sendo o número imediatamente a seguir ao maior valor dos identificadores das facturas. Segunda, o identificador do fornecedor pode ser assumido, dado que a vista *FacturasFornecedorIBM*

selecciona explicitamente as facturas do fornecedor *IBM*. Terceira, as condições de pagamento de uma factura podem ser assumidas como sendo a condição de pagamento predefinida do fornecedor em causa. Quarta, a data de vencimento de uma factura de uma factura pode ser calculada com base na data da factura e do prazo de pagamento da factura.

Depois de declarar as variáveis, o *trigger* pesquisa na tabela *INSERTED* para obter o número de registos inseridos. Este *trigger* só funciona se tiver sido inserido apenas um registo. Caso tenham sido inseridos mais do que um registo, é apresentada uma mensagem de erro.

No caso de ter sido inserido apenas um registo, o *trigger* obtém os valores para três colunas que foram especificadas na instrução *INSERT* que o faz activar e regista os respectivos valores em três variáveis. Depois, se nenhuma das variáveis contiver um valor *NULL*, o *trigger* determina os valores para as colunas que não foram especificadas na instrução de *INSERT* que o activa.

Depois, o *trigger* determina o valor a atribuir para o identificador da factura a inserir, baseando-se no maior valor dos identificadores já atribuídos.

Dado que um *trigger INSTEAD OF* é executado em vez da instrução *DML* que o activou, a instrução nunca é executada. Por esse motivo, é incluída, no final do corpo do *trigger*, a instrução *INSERT* que permite inserir o registo na tabela *Facturas*. Sem esta instrução definida no corpo do *trigger* a inserção na tabela *Facturas* nunca seria efectuada.

Os *triggers* também podem ser utilizados para verificar/impor regras de base de dados que visam a consistência dos dados, que não podem ser impostas através da especificação de restrições (*Constraints*). Por exemplo, a soma das quantias dos itens de uma factura deve ser igual ao total (dos itens) da factura registado na tabela *Facturas*.

1.8 Execute as seguintes instruções de *SELECT* e *UPDATE*.

```
SELECT * from ItensFactura
WHERE IDFactura = 19 AND SequenciaItem = 2

UPDATE ItensFactura
SET MontanteItem = 765
WHERE IDFactura = 19 AND SequenciaItem = 2


SELECT * from ItensFactura
WHERE IDFactura = 19 AND SequenciaItem = 2


SELECT *
FROM Facturas JOIN (SELECT IDFactura, SUM(MontanteItem) AS SomaItens
                    FROM ItensFactura
                    GROUP BY IDFactura
                    HAVING IDFactura = 19) AS Itens
            ON Facturas.IDFactura = Itens.IDFactura
```

O resultado é o seguinte.

Results | Messages

	IDFactura	SequenciaItem	ContaBalanço	MontanteItem	DescriçãoItem
1	19	2	540	765,13	SCMD Flyer

	IDFactura	SequenciaItem	ContaBalanço	MontanteItem	DescriçãoItem
1	19	2	540	765,00	SCMD Flyer

	IDFactura	Fornecedor	NúmeroFactura	DataFactura	TotalFactura	Pagamento	Crédito	CondiçãoPagamento	DataVencimentoFactura	DataPagamento	IDFactura	SomaItens
1	19	121	97/522	2006-04-30 00:00:00	1962,13	0,00	400,00	4	2006-07-10 00:00:00	NULL	19	1962,00

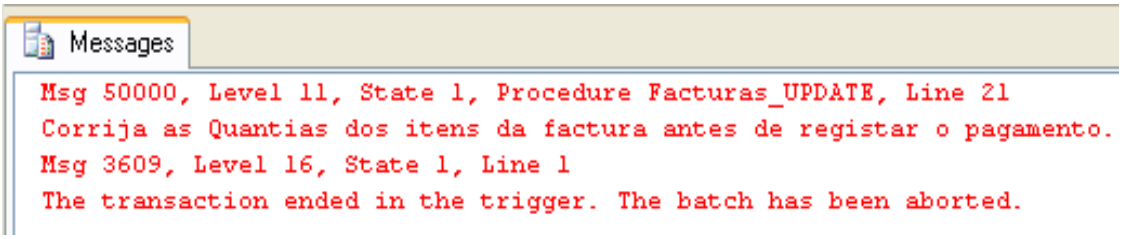
1.9 *Script* que cria um *trigger* para validar as quantias dos itens de uma factura quando se regista o pagamento da factura.

```
USE Pagamentos

IF OBJECT_ID('Facturas_UPDATE') IS NOT NULL
    DROP TRIGGER Facturas_UPDATE

GO

CREATE TRIGGER Facturas_UPDATE
    ON Facturas
    AFTER UPDATE
AS
IF EXISTS           --Verificar se o Pagamento foi alterado
   (SELECT *
    FROM Deleted JOIN Facturas
      ON Deleted.IDFactura = Facturas.IDFactura
    WHERE Deleted.Pagamento <> Facturas.Pagamento)
  BEGIN
    IF EXISTS
--Verificar se a soma das quantias dos itens da factura correspondem
--ao total da factura
       (SELECT *
        FROM Facturas JOIN
            (SELECT IDFactura, SUM(MontanteItem) AS SomaItens
             FROM ItensFactura
             GROUP BY IDFactura) AS Itens
          ON Facturas.IDFactura = Itens.IDFactura
        WHERE (Facturas.TotalFactura <> Itens.SomaItens) AND
              (Itens.IDFactura IN (SELECT IDFactura FROM Deleted)))
      BEGIN
        RAISERROR('Corrija as Quantias dos itens da factura antes de
                   registar o pagamento.', 11, 1)
        ROLLBACK TRAN
      END
  END
```

1.10 Instrução *UPDATE* que activa o *trigger* criado no passo anterior.

```
UPDATE Facturas
SET Pagamento = 1963.13, DataPagamento = getdate()
WHERE IDFactura = 19
```

A seguir, apresenta-se o resultado da execução da instrução de UPDATE apresentada no passo anterior.



Note que o *trigger* criado no passo 1.9 faz sentido, pois regista-se em primeiro lugar na base de dados uma factura. Apenas depois de ter sido registada a factura é que se podem inserir os dados relativos aos seus itens, de modo a não violar a integridade referencial.

Este exemplo pretende simular uma situação em que se regista uma factura (19) com um determinado total. Depois, registam-se os seus (2) itens, de modo a que a soma das suas quantias não seja igual ao total da factura correspondente. Assim, quando se for, mais tarde, registar o montante pago, o *trigger* irá detectar a situação anómala e irá gerar uma mensagem adequada e impedir que seja efectuada a alteração.

Para apagar um *trigger*, utilize a seguinte sintaxe.

```
DROP TRIGGER nome_do_trigger [, …]
```

Para alterar a definição de um *trigger*, elimine-o e, depois volte a criá-lo. Alternativamente, pode utilizar a seguinte sintaxe.

```
ALTER TRIGGER nome_do_trigger
ON {nome_da_tabela | nome_da_vista}
[WITH {ENCRYPTION | cláusula_EXECUTE_AS | ENCRYPTION , cláusula_EXECUTE_AS }]
{FOR | AFTER | INSTEAD OF} [INSERT] [,] [UPDATE] [,] [DELETE]
AS instruções_SQL
```

Quando se elimina um *trigger*, as permissões de segurança que lhe estão associadas são também eliminadas.

A menos que o trigger e a tabela a que ele está associado pertençam ao esquema predefinido (*default schema*), deve-se incluir o nome do esquema nas instruções de *DROP TRIGGER* e de *ALTER TRIGGER*.

QUESTÕES

2 Implemente as seguintes instruções.

2.1 Demonstre que, para os vários casos possíveis de violação da integridade referencial, os *triggers* criados no passo 1.4 impõem, de facto, a integridade referencial entre as tabelas *CópiaFornecedores* e *CópiaFacturas*.

2.2 Com a falta de pagamento por parte de alguns fornecedores a empresa decidiu "bloquear" a emissão de novas facturas para clientes com valor em dívida superior a 2500€. Crie um trigger que implemente esta regra tendo em conta as seguintes alíneas:

2.2.1 O valor em dívida deverá ser calculado através de uma UDF do tipo escalar, tendo como parâmetro de entrada o *IDFornecedor*.

Resultado

(Abaixo o valor da dívida utilizando a UDF para os fornecedores com ID 82 e 122, respetivamente.)

	(No column name)
1	3705.75

	(No column name)
1	3839.74

2.2.2 O trigger deverá contar o número de registos inseridos, emitindo o seguinte erro 'Insira apenas um registo.' caso a contagem seja superior a um registo.

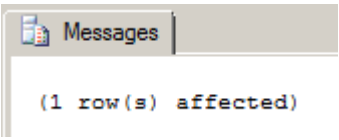
2.2.3 De igual modo, o trigger deverá validar o valor em dívida do fornecedor em causa, recorrendo à função criada na alínea 2.2.1. Caso o valor em dívida seja superior a 2500€, o trigger deverá emitir o erro 'Divida superior a 2500€. Conta suspensa!' recorrendo à função RAISERROR. Se o valor em dívida for inferior ou igual a 2500€ o trigger deverá inserir a factura normalmente.

2.2.4 Para salvaguarda do script, implemente o código que permite verificar a existência do trigger, eliminando-o se necessário antes da sua criação.

