Lista de Exercícios Resolvidos para Cursores

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

Criada em 02/09/2024.

Criação da Base de Dados e Tabela

```sql
-- Criação da base de dados
CREATE DATABASE db02092024;
USE db02092024;

-- Tabela especializada em localizar partículas (componentes de games)
-- num espaço 2-D [50,100] x [50,100].
CREATE TABLE tb_coordenadas(
    id      INT PRIMARY KEY IDENTITY(1,1),
    nome    VARCHAR(MAX),
    x       FLOAT,
    y       FLOAT
);
```

Exercícios

Exercício 1

Objetivo: Introduzir 100 linhas aleatórias na tabela `tb_coordenadas` considerando que:

- O nome deve ser composto por "partícula i", sendo i o número da linha a ser inserida.
- `x` e `y` devem ser valores aleatórios entre 50 e 100.

Exercício 2

Objetivo: Fazer uma cópia da tabela `tb_coordenadas` para uma tabela `tb_copia` de mesma estrutura. Esta cópia deve ser feita utilizando cursores.

Exercício 3

Objetivo: Utilizando cursores, detectar quais são as DUAS partículas mais próximas e quais são as DUAS mais distantes entre si.

Observação: A distância entre duas partículas (x_1, y_1) e (x_2, y_2) deve ser calculada como:

$$\text{Distância} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Exercício 4

Objetivo: Efetuar as seguintes rotinas:

1. Deletar quatro linhas da tabela `tb_coordenadas` de forma aleatória. Isto é, definir aleatoriamente quatro índices da tabela e remover as linhas correspondentes.

2. Apresentar quais foram as linhas removidas de `tb_coordenadas`.

3. Utilizando cursores, recuperar as linhas deletadas de `tb_coordenadas` a partir de uma varredura na tabela `tb_copia`.

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 cursores ---
@ Prof. Dr. Rodrigo Colnago Contreras
@ rodrigo.contreras@unesp.br
@ Criada em 02/09/2024
-------------------------------------------------
*/

CREATE DATABASE db02092024;
USE  db02092024;

/*
Tabela especializada em localizar partículas (componentes de games)
num espaço 2-D [50,100] x [50,100].
*/
CREATE TABLE tb_coordenadas(
    id      INT PRIMARY KEY IDENTITY(1,1),
    nome    VARCHAR(MAX),
    x       FLOAT,
    y       FLOAT
);

/* ---------------------------------- Exercícios
  ---------------------------------- */

/*
Exercício 1: Introduzir 100 linhas aleatórias na tabela tb_coordenadas considerando
   que:

i.  O nome deve ser composto por "partícula i", sendo 'i' o número da linha a ser
  inserida.
ii. 'x' e 'y' devem ser valores aleatórios entre 50 e 100.
*/

-- Inserir 100 linhas aleatórias
DECLARE @i INT = 0,
        @n INT = 100;
WHILE @i < @n
BEGIN
    DECLARE @nome   VARCHAR(MAX) = CONCAT('partícula ', @i+1),
            @x      FLOAT = (50 + 50 * RAND()),
            @Y      FLOAT = (50 + 50 * RAND());
    INSERT INTO tb_coordenadas
    VALUES (@nome, @x, @y);

    SET @i = @i + 1;
END;

/*
Exercício 2: Faça uma cópia da tabela tb_coordenadas para uma tabela tb_copia de
  mesma estrutura.
Esta cópia deve ser feita utilizando cursores.
*/

-- Fazer a cópia dessa tabela para uma tb_copia usando cursores
```

```sql
-------------------------------------------------------------
-- Utilizando @@FETCH_STATUS
-------------------------------------------------------------

-- Copiar a estrutura
CREATE TABLE tb_copia(
    id      INT PRIMARY KEY IDENTITY(1,1),
    nome    VARCHAR(MAX),
    x       FLOAT,
    y       FLOAT
);
SET IDENTITY_INSERT tb_copia ON;


