TAREFA ORIENTADA S13

Procedimentos armazenados

Objetivos:

- Criar Procedimentos armazenados
- Chamar procedimentos armazenados
- Utilizar parâmetros de entrada e de saída

Um procedimento armazenado representa um objeto executável da base de dados que contém uma ou mais instruções SQL. Os procedimentos armazenados são pré compilados, isto é, quando um procedimento armazenado é executado pela primeira vez, são também compiladas e executadas as instruções SQL nele contidas. Depois, o procedimento é armazenado, sob a forma compilada, na base de dados. Nas execuções subsequentes, as instruções SQL são executadas sem necessidade de serem compiladas. Deste modo, a execução de um procedimento armazenado torna-se mais rápida, quando comparada, por exemplo, com a execução de um *script* SQL equivalente.

Para criar um procedimento armazenado utilize a seguinte sintaxe:

```
CREATE {PROC | PROCEDURE} nome_do_procedimento_armazenado
[declarações de parâmetros]
[WITH [RECOMPILE] [, ENCRYPTION] [, cláusula EXECUTE AS]]
AS instruções_SQL
```

A instrução *CREATE PROC* deve ser a primeira e a única instrução contida num grupo de um *script*. Se criar um *script* com mais do que um procedimento armazenado, deve incluir as instruções *CREATE PROC* entre comandos *GO*.

Pode utilizar parâmetros para passar um ou mais valores de ou para o procedimento armazenado. Mais à frente voltaremos a este assunto.

Para criar um procedimento armazenado temporário, preceda o nome do procedimento com o carácter # para um procedimento local e com dois caracteres ## para um procedimento global. Um procedimento armazenado temporário é guardado na base de dados do sistema *tempdb* e existe apenas enquanto a sessão corrente estiver aberta.

A cláusula *AS* contém as instruções SQL a ser executadas no procedimento armazenado.

A opção *RECOMPILE* impede o sistema de pré compilar o procedimento armazenado, isto é, o procedimento tem que ser compilado de cada vez que é executado. Dado que isto reduz o desempenho, não se utiliza, frequentemente, esta opção.

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

A opção *EXECUTE AS* foi introduzida na versão SQL SERVER 2005. Pode utilizá-la para permitir que os utilizadores executem o procedimento armazenado com as permissões especificadas nesta cláusula.

Uma das vantagens de utilizar procedimentos armazenados é que, através da sua utilização, as aplicações ou os utilizadores finais não necessitam de conhecer detalhadamente a estrutura da base de dados.

Outra vantagem da utilização de procedimentos armazenados é que eles permitem restringir e controlar o acesso a uma base de dados. Por exemplo, pode permitir a uma aplicação ou a um utilizador que chame determinados procedimentos armazenados e impedi-los de executar outras instruções SQL. Deste modo, pode restringir o acesso apenas aos registos, colunas e tabelas contemplados nesses procedimentos armazenados. Para os sistemas em que a segurança é crítica, este pode ser um modo interessante de restringir o acesso aos dados.

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

1.1 *Script* que cria um procedimento armazenado.

```
USE Pagamentos

IF OBJECT_ID('spRelatórioFacturas') IS NOT NULL
    DROP PROC spRelatórioFacturas
GO

CREATE PROC spRelatórioFacturas
AS
SELECT Nome, NúmeroFactura, DataFactura, TotalFactura
FROM Facturas JOIN Fornecedores
     ON Fornecedor = IDFornecedor
WHERE TotalFactura - Pagamento - Crédito > 0
ORDER BY Nome
```

O procedimento armazenado criado neste exemplo contém apenas uma instrução *SELECT* que devolve dados das tabelas *Facturas* e *Fornecedores*. Mais precisamente, devolve o nome dos fornecedores e o número, a data e o total das respetivas faturas ainda por saldar.

Para chamar/executar um procedimento armazenado utilize a instrução *EXEC*. Se chamar um procedimento armazenado na primeira linha de um *script*, pode omitir a instrução *EXEC* e pode utilizar apenas o nome do procedimento. Todavia, é uma boa prática utilizar sempre este comando, para tornar a leitura do código mais simples.

1.2 Instrução que chama/executa o procedimento armazenado criado no passo anterior.

```
EXEC spRelatórioFacturas
```

O resultado desta instrução é o seguinte.