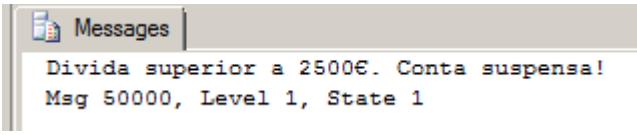
2.2.5 Verifique o correto funcionamento do trigger introduzindo a informação abaixo apresentada.

Fornecedor	Número Factura	Data Factura	Data Vencimento Factura	Total Factura	Condição Pagamento
82	LS2015-223	Data atual	Data atual + 30dias	1235.25	1
122	LS2015-224	Data atual	Data atual + 15 dias	2400.30	4

Resultado da 1ª linha (registo inserido com sucesso)



Resultado da 1ª linha (registo não inserido, conta suspensa)



2.3 Recorrendo à base de dados *Editora*, crie um trigger responsável por atualizar a quantidade de produtos em stock mediante inserção, atualização ou exclusão de registos na tabela *ItensFacturasED*.

Isto é:

- Por cada linha inserida na tabela *ItensFacturasED* deve ser subtraído o valor da quantidade registada, no campo *Quantidade* na tabela *Produtos*;
- Por cada linha eliminada na tabela *ItensFacturasED* deve ser atualizado positivamente o campo *Quantidade* na tabela *Produtos*, em igual valor;
- Por cada linha atualizada na tabela *ItensFacturasED* deve ser atualizada a diferença entre o valor anterior e o novo valor no campo *Quantidade* na tabela *Produtos*.

2.3.1 Atualize a quantidade do Produto 'DB2R' da fatura com ID 27 para 10 valores. Verifique a quantidade final na tabela *Produtos*.

Resultado
(tabela ItensFacturasED e Produtos antes e depois da atualização do produto DB2R)

Results | Messages

	Factura	Produto	PreçoUnitário	Quantidade	TotalItem
1	27	DB2R	45.00	4	45.00
2	27	MCBL	62.50	1	62.50

	Código	Designação	PreçoUnitário	Quantidade
1	DB2R	DB2 for the COBOL Programmer, Part 2 (2nd Edition)	45.00	621

	Factura	Produto	PreçoUnitário	Quantidade	TotalItem
1	27	DB2R	45.00	10	45.00
2	27	MCBL	62.50	1	62.50

	Código	Designação	PreçoUnitário	Quantidade
1	DB2R	DB2 for the COBOL Programmer, Part 2 (2nd Edition)	45.00	615

2.3.2 Insira um novo registo para a fatura com ID 27 para o produto com o código 'DB1R', 125 quantidades e com Total Item de 5250.

Resultado

(tabela ItensFacturasED e Produtos antes e depois da inserção do produto DB1R)

Results | Messages

	Factura	Produto	PreçoUnitário	Quantidade	TotalItem
1	27	DB2R	45.00	10	45.00
2	27	MCBL	62.50	1	62.50

	Código	Designação	PreçoUnitário	Quantidade
1	DB1R	DB2 for the COBOL Programmer, Part 1 (2nd Edition)	42.00	4825

	Factura	Produto	PreçoUnitário	Quantidade	TotalItem
1	27	DB1R	42.00	125	5250.00
2	27	DB2R	45.00	10	45.00
3	27	MCBL	62.50	1	62.50

	Código	Designação	PreçoUnitário	Quantidade
1	DB1R	DB2 for the COBOL Programmer, Part 1 (2nd Edition)	42.00	4700

2.3.3 Faça as alterações necessárias ao trigger, para que este atualize corretamente o campo *TotalItem* da tabela *ItensFacturasED*. Para verificar o correcto funcionamento do trigger, actualize a quantidade do produto 'DB2R' da fatura com ID 27 para 7 quantidades. Verifique que o trigger calculou corretamente o valor do campo *TotalItem*.

Resultado

(tabela ItensFacturasED e Produtos antes e depois da atualização do produto DB2R para 7 quantidades – correcção do valor do campo TotalItem automaticamente)

Results | Messages

	Factura	Produto	PreçoUnitário	Quantidade	TotalItem
1	27	DB1R	42.00	125	5250.00
2	27	DB2R	45.00	10	45.00
3	27	MCBL	62.50	1	62.50

	Código	Designação	PreçoUnitário	Quantidade
1	DB2R	DB2 for the COBOL Programmer, Part 2 (2nd Edition)	45.00	615

	Factura	Produto	PreçoUnitário	Quantidade	TotalItem
1	27	DB1R	42.00	125	5250.00
2	27	DB2R	45.00	7	315.00
3	27	MCBL	62.50	1	62.50

	Código	Designação	PreçoUnitário	Quantidade
1	DB2R	DB2 for the COBOL Programmer, Part 2 (2nd Edition)	45.00	618