-- Declaração
DECLARE cursor_ex CURSOR
FOR
SELECT * FROM tb_coordenadas;

-- Abertura
OPEN cursor_ex;

-- Utilização
DECLARE @id INT, @nome VARCHAR(MAX), @x FLOAT, @y FLOAT;

FETCH NEXT FROM cursor_ex INTO @id, @nome, @x, @y;

WHILE @@FETCH_STATUS = 0 -- É o fim do cursor = False
BEGIN

    INSERT INTO tb_copia (id, nome, x, y)
    VALUES (@id, @nome, @x, @y);

    FETCH NEXT FROM cursor_ex INTO @id, @nome, @x, @y;
END;
-- Fechamento

CLOSE cursor_ex;
DEALLOCATE cursor_ex;

SET IDENTITY_INSERT tb_copia OFF;

SELECT * FROM tb_copia;


-------------------------------------------------------------
-- Utilizando o tamanho da informação tabular do cursor --
-------------------------------------------------------------

-- Copiar a estrutura
CREATE TABLE tb_copia2(
    id      INT PRIMARY KEY IDENTITY(1,1),
    nome    VARCHAR(MAX),
    x       FLOAT,
    y       FLOAT
);
```

```
SET IDENTITY_INSERT tb_copia2 ON;



-- Declaração
DECLARE cursor_ex2 CURSOR
FOR
SELECT * FROM tb_coordenadas;

-- Abertura
OPEN cursor_ex2;

-- Utilização
DECLARE @i INT = 1, @tamanho INT = (SELECT COUNT(*) FROM tb_coordenadas);
DECLARE @id INT, @nome VARCHAR(MAX), @x FLOAT, @y FLOAT;

WHILE @i <= @tamanho
BEGIN

    FETCH NEXT FROM cursor_ex2 INTO @id, @nome, @x, @y;

    INSERT INTO tb_copia2 (id, nome, x, y)
    VALUES (@id, @nome, @x, @y);

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

CLOSE cursor_ex2;
DEALLOCATE cursor_ex2;

SET IDENTITY_INSERT tb_copia2 OFF;

SELECT * FROM tb_copia2;
/*
Exercício 3: Utilizando cursores, detecte quais são as DUAS partículas mais
  próximas e quais são as DUAS mais distantes entre si.
Obs.: A distância entre duas partículas (x1,y1) e (x2,y2) deve ser calculada como
  SQRT(POWER(x1-x2,2.0) + POWER(Y1-Y2,2.0)).
*/

-- Vamos resolver este exercício em duas etapas: uma para detectar as duas mais
  próximas e outra para detectar as mais distantes.

-- Para detectar as mais próximas:
-- Vamos utilizar um cursor para fazer a varredura na tabela tb_coordenadas.

-- Declaração
DECLARE cursor_ex3 CURSOR
FOR
SELECT * FROM tb_coordenadas;

-- Abertura

OPEN cursor_ex3;
-- Varredura
```

```
DECLARE @id INT, @nome VARCHAR(MAX), @x FLOAT, @y FLOAT;

FETCH NEXT FROM cursor_ex3 INTO @id, @nome, @x, @y;

DECLARE @min_dist FLOAT = 50 * SQRT(2),
        @min_id_1 INT,
        @min_id_2 INT;

WHILE @@FETCH_STATUS = 0
BEGIN

    -- Vamos precisar fazer uma nova varredura para comparar a linha fixada no cursor
    -- com as demais linhas da tabela.
    -- Então vamos abrir um novo cursor para isso:
    --Declaração
    DECLARE cursor_ex3_2 CURSOR
    FOR
    SELECT * FROM tb_coordenadas;
    --Abertura
    OPEN cursor_ex3_2;
    --Varredura
    DECLARE @id_2 INT, @nome_2 VARCHAR(MAX), @x_2 FLOAT, @y_2 FLOAT;
    FETCH NEXT FROM cursor_ex3_2 INTO @id_2, @nome_2, @x_2, @y_2;
    WHILE @@FETCH_STATUS = 0
    BEGIN

        IF @id_2 > @id --  Menos eficiente mas funcional: @id_2 != @id
        BEGIN
            DECLARE @dist FLOAT = SQRT(POWER(@x - @x_2,2.0) + POWER(@y - @y_2,2.0));

            IF @dist < @min_dist
            BEGIN
                SET @min_dist = @dist;
                SET @min_id_1 = @id;
                SET @min_id_2 = @id_2;
            END;
        END;

        FETCH NEXT FROM cursor_ex3_2 INTO @id_2, @nome_2, @x_2, @y_2;
    END;
    --Fechamento
    CLOSE cursor_ex3_2;
    DEALLOCATE cursor_ex3_2;


    FETCH NEXT FROM cursor_ex3 INTO @id, @nome, @x, @y;
END;

-- Fechamento
CLOSE cursor_ex3;
DEALLOCATE cursor_ex3;

SELECT  @min_dist AS 'Distância Mínima';
```

```
SELECT * FROM tb_coordenadas
WHERE id in (@min_id_1, @min_id_2);

--------------------------------------------------------------------------------
  --

-- Agora para detectar as duas partículas mais distantes entre si, o código é
  análogo
--mudando apenas a atualização da distância armazenada.

DECLARE cursor_ex3 CURSOR
FOR
SELECT * FROM tb_coordenadas;

-- Abertura

OPEN cursor_ex3;
-- Varredura

DECLARE @id INT, @nome VARCHAR(MAX), @x FLOAT, @y FLOAT;

FETCH NEXT FROM cursor_ex3 INTO @id, @nome, @x, @y;

DECLARE @max_dist FLOAT = 0,
        @max_id_1 INT,
        @max_id_2 INT;

WHILE @@FETCH_STATUS = 0
BEGIN

    -- Vamos precisar fazer uma nova varredura para comparar a linha fixada no
      cursor
    -- com as demais linhas da tabela.
    -- Então vamos abrir um novo cursor para isso:
    --Declaração
    DECLARE cursor_ex3_2 CURSOR
    FOR
    SELECT * FROM tb_coordenadas;
    --Abertura
    OPEN cursor_ex3_2;
    --Varredura
    DECLARE @id_2 INT, @nome_2 VARCHAR(MAX), @x_2 FLOAT, @y_2 FLOAT;
    FETCH NEXT FROM cursor_ex3_2 INTO @id_2, @nome_2, @x_2, @y_2;
    WHILE @@FETCH_STATUS = 0
    BEGIN

        IF @id_2 > @id --  Menos eficiente mas funcional: @id_2 != @id
        BEGIN
            DECLARE @dist FLOAT = SQRT(POWER(@x - @x_2,2.0) + POWER(@y -
              @y_2,2.0));

            IF @dist > @max_dist
            BEGIN
                SET @max_dist = @dist;
                SET @max_id_1 = @id;
                SET @max_id_2 = @id_2;
```

```
            END;
        END;

        FETCH NEXT FROM cursor_ex3_2 INTO @id_2, @nome_2, @x_2, @y_2;
    END;
    --Fechamento
    CLOSE cursor_ex3_2;
    DEALLOCATE cursor_ex3_2;


    FETCH NEXT FROM cursor_ex3 INTO @id, @nome, @x, @y;
END;

-- Fechamento
CLOSE cursor_ex3;
DEALLOCATE cursor_ex3;

SELECT  @max_dist AS 'Distância Máxima';
SELECT * FROM tb_coordenadas
WHERE id in (@max_id_1, @max_id_2);


/*
Exercício 4: Efetue as seguintes rotinas:

i. Delete quatro linhas da tabela tb_coordenadas de forma aleatória. Isto é,
defina aleatoriamente quatro índices da tabela e remova as linhas correspondentes.

ii. Apresente quais foram as linhas removidas de tb_coordenadas.

iii. Utilizando cursores, recupere as linhas deletadas de tb_coordenadas a partir
de uma varredura na tabela tb_copia.