Results | Messages

	Nome	NúmeroFactura	DataFactura	TotalFactura
1	Mc Graw Hill	97/553B	2006-04-26 00:00:00	313,55
2	Mc Graw Hill	97/553	2006-04-27 00:00:00	904,14
3	Mc Graw Hill	97/522	2006-04-30 00:00:00	1962,13
4	Patinter	989319-497	2006-04-17 00:00:00	2312,20
5	Patinter	989319-487	2006-04-18 00:00:00	1927,54

É possível chamar um procedimento armazenado a partir de outro procedimento armazenado. Pode mesmo chamar um procedimento armazenado a partir de si próprio. A esta técnica atribui-se comummente a designação de chamada recursiva. Contudo, esta técnica é pouco utilizada na programação SQL.

1.3 *Script* que cria um procedimento armazenado que copia uma tabela.

```
USE Pagamentos

IF OBJECT_ID('spCópiaFacturas') IS NOT NULL
   DROP PROC spCópiaFacturas
GO

CREATE PROC spCópiaFacturas
AS
    IF OBJECT_ID('CópiaFacturas') IS NOT NULL
       DROP TABLE CópiaFacturas
    SELECT *
    INTO CópiaFacturas
    FROM Facturas
```

Neste exemplo, começa-se por verificar se o procedimento armazenado *spCópiaFacturas* já existe. Em caso afirmativo, procede-se à sua eliminação. Depois, cria-se novamente o procedimento. Verifica-se se já existe uma tabela com a designação *CópiaFacturas*. Em caso afirmativo, procede-se à sua eliminação. Depois, cria-se novamente a tabela, a partir da tabela *Facturas*.

1.4 Instrução que chama/executa o procedimento armazenado criado no passo anterior.

```
EXEC spCópiaFacturas
```

1.5 Execute a seguinte instrução *SELECT*.

```
Select *
FROM CópiaFacturas
```

A seguir, apresenta-se o resultado da execução da instrução *SELECT* anterior.

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

Até ao momento, os exemplos apresentados não contemplaram a utilização de parâmetros nos procedimentos armazenados.

Para declarar um parâmetro dentro de um procedimento armazenado, coloque o nome do parâmetro, seguido do tipo de dados que lhe é associado. O nome do parâmetro deve começar com o carácter @ e o seu tipo de dados pode ser um qualquer tipo de dados aceite pelo SQL SERVER, exceto *Table*. Os parâmetros são sempre locais ao procedimento. Se utilizar mais do que um parâmetro, as suas declarações devem ser separadas por vírgulas.

Os procedimentos armazenados permitem a utilização de dois tipos distintos de parâmetros: de entrada e de saída. Um parâmetro de entrada é passado para o procedimento armazenado a partir, por exemplo, de uma aplicação.

Um parâmetro de saída é passado do procedimento armazenado para, por exemplo, uma aplicação e é identificado pela palavra-chave *OUTPUT*, no momento da sua declaração. Se esta palavra for omitida na declaração do parâmetro, assume-se que se trata de um parâmetro de entrada.

Pode declarar um parâmetro de entrada de modo a que ele requeira um valor ou que o seu valor seja opcional. No primeiro caso, se não for passado nenhum valor ao parâmetro, é devolvido um erro. Pode identificar um parâmetro opcional atribuindo-lhe um valor predefinido no momento da sua declaração. Depois, esse valor irá ser utilizado, caso não seja passado nenhum valor da aplicação para o procedimento armazenado. Embora também possa definir parâmetros de saída opcionais, não existem, usualmente, razões para o fazer. É uma boa prática de programação declarar primeiro os parâmetros que requerem a passagem de valores, seguidos pelos parâmetros opcionais.

A seguir, apresenta-se a sintaxe para a declaração de parâmetros num procedimento armazenado.

```
@nome_do_parâmetro1 Tipo_de_dados [= valor_predefinido] [OUTPUT]
[,@nome_do_parâmetro2 Tipo_de_dados [= valor_predefinido] [OUTPUT]] …
```

A seguir, apresentam-se algumas declarações típicas de parâmetros.

```
-- parâmetro de entrada que aceita uma data
@DataEnc smalldatetime
-- parâmetro de entrada opcional que aceita uma cadeia de caracteres
@NomeFornecedor varchar(40) = NULL
-- parâmetro de saída que passa um valor monetário
@TotalFactura Money OUTPUT
```

Também pode utilizar parâmetros de saída como parâmetros de entrada, isto é, pode passar um valor, a partir de uma aplicação, para o procedimento armazenado, através de um parâmetro de saída. Contudo, esta é uma forma inusual de usar os parâmetros de saída. Para evitar

confusões, deve utilizar os parâmetros de saída apenas para passar valores do procedimento para, por exemplo, uma aplicação.

Dentro de um procedimento armazenado, os parâmetros são tratados como variáveis. Embora se possa modificar o valor de um parâmetro de entrada dentro do procedimento, essa alteração não é devolvida para a aplicação que passou o parâmetro, nem tem qualquer efeito sobre ela. Todavia, quando o procedimento termina, os valores dos seus parâmetros de saída são passados para a aplicação que chamou o procedimento armazenado.

1.6 Criação de um procedimento armazenado que utiliza dois parâmetros de entrada opcionais e um parâmetro de saída.

```
USE Pagamentos

IF OBJECT_ID('spTotalFacturaFornecedor') IS NOT NULL
    DROP PROC spTotalFacturaFornecedor
GO

CREATE PROC spTotalFacturaFornecedor
       @Total money OUTPUT,
       @Data smalldatetime = NULL,
       @NomeFornecedor varchar(40) = '%'
AS

IF @Data IS NULL
   SELECT @Data = MIN(DataFactura) FROM Facturas

SELECT @Total = SUM(TotalFactura)
FROM Facturas JOIN Fornecedores
    ON Fornecedor = IDFornecedor
WHERE ((DataFactura >= @Data) AND (Nome LIKE @NomeFornecedor))
```

Utilizando o *Management Studio* para passar os valores aos parâmetros de entrada de um procedimento, utilize a instrução *EXEC*. Pode passar os valores por posição ou por nome (dos parâmetros).

Para passar valores por posição, liste-os, separados por vírgula, na mesma ordem em que eles estão declarados no procedimento armazenado. Quando utiliza esta técnica, pode omitir os valores a passar aos parâmetros opcionais apenas se eles estiverem declarados depois dos parâmetros que requerem valores de entrada.

Para passar valores por nome, utilize o seu nome seguido do sinal de = e do valor a passar. Quando usa esta técnica, pode listar os valores a passar, separados por vírgulas, em qualquer ordem e pode omitir facilmente os valores a passar para os parâmetros opcionais.

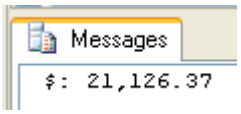
Para utilizar o valor passado por um parâmetro de saída de um procedimento armazenado numa aplicação (por exemplo, no *Management Studio*), tem que declarar uma variável, com um tipo de dados compatível, para armazenar o valor. Depois, pode utilizar o nome dessa variável na instrução *EXEC*, usada para chamar o procedimento armazenado, seguido da palavra-chave *OUTPUT*.

1.7 Instruções que permitem passar valores de parâmetros por posição.

```
USE Pagamentos

DECLARE @MeuTotal money
EXEC spTotalFacturaFornecedor @MeuTotal OUTPUT, '2006-01-01', 'P%'
PRINT '$: ' + CONVERT(varchar,@MeuTotal,1)
```

A seguir, apresenta-se o resultado da execução das instruções do passo anterior.

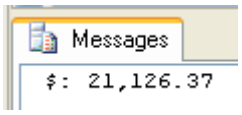


1.8 Instruções que permitem passar valores de parâmetros por nome.

```
USE Pagamentos

DECLARE @MeuTotal money
EXEC spTotalFacturaFornecedor @Data = '2006-01-01',
                              @NomeFornecedor = 'P%',
                              @Total = @MeuTotal OUTPUT
PRINT '$: ' + CONVERT(varchar,@MeuTotal,1)
```

A seguir, apresenta-se o resultado da execução das instruções do passo anterior.

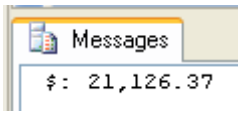


1.9 Instrução que omite o valor a passar a um parâmetro opcional.

```
USE Pagamentos

DECLARE @MeuTotal money
EXEC spTotalFacturaFornecedor @NomeFornecedor = 'P%',
                              @Total = @MeuTotal OUTPUT
PRINT '$: ' + CONVERT(varchar,@MeuTotal,1)
```

A seguir, apresenta-se o resultado da execução das instruções do passo anterior.



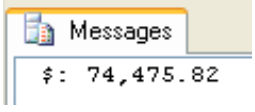
1.10 Instrução que omite os valores a passar aos dois parâmetros opcionais.

```
USE Pagamentos

DECLARE @MeuTotal money
EXEC spTotalFacturaFornecedor @MeuTotal OUTPUT

PRINT '$: ' + CONVERT(varchar,@MeuTotal,1)
```

A seguir, apresenta-se o resultado da execução das instruções do passo anterior.



