Stored Procedures (SQL/PL) - Aula 09

Gabriel de Paula Gaspar Pinto

Exercício 1

Stored procedures, no SQL, é um segmento de código SQL armazenado no próprio banco de dados e pode ser chamado por um programa, pelo banco de dados ou até mesmo por outra stored procedure.

Exercício 2

Os modos possíveis de passagem de parâmetros são o IN, OUT e INOUT. Um exemplo de IN, é quando um gerente de um banco quiser ver as transações de um cliente específico, o CPF de tal cliente será passado como IN. Um exemplo de OUT, é a pesquisa do salário de um funcionário, no qual a procedure retorna o salário por um parâmetro OUT. Por fim, um exemplo de INOUT, é um sistema de caixa de mercado, que calcula o preço de um item e aplica taxas e descontos (prática comum nos Estados Unidos), no qual o preço de um produto na prateleira é passado como IN e o preço com taxas/impostos e descontos retorna como OUT.

Exercício 3

a) Para criar a procedure, foi usado o seguinte código:

```
DELIMITER //
    CREATE PROCEDURE somar(IN num1 INT, IN num2 INT, OUT soma INT)
    BEGIN
        SET resultado = num1 + num2
    END
DELIMITER ;

```

Agora, para usar a procedure feita, foi usado o seguinte código:

```
CALL somar(5, 7, @soma);
SELECT @soma AS 'Soma'

```

b) Para criar a procedure, foi usado o seguinte código:

```
DELIMITER //
    CREATE PROCEDURE par_ou_impar(IN num INT, OUT resultado VARCHAR(5))
    BEGIN
        IF MOD(num, 2) = 0 THEN
            SET resultado = 'Par';
        ELSE
            SET resultado = 'Impar';
        END IF;
    END //
DELIMITER ;

```

Agora, para usar esta procedure, foi usado o seguinte código:

```
CALL par_ou_impar(58769, @resultado);
SELECT @resultado AS 'Par ou Impar'

```

Exercício 4

a) Para criar a função, foi utilizado o código abaixo. O MySQL estava reclamando da ausência do "DETERMINISTIC", então eu o coloquei.

```
            DELIMITER //
            CREATE FUNCTION idade_garota (nasc DATE) RETURNS INT DETERMINISTIC
            BEGIN
                DECLARE idade INT;
                SET idade = TIMESTAMPDIFF(YEAR, nasc, CURDATE());
                RETURN idade;
            END //
            DELIMITER ;

```

Para retornar a idade de todas as garotas, foi usado a seguinte query:

```
            SELECT nome, idade_garota(nascimento) AS idade FROM garota;

```

Mas, para retornar a idade de uma garota específica, usando o nome como parâmetro, a query fica:

```
            SELECT idade_garota(nascimento) AS idade FROM garota WHERE nome = 'Ana'

```

b) Para criar a stored procedure da pontuação (Lista 4), foi usada a seguinte query:

```
            DELIMITER //
            CREATE PROCEDURE pontuacao (IN id_nome VARCHAR(50), OUT idade INT, OUT media DECIMAL
    (10, 2), OUT total DECIMAL(10,2))
            BEGIN
                SELECT TIMESTAMPDIFF(YEAR, nascimento, CURDATE()) INTO idade FROM garota WHERE
    nome = id_nome;
                SELECT ROUND(AVG(valor_venda), 2) INTO media FROM garota, vendas_biscoito WHERE
    garota.id_garota = vendas_biscoito.id_garota AND garota.nome = id_nome;
                SET total = ROUND(media - (idade * 0.5), 2);
            END //
            DELIMITER ;

```

Para chamar a procedure, usa-se a seguinte query, com o nome como parâmetro, para facilitar o uso da procedure.

```
            CALL pontuacao('Beatriz', @idade, @media, @total);
            SELECT @idade AS 'Idade', @media AS 'Media', @total AS 'Pontuacao Total';

