**MySQL Procedimentos e Funções**

Neste artigo veremos como criar e utilizar [**Procedimentos e Funções no MySQL**](https://www.devmedia.com.br/procedures-e-funcoes-no-mysql/2550) com auxílio da ferramenta PhpMyAdmin. Antes de começarmos a criação destas rotinas “armazenadas” precisamos entender o que são Procedimentos (Procedures) e Funções (Functions) no **MySql** e porque optamos por [**usar o PhpMyAdmin**](https://www.devmedia.com.br/manipulando-o-mysql-com-o-phpmyadmin/2276).

Procedures e Functions são rotinas armazenadas no **banco de dados** que executam um conjunto definido de passos sequenciais. O conceito é análogo a criação de um método em uma linguagem de programação, onde você pode chamar este método diversas vezes sem necessidade de reescrever código.

Vamos pensar na real necessidade de tais recursos disponíveis em um [**Sistema Gerenciador de Banco de Dados**](https://www.devmedia.com.br/cursos/banco-de-dados), em específico no [**MySQL**](https://www.devmedia.com.br/curso/curso-completo-de-mysql/281).

Imagine a seguinte situação: Por algum motivo um determinado registro pode ser inserido de forma errada no banco de dados, por erro de digitação do usuário que está operando um Sistema X. Você não tem acesso ao código fonte do sistema e não tem como tratar esse erro na sua “fonte” e evitar com que ele ocorra, mas você tem total domínio sobre o banco de dados. Sabendo que este erro ocorre com frequência e você precisa realizar um procedimento de limpeza pelo menos uma vez por semana, o mais fácil será você criar uma rotina que realiza essa limpeza.

Vejamos a solução “errada” na **Listagem 1**.

|  |
| --- |
| UPDATE tabela\_x SET registro\_1 = (registro\_1 + 3) WHERE registro\_2 > 10;  DELETE FROM tabela\_x WHERE registro\_1 < 5;  UPDATE tabela\_y SET registro\_1 = (SELECT MAX(registro\_1) FROM tabela\_x); |

**Listagem 1**. Solução errada

A **Listagem 1** demonstra uma solução genérica e errada para um problema qualquer. Suponha que seja o problema de registro errado que apresentamos logo acima. Toda vez que for necessário fazer a limpeza, o DBA (Database Administrator) deverá executar todos esses comandos (imagine se ele esquecer de algum desses, a consequência pode ser catastrófica). No momento pode parecer fácil, mas lembre-se que isto é apenas um exemplo e na vida real este tipo de correções podem requerer 100, 200 ou mais linhas dependendo da complexidade de tal correção.

Pensando em termos de rotinas poderíamos fazer como no código da **Listagem 2**.

|  |
| --- |
| rotina\_1 {    UPDATE tabela\_x SET registro\_1 = (registro\_1 + 3) WHERE registro\_2 > 10;    DELETE FROM tabela\_x WHERE registro\_1 < 5;    UPDATE tabela\_y SET registro\_1 = (SELECT MAX(registro\_1) FROM tabela\_x);  }    --Executando a rotina\_1  CALL rotina\_1; |

**Listagem 2**. Usando rotina genérica para solucionar a **Listagem 1**

É óbvio que a **Listagem 2** é apenas um exemplo sem muitos detalhes técnicos e de sintaxe da linguagem, mas vale para mostrar que “encapsulamos” todos os códigos da **Listagem 1** na nossa rotina\_1 apresentada na **Listagem 2**. Agora o DBA apenas precisará chamar a rotina\_1 sempre que precisar realizar aquela tal limpeza, ou melhor, como já está tudo pronto, outra pessoa, que não seja o DBA, também poderá executar a rotina\_1tendo certeza que tudo será feito como previsto.

Voltando ao nosso assunto principal, qual a real diferença entre Procedures e Functions? Procedimentos só podem ser chamamos usando a palavra reservada CALL e não retornam nada, enquanto Funções podem ser chamadas [**dentro de um SELECT**](http://www.ricardoarrigoni.com.br/sql-select-guia-para-iniciantes/), por exemplo, e podem retornar um valor. Vamos ver nas próximas seções mais detalhes.

**Stored Procedures**

Primeiro vamos ver um procedimento bem simples e explicar o mesmo com detalhes para só então vermos outros mais complexos. Acompanhe a **Listagem 3**.

|  |
| --- |
| #Criando Procedure   DELIMITER $   CREATE PROCEDURE proc()   BEGIN   SELECT "hello from proc";   END   $    #Chamando a Procedure  CALL proc();    #Resultado  hello from proc |

**Listagem 3**. Exemplo Simples de Procedure

A palavra-chave DELIMITER define um caractere delimitador que é responsável por dizer onde começa e onde termina a nossa função. Em nosso caso definimos o $$ como delimitador, mas poderia ser um outro caractere de sua escolha.

Depois temos a assinatura da nossa função em CREATE PROCEDURE proc() que, em nosso caso, não tem nenhum parâmetro e é bem simples. Os delimitadores do “corpo” da nossa rotina são o BEGIN e o END e dentro destes colocaremos o que nossa rotina deve fazer. Em nosso caso, ela apenas emite uma mensagem hello from proc.

No segundo bloco temos a chamada ao procedimento com o CALL proc();, fazendo assim com que o SGBD execute a função que criamos anteriormente resultando na mensagem hello from proc.

**Functions**

Acompanhe um exemplo de function na **Listagem 4**.

|  |
| --- |
| #Criando FUNCTION   DELIMITER $   CREATE FUNCTION func() RETURNS CHAR(100)   BEGIN    RETURN "hello from func";   END   $    #Chamando func()  SELECT func(); |

**Listagem 4**. Exemplo Simples de Function

Assim como na **Listagem 3,**ainda usamos o DELIMITER com o mesmo objetivo, porém, agora mudamos a palavra-chave de PROCEDURE para FUNCTION e adicionamos RETURNS CHAR(100), ou seja, nossa função deverá retornar um CHAR com tamanho máximo de 100 caracteres.

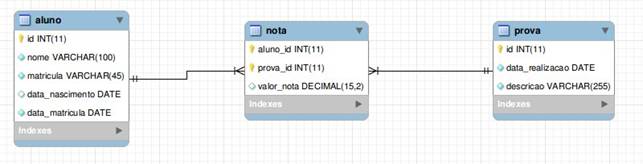
Em geral, tanto a rotina da **Listagem 3** quanto da **Listagem 4** possuem o mesmo objetivo: mostrar no console uma frase qualquer. Porém na função temos a vantagem de poder utilizá-la dentro de um SELECT, como mostrado na **Listagem 4**.

**Caso Real com PhpMyAdmin -Criando nosso Banco de Dados**

Para todos os exemplos mostrados, daqui em diante usaremos a ferramenta PhpMyAdmin que geralmente pode ser acessada através do endereço http://localhost/phpMyAdmin. Mas nada impede que você use outra ferramenta ou até mesmo o próprio terminal do MySQL, fique a vontade.

**Nosso Cenário**: Para ilustrar um exemplo real trabalharemos com a necessidade de criar um banco de dados para armazenar o cadastro e notas dos alunos de uma escola chamada ABC. O banco de dados terá que prover um Cadastro de Provas, Alunos e Notas. Cada Aluno pode ter nenhuma ou várias notas e uma nota só pode ser de uma prova, ou seja, o Aluno “João” tem a nota 8.0 na prova 0010 que foi realizada em 20/06/2014.

Primeiramente vamos criar nossa modelagem para posteriormente iniciar a criação dos scripts necessários para definição e população do nosso banco de dados. Observe a **Figura 1**.

**Figura 1**. Modelagem do Banco

Dada a modelagem acima, podemos criar nosso banco de dados. No PhpMyAdmin, logo que você o acessa, clique na aba “SQL” e você terá um console para digitar os comandos que necessitar. Crie o banco de dados como na **Listagem 5** através deste console e não se esqueça de clicar em executar para que o comando tenha efeito, conforme ilustra a **Figura 2**.

|  |
| --- |
| CREATE DATABASE escola\_abc CHARACTER SET = 'utf8' COLLATE  = 'utf8\_general\_ci' |

**Listagem 5**. Criando Banco de dados

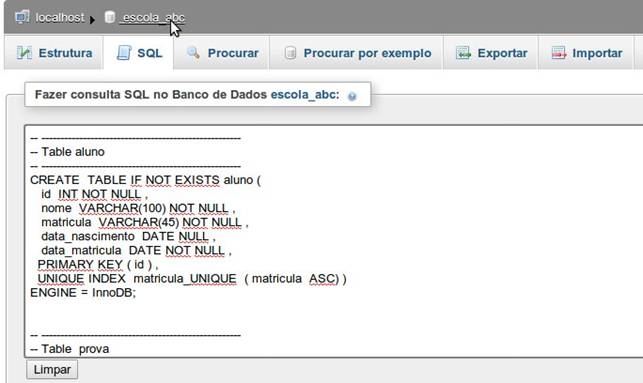
**Figura 2**. Console para Criação do Banco de dados no PhpMyAdmin

Criado nosso banco de dados vamos as definições da sua estrutura. Antes de digitar os comandos da **Listagem 6**, certifique-se que o banco de dados escola\_abc está selecionado. No Topo do PhpMyAdmin você deverá ver algo como: localhost → escola\_abc. Se estiver escrito apenas localhost significa que o banco de dados não foi selecionado.

Com o banco de dados selecionado corretamente, selecione novamente a aba SQL, pois agora executaremos comandos dentro do banco escola\_abc, apenas (**Figura 3**).

|  |
| --- |
| -- -----------------------------------------------------  -- Table aluno  -- -----------------------------------------------------  CREATE  TABLE IF NOT EXISTS aluno (     id  INT NOT NULL ,     nome  VARCHAR(100) NOT NULL ,     matricula  VARCHAR(45) NOT NULL ,     data\_nascimento  DATE NULL ,     data\_matricula  DATE NOT NULL ,    PRIMARY KEY ( id ) ,    UNIQUE INDEX  matricula\_UNIQUE  ( matricula  ASC) )  ENGINE = InnoDB;      -- -----------------------------------------------------  -- Table  prova  -- -----------------------------------------------------  CREATE  TABLE IF NOT EXISTS prova (     id  INT NOT NULL ,     data\_realizacao  DATE NOT NULL ,     descricao  VARCHAR(255) NOT NULL ,    PRIMARY KEY ( id ) )  ENGINE = InnoDB;      -- -----------------------------------------------------  -- Table nota  -- -----------------------------------------------------  CREATE  TABLE IF NOT EXISTS nota (     aluno\_id  INT NOT NULL ,     prova\_id  INT NOT NULL ,     valor\_nota  DECIMAL(15,2) NULL ,    PRIMARY KEY ( aluno\_id ,  prova\_id ) ,    INDEX  fk\_aluno\_has\_prova\_prova1  ( prova\_id  ASC) ,    INDEX  fk\_aluno\_has\_prova\_aluno1  ( aluno\_id  ASC) ,    CONSTRAINT  fk\_aluno\_has\_prova\_aluno1      FOREIGN KEY ( aluno\_id  )      REFERENCES aluno ( id  )      ON DELETE NO ACTION      ON UPDATE NO ACTION,    CONSTRAINT  fk\_aluno\_has\_prova\_prova1      FOREIGN KEY ( prova\_id  )      REFERENCES prova  ( id  )      ON DELETE NO ACTION      ON UPDATE NO ACTION)  ENGINE = InnoDB; |

**Listagem 6**. Criação da Estrutura – DDL

**Figura 3**. Console SQL com banco de dados escola\_abc selecionado

Para finalizