Lista de Exercícios Resolvidos para Modularização

Prof. Dr. Rodrigo Colnago Contreras
rodrigo.contreras@unesp.br
Lattes Google Scholar ORCID

Criada em 02/09/2024.

Instruções Iniciais

Criação do banco de dados e da tabela alvo dos nossos desafios. Considere-os na solução dos exercícios da sequência.

```sql
-- Tentativa de criação do BD:
BEGIN TRY
    CREATE DATABASE db02092024;
END TRY
BEGIN CATCH
    PRINT('Banco já existente!');
END CATCH;

-- Utilização do BD:
USE db02092024;

-- Criação de uma tabela com 6 valores temporais referentes
-- ao primeiro semestre de uma série financeira.
IF NOT EXISTS(SELECT * FROM sys.tables WHERE name = 'tb_serie_temporal')
BEGIN
    CREATE TABLE tb_serie_temporal(
        id INT PRIMARY KEY IDENTITY(1,1),
        jan DECIMAL(10,2),
        fev DECIMAL(10,2),
        mar DECIMAL(10,2),
        abr DECIMAL(10,2),
        mai DECIMAL(10,2),
        jun DECIMAL(10,2)
    );
END;
```

Exercícios

Functions

1. Acrescente 100 séries temporais aleatórias na tabela, sendo que os valores para as colunas devem ser valores do tipo `DECIMAL(10,2)` gerados no intervalo `[0,10]`.

2. Crie uma `FUNCTION` que calcula a média de uma linha da tabela `tb_serie_temporal` a partir de um `id`. Obs.: $\text{média}(x_1, x_2, ..., x_n) = \frac{\sum_{i=1}^{n} x_i}{n}$.

3. Construa uma `FUNCTION` que calcula as medidas resumo `MÉDIA`, `VARIÂNCIA` e `DESVIO PADRÃO POPULACIONAIS`, sendo todos do tipo `DECIMAL(10,2)` entre as colunas de uma linha da tabela `tb_serie_temporal` a partir de um `id`. O retorno deve ser dado na forma de uma tabela.

   - Variância Populacional: $\text{var}(x_1, x_2, ..., x_n) = \frac{\sum_{i=1}^{n}(x_i - \text{média}(x_1, x_2, ..., x_n))^2}{n}$;
   - Desvio Padrão: $\text{std}(x_1, x_2, ..., x_n) = \sqrt{\text{var}(x_1, x_2, ..., x_n)}$.

Procedures

4. Adicione uma coluna na tabela `tb_serie_temporal` chamada de `jul` do tipo `DECIMAL(10,2)` e escreva uma `PROCEDURE add_mes` que seja definida como a média das linhas anteriores. A `PROCEDURE` deve apresentar a tabela atualizada.

5. Escreva uma `PROCEDURE` que insira uma quantidade inteira `@n` de valores aleatórios na tabela `tb_serie_temporal` seguindo as mesmas condições estabelecidas no Exercício 1. Na sequência, adicione 10, 100 e 1000 linhas na tabela.

   Obs.: Repare que, agora, existe uma coluna `jul` a mais na tabela.

6. Escreva uma `PROCEDURE` que modifique a coluna `jul` acrescentando um ruído aleatório em `[0.0,@alpha]`, com `@alpha` entre 0 e 1 fornecido pelo usuário, em todas as suas entradas.

7. Escreva uma `PROCEDURE` que delete todas as linhas que possuam um valor para as colunas `jan`, `fev`, `mar`, `abr`, `mai`, `jun`, `jul` que sejam respectivamente maiores que `@jan`, `@fev`, `@mar`, `@abr`, `@mai`, `@jun`, `@jul` definidas como parâmetros de entrada desta `PROCEDURE`. Além disso, esta `PROCEDURE` deve retornar a quantidade de linhas que foram removidas da tabela durante a execução da rotina.

8. Escreva uma `PROCEDURE` que crie uma tabela temporária com as colunas `mes` do tipo `VARCHAR(3)` e `mAVG`, `mVAR`, `mSTD`, `mMAX` e `mMIN` como sendo valores do tipo `FLOAT` e, para cada coluna na tabela `tb_serie_temporal`, acrescente uma linha nesta tabela temporária com as respectivas medidas estatísticas `média`, `variância`, `desvio padrão`, `máximo` e `mínimo`.

9. Refaça o exercício anterior utilizando o comando `EXEC sp_executesql` para evitar repetições desnecessárias.

   Dicas:

   - Use o comando `EXEC sp_executesql` para fixar uma coluna da tabela de maneira dinâmica.
   - Crie uma `PROCEDURE` auxiliar que recupera as estatísticas de uma coluna a partir do nome da mesma e que insira tais valores na tabela `#tb_temp`.

10. Crie uma `FUNCTION` e uma `PROCEDURE` que retornem o maior valor de uma dada coluna na tabela `tb_serie_temporal`.

    Obs.: O comando `EXEC` só pode ser utilizado em `PROCEDURES`. `FUNCTIONS` exigem estruturas estáticas e muito mais simples.

Triggers

11. Crie uma `TRIGGER` que impeça a remoção de linhas nas quais a coluna `jun` apresente um valor superior a 7.5.

12. Crie uma `TRIGGER` que limite o valor de `jun` de um novo dado inserido para 7.5.

13. Crie uma `TRIGGER` que impossibilite a atualização de uma linha no caso de que o valor de `jun` seja maior que 7.5.

    Dica: Se um registro é atualizado, o antigo valor entra na tabela DELETED e o novo valor entra na tabela INSERTED.

    Dica: Refaça este exercício utilizando cursores.

14. Crie uma tabela que contabilize a quantidade de linhas inseridas, removidas e/ou atualizadas na tabela `tb_serie_temporal`. Esta tabela deve ter 4 colunas:

    - `id INT`: que associa um identificador na linha;
    - `tipo VARCHAR(MAX)`: que destaca o tipo de modificação na tabela (inserção, remoção ou atualização);
    - `qtde INT`: apresenta a quantidade de linhas modificadas;
    - `horario DATETIME`: que aponta o momento em que a modificação foi efetuada.

Obs.: Para concluir este exercício, será necessário implementar uma `TRIGGER`. Neste caso, escreva apenas uma `TRIGGER`. Dica: É possível definir uma mesma `TRIGGER` para mais de um tipo de evento.

15. Escreva uma `TRIGGER` que garanta que todos os registros inseridos na tabela `tb_serie_temporal` tenham colunas, com exceção da chave primária, com valores apenas entre 0 e 10. Dica: Construa uma função que projete valores dentro do intervalo `[0,10]`.

Sugestões

Sugere-se que o aluno resolva a lista antes de consultar a solução completa. Se encontrar dificuldades, é recomendável visualizar apenas parte da solução.

As soluções foram conduzidas em Transact-SQL (T-SQL) no MSSQL Server 2022. Atente-se para possíveis alterações com respeito a versões do *software*.

Com o surgimento de dúvidas, o professor deverá ser consultado.

Solução dos Exercícios em T-SQL

```
/*
--- Lista de Exercícios Resolvidos para módulos  ---
--- FUNCTIONS, PROCEDURES e TRIGGERS no T-SQL    ---
@ Prof. Dr. Rodrigo Colnago Contreras
@ rodrigo.contreras@unesp.br
@ Criada em 02/09/2024
-------------------------------------------------
*/


-- Tentativa de criação do BD:
BEGIN TRY
    CREATE DATABASE db02092024;
END TRY
BEGIN CATCH
    PRINT('Banco já existente!');
END CATCH;

-- Utilização do BD:
USE db02092024;

-- Criação de uma tabela com 6 valores temporais referentes
--ao primeiro semestre de uma série financeira.
IF NOT EXISTS(SELECT * FROM sys.tables WHERE name = 'tb_serie_temporal')
BEGIN
    CREATE TABLE tb_serie_temporal(
        id INT PRIMARY KEY IDENTITY(1,1),
        jan DECIMAL(10,2),
        fev DECIMAL(10,2),
        mar DECIMAL(10,2),
        abr DECIMAL(10,2),
        mai DECIMAL(10,2),
        jun DECIMAL(10,2)
    );
END;

/*
-----------------------------------
---------- FUNCTIONS --------------
-----------------------------------
*/

/*
Exercício 1: Acrescente 100 séries temporais aleatórias na tabela, sendo que
os valores para as colunas devem ser valores do tipo DECIMAL(10,2) gerados no
intervalo [0,10].
*/

DECLARE @i INT = 0,
        @n INT = 100;
WHILE @i < @n
BEGIN
    DECLARE @jan DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10),
            @fev DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10),
            @mar DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10),
            @abr DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10),
```

```
            @mai DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10),
            @jun DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10);

    INSERT INTO tb_serie_temporal
    VALUES (@jan, @fev, @mar, @abr, @mai, @jun);

    SET @i = @i + 1;
END;

SELECT * FROM tb_serie_temporal;

/*
Exercício 2: Crie uma FUNCTION que calcula a média de uma linha da tabela
tb_serie_temporal a partir de um id.
*/

CREATE OR ALTER FUNCTION dbo.calcula_media(@id INT)
RETURNS DECIMAL(10,2)
AS
BEGIN
    DECLARE @jan DECIMAL(10,2) = (SELECT jan FROM tb_serie_temporal WHERE id =
      @id),
            @fev DECIMAL(10,2) = (SELECT fev FROM tb_serie_temporal WHERE id =
              @id),
            @mar DECIMAL(10,2) = (SELECT mar FROM tb_serie_temporal WHERE id =
              @id),
            @abr DECIMAL(10,2) = (SELECT abr FROM tb_serie_temporal WHERE id =
              @id),
            @mai DECIMAL(10,2) = (SELECT mai FROM tb_serie_temporal WHERE id =
              @id),
            @jun DECIMAL(10,2) = (SELECT jun FROM tb_serie_temporal WHERE id =
              @id);

    RETURN CONVERT(DECIMAL(10,2),(@jan + @fev + @mar + @abr + @mai + @jun) / 6.0);
END;

DECLARE @media DECIMAL(10,2) = dbo.calcula_media(8);
PRINT(@media)

/*
Exercício 3: Construa uma FUNCTION que calcula as medidas resumo MÉDIA, VARIÂNCIA e

DESVIO PADRÃO POPULACIONAIS, sendo todos do tipo DECIMAL(10,2) entre as colunas de
uma linha da tabela tb_serie_temporal a partir de um id.
O retorno deve ser dado na forma de uma tabela.

Fórmulas:
i. Variância Populacional: SUM_i(x_i - média)^2 / n;
ii. Desvio Padrão: SQRT(Variância Populacional).
*/

CREATE OR ALTER FUNCTION dbo.calcula_statistics(@id INT)
RETURNS @tb_retorno TABLE(
            media DECIMAL(10,2),
            variancia DECIMAL(10,2),
            std DECIMAL(10,2)
```

```
        )
AS
BEGIN
    DECLARE @jan DECIMAL(10,2) = (SELECT jan FROM tb_serie_temporal WHERE id =
      @id),
        @fev DECIMAL(10,2) = (SELECT fev FROM tb_serie_temporal WHERE id = @id),
