# ✓Banco de Dados II

Base de Dados MySQL

Interface gráfica MySQL WorkBench

# Stored Procedures

## Procedimento Armazenado

Sequência de instruções SQL declarativas armazenada dentro do catálogo de banco de dados.

Pode ser invocado por **gatilhos (triggers)**, outros **procedimentos armazenados** ou por outras **aplicações**, tais como: Java, C #, PHP etc.

Para facilitar a leitura o procedimento armazenado será chamado aqui apenas por procedimento.

# Vantagens

**Desempenho:** Uma vez criados, os procedimentos ficam armazenados. Eles são compilados na demanda e ficam em cache. Se um aplicativo usar o procedimento várias vezes na conexão, a versão compilada é usada.

Isso reduz o tráfego servidor de aplicativos/ BD: Em vez de enviar várias instruções SQL, a aplicação envia somente nome e parâmetros do procedimento.

Os procedimentos são reutilizáveis e transparentes a todas as aplicações.

**Segurança:** O DBA pode conceder permissões à aplicações adequadas para acessar procedimentos no BD, sem dar permissão para as tabelas de banco de dados subjacentes.

# Desvantagens

Se utilizar muitos procedimentos, o uso de memória de cada conexão utilizada aumentará substancialmente. E se utilizar um grande número de operações lógicas no procedimento, o uso de CPU vai aumentar.

É difícil para depurar procedimentos.

Apenas alguns SGBDs permitem depurar procedimentos. Infelizmente, o MySQL não fornece ferramentas eficazes para tal.

Requer habilidades que desenvolvedores de aplicativos iniciantes podem não possuir (precisa dominar  banco de dados).

# Criação

```
DELIMITER $$
CREATE PROCEDURE MostrarProdutos()
   BEGIN
   SELECT *  FROM products;
   END $$
DELIMITER ;
```

O primeiro comando é **DELIMITER** $$, que não está relacionada com a sintaxe do procedimento. A declaração DELIMITER muda o delimitador padrão que é ponto e vírgula (;) para outro, neste caso, para  $$.

É necessário mudar o delimitador para passar o procedimento armazenado para o servidor como um todo ao invés de deixar o MYSQL interpretar cada instrução de cada vez.

Seguindo a palavra-chave **END**, usa-se o demilitador escolhido ($$) para indicar o fim do procedimento armazenado.

O último comando **DELIMITER ;** muda o delimitador de volta para o padrão.

# Execução

Para executar:

```
CALL STORED_PROCEDURE_NAME();
```

Exemplo:

```
CALL MostrarProdutos();
```

# Exclusão

```
DROP PROCEDURE [IF EXISTS];
```

# Variáveis

Uma variável é um objeto de dados nomeado, cujo valor pode mudar durante a execução do procedimento.

Usualmente as variáveis em procedimentos mantém os resultados imediatos.

Essas variáveis são locais para o procedimento.

Declaração da variável dentro de um procedimento:

```
DECLARE variable_name datatype(size) DEFAULT default_value;
```

Exemplos:

```
DECLARE total_sale INT DEFAULT 0;
DECLARE x, y INT DEFAULT 0;
```

# Atribuição de Variáveis

Uma vez declarada uma variável, ela pode ser utilizada. Para atribuir a variável outro valor, use a instrução SET:

```
DECLARE total_count INT DEFAULT 0;
SET total_count = 10;
```

Usa-se SELECT INTO para atribuir o resultado de uma consulta a uma variável. A consulta deve retornar um valor escalar (único):

```
DECLARE total_products INT DEFAULT 0;
SELECT COUNT(*) INTO total_products FROM products;
```

# Escopo de Variáveis

Uma variável tem o seu próprio espaço, que define o seu tempo de vida.

Uma variável declarada em um procedimento, não será mais acessível quando a instrução END for executada.

Uma variável que começa com o sinal @ é variável de sessão. Ela estará disponível e acessível até que a sessão termine.

# Parâmetros (IN, OUT, INOUT)

Parâmetros aumentam a utilidade do procedimento. Em MySQL, um parâmetro tem um dos três modos IN, OUT ou INOUT. A definição de uma lista parâmetros deve ser separada por vírgulas.

**Parâmetro IN (padrão)**

O programa chamador deve passar um argumento para o procedimento armazenado.

O valor de um parâmetro de entrada está protegido: mesmo que o valor do parâmetro seja modificado dentro do procedimento, o valor original é mantido após término do procedimento pois o procedimento funciona com uma cópia do valor.

## Parâmetro OUT

O valor de um parâmetro OUT pode ser alterado durante a execução do procedimento e seu valor final é passado ao programa chamador.

Observe que o procedimento não lê do programa chamador o valor inicial do parâmetro OUT.

## Parâmetro INOUT

Um parâmetro INOUT é a combinação de parâmetro IN e OUT.

O programa chamador passa o argumento, e o procedimento armazenado pode modificar o parâmetro INOUT e passar o novo valor de volta para o programa chamador.

# Parâmetros - Síntase

Síntaxe:

**MODE param_name param_type(param_size)**

Onde:

MODE poder ser IN, OUT ou INOUT.

param_name é o nome do parâmetro. Deve seguir as regras de nomenclatura do nome da coluna no MySQL.

Para finalizar, informa-se o tipo e tamanho dos dados que pode ser qualquer tipo de dados do MySQL.