```

Exercício 5

a) Para criação da função de raiz quadrada, foi usada a seguinte query:

```
        DELIMITER $$

        CREATE FUNCTION RaizQuadradaAproximada(c FLOAT)
        RETURNS FLOAT DETERMINISTIC
        BEGIN
            DECLARE x FLOAT;
            DECLARE funcao FLOAT;
            DECLARE erro FLOAT;

            SET x = c / 2.0;
            SET erro = ABS(c - (x * x));

            WHILE erro > 0.0001 DO
                SET funcao = 0.5 * (x + (c / x));
                SET x = funcao;
                SET erro = ABS(c - (x * x));
            END WHILE;

            RETURN x;
        END$$

        DELIMITER ;

```

Para retornar os valores, se usa a seguinte query, com "c"sendo a constante que você quer calcular a sua raiz quadrada:

```
SELECT RaizQuadradaAproximada(c);
```

b) A criação da função que calcula a fórmula de Bhaskara, se deu pela seguinte query

```
DELIMITER $$

CREATE FUNCTION bhaskara(a FLOAT, b FLOAT, c FLOAT)
RETURNS VARCHAR(200) DETERMINISTIC
BEGIN
    DECLARE delta FLOAT;
    DECLARE raiz FLOAT;
    DECLARE x1 FLOAT;
    DECLARE x2 FLOAT;
    DECLARE resultado VARCHAR(200);

    IF a = 0 THEN
        RETURN 'Impossível calcular';
    END IF;

    SET delta = b * b - 4 * a * c;

    IF delta < 0 THEN
        RETURN 'Impossível calcular';
    END IF;

    SET raiz = SQRT(delta);
    SET x1 = (-b + raiz) / (2 * a);
    SET x2 = (-b - raiz) / (2 * a);

    SET resultado = CONCAT('R1 = ', ROUND(x1, 5), ', R2 = ', ROUND(x2, 5));

    RETURN resultado;
END$$

DELIMITER ;
```

Para retornar, usa-se a seguinte query, com a, b e c sendo as mesmas variáveis da fórmula de Bhaskara:

```
SELECT bhaskara(a, b, c);
```

c) Para criar a função de números romanos, foi usado a seguinte query:

```
DELIMITER $$

CREATE FUNCTION romano(arabico INT)
RETURNS VARCHAR(100) DETERMINISTIC
BEGIN
    DECLARE resultado VARCHAR(100) DEFAULT '';

    IF arabico < 1 OR arabico > 3000 THEN
        RETURN 'Numero menor que 1 ou maior que 3000';
    END IF;

    WHILE arabico >= 1000 DO
        SET resultado = CONCAT(resultado, 'M');
        SET arabico = arabico - 1000;
    END WHILE;

    IF arabico >= 900 THEN
        SET resultado = CONCAT(resultado, 'CM');
        SET arabico = arabico - 900;
    END IF;

    IF arabico >= 500 THEN
        SET resultado = CONCAT(resultado, 'D');
        SET arabico = arabico - 500;
    END IF;

```

```
            IF arabico >= 400 THEN
                SET resultado = CONCAT(resultado, 'CD');
                SET arabico = arabico - 400;
            END IF;

            WHILE arabico >= 100 DO
                SET resultado = CONCAT(resultado, 'C');
                SET arabico = arabico - 100;
            END WHILE;

            IF arabico >= 90 THEN
                SET resultado = CONCAT(resultado, 'XC');
                SET arabico = arabico - 90;
            END IF;

            IF arabico >= 50 THEN
                SET resultado = CONCAT(resultado, 'L');
                SET arabico = arabico - 50;
            END IF;

            IF arabico >= 40 THEN
                SET resultado = CONCAT(resultado, 'XL');
                SET arabico = arabico - 40;
            END IF;

            WHILE arabico >= 10 DO
                SET resultado = CONCAT(resultado, 'X');
                SET arabico = arabico - 10;
            END WHILE;

            IF arabico = 9 THEN
                SET resultado = CONCAT(resultado, 'IX');
                SET arabico = arabico - 9;
            END IF;

            IF arabico >= 5 THEN
                SET resultado = CONCAT(resultado, 'V');
                SET arabico = arabico - 5;
            END IF;

            IF arabico = 4 THEN
                SET resultado = CONCAT(resultado, 'IV');
                SET arabico = arabico - 4;
            END IF;

            WHILE arabico >= 1 DO
                SET resultado = CONCAT(resultado, 'I');
                SET arabico = arabico - 1;
            END WHILE;

            RETURN resultado;
        END$$

        DELIMITER ;