Além de passar os valores dos parâmetros de saída para uma aplicação, o procedimento armazenado também devolve um valor. De modo predefinido, este valor é zero. Contudo, pode utilizar o comando *RETURN* para devolver outro número. O comando *RETURN* sai imediatamente do procedimento armazenado e devolve o valor à aplicação que chamou o procedimento. Por exemplo, se o procedimento atualizar registos, pode devolver o número de registos que foram atualizados. Para tal, pode usar a função @@*ROWCOUNT*.

Para utilizar, o valor devolvido pelo procedimento armazenado, tem que declarar uma variável na aplicação que chama o procedimento. Depois, utilize essa variável na instrução *EXEC*, seguida de um sinal de igual e do nome do procedimento.

1.11 Criação de um procedimento armazenado que devolve um valor.

```
USE Pagamentos

IF OBJECT_ID('spContagemFacturas') IS NOT NULL
    DROP PROC spContagemFacturas
GO

CREATE PROC spContagemFacturas
       @Data smalldatetime = NULL,
       @NomeFornecedor varchar(40) = '%'
AS

IF @Data IS NULL
   SELECT @Data = MIN(DataFactura) FROM Facturas

DECLARE @NúmeroFacturas int

SELECT @NúmeroFacturas = COUNT(IDFactura)
FROM Facturas JOIN Fornecedores
    ON Fornecedor = IDFornecedor
WHERE ((DataFactura >= @Data) AND (Nome LIKE @NomeFornecedor))

RETURN @NúmeroFacturas
```

Este procedimento armazenado devolve o número de faturas do fornecedor especificado para o segundo parâmetro de entrada do procedimento e cuja data é posterior ou igual à data especificada para o primeiro parâmetro de entrada. Como este parâmetro é opcional, se não for especificada nenhuma data na chamada do procedimento, vai ser considerada a data mais antiga das faturas armazenadas na base de dados. Dado que o segundo parâmetro

também é opcional, pois é-lhe atribuído um valor na sua declaração, se não for especificado nenhum nome de fornecedor na chamada do procedimento, vão ser considerados todos os fornecedores.

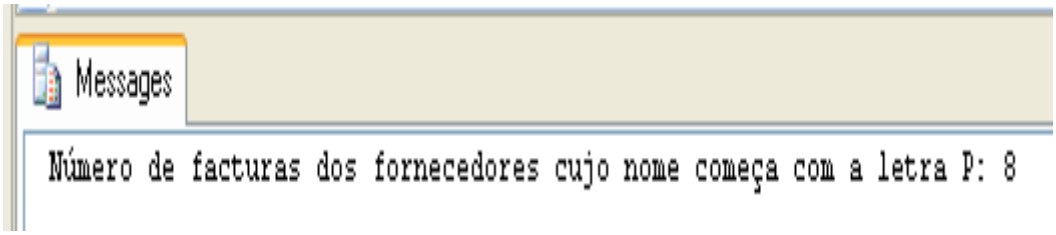
Note que os parâmetros utilizados neste procedimento são idênticos aos utilizados no procedimento anterior. Todavia, dado que este procedimento utiliza o comando *RETURN* para devolver um valor numérico, não existe a necessidade de usar um parâmetro de saída.

1.12 *Script* que chama o procedimento armazenado criado no passo anterior.

```
USE Pagamentos

DECLARE @MeuNúmeroFacturas int
EXEC @MeuNúmeroFacturas = spContagemFacturas '2006-01-01', 'P%'
PRINT 'Número de facturas dos fornecedores cujo nome começa com
      a letra P: ' + CONVERT(varchar, @MeuNúmeroFacturas)
```

A seguir, apresenta-se o resultado da execução das instruções do passo anterior.



Quando chamado com as instruções do passo anterior, o procedimento armazenado criado no passo 1.11 devolve o número de faturas dos fornecedores cujo nome começa com a letra P e cujas datas são posteriores a 01-01-2006.

Pode utilizar a instrução *RAISERROR* para devolver uma mensagem de erro personalizada. A seguir, apresenta-se a sintaxe do comando *RAISERROR*.

RAISERROR ({id_mensagem | mensagem}, severidade, estado)

O argumento *id_mensagem* especifica o número do erro.

