13 – Introdução

- Para o desenvolvimento de stored procedures ou qualquer outro tipo de sub-rotina no banco de dados MySQL é necessário o domínio da estrutura de sua linguagem de script.

- O primeiro comando que deve ser analisado é o **DELIMITER**, que tem a função de especificar qual caractere será o finalizador, ou delimitador da rotina.

- O MySQL reconhece como delimitador o caractere “;” (ponto e vírgula), entretanto, no meio da sub-rotina, pode ser necessário utilizar este caractere para encerrar os laços e condições existentes.

- Para evitar erros, deve-se criar um outro delimitador, por meio do comando DELIMITER, para que o MySQL saiba onde se inicia e onde termina a rotina. Por padrão, utiliza-se o caractere “$”, mas não existe uma regra, fica a critério do programador.

13 – Introdução

- Quando o código interno de uma sub-rotina possui duas linhas ou mais, estas devem estar dentro das instruções **BEGIN** e **END**.

- Para criação de variáveis locais, utiliza-se o comando **DECLARE**, seguido do nome da variável e seu tipo.

- Para atribuir valor nas variáveis da rotina, o comando **SET** deve ser utilizado, seguido do nome da variável e do valor a ser atribuído.

- E para criar uma sub-rotina do tipo procedure, utiliza-se o comando **CREATE PROCEDURE,** seguido do nome e dos parâmetros do procedimento.

- Na criação de uma procedure, pode-se utilizar antes do comando CREATE, o comando DROP **PROCEDURE IF EXISTS** seguido do nome da procedure. Desta forma, caso já exista uma procedure com o mesmo nome, ela será excluída para a criação de uma nova.

13.1 – Sintaxe de uma Stored Procedure

- A sintaxe geral de uma stored procedure é mostrada no quadro abaixo:

```
DELIMITER $
DROP PROCEDURE IF EXISTS <nome da procedure>$
CREATE PROCEDURE <nome da procedure>
BEGIN
  <corpo da procedure>
END$
DELIMITER ;
```

13.1 – Sintaxe de uma Stored Procedure

- O exemplo abaixo mostra o código de uma sored procedure básica, onde é exibido na tela do usuário a frase “Hello World”. O comando **CALL** efetua a chamada da procedure:

```
DELIMITER $
DROP PROCEDURE IF EXISTS hello_world$
CREATE PROCEDURE hello_world()
BEGIN
  SELECT 'Hello World';
END$
DELIMITER ;
```

```
CALL Hello_World;
```

Hello World	
Hello World	

13.2 – Exclusão de uma Stored Procedure

- Para excluir uma stored procedure, utiliza-se o comando **DROP PROCEDURE** seguido do nome da procedure. O exemplo abaixo ilustra o código necessário para excluir a procedure Hello_Word:

```
DROP PROCEDURE hello_world;
```

13.3 – Listar Sub-Rotinas do Banco de Dados

- Para listar todas as sub-rotinas (procedure, function e triggers) existentes no banco (todos os databases), pode-se utilizar o comando **SHOW PROCEDURE STATUS**.

- Caso seja necessário uma listagem mais restrita, é possível selecionar o database e a sub-rotina que deverá ser consultado. No exemplo abaixo, serão mostradas apenas as procedures do database que o usuário está conectado:

```
SHOW PROCEDURE STATUS
```

```
SHOW PROCEDURE STATUS WHERE DB = DATABASE() AND Type = 'PROCEDURE'
```

13.4 – Stored Procedure com SQL

- No corpo de uma stored procedure podem ser utilizados todos os comandos SQL tradicionais, tais como: **SELECT, INSERT, UPDATE e DELETE, SUBQUERYS, GROUP BY,** funções de data e hora, etc. Esses recursos SQL podem ser somados aos oferecidos pelas sub-rotinas (IF, WHILE, variáveis, etc.), garantindo grande flexibilidade ao desenvolvedor. O exemplo abaixo mostra o uso de SQL em uma stored procedure:

```
DELIMITER $
CREATE PROCEDURE selecao()
BEGIN
   SELECT * FROM cliente WHERE id_cliente<=3;
END$
DELIMITER ;
```

id_cliente	nome	endereco	id_cidade	telefone	email
1	Paulo Soares	Rua A, 48	1	4499577872	paulo@paulo.com.br
2	Edmundo da Silva	Rua B, 23	5	4199545876	ed@edmundo.com
3	Paulo Nunes	Rua P, 78	2	2399763456	paulo@nunes.com