*/

-- i. ------------------------------------------------------------
SELECT COUNT(*) AS 'Tamanho inicial da Tabela' FROM tb_coordenadas;
DECLARE @i INT = 0;
WHILE @i < 4
BEGIN
    DECLARE @tamanho INT = (SELECT COUNT(*) FROM tb_coordenadas);
    -- gerar um ID entre 1 e o tamanho da tabela (IDs já eliminados inclusos)
    DECLARE @id_gerado INT = CONVERT(INT, RAND() * (@tamanho - 1) + 1);

    BEGIN TRY -- A cláusula TRY é utilizada para o caso em que o ID gerado não
      existe (já foi removido, por ex.)
        -- Tente remover
        DELETE FROM tb_coordenadas WHERE id = @id_gerado;

        SET @i = @i + 1;
    END TRY
    BEGIN CATCH
        PRINT('ID não existente gerado');
    END CATCH;
END;
SELECT COUNT(*) AS 'Tamanho final da Tabela' FROM tb_coordenadas;
```

```sql
-- ii. ----------------------------------------------------------

-- Declaração
DECLARE cursor_ex4 CURSOR
FOR
SELECT id FROM tb_copia;

-- Abertura
OPEN cursor_ex4;

-- Varredura
DECLARE @id INT;
FETCH NEXT FROM cursor_ex4 INTO @id;

IF NOT EXISTS(SELECT * FROM sys.tables WHERE name = 'id_removidos')
BEGIN
    CREATE TABLE id_removidos(id INT PRIMARY KEY);
END
ELSE
BEGIN
    TRUNCATE TABLE id_removidos;
END;

WHILE @@FETCH_STATUS = 0
BEGIN
    DECLARE @qtde INT = (SELECT COUNT(*) FROM tb_coordenadas WHERE id = @id);

    IF @qtde = 0
    BEGIN
        INSERT INTO id_removidos VALUES(@id);
    END;

    FETCH NEXT FROM cursor_ex4 INTO @id;
END;
-- Fechamento
CLOSE cursor_ex4;
DEALLOCATE cursor_ex4;

SELECT id AS 'IDs que foram aleatoriamente deletados' FROM id_removidos;
SELECT * FROM tb_copia tc
JOIN id_removidos ir
ON tc.id = ir.id;

-- iii. ---------------------------------------------------------

-- Declaração
DECLARE cursor_ex4 CURSOR
FOR
SELECT * FROM tb_copia;

-- Abertura
OPEN cursor_ex4;

-- Varredura
DECLARE @id INT, @nome VARCHAR(MAX), @x FLOAT, @y FLOAT;
```

```
FETCH NEXT FROM cursor_ex4 INTO @id, @nome, @x, @y;

IF NOT EXISTS(SELECT * FROM sys.tables WHERE name = 'id_removidos')
BEGIN
    CREATE TABLE id_removidos(id INT PRIMARY KEY);
END
ELSE
BEGIN
    TRUNCATE TABLE id_removidos;
END;

WHILE @@FETCH_STATUS = 0
BEGIN
    DECLARE @qtde INT = (SELECT COUNT(*) FROM tb_coordenadas WHERE id = @id);

    IF @qtde = 0 -- id foi removido
    BEGIN
        SET IDENTITY_INSERT tb_coordenadas ON;
        INSERT INTO tb_coordenadas (id, nome, x, y)
        VALUES(@id, @nome, @x, @y);
    END;

    FETCH NEXT FROM cursor_ex4 INTO @id, @nome, @x, @y;
END;

SET IDENTITY_INSERT tb_coordenadas OFF;
-- Fechamento
CLOSE cursor_ex4;
DEALLOCATE cursor_ex4;

SELECT * FROM tb_coordenadas;
```