        @mar DECIMAL(10,2) = (SELECT mar FROM tb_serie_temporal WHERE id = @id),
        @abr DECIMAL(10,2) = (SELECT abr FROM tb_serie_temporal WHERE id = @id),
        @mai DECIMAL(10,2) = (SELECT mai FROM tb_serie_temporal WHERE id = @id),
        @jun DECIMAL(10,2) = (SELECT jun FROM tb_serie_temporal WHERE id = @id);

    DECLARE @media DECIMAL(10,2) = dbo.calcula_media(@id);

    DECLARE @var FLOAT;
    SET @var = POWER(@jan-@media,2.0) + POWER(@fev-@media,2.0) + POWER(@mar-
      @media,2.0) +
               POWER(@abr-@media,2.0) + POWER(@mai-@media,2.0) + POWER(@jun-
                 @media,2.0);
    SET @var = @var / 6;
    DECLARE @var_final DECIMAL(10,2) = CONVERT(DECIMAL(10,2),@var);

    DECLARE @std DECIMAL(10,2) = CONVERT(DECIMAL(10,2), SQRT(@var));


    INSERT INTO @tb_retorno VALUES(@media, @var_final, @std);

    RETURN;
END;


SELECT * FROM dbo.calcula_statistics(4);


/*
-----------------------------------
---------- PROCEDURES -------------
-----------------------------------
*/

/*Exercício 4: Adicione uma coluna na tabela tb_serie_temporal chamada de 'jul' do
  tipo
DECIMAL(10,2) e escreva uma procedure add_mes que  que seja definida como a média
  das
linhas anteriores.
A PROCEDURE deve apresentar a tabela atualizada.
*/

-- Primeiro, adicionamos a coluna
ALTER TABLE tb_serie_temporal
ADD jul DECIMAL(10,2) NULL;

CREATE OR ALTER PROCEDURE dbo.add_mes
AS
BEGIN

    UPDATE tb_serie_temporal
```

```
    SET jul = dbo.calcula_media(id);

    SELECT * FROM tb_serie_temporal;
END;

EXEC dbo.add_mes;


/*
Exercício 5: Escreva uma PROCEDURE que insira uma quantidade inteira @n de valores
  aleatórios
na tabela tb_serie_temporal seguindo as mesmas condições estabelecidas no exercício
   1. Na
sequência, adicione 10, 100 e 1000 linhas na tabela.
Obs.: Repare que agora, existe uma coluna 'jul' a mais na tabela.
*/
CREATE OR ALTER PROCEDURE add_rows
@n INT
AS
BEGIN
    DECLARE @i INT = 0;
    WHILE @i < @n
    BEGIN
        DECLARE @jan DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10),
                @fev DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10),
                @mar DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10),
                @abr DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10),
                @mai DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10),
                @jun DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10),
                @jul DECIMAL(10,2) = CONVERT(DECIMAL(10,2),RAND()*10);

        INSERT INTO tb_serie_temporal
        VALUES (@jan, @fev, @mar, @abr, @mai, @jun, @jul);

        SET @i = @i + 1;
    END;
END;

EXEC add_rows 10;
EXEC add_rows 100;
EXEC add_rows 1000;

SELECT COUNT(*) FROM tb_serie_temporal;


/*
Exercício 6: Escreva uma procedure que modifique a coluna 'jul' acrescentando um
  ruído aleatório em [0.0,@alpha],
com @alpha entre 0 e 1 fornecido pelo usuário, em todas as suas entradas.
*/

CREATE OR ALTER PROCEDURE add_ruido
@alpha FLOAT -- valor entre 0.0 e 1.0
AS
BEGIN
    IF ((@alpha < 0) OR (@alpha > 1))
```

```
    BEGIN
        RAISERROR('Valor fora do intervalo permitido para @alpha', 16, 1);
    END;
    ELSE
    BEGIN

        SELECT jul FROM tb_serie_temporal;

        UPDATE tb_serie_temporal
        SET jul = jul + (RAND() * @alpha);
    END;

    SELECT jul FROM tb_serie_temporal;
END;

EXEC add_ruido 0.1;


EXEC add_ruido 2;

/*
Exercício 7: Escreva uma procedure que delete todas as linhas que possuam um valor para as colunas
jan, fev, mar, abr, mai, jun, jul que sejam respectivamente maiores que @jan, @fev, @mar, @abr,
@mai, @jun, @jul definidas como parâmetros de entrada desta procedure.
Além disso, esta procedure deve retornar a quantidade de linhas que foram removidas da tabela
durante a execução da rotina.
*/
CREATE OR ALTER PROCEDURE remove_linhas
@jan  DECIMAL(10,2),
@fev  DECIMAL(10,2),
@mar  DECIMAL(10,2),
@abr  DECIMAL(10,2),
@mai  DECIMAL(10,2),
@jun  DECIMAL(10,2),
@jul  DECIMAL(10,2),
@qtde INT OUTPUT
AS
BEGIN
    -- Variável para armezenar o tamanho inicial da tabela
    DECLARE @qtde_inicial INT = (SELECT COUNT(*) FROM tb_serie_temporal);

    DELETE FROM tb_serie_temporal
    WHERE ((jan < @jan) OR (fev < @fev) OR (mar < @mar) OR (abr < @abr) OR (mai < @mai) OR (jun < @jun) OR (jul < @jul));

    -- Variável para armezenar o tamanho final da tabela
    DECLARE @qtde_final INT = (SELECT COUNT(*) FROM tb_serie_temporal);

    SET @qtde = @qtde_inicial - @qtde_final;
END;

DECLARE @retorno INT;
EXEC remove_linhas 0.5, 0.75, 0.5, 0.5, 0.5, 0.5, 0.5, @retorno OUTPUT;
```

```
PRINT(CONCAT('Quantidade de linhas removidas: ', @retorno));

/*
Exercício 8: Escreva uma PROCEDURE que crie uma tabela temporária com as colunas
'mes' do tipo VARCHAR(3) e 'mAVG', 'mVAR', 'mSTD', 'mMAX' e 'mMIN' como sendo
valores do tipo FLOAT e, para cada coluna na tabela tb_serie_temporal acrescente
uma linha nesta tabela temporária com as respectivas medidas estatísticas média,
variância, desvio padrão, máximo e mínimo.
*/

CREATE OR ALTER PROCEDURE calc_st
AS
BEGIN
    IF NOT EXISTS(SELECT * FROM sys.tables WHERE name = '#tb_temp')
    BEGIN
        CREATE TABLE #tb_temp(
            mes VARCHAR(3) PRIMARY KEY,
            mAVG FLOAT,
            mVAR FLOAT,
            mSTD FLOAT,
            mMAX FLOAT,
            mMIN FLOAT
        );
    END
    ELSE
    BEGIN
        TRUNCATE TABLE #tb_temp;
    END;

    DECLARE @avg FLOAT,
            @var FLOAT,
            @std FLOAT,
            @max FLOAT,
            @min FLOAT;

    SELECT @avg = AVG(jan), @var = VAR(jan), @std = SQRT(VAR(jan)), @max = MAX
      (jan), @min = MIN(jan)
    FROM tb_serie_temporal;

    INSERT INTO #tb_temp VALUES ('jan', @avg, @var, @std, @max, @min);

    SELECT @avg = AVG(fev), @var = VAR(fev), @std = SQRT(VAR(fev)), @max = MAX
      (fev), @min = MIN(fev)
    FROM tb_serie_temporal;

    INSERT INTO #tb_temp VALUES ('fev', @avg, @var, @std, @max, @min);


    SELECT @avg = AVG(mar), @var = VAR(mar), @std = SQRT(VAR(mar)), @max = MAX
      (mar), @min = MIN(mar)
    FROM tb_serie_temporal;

    INSERT INTO #tb_temp VALUES ('mar', @avg, @var, @std, @max, @min);

    SELECT @avg = AVG(abr), @var = VAR(abr), @std = SQRT(VAR(abr)), @max = MAX
      (abr), @min = MIN(abr)
```

```
    FROM tb_serie_temporal;

    INSERT INTO #tb_temp VALUES ('abr', @avg, @var, @std, @max, @min);

    SELECT @avg = AVG(mai), @var = VAR(mai), @std = SQRT(VAR(mai)), @max = MAX
      (mai), @min = MIN(mai)
    FROM tb_serie_temporal;

    INSERT INTO #tb_temp VALUES ('mai', @avg, @var, @std, @max, @min);


    SELECT @avg = AVG(jun), @var = VAR(jun), @std = SQRT(VAR(jun)), @max = MAX
      (jun), @min = MIN(jun)
    FROM tb_serie_temporal;

    INSERT INTO #tb_temp VALUES ('jun', @avg, @var, @std, @max, @min);


    SELECT @avg = AVG(jul), @var = VAR(jul), @std = SQRT(VAR(jul)), @max = MAX
      (jul), @min = MIN(jul)
    FROM tb_serie_temporal;

    INSERT INTO #tb_temp VALUES ('jul', @avg, @var, @std, @max, @min);

    SELECT * FROM #tb_temp;
END;

EXEC calc_st;

/*
Exercício 9: Refaça o exercício anterior utilizando o comando EXEC sp_executesql