O argumento *mensagem* especifica a mensagem de erro. Quando é especificada uma mensagem de erro, é atribuído o valor 50000, de modo predefinido, ao argumento *id_mensagem*.

O argumento *severidade* indica a severidade do erro. Se for uma valor entre 1 e 10, o SQL SERVER considera que o erro é apenas informativo. Consequentemente, a execução do programa não pode "saltar" para um bloco *CATCH*. Se o erro tiver uma severidade entre 11 e 19, o SQL SERVER considera que o erro requer ação. Consequentemente, a execução do programa "salta" para um bloco *CATCH*. Se o erro tiver uma severidade entre 20 e 25, o SQL SERVER considera que o erro deve ser fatal. Nesse caso, a ligação à base de dados é imediatamente terminada.

O argumento *estado* é estritamente informativo e não tem qualquer significado para o sistema. Pode utilizar um qualquer valor entre 1 e 127 para representar o estado em que o sistema estava quando ocorreu o erro. Na maioria dos casos, é utilizado o valor 1 para este argumento.

1.13 Criação de um procedimento armazenado que devolve um valor.

```
USE Pagamentos

GO

IF OBJECT_ID('spInserirFacturas') IS NOT NULL
    DROP PROC spInserirFacturas

GO

CREATE PROC spInserirFacturas

 @IDFactura         int,
 @IDFornecedor      int,           @NúmeroFactura          varchar(50),
 @DataFactura       smalldatetime, @TotalFactura           money,
 @CondiçãoPagamento int,           @DataVencimentoFactura smalldatetime
AS

IF EXISTS(SELECT * FROM Fornecedores WHERE IDFornecedor = @IDFornecedor)
    BEGIN
        INSERT Facturas
        VALUES (@IDFactura, @IDFornecedor, @NúmeroFactura,
                @DataFactura, @TotalFactura, 0, 0,
                @CondiçãoPagamento, @DataVencimentoFactura, NULL)
    END
ELSE
    BEGIN
        RAISERROR('Fornecedor não válido', 11, 1)
    END
```

O procedimento armazenado, após receber os valores para os seus parâmetros de entrada, começa por verificar se o valor passado para o parâmetro @IDFornecedor existe na tabela *Fornecedores*, antes de efetuar a inserção de dados. Deste modo, nunca surgirá um erro provocado pela introdução de uma chave forasteira que viole a regra da integridade referencial. Em vez disso, se o valor passado para o parâmetro *@IDFornecedor* for inválido faz com que o comando *RAISERROR* gere um erro, devolva uma mensagem de erro personalizada e atribuía o valor 5000 ao erro. Dado que, nesse caso, é também atribuída uma severidade de 11 ao erro gerado, o erro pode ser "apanhado" por uma instrução *CATCH*.

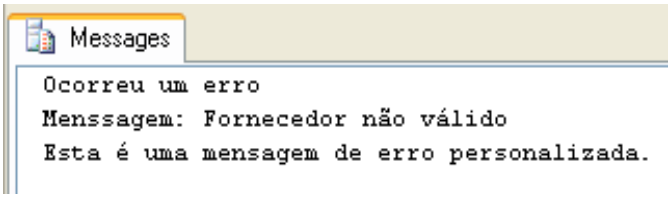
Isso é o que acontece no *sript* apresentado a seguir. Note que irá ser aproveitado, na instrução *CATCH*, o número do erro, através da utilização da função *ERROR_NUMBER*.

1.14 *Script* que chama o procedimento armazenado criado no passo anterior.

```
USE Pagamentos

BEGIN TRY
    EXEC spInserirFacturas
         21,799,'ZXK-799','2006-07-01',299.95,1,'2005-08-01'
END TRY
BEGIN CATCH
    PRINT 'Ocorreu um erro'
    PRINT 'Menssagem: ' + CONVERT(varchar, ERROR_MESSAGE())
    IF ERROR_NUMBER() = 50000
        PRINT 'Esta é uma mensagem de erro personalizada.'
END CATCH
```

A seguir, apresenta-se o resultado da execução das instruções do passo anterior.