13.5 – Uso de Variáveis em Stored Procedure

- Os tipos de variáveis que podem ser declarados em uma stored procedure são os mesmos que o MySQL comporta para definir os tipos das colunas. Toda variável precisa ser declarada com o comando **DECLARE**. Para atribuir valor a variável, utiliza-se o comando **SET**, como mostra o exemplo abaixo:

```
DELIMITER $
CREATE PROCEDURE variavel()
BEGIN
  DECLARE mensagem VARCHAR(11);
  SET mensagem = 'Alo, Mundo!';
  SELECT mensagem;
END$
DELIMITER ;
```

mensagem
Alo, Mundo!

13.5 – Uso de Variáveis em Stored Procedure

- O exemplo abaixo mostra o uso de diversos tipos de variáveis. Com o comando DEFAULT é possível atribuir um valor inicial a variável.

```
DELIMITER $
CREATE PROCEDURE tipo_variavel()
BEGIN
  DECLARE valor INT;
  DECLARE resultado BIGINT;
  DECLARE data DATE DEFAULT CURDATE();
  DECLARE mensagem VARCHAR(30) DEFAULT 'Calculo Realizado em ';
  SET valor=5;
  SET resultado=POWER(valor,3);
  SELECT concat(mensagem, data,' =', resultado);
END$
DELIMITER ;
```

concat(mensagem, data,' =', resultado)
Calculo Realizado em 2016-11-17 =125

13.6 – Parâmetros

- Por meio dos parâmetros é possível interagir com as sub-rotinas. Em se tratando de stored procedure, os parâmetros são passados dentro de parênteses que ficam logo após a definição do nome da procedure. Existem basicamente 3 formas de parâmetros, são eles:

1. **IN**: Define os parâmetros de entrada da procedure.

2. **OUT**: Define os parâmetros de saída da procedure.

3. **INOUT**: Possui tanto a função de entrada (IN) como de saída (OUT). Desta forma, é possível passar um valor como parâmetro de entrada para procedure, que por sua vez irá modificar e retornar este valor como parâmetro de saída.

13.6.1 – Parâmetro de Entrada (IN)

- O exemplo abaixo mostra uma stored procedure que calcula a raiz quadrada de um valor inteiro passado como parâmetro de entrada:

```
DELIMITER $
CREATE PROCEDURE raiz(IN valor INT)
BEGIN
  DECLARE resultado FLOAT;
  SET resultado=SQRT(valor);
  SELECT resultado;
END$
DELIMITER ;
```

```
CALL raiz(9);
```

resultado
3

13.6.2 – Parâmetro de Saída (OUT)

- O exemplo abaixo mostra uma procedure que faz a soma dos dois parâmetros de entrada (A e B) e atribui o resultado no parâmetro de saída (S). Para receber 'S', é criada uma variável de usuário '@saida'. Este tipo de variável armazena o valor até o fim da thread do usuário:

```
DELIMITER $
CREATE PROCEDURE soma(IN A INT, IN B INT, OUT S INT)
BEGIN
  SET S = A + B;
END$
DELIMITER ;
```

```
CALL soma(3, 5, @saida);
SELECT @saida;
```

@saida
8

13.6.3 – Parâmetro de Entrada/Saída (INOUT)

- O exemplo abaixo mostra uma procedure que incrementa o valor passado como parâmetro de entrada e saída (INOUT):

```
DELIMITER $
CREATE PROCEDURE incrementa(INOUT cont INT)
BEGIN
  SET cont =  cont + 1;
END$
DELIMITER ;
```

```
SET @saida=10;
CALL incrementa(@saida);
SELECT @saida;
```

@saida
11

13.7 – Execução Condicional (IF)