  para evitar
repetições desnecessárias.

Dicas:
    1. Use o comando EXEC sp_executesql  <COMANDO SQL>,
                                    <DECLARAÇÃO (SEM DECLARE) DE TODAS AS
                    VARIÁVEIS DE SAÍDA COMO NVARCHAR>,
                                    <VARIÁVEL DE SAÍDA 1> OUTPUT,
                                    <VARIÁVEL DE SAÍDA 2> OUTPUT,
                                    ...
                                    <VARIÁVEL DE SAÍDA n> OUTPUT;
        Obs.: Todas as variáveis de saída devem ser declaradas no mesmo unicode do
          segundo parâmetro.
        Por ex.:
        EXEC sp_executesql @comando, N'@var1 INT OUTPUT, @var2 FLOAT OUTPUT, @var3
          VARCHAR(MAX) OUTPUT',
                                @var1 OUTPUT, @var2 OUTPUT, @var3 OUTPUT;

    2. Crie uma PROCEDURE auxiliar que recupera as estatísticas de uma coluna a
      partir
    do nome da mesma e que insira tais valores na tabela #tb_temp.

Com este comando, você pode fixar uma coluna da tabela de maneira dinâmica.
*/
CREATE OR ALTER PROCEDURE auxilia_add_st
```

```
@col VARCHAR(3)
AS
BEGIN

    DECLARE @avg FLOAT,
            @var FLOAT,
            @std FLOAT,
            @max FLOAT,
            @min FLOAT;

    DECLARE @sqlcmd NVARCHAR(MAX) = N'SELECT @avg = AVG('+@col+'),
                                             @var = VAR('+@col+'),
                                             @std = SQRT(VAR('+@col+')),
                                             @max = MAX('+@col+'),
                                             @min = MIN('+@col+')
                                             FROM tb_serie_temporal;';

    EXEC sp_executesql @sqlcmd, N'@avg FLOAT OUTPUT, @var FLOAT OUTPUT, @std FLOAT
      OUTPUT, @max FLOAT OUTPUT, @min FLOAT OUTPUT',
                                @avg OUTPUT, @var OUTPUT, @std OUTPUT, @max OUTPUT,
                    @min OUTPUT;

    INSERT INTO #tb_temp VALUES (@col, @avg, @var, @std, @max, @min);

END;

CREATE OR ALTER PROCEDURE cria_tb_temp_com_stat
AS
BEGIN
    IF NOT EXISTS(SELECT * FROM sys.tables WHERE name = '#tb_temp')
    BEGIN
        CREATE TABLE #tb_temp(
            mes VARCHAR(3) PRIMARY KEY,
            mAVG FLOAT,
            mVAR FLOAT,
            mSTD FLOAT,
            mMAX FLOAT,
            mMIN FLOAT
        );
    END
    ELSE
    BEGIN
        TRUNCATE TABLE #tb_temp;
    END;

    EXEC auxilia_add_st 'jan';
    EXEC auxilia_add_st 'fev';
    EXEC auxilia_add_st 'mar';
    EXEC auxilia_add_st 'abr';
    EXEC auxilia_add_st 'mai';
    EXEC auxilia_add_st 'jun';
    EXEC auxilia_add_st 'jul';

    SELECT * FROM #tb_temp;
END;
```

```
EXEC cria_tb_temp_com_stat;

/*
Exercício 10: Crie uma function e uma procedure que retornem o maior valor de uma
  dada coluna
na tabela tb_serie_temporal.

Obs.: O comando EXEC só pode ser utilizado em PROCEDURES. FUNCTIONS exigem
  estruturas estáticas
e muito mais simples.
*/

-- FUNCTION que faz isso
CREATE OR ALTER FUNCTION dbo.get_maior_func(@col VARCHAR(3))
RETURNS DECIMAL(10,2)
AS
BEGIN
    DECLARE @maxval DECIMAL(10,2);
    IF @col = 'jan'
    BEGIN
        SET @maxval = (SELECT MAX(jan) FROM tb_serie_temporal);
    END
    ELSE
    BEGIN

        IF @col = 'fev'
        BEGIN
            SET @maxval =  (SELECT MAX(fev) FROM tb_serie_temporal);
        END
        ELSE
        BEGIN
            IF @col = 'mar'
            BEGIN
                SET @maxval = (SELECT MAX(mar) FROM tb_serie_temporal);
            END
            ELSE
            BEGIN

                IF @col = 'abr'
                BEGIN