1.15 Criação de um procedimento armazenado que faz a validação de dados para a inserção de uma nova fatura na base de dados.

```
USE Pagamentos

GO

IF OBJECT_ID('spInserirFacturasValidação') IS NOT NULL
    DROP PROC spInserirFacturasValidação
GO

CREATE PROC spInserirFacturasValidação
       @IDFactura               int = NULL OUTPUT,
       @Fornecedor              int = NULL,
       @NúmeroFactura           varchar(50) = NULL,
       @DataFactura             smalldatetime = NULL,
       @TotalFactura            money = NULL,
       @Pagamento               money = NULL,
       @Crédito                 money = NULL,
       @CondiçãoPagamento       int = NULL,
       @DataVencimentoFactura   smalldatetime = NULL,
       @DataPagamento           smalldatetime = NULL
AS

IF NOT EXISTS (SELECT * FROM Fornecedores WHERE IDFornecedor = @Fornecedor)
    RAISERROR('Número de fornecedor inválido.', 11, 1)
IF @NúmeroFactura IS NULL
    RAISERROR('Número de factura inválido.', 11, 1)
IF @DataFactura IS NULL OR @DataFactura > GETDATE()
        OR DATEDIFF(dd, @DataFactura, GETDATE()) > 30
    RAISERROR('Data de factura inválida.', 11, 1)
IF @TotalFactura IS NULL OR @TotalFactura <= 0
    RAISERROR('Total de factura inválido.', 11, 1)
IF @Pagamento IS NULL
    SET @Pagamento = 0
IF @Crédito IS NULL
    SET @Crédito = 0
IF @Crédito > @TotalFactura
    RAISERROR('Crédito inválido.', 11, 1)
IF @Pagamento > @TotalFactura - @Crédito
    RAISERROR('Pagamento inválido.', 11, 1)
IF NOT EXISTS (SELECT * FROM CondiçõesPagamento WHERE IDCondição =
@CondiçãoPagamento)
    IF @CondiçãoPagamento IS NULL
        SELECT @CondiçãoPagamento = CondiçãoPagamentoPredefinida
        FROM Fornecedores
        WHERE IDFornecedor = @Fornecedor
    ELSE  -- @CondiçãoPagamento IS NOT NULL
        RAISERROR('Condição de pagamento inválida.', 11, 1)
IF @DataVencimentoFactura IS NULL
    SET @DataVencimentoFactura = @DataVencimentoFactura +
        (SELECT PrazoPagamento FROM CondiçõesPagamento WHERE IDCondição =
@CondiçãoPagamento)
ELSE  -- @DataVencimentoFactura IS NOT NULL
    IF @DataVencimentoFactura < @DataFactura OR
            DATEDIFF(dd, @DataVencimentoFactura, @DataFactura) > 180
        RAISERROR('data de vencimento da factura inválida.', 11, 1)
IF @DataPagamento < @DataFactura OR
             DATEDIFF(dd, @DataPagamento, GETDATE()) > 14
    RAISERROR('Data de pagamento inválida.', 11, 1)

INSERT Facturas
VALUES (@IDFactura, @Fornecedor, @NúmeroFactura, @DataFactura,
@TotalFactura,
```

```
        @Pagamento, @Crédito, @CondiçãoPagamento, @DataVencimentoFactura,
        @DataPagamento)
```

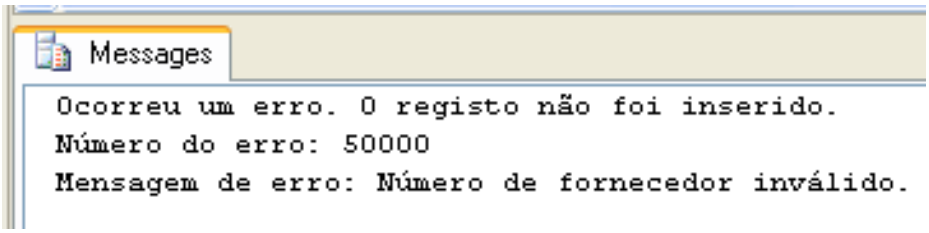
1.16 *Script* que chama o procedimento armazenado criado no passo anterior.

```
USE Pagamentos

GO

BEGIN TRY
    DECLARE @IDFactura int
    EXEC spInserirFacturasValidação
         @IDFactura = 1,
         @Fornecedor = 799,
         @NúmeroFactura = 'RZ99381',
         @DataFactura = '2006-09-12',
         @TotalFactura = 1292.45
    PRINT 'O registo foi inserido com sucesso.'
END TRY
BEGIN CATCH
    PRINT 'Ocorreu um erro. O registo não foi inserido.'
    PRINT 'Número do erro: ' + CONVERT(varchar, ERROR_NUMBER())
    PRINT 'Mensagem de erro: ' + CONVERT(varchar, ERROR_MESSAGE())
END CATCH
```

A seguir, apresenta-se o resultado da execução das instruções do passo anterior



Para além da capacidade de alterar dados de forma permanente, os procedimentos armazenados também proporcionam uma forma eficiente para aceder dados.