# Exemplos

**EXEMPLO IN**

```
DELIMITER $$
CREATE PROCEDURE Lista_Func_no_Cargo(IN codcarg CHAR(3))
BEGIN
  SELECT * FROM func WHERE codigo_carg = codcarg;
END $$
DELIMITER;
```

**CHAMADA**

```
CALL Lista_Func_no_Cargo('DBA');
```

**EXEMPLO OUT**

```
DELIMITER $$
CREATE PROCEDURE Qtde_Func(OUT qtde INT)
BEGIN
  SELECT count(*) INTO qtde FROM func;
END $$
DELIMITER;
```

**CHAMADA**

```
CALL Qtde_Func(@qt);
SELECT @qt…
```

# Estruturas de Decisão

```
IF if_expression THEN commands
   [ELSEIF elseif_expression THEN commands]
   [ELSE commands]
END IF;

CASE  case_expression
   WHEN when_expression_1 THEN commands
   WHEN when_expression_2 THEN commands
   ...
   ELSE commands
END CASE;

WHILE expression DO
   Statements
END WHILE;

REPEAT
Statements
UNTIL expression
END REPEAT;
```

# Exemplo IF

```
DELIMITER $$
CREATE PROCEDURE GetCustomerLevel(
    in  p_customerNumber int(11), out p_customerLevel  varchar(10))
BEGIN
    DECLARE creditlim double;
    SELECT creditlimit INTO creditlim FROM customers
    WHERE customerNumber = p_customerNumber;
    IF creditlim > 50000 THEN
SET p_customerLevel = 'PLATINUM';
    ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN
        SET p_customerLevel = 'GOLD';
    ELSEIF creditlim < 10000 THEN
        SET p_customerLevel = 'SILVER';
    END IF;
 END$$
```

# Exemplo CASE

```
DELIMITER $$
CREATE PROCEDURE GetCustomerShipping(in  p_customerNumber int(11),
out p_shipping       varchar(50))
BEGIN
    DECLARE customerCountry varchar(50);
    SELECT country INTO customerCountry
FROM customers
WHERE customerNumber = p_customerNumber;
     CASE customerCountry
WHEN  'USA' THEN
   SET p_shipping = '2-day Shipping';
WHEN 'Canada' THEN
   SET p_shipping = '3-day Shipping';
ELSE
   SET p_shipping = '5-day Shipping';
END CASE;
END$$
```

# Exemplo WHILE

```
DELIMITER $$
DROP PROCEDURE IF EXISTS WhileLoopProc$$
CREATE PROCEDURE WhileLoopProc()
    BEGIN
        DECLARE x  INT;
        DECLARE str  VARCHAR(255);
        SET x = 1;
        SET str =  '';
        WHILE x  <= 5 DO
            SET  str = CONCAT(str,x,',');
            SET  x = x + 1;
        END WHILE;
          SET str = SUBSTRING(str, 1, CHAR_LENGTH(str) - 1); -- Remove a última vírgula
        SELECT str;
    END $$
DELIMITER ;
```

# Exemplo REPEAT

```
DELIMITER $$
DROP PROCEDURE IF EXISTS RepeatLoopProc$$
CREATE PROCEDURE RepeatLoopProc()
    BEGIN
        DECLARE x  INT;
        DECLARE str  VARCHAR(255);
        SET x = 1;
        SET str =  '';
        REPEAT
            SET  str = CONCAT(str,x,',');
            SET  x = x + 1;
        UNTIL x  > 5;
        END REPEAT;
        SET str = SUBSTRING(str, 1, CHAR_LENGTH(str) - 1); -- Remove a última vírgula
        SELECT str;
    END$$
DELIMITER ;
```

# Estruturas de Repetição

Instruções: **LOOP, LEAVE, ITERATE**

**LOOP** permite executar um bloco de código repetidamente com a flexibilidade adicional de usar uma LABEL (etiqueta) para o loop, para posterior referência.

**LEAVE** permite sair do loop imediatamente, funciona como à instrução break em linguagens tais como PHP, C++, Java.

**ITERATE** permite ignorar o código inteiro sob ele e iniciar uma nova iteração. A declaração ITERATE é semelhante à instrução continue em PHP, C++, Java.

# Exemplo de Repetição

```
DELIMITER $$
DROP PROCEDURE IF EXISTS LOOPLoopProc$$
CREATE PROCEDURE LOOPLoopProc()
    BEGIN
        DECLARE x  INT;
        DECLARE str  VARCHAR(255);
        SET x = 1;
        SET str =  '';
        loop_label:  LOOP
            IF  x > 10 THEN
                LEAVE  loop_label;
            END  IF;
            SET  x = x + 1;
            IF  (x mod 2) <> 0 THEN
                ITERATE  loop_label;
            ELSE
                SET  str = CONCAT(str,x,',');
            END  IF;
         END LOOP;
         SET str = SUBSTRING(str, 1, CHAR_LENGTH(str) - 1); -- Remove a última vírgula
        SELECT str;
    END$$
DELIMITER ;
```

# Nesta aula

**Instruções**

DELIMITER
CREATE PROCEDURE nome()
BEGIN
END
CALL nome();
DECLARE variable_name datatype(size) DEFAULT default_value;
SET total_count = 99;
SELECT COUNT(*) INTO total_products FROM products;
MODE param_name param_type(param_size)
IF expression THEN commands
CASE  case_expression
WHILE expression DO
REPEAT
LOOP,
LEAVE,
ITERATE

**Conceitos**

Procedure – criação e execução
Variável, seu escopo, variável de sessão
Parâmetros