                    SET @maxval = (SELECT MAX(abr) FROM tb_serie_temporal);
                END
                ELSE
                BEGIN

                    IF @col = 'mai'
                    BEGIN
                        SET @maxval = (SELECT MAX(mai) FROM tb_serie_temporal);
                    END
                    ELSE
                    BEGIN

                        IF @col = 'jun'
                        BEGIN
                            SET @maxval = (SELECT MAX(jun) FROM tb_serie_temporal);
                        END
```

```
                        ELSE
                        BEGIN
                            SET @maxval = -1000;
                        END;
                    END;
                END;
            END;
        END;
    END;
    RETURN @maxval;
END;

DECLARE @retorno DECIMAL(10,2) = dbo.get_maior_func('fev');
PRINT(@retorno);



CREATE OR ALTER PROCEDURE get_maior_proc
@col VARCHAR(3),
@maxval DECIMAL(10,2) OUTPUT
AS
BEGIN
    DECLARE @sql NVARCHAR(MAX);

    SET @sql = N'SELECT @maxval = MAX(' + @col + ') FROM tb_serie_temporal';

    EXEC sp_executesql @sql, N'@maxval DECIMAL(10,2) OUTPUT', @maxval OUTPUT;
END;

DECLARE @retorno_maxval DECIMAL(10,2);
DECLARE @mes VARCHAR(3) = 'fev';
EXEC get_maior_proc @mes, @retorno_maxval OUTPUT;
PRINT(@retorno_maxval);

/*
-----------------------------------
------------ TRIGGERS -------------
-----------------------------------
*/

/*
Exercício 11: Crie uma TRIGGER que impeça a remoção de linhas nas quais a coluna
'jun' apresente um valor superior a 7.5.
*/

CREATE OR ALTER TRIGGER tg_impede_remocao
ON tb_serie_temporal
INSTEAD OF DELETE -- Substituição da remoção
AS
BEGIN

    DELETE FROM tb_serie_temporal
    WHERE id IN (SELECT id FROM DELETED WHERE jun <= 7.5);

END;
```

```sql
SELECT * FROM tb_serie_temporal;

DELETE FROM tb_serie_temporal WHERE id = 1;

/*
Exercício 12: Crie uma TRIGGER que limite o valor de 'jun' de um novo dado inserido
para 7.5.
*/

CREATE OR ALTER TRIGGER limita_jun
ON tb_serie_temporal
AFTER INSERT
AS
BEGIN

    UPDATE tb_serie_temporal
    SET jun = 7.5
    WHERE ((id IN (SELECT id FROM INSERTED)) AND (jun > 7.5));

END;

INSERT INTO tb_serie_temporal
VALUES (0,0,0,0,0,10,0);

SELECT * FROM tb_serie_temporal
WHERE jan = 0 AND fev = 0 AND mar = 0;


/*
Exercício 13: Crie uma trigger que impossibilite a atualização de uma linha
no caso de que o valor de 'jun' seja maior que 7.5.

Dica: Se um registro é atualizado, o antigo valor entra na tabela DELETED e o
novo valor entra na tabela INSERTED.

Dica: Refaça este exercício utilizando cursores.
*/

CREATE OR ALTER TRIGGER analisa_atualizacao
ON tb_serie_temporal
INSTEAD OF UPDATE
AS
BEGIN

    UPDATE tst
    SET tst.jan = tbins.jan,
        tst.fev = tbins.fev,
        tst.mar = tbins.mar,
        tst.abr = tbins.abr,
        tst.mai = tbins.mai,
        tst.jun = tbins.jun
    FROM tb_serie_temporal tst
    INNER JOIN INSERTED tbins
    ON tst.id = tbins.id
    WHERE tbins.jun <= 7.5;
```

```
END;

SELECT * FROM tb_serie_temporal WHERE id < 10;

UPDATE tb_serie_temporal
SET jun = 7
WHERE id = 9;

SELECT * FROM tb_serie_temporal WHERE id = 9;

/*
Exercício 14: Crie uma tabela que contabilize a quantidade de linhas
inseridas, removidas e/ou atualizadas na tabela tb_serie_temporal.
Esta tabela deve ter 4 colunas:

i. id INT: que associa um identificador na linha;
ii. tipo VARCHAR(MAX): que destaca o tipo de modificação na tabela
(inserção, remoção ou atualização);
iii. qtde INT: apresenta a quantidade de linhas modificadas;
iv. horario DATETIME: que aponta o momento em que a modificação foi
