Banco de Dados

Rebeca Barros

Programas Armazenados

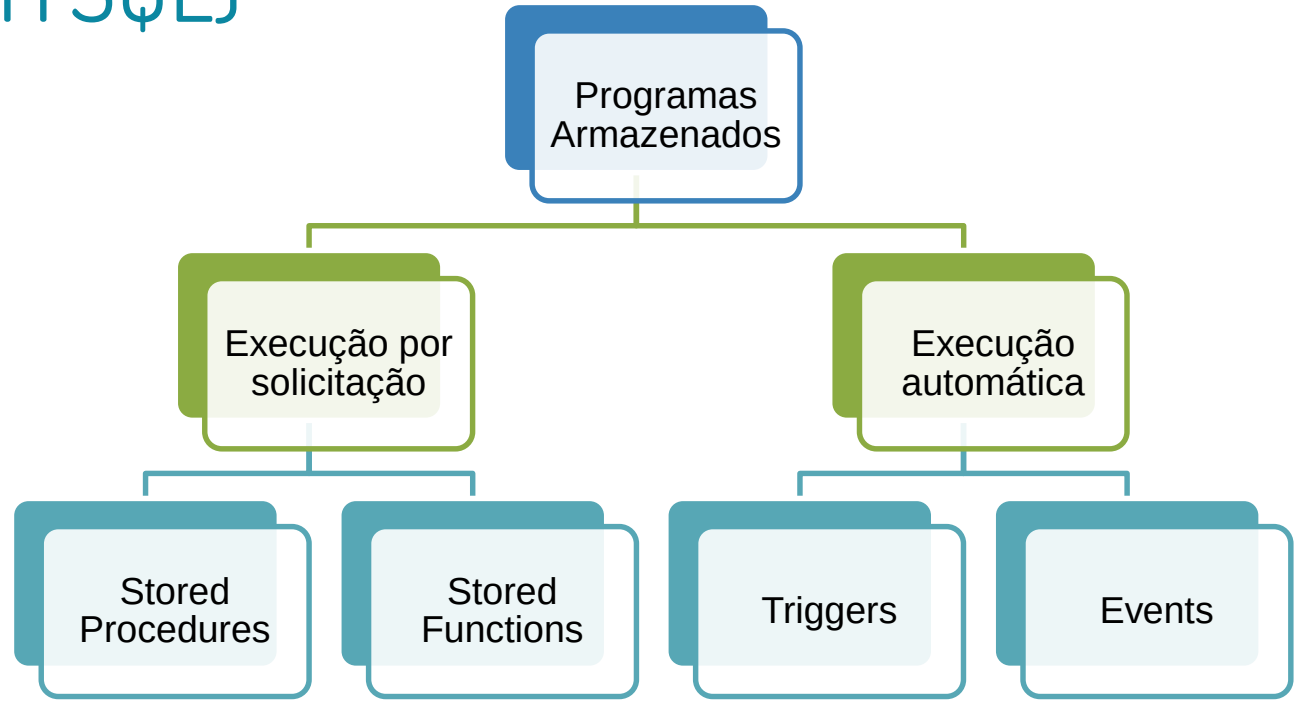
Procedimentos Armazenados (Stored Procedures)

Definição

- O SQL e a maioria dos SGBDs permitem codificar programas dentro do banco de dados com funcionalidades similares a linguagens de programação como PHP, Java ou C.

- Embora limitada quando comparada com essas linguagens, programar em um SGBD (como MySQL) pode ser poderoso e flexível no que diz respeito ao seu principal propósito: realizar alterações e consultas no banco de dados.

Tipos de Programas Armazenados (MYSQL)



Stored Procedures

- Um objeto do banco de dados que contém um bloco de código SQL procedural. Podem ser usados para modificar os dados que estão armazenados no banco.
- Quando o comando CREATE PROCEDURE é executado, o MySQL compila o código e armazena o código compilado no banco de dados.

Sintaxe

```
CREATE PROCEDURE nome_procedure
(
    [variavel1 tipo,
    variavel2 tipo]
)
bloco de instruções sql
```

Sintaxe – Mostrando dados

```
DELIMITER //
CREATE PROCEDURE teste()

BEGIN
    SELECT 'Isso é um teste' AS mensagem;
END
//
DELIMITER ;
```

Variáveis

- Objeto de dado cujo valor pode mudar durante a execução da *Stored Procedure*;
- Variáveis devem ser declaradas antes de serem utilizadas;

Sintaxe - Variáveis

DECLARE nome_variavel tipo(tamanho) **DEFAULT** valor_padrao;

Onde:

- **DECLARE** é a palavra chave para declarar uma nova variável;
- nome_variavel é o nome da variável. A regra pra nomear variáveis é a mesma para nome das colunas no MySQL;
- Tipo e tamanho são o tipo e tamanho da variável, sendo válido os valores do SGBD como INT, VARCHAR ou DATETIME;
- Variáveis inicial com valor *null,* é possível setar um valor padrão com a palavra chave **DEFAULT**

Sintaxe - Variáveis

DECLARE vendedor_id_var INT;

Declara uma variável chamada ‘vendedor_id_var’ que é do tipo inteiro.

Sintaxe - Variáveis

Atribuir valor a uma variável pode ser feito de duas formas:

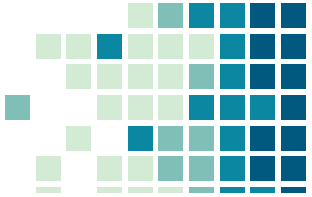
- Comando **SET**
  - **SET vendedor_id_var = 95;**

O valor de vendedor_id_var passa a ser 95 após esse comando.

- **SELECT INTO,** para atribuir a variável o resultado de alguma consulta;
  - **SELECT MAX(fatura_total) INTO max_fatura_total FROM faturas WHERE vendedor_id = 95;**

O valor de max_fatura_total será o maior valor da coluna fatura_total da tabela faturas onde o vendedor_id for 95.

Sintaxe – Variáveis



```
CREATE PROCEDURE teste()

BEGIN
    DECLARE max_fatura_total DECIMAL(9,2);
    DECLARE min_fatura_total DECIMAL(9,2);
    DECLARE diferenca DECIMAL(9,4);
    DECLARE count_fatura_id INT;
    DECLARE vendedor_id_var INT;

    SET vendedor_id_var = 95;

    SELECT MAX(fatura_total), MIN(fatura_total), COUNT(fatura_id)
    INTO max_fatura_total, min_fatura_total, count_fatura_id
    FROM faturas WHERE vendedor_id = vendedor_id_var;
```

Sintaxe – Variáveis

```
SELECT MAX(fatura_total), MIN(fatura_total), COUNT(fatura_id)
INTO max_fatura_total, min_fatura_total, count_fatura_id
FROM faturas WHERE vendedor_id = vendedor_id_var;

SET diferenca = (max_fatura_total - min_fatura_total) /
min_fatura_total * 100 ;

SELECT CONCAT('$', max_fatura_total) AS 'Fatura Max',
       CONCAT('$', min_fatura_total) AS 'Fatura Min',
       CONCAT('%', ROUND(diferenca, 2)) AS 'Diferenca',
       count_fatura_id AS 'Numero de faturas';
END//
```

Escopo - Variáveis

- Cada variável tem sobre próprio escopo, ou seja, o espaço onde ela será válida e poderá ser utilizada.
- Se você declarar uma variável dentro da *stored procedure* ela será válida até atingir o END da procedure.
- Se você declarar uma variável dentro de um bloco BEGIN/END, ela só será válida dentro daquele bloco.
- Uma variável de usuário, cujo nome começa com o símbolo de @, fica disponível globalmente e é válida durante toda a sessão daquele usuário.

Parâmetros

- Quase toda *stored procedure* faz uso de parâmetros. São valores que são passados para a *procedure* pelo programa que a chamou, ou que retornam como resposta para esse mesmo programa.
- No MySQL, um parâmetro pode ser de 3 tipos: **IN**, **OUT** ou **INOUT**

Parâmetros

- **IN** é o tipo padrão. Quando definido, o programa que faz a chamada da *procedure* deve passar um argumento;
- **OUT** – o valor passado como retorno ao programa que chamou a *procedure;*
- **INOUT** – uma combinação dos dois anteriores. Significa que o programa que chamar a *procedure* poderá passar um argumento, a *procedure* poderá modificá-lo e passar o novo valor de volta ao programa.

Sintaxe - Parâmetros

TIPO nome_parametro tipo(tamanho)

Onde:

- Tipo poderá ser IN, OUT ou INOUT;
- Nome, tipo e tamanho seguem as mesmas regras pra variáveis e nomes de colunas no MySQL;
- Cada parâmetro de uma *stored procedure* deve ser separado entre vírgula (,).

```
DROP PROCEDURE IF EXISTS cdPorGravadora;

DELIMITER //

CREATE PROCEDURE cdPorGravadora(
    IN p_nomeG VARCHAR(25),
    OUT p_total INT)

BEGIN
    DECLARE v_idGravadora INT;
    SELECT idGravadora INTO v_idGravadora FROM gravadora where
    nomeGravadora = p_nomeG;
    SELECT count(idCD) INTO p_total FROM cd WHERE
    Gravadora_idGravadora = v_idGravadora group by Gravadora_idGravadora;
END//
DELIMITER ;

CALL cdPorGravadora('BMG', @p_total);

SELECT @p_total AS 'Resultado';
```

Estrutura de Decisão – IF/ELSE

- Assim como na maior parte das linguagens de programação, o comando **IF** no MySQL permite que um conjunto de comandos seja executado de acordo com uma condição ou valor de uma expressão.