- Assim como na maioria das linguagens de programação, é possível utilizar o comando **IF – THEN – ELSE ou ELSEIF** para efetuar comparações e condições nas stored procedures.

```
DELIMITER $
CREATE PROCEDURE decisao(IN A INT, IN B INT)
BEGIN
  DECLARE X FLOAT;
  SET X=A+B;
  IF (X<10) THEN SELECT X;
  ELSE
        BEGIN
          SET X=X+1;
          SELECT X;
        END;
  END IF;
END$
DELIMITER ;
```

```
CALL decisao(7,9);
```

X
17

13.8 – Execução Condicional (CASE)

- O exemplo abaixo ilustra uma procedure que retorna o dia da semana em português, utilizando o comando CASE:

```
DELIMITER $
CREATE PROCEDURE dia_semana()
BEGIN
 DECLARE valor INTEGER;
 SET valor = DAYOFWEEK(CURDATE());
 CASE
  WHEN  valor = 1 THEN SELECT 'Domingo';
  WHEN  valor = 2 THEN SELECT 'Segunda-feira';
  WHEN  valor = 3 THEN SELECT 'Terca-feira';
  WHEN  valor = 4 THEN SELECT 'Quarta-feira';
  WHEN  valor = 5 THEN SELECT 'Quinta-feira';
  WHEN  valor = 6 THEN SELECT 'Sexta-feira';
  WHEN  valor = 7 THEN SELECT 'Sabado';
 END CASE;
END$
DELIMITER ;
```

```
CALL dia_semana;
```

Sexta-feira
Sexta-feira

13.9 – Estrutura de Looping(LOOP)

- O exemplo abaixo mostra uma procedure que incrementa o valor passado como parâmetro de entrada e saída (INOUT):

```
DELIMITER $
CREATE PROCEDURE fatorial1(IN valor INTEGER)
BEGIN
  DECLARE fator, i INTEGER;
  SET fator = 1;
  SET i = 1;
  calculo: LOOP
    SET fator = fator * I;
    SET i = i + 1;
    IF (i > valor) THEN
      LEAVE calculo;
    END IF;
  END LOOP calculo;
  SELECT fator;
END$
DELIMITER ;
```

```
CALL fatorial1(5);
```

fator
120

13.10 – Estrutura de Looping(REPEAT)

- O exemplo abaixo mostra uma procedure que incrementa o valor passado como parâmetro de entrada e saída (INOUT):

```
DELIMITER $
CREATE PROCEDURE fatorial2(IN valor INTEGER)
BEGIN
  DECLARE fator, i INTEGER;
  SET fator = 1;
  SET i = 1;
  calculo: REPEAT
    SET fator = fator * i;
    SET i = i + 1;
  UNTIL (i > valor)
  END REPEAT calculo;
  SELECT fator;
END$
DELIMITER ;
```

```
CALL fatorial2(5);
```

fator
120

13.11 – Estrutura de Looping(WHILE)

- O exemplo abaixo mostra uma procedure que incrementa o valor passado como parâmetro de entrada e saída (INOUT):

```
DELIMITER $
CREATE PROCEDURE fatorial3(IN valor INTEGER)
BEGIN
  DECLARE fator, i INTEGER;
  SET fator = 1;
  SET i = 1;
  calculo: WHILE (i <= valor) DO
    SET fator = fator * i;
    SET i = i + 1;
  END WHILE calculo;
  SELECT fator;
END$
DELIMITER ;
```

```
CALL fatorial3(5);
```

fator
120

13.12 – Preenchimento de Tabela

- O exemplo abaixo mostra uma procedure com a finalidade de criar e preencher uma tabela vazia com dados:

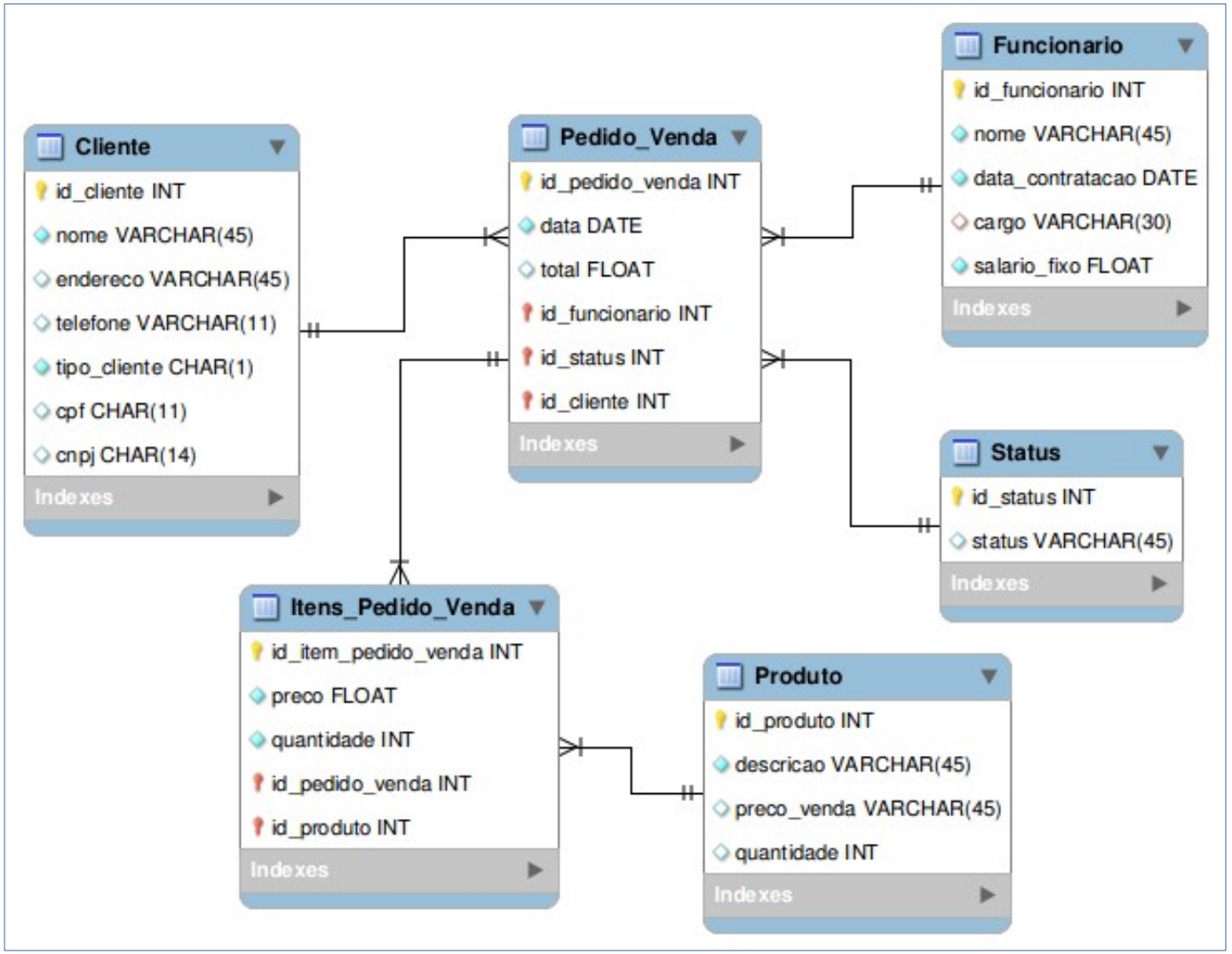
```
DELIMITER $
CREATE PROCEDURE gera_dados()
BEGIN
	DECLARE i INT DEFAULT 1;
	SET autocommit=0;
	DROP TABLE IF EXISTS teste;
	CREATE TABLE teste(
		id_teste INT PRIMARY KEY auto_increment,
		dados VARCHAR(30)) ENGINE=innodb;
	WHILE (i<=10) DO
	BEGIN
		INSERT INTO teste VALUES(NULL, CONCAT("dado ",i));
		SET i=i+1;
	END;
	END WHILE;
END$
```

```
CALL gera_dados();
SELECT * FROM teste;
```

```
+----------+---------+
| id_teste | dados   |
+----------+---------+
|        1 | dado 1  |
|        2 | dado 2  |
|        3 | dado 3  |
|        4 | dado 4  |
|        5 | dado 5  |
|        6 | dado 6  |
|        7 | dado 7  |
|        8 | dado 8  |
|        9 | dado 9  |
|       10 | dado 10 |
+----------+---------+
```