efetuada.

Obs.: Para concluir este exercício, será necessário implementar uma TRIGGER.
Neste caso, escreva apenas UMA TRIGGER.

Dica: É possível definir uma mesma TRIGGER para mais de um tipo de
evento. Por exemplo, se quisermos que algo ocorra no momento de uma
inserção, remoção ou atualização, então definimos a TRIGGER como

CREATE OR ALTER TRIGGER on_events
ON tabela
[AFTER | INTEAD OF] INSERT, DELETE, UPDATE
AS
...

*/

CREATE TABLE tb_atualizacoes(
    id      INT PRIMARY KEY IDENTITY(1,1),
    tipo    VARCHAR(MAX),
    qtde    INT,
    horario DATETIME
);

CREATE OR ALTER PROCEDURE atualiza_tb_atualizacoes
@tipo VARCHAR(MAX),
@qtde INT
AS
BEGIN
    INSERT INTO tb_atualizacoes
    VALUES(@tipo, @qtde, GETDATE());
END;

CREATE OR ALTER TRIGGER on_modify
ON tb_serie_temporal
```

```
AFTER INSERT, DELETE, UPDATE
AS
BEGIN
    DECLARE @qtde_removidos INT = (SELECT COUNT(*) FROM DELETED);
    DECLARE @qtde_inseridos INT = (SELECT COUNT(*) FROM INSERTED);

    DECLARE @tipo VARCHAR(MAX),
            @qtde INT;

    IF ((@qtde_removidos = 0) AND (@qtde_inseridos != 0))
    BEGIN
        SET @tipo = 'Inserção';
        SET @qtde = @qtde_inseridos;
    END
    ELSE
    BEGIN
        IF ((@qtde_removidos != 0) AND (@qtde_inseridos = 0))
        BEGIN
            SET @tipo = 'Remoção';
            SET @qtde = @qtde_removidos;
        END
        ELSE -- quantidade de removidos = quantidade de inseridos != 0
             -- CASO DE ATUALIZAÇÃO!
        BEGIN
            SET @tipo = 'Atualização';
            SET @qtde = @qtde_removidos;
        END;
    END;

    IF @qtde != 0 -- Elimina as 'tentativas' de atualização controladas pelas
      demais TRIGGERs
        EXEC atualiza_tb_atualizacoes @tipo, @qtde;
END;

SELECT * FROM tb_atualizacoes;

EXEC add_rows 10;
-- Repare que alguns tipos são de atualização
-- Isso acontece, pois já definimos triggers
-- que limitam os valores inseridos numa tabela por atualizações...

DELETE FROM tb_serie_temporal WHERE id >= 1000;

DELETE FROM tb_atualizacoes;

/*
Exercício 15: Escreva uma TRIGGER que garanta que todos os registros inseridos na
  tabela
tb_serie_temporal tenham colunas, com exceção da chave primária, com valores apenas
   entre 0 e 10.

Dica: Construa uma função que projete valores dentro do intervalo [0,10]. Isto é,
f(x) = {10, se x > 10;
        0,  se x < 0;
        x,  se 0 <= x <= 10.
Na sequência, utilize uma atualização composta no caso de alguma coluna não atender
```

```
   aos
requisitos estabelecidos.
*/

CREATE OR ALTER FUNCTION dbo.ajeita_limites(@x DECIMAL(10,2), @inf DECIMAL(10,2),
  @sup DECIMAL(10,2))
RETURNS DECIMAL(10,2)
AS
BEGIN

    IF @sup < @inf
    BEGIN
        DECLARE @temp DECIMAL(10,2) = @inf;
        SET @inf = @sup;
        SET @sup = @temp;
    END;

    DECLARE @retorno DECIMAL(10,2) = @x;

    IF @x < @inf
        SET @retorno = @inf;
    ELSE
        IF @x > @sup
            SET @retorno = @sup;

    RETURN @retorno;
END;
GO

CREATE OR ALTER TRIGGER on_insert_tb_serie_temporal
ON tb_serie_temporal
AFTER INSERT
AS
BEGIN

    UPDATE tb_serie_temporal
    SET jan = dbo.ajeita_limites(jan,0,10),
        fev = dbo.ajeita_limites(fev,0,10),
        mar = dbo.ajeita_limites(mar,0,10),
        abr = dbo.ajeita_limites(abr,0,10),
        mai = dbo.ajeita_limites(mai,0,10),
        jun = dbo.ajeita_limites(jun,0,10),
        jul = dbo.ajeita_limites(jul,0,10)
    WHERE (id IN (SELECT id FROM inserted));

END;
GO

INSERT INTO tb_serie_temporal
VALUES (-1,-1,11,11,-1,12,11);
GO

SELECT * FROM tb_serie_temporal ORDER BY id DESC;
GO
```