1.17 Seguidamente vamos criar um SP para devolver os nomes das três primeiras bases de dados (em função do seu identificador). Para tal, vai-se aceder a dados de tabelas de sistema.

```
CREATE PROCEDURE GetTOP3DBNames
AS
SELECT top 3 name, database_id
```

```
FROM sys.databases
order by database_id
```

1.18 A execução do SP criado permite visualizar o resultado dos dados acedidos. Informe-se acerca do papel destas três bases de dados do SQL Server.

```
exec GetTOP3DBNames
```

	name	database_id
1	master	1
2	tempdb	2
3	model	3

1.19 Vamos agora proceder à inserção dos dados devolvidos pelo SP numa tabela (que vamos criar para o efeito). Como resultado, serão inseridos na tabela 3 registos.

```
CREATE TABLE #TestTable ([name] NVARCHAR(256), [database_ID] INT);
INSERT INTO #TestTable
EXEC GetTOP3DBNames
```

1.20 Vamos agora consultar o conteúdo da tabela.

```
SELECT *
FROM #TestTable;
```

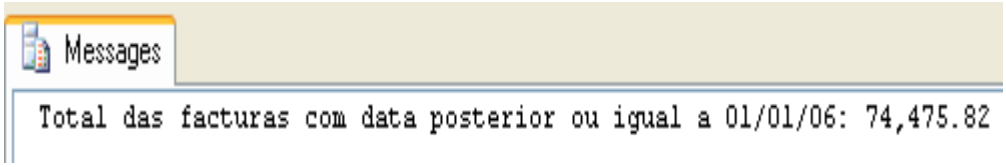
	name	database_id
1	master	1
2	tempdb	2
3	model	3

QUESTÕES

2 Implemente as seguintes instruções.

2.1 Crie um procedimento armazenado, com o nome *spTotalFactura*, que utilize um parâmetro de entrada, com a designação *Data*, para receber uma data e um parâmetro de saída, com a designação *Total*, para passar um valor monetário. O procedimento deve passar para o exterior, através do seu parâmetro de saída, a soma dos montantes das faturas cuja data é posterior ou igual à data fornecida ao procedimento armazenado, através do seu parâmetro de entrada.

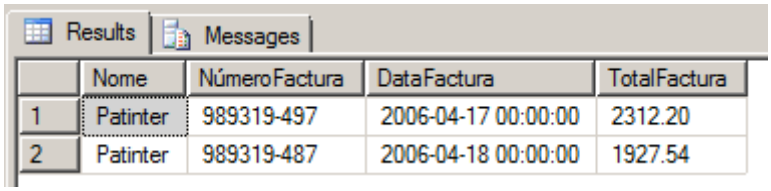
2.2 Implemente as instruções necessárias, de modo a obter a seguinte mensagem no MS SQL SERVER Management Studio.



2.3 Com base no exercício 1.1, crie um procedimento armazenado com o nome *spFaturasEmDividaPorIDFornecedor*, que devolva o Nome do Fornecedor, o Número, a Data e o Total das Factura em dívida, i.e., faturas cujo Total não foi totalmente liquidado. Este resultado deve ser filtrado ainda a um dado IDFornecedor. O IDFornecedor deve ser um parâmetro de entrada deste procedimento armazenado.

2.4 Execute o procedimento armazenado anterior, sem recorrer à declaração de variáveis.

Resultado



	Nome	NúmeroFactura	DataFactura	TotalFactura
1	Patinter	989319-497	2006-04-17 00:00:00	2312.20
2	Patinter	989319-487	2006-04-18 00:00:00	1927.54

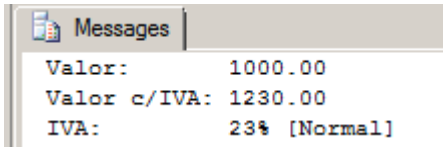
2.5 Crie um procedimento armazenado, com o nome *spAplicaIVA*, que implemente os quatro parâmetros abaixo descritos.

Nome	**Tipo de dados**	**Tipo de parâmetro**
Valor	Monetário	Entrada
TaxaIVA	Texto 8 caracteres	Entrada
ValorComIVA	Monetário	Saída
ValorDoIVA	Decimal	Saída

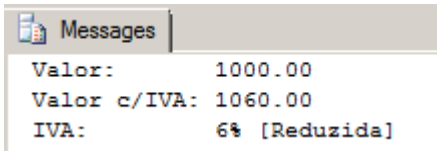
O procedimento recebe um valor monetário (*Valor*) e a taxa de iva (*TaxaIVA*) pretendida ('Normal' ou 'Reduzida'). Posteriormente, se a *TaxaIVA* for 'Normal' deve atribuir o valor de 0.23 ao parâmetro *ValorDoIVA*. Por outro lado, para a *TaxaIVA* = 'Reduzida' deverá atribuir ao *ValorDoIVA* o valor de 0.06. Após determinar a *TaxaIVA* correta, o procedimento deve ainda calcular e devolver para o exterior o *ValorComIVA*.

2.6 Implemente as instruções necessárias, de modo a obter as seguintes mensagens no MS SQL SERVER Management Studio.

Resultado – Taxa 'Normal'



Resultado – Taxa 'Reduzida'

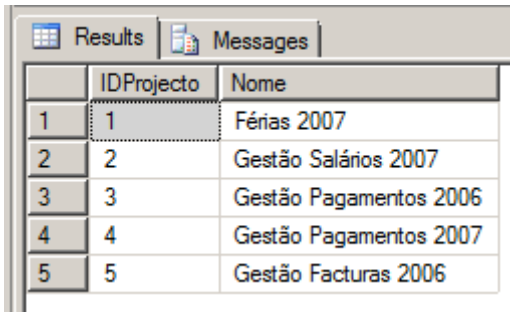


2.7 Crie um procedimento armazenado, com o nome *spTopReg*, que utilize um parâmetro de entrada, com a designação *Tabela* (para receber o nome de um objeto tabela do Sql Server) e um parâmetro de entrada *NRegs* (para receber um número inteiro). Por omissão o parâmetro de entrada *NRegs* deve assumir o valor 5.

O procedimento deve ser capaz de, recorrendo aos parâmetros de entrada, executar uma consulta dinâmica contendo as *NRegs* primeiras linhas da tabela com o nome armazenado no parâmetro *Tabela*.

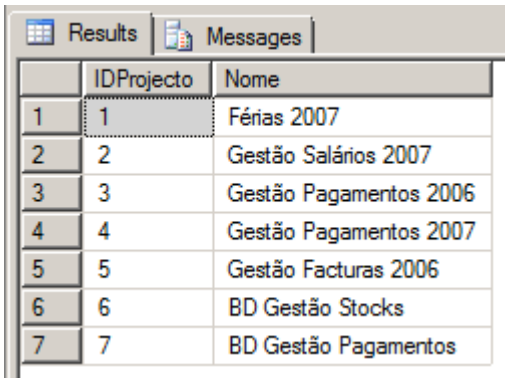
2.8 Execute o procedimento armazenado da questão anterior de três modos distintos (sem declaração de variáveis, com declaração de variáveis e ainda com alteração da ordem dos parâmetros de entrada do procedimento armazenado). Este exemplo recorre à tabela 'Projectos' da base de dados 'Pagamentos'. Implemente todas as instruções necessárias, de modo a obter os resultados abaixo implementados.

Resultados



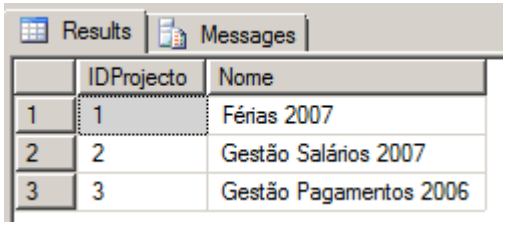
Results | Messages

	IDProjecto	Nome
1	1	Férias 2007
2	2	Gestão Salários 2007
3	3	Gestão Pagamentos 2006
4	4	Gestão Pagamentos 2007
5	5	Gestão Facturas 2006



Results | Messages

	IDProjecto	Nome
1	1	Férias 2007
2	2	Gestão Salários 2007
3	3	Gestão Pagamentos 2006
4	4	Gestão Pagamentos 2007
5	5	Gestão Facturas 2006
6	6	BD Gestão Stocks
7	7	BD Gestão Pagamentos



Results | Messages

	IDProjecto	Nome
1	1	Férias 2007
2	2	Gestão Salários 2007
3	3	Gestão Pagamentos 2006