13.12 – Exemplos Práticos

- Os próximos exemplos de stored procedure serão baseados no banco de dados abaixo:



13.12 – Exemplos Práticos

- A stored procedure abaixo mostra quantos pedidos um determinado cliente (pessoa física) efetuou. Será passado como parâmetro o CPF do cliente. O comando **INTO** atribui o valor retornado pelo **SELECT** para a variável:

```
DELIMITER $
CREATE PROCEDURE numero_pedidos(IN cpf_cliente char(11))
BEGIN
  DECLARE num, cod INT;
  SELECT id_cliente INTO cod FROM cliente WHERE cpf=cpf_cliente;
  SELECT COUNT(*) INTO num FROM pedido_venda WHERE id_cliente=cod;
  SELECT concat('O número de pedidos do cliente é: ',num);
END$
DELIMITER ;
```

```
CALL numero_pedidos('05699445977');
```

concat('O número de pedidos do cliente
O número de pedidos do cliente é: 2

13.12 – Exemplos Práticos

- A stored procedure abaixo efetua um aumento de uma porcentagem (passada como parâmetro) no salário de um determinado funcionário (id_funcionario passado como parâmetro). Se o id_funcionario for igual a ‘0’, aumenta para todos:

```
DELIMITER $
CREATE PROCEDURE reajuste_salario(IN id INT, IN valor FLOAT)
BEGIN
        DECLARE max_id INT DEFAULT 1;
        IF (id=0) THEN
   BEGIN
                SELECT MAX(id_funcionario) INTO max_id FROM funcionario;
      WHILE (max_id>0) DO
                        UPDATE funcionario SET salario_fixo=salario_fixo+(salario_fixo*valor/100) WHERE
id_funcionario=max_id;
         SET max_id=max_id-1;
                END WHILE;
        END;
```

13.12 – Exemplos Práticos

```
   ELSE
            UPDATE funcionario SET salario_fixo=salario_fixo+(salario_fixo*valor/100) WHERE
id_funcionario=id;
        END IF;
   SELECT "Reajustado com Sucesso";
END$
```

#	Reajustado com Sucesso
1	Reajustado com Sucesso

13.13 – SQL Dinâmico com Prepared

- O banco de dados possui a facilidade de permitir que uma determinada instrução sql seja preparada no servidor, para que então o usuário precise apenas passar os valores como parâmetro.

- Para lançar a instrução ao servidor, utiliza-se o comando **PREPARE**. Posteriormente, como o comando **EXECUTE**, é possível executar o comando. Com **DEALLOCATE**, o comando será excluído do servidor.

- Os comandos preparados no servidor ficarão em memória apenas durante a seção do usuário, sendo que assim que esta for encerrado o comando será descartado, mesmo sem o uso do DEALLOCATE.

13.13 – SQL Dinâmico com Prepared

- O exemplo abaixo mostra a preparação de um comando para excluir um funcionário. Com o “@” é possível criar as variáveis para serem usadas:

```
PREPARE excluir_funcionario FROM “DELETE FROM funcionario WHERE id_funcionario=?
SET @id=2;
EXECUTE excluir_funcionario USING @id;
DEALLOCATE PREPARE excluir_funcionario
```

13.13 – SQL Dinâmico com Prepared

- O exemplo abaixo mostra como uma procedure pode ser criada para facilitar comandos de alteração, exclusão e até mesmo inserção dos dados. O código abaixo mostra a criação de uma procedura que facilita o UPDATE, não exigindo que se conheça a sintaxe para alteração:

```
DELIMITER $
CREATE PROCEDURE atualiza_salario (tabela VARCHAR(100), coluna VARCHAR(100),
novo_valor FLOAT, condicao_where  VARCHAR(128))
BEGIN
   DECLARE comando VARCHAR(255);
   SET comando=CONCAT_WS(' ',' UPDATE', tabela, 'SET', coluna, '=', novo_valor, 'WHERE', condicao_where);
   SET @sql=comando;
   PREPARE s1 FROM @sql;
   EXECUTE s1;
   DEALLOCATE PREPARE s1;
END$
```

```
CALL atualiza_salario('funcionario','salario_fixo','3220','id_funcionario=2');
```

13.14 – Cursores SQL

- Para manipular uma instrução SELECT que retorna mais de uma linha, é preciso que um cursor SQL seja implementado.