IF expressão **THEN**

bloco de código;

END IF;

```sql
CREATE PROCEDURE teste()
BEGIN
    DECLARE vencimento_primeira_fatura DATE;
    SELECT MIN(fatura_data_vencimento)
    INTO vencimento_primeira_fatura
    FROM faturas
    WHERE fatura_total - pagamento_total - credito_total > 0;

    IF vencimento_primeira_fatura < NOW() THEN
        SELECT 'Faturas pendentes vencidas';
    ELSEIF vencimento_primeira_fatura = NOW() THEN
        SELECT 'Faturas pendentes vencem hoje';
    ELSE
        SELECT 'Nenhuma fatura atrasada';
    END IF;
END//
```

Estrutura de Decisão – IF/ELSE

- Fazendo uso do banco de dados '**imobiliariabd**', escreva uma procedure que receba como parâmetros o id do imóvel e o id do comprador e que baseado no valor das ofertas retorne uma mensagem seguindo as seguintes regras:
  - Se a oferta for maior que o preço do imóvel, retorne "Negócio Fechado";
  - se a oferta for menor que o preço em até 5%, retorne "Negociação em andamento";
  - se a oferta for menor que o preço além dos 5%, retorne "Sem negócio".

Estrutura de Decisão – CASE

- Uma outra forma de criar estruturas condicionais no MySQL é o comando CASE:

```
CASE expressão
  WHEN condição_1 THEN comandos
  WHEN condição 2 THEN comandos
  ...
  ELSE comandos
END CASE;
```

```
CREATE PROCEDURE teste()
BEGIN
    DECLARE termos_id_var INT;
    SELECT termos_id INTO termos_id_var
    FROM faturas WHERE fatura_id = 4;
    CASE termos_id_var
        WHEN 1 THEN
            SELECT 'Vencimento 10 dias' AS Termos;
        WHEN 2 THEN
            SELECT 'Vencimento 20 dias' AS Termos;
        WHEN 3 THEN
            SELECT 'Vencimento 30 dias' AS Termos;
        ELSE
            SELECT 'Vencimento mais de 30 dias' AS Termos;
    END CASE;
END//
```

Estrutura de Decisão – CASE

- Fazendo uso do banco de dados '**imobiliariabd**', escreva uma procedure que receba como parâmetro o Estado de um imóvel e exiba o principal número de contato da imobiliária naquela UF.
  - Se o imóvel for de SP o número é (11) 3333-4444;
  - se ele for do RJ, o número é (21) 2222-5555;
  - Se ele for da BA, o número é (71) 7777-3333;
  - e se ele morar em qualquer outro estado o número é (11) 0800-386-0034.

Estrutura de Repetição – WHILE

- MySQL também possui estruturas de repetição, aquelas que permitem a execução de um bloco de código repetidamente baseado em uma condição.

WHILE expressão **DO**

bloco de código;

END WHILE;

```
CREATE PROCEDURE teste()

BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE s VARCHAR(400) DEFAULT '';

    WHILE i < 4 DO
        SET s = CONCAT(s, 'i=', i, ' | ');
        SET i = i + 1;
    END WHILE;

    SELECT s AS mensagem;

END//
```

```
CREATE PROCEDURE teste()

BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE s VARCHAR(400) DEFAULT '';

    REPEAT
        SET s = CONCAT(s, 'i=', i, ' | ');
        SET i = i + 1;
    UNTIL i = 4
    END REPEAT;

    SELECT s AS mensagem;

END//
```

Programas Armazenados

Funções

Funções

- Um tipo especial de programa armazenado que retorna um **único** valor. Funções podem ser usadas para encapsular fórmulas comuns ou regras de negócios que são reutilizadas por outros comandos SQL ou *stored procedures*.

Funções

- As duas principais diferenças entre uma *stored procedure* e uma função são:
  - Um função no MySQL sempre retornará um valor único (ao contrário de alguns outros sgbds);

Sintaxe

```
CREATE FUNCTION nome_funcao (
    [parametro1 tipo, parametro2 tipo]
)

RETURNS tipo [CARACTERISTICAS]
bloco de instruções sql;
```

- Onde características podem ser: **NOT DETERMINIST, DETERMINIST, CONTAINS SQL, NO SQL, READS SQL DATA, MODIFIES SQL DATA**

```
CREATE FUNCTION get_vendedor_id
(
    vendedor_nome_param VARCHAR(50)
)
RETURNS INT
BEGIN
    DECLARE vendedor_id_var INT;

    SELECT vendedor_id
    INTO vendedor_id_var
    FROM vendedores
    WHERE vendedor_nome = vendedor_nome_param;

    RETURN (vendedor_id_var);
END//
```

Funções

- Reescreva na forma de Função, a procedure que retorna a informação sobre o aceite da oferta de acordo com o valor da mesma;