```

Para retornar um valor, usa-se a seguinte query, com arabico sendo o valor que você quer converter de arábico para romano:

```
        SELECT romano(arabico);

```

Exercício 6

a) A criação da query:

```
        DELIMITER $$

        CREATE PROCEDURE detalha_usuario(IN in_id INT)
        BEGIN
        SELECT *
        FROM usuarios
```

```
        WHERE id_conta = in_id;
        END$$

        DELIMITER ;

```

Para retornar a procedure

```
        CALL detalha_usuario(id);

```

b) A criação da query:

```
        DELIMITER $$

        CREATE PROCEDURE saldo_corrente(IN in_id INT, OUT out_saldo DECIMAL(10,2))
        BEGIN
        SELECT IFNULL(SUM(saldo),0)
        INTO out_saldo
        FROM conta_corrente
        WHERE id_conta = in_id;
        END$$

        DELIMITER ;

```

Para retornar a procedure

```
        SET @s = 0;
        CALL saldo_corrente(id, @s);
        SELECT @s AS saldo_corrente;

```

c) A criação da query:

```
        DELIMITER $$

        CREATE PROCEDURE saldo_total(IN  in_id INT, OUT out_total DECIMAL(10,2))
        BEGIN
        SELECT IFNULL((SELECT SUM(saldo) FROM conta_corrente  WHERE id_conta = in_id),0) + IFNULL
    ((SELECT SUM(saldo) FROM conta_poupanca WHERE id_conta = in_id),0)
        INTO out_total;
        END$$

        DELIMITER ;

```

Para retornar a procedure

```
        SET @t = 0;
        CALL saldo_total(id, @t);
        SELECT @t AS saldo_total;

```

d) A criação da query:

```
        DELIMITER $$

        CREATE PROCEDURE lista_usuario_email(IN in_id INT)
        BEGIN
        SELECT CONCAT(primeiro_nome, ' ', sobrenome, ' - ', email) AS `Nome / e-mail`
        FROM usuarios
        WHERE id_conta = in_id;
        END$$

        DELIMITER ;

```

Para retornar a procedure

```
        CALL lista_usuario_email(id);

```

Exercício 7

Os adicionar os requisitos descritos a query dada pelo exercício, ela fica da seguinte maneira:

```
DELIMITER $$

CREATE PROCEDURE transfiraCorrenteParaPoupanca(IN id INT, IN valor DECIMAL(10,2))
BEGIN
DECLARE saldo_atual DECIMAL(10,2);

START TRANSACTION;

SELECT saldo INTO saldo_atual
FROM conta_corrente
WHERE id_conta = id;

SELECT 'Antes da transferencia:' AS info;
SELECT saldo AS saldo_corrente FROM conta_corrente WHERE id_conta = id;
SELECT saldo AS saldo_poupanca FROM conta_poupanca WHERE id_conta = id;

IF valor < 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Valor invalido: nao pode ser negativo';
ELSEIF saldo_atual < 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Saldo da conta corrente esta negativo';
ELSEIF valor > saldo_atual THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Valor maior que o saldo disponivel';
END IF;

UPDATE conta_corrente
SET saldo = saldo - valor
WHERE id_conta = id;

UPDATE conta_poupanca
SET saldo = saldo + valor
WHERE id_conta = id;

COMMIT;

SELECT 'Depois da transferencia:' AS info;
SELECT saldo AS saldo_corrente FROM conta_corrente WHERE id_conta = id;
SELECT saldo AS saldo_poupanca FROM conta_poupanca WHERE id_conta = id;

END$$

DELIMITER ;

```

Para as validações, ao invés de utilizar Views, eu considerei o SIGNAL SQLSTATE '45000' uma opção melhor, pelo fato de ser mais simples do que criar uma view, já terminando a execução da procedure e, consequentemente, terminando a execução da transação, de modo que nenhuma alteração no banco é feita, além de imprimir o erro descrito na query. Ao usar o SIGNAL SQLSTATE também melhora a legibilidade e a manutenibilidade do código, considerando que todo o código está em um lugar só.