- Um cursor é um objeto que fornece acesso a um conjunto de resultados retornados por uma instrução SELECT inserida em sua estrutura. É possível utilizar o cursor para percorrer as linhas do conjunto de resultados e efetuar alterações para cada linha individualmente.

- O banco permite apenas buscar cada linha no conjunto de resultados do primeiro para o último registro. Não é possível buscar da última para a primeira linha e não é possível saltar diretamente para uma linha específica no conjunto de resultados.

13.14.1 – Estrutura dos Cursores SQL

- O cursor geralmente é criado dentro de uma stored procedure, embora possa ser criado em stored functions e triggers, sendo a sua estrutura mostrada no quadro abaixo:

```
//O cursor deve ser declarado após a declaração de todas as variáveis, senão gera erro
DECLARE <nome> CURSOR FOR <instrução select>


Exemplo:
CREATE PROCEDURE atualiza_salario (id INT)
BEGIN
    DECLARE salario FLOAT;
    DECLARE taxa FLOAT;
    DECLARE c1 CURSOR FOR SELECT id_funcionario, nome FROM funcionario WHERE id_funcionario=id;
```

13.14.2 – Estado dos Cursores SQL

- Os cursores possuem basicamente três estados de operação:

1. **OPEN**: Inicializa o conjunto de resultados do cursor. Ele precisa ser aberto sempre antes de se buscar resultados. Sua sintaxe é: **OPEN** <nome do cursor>

2. **FETCH**: Recupera a linha corrente do cursor e em seguida move o ponteiro para a próxima linha. A variável qu ereceberá o valor corrente deve ser compatível com o tipo de coluna da tabela. Sua sintaxe é: **FETCH** <nome do cursor> **INTO** <variavel>

3. **CLOSE**: Desativa o cursor e libera a memória associada a sua execução. Sua sintaxe é:
CLOSE <nome do cursor>

13.14.3 – Tratar Erro dos Cursores SQL

- Para evitar erros, não se deve permitir que o cursor tente posicionar seu ponteiro para um registro posterior ao último. Po risso, tudo cursor deve tratar esta situação, verificando em cada iteração se já é o último registro. Como mostra o exemplo abaixo:

```
//Cria a variável de controle para testar se é o último registro ou não. Seu valor inicial é 0
DECLARE fim_departamento INT DEFAULT 0;


//Muda o valor da variável para 1 caso o erro NOT FOUND seja gerado (erro por tentar posicionar //ponteiro
após último registro
DECLARE CONTINUE HANDLER FOR NOT FOUND SET fim_departamento=1;


//Cada iteração do LOOP verifica se é o último registro
loop1:LOOP
        FETCH cursor1 INTO codigo, salario_atual;
        IF fim_departamento=1 THEN
                LEAVE loop1;
        END IF;
```

13.14.3 – Tratar Erro dos Cursores SQL

- O exemplo abaixo mostra uma procedure para atualizar o salário de todos os empregados, fazendo o uso de um cursor para ler todos os registros:

```
DELIMITER $
CREATE PROCEDURE atualiza_salario (IN reajuste INT)
BEGIN
    DECLARE fim_departamento INT DEFAULT 0;
    DECLARE codigo INT;
    DECLARE salario_atual FLOAT;
    DECLARE cursor1 CURSOR FOR SELECT id_funcionario, salario_fixo FROM funcionario;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET fim_departamento=1;
    OPEN cursor1;
```

13.14.3 – Tratar Erro dos Cursores SQL

```
    loop1:LOOP
        FETCH cursor1 INTO codigo, salario_atual;
        IF fim_departamento=1 THEN
            LEAVE loop1;
        END IF;
        UPDATE funcionario SET salario_fixo=salario_atual+(salario_atual*reajuste/100)
            WHERE id_funcionario=codigo;
    END LOOP;
    CLOSE cursor1;
    SET fim_departamento=0;
END$
```

```
CALL atualiza_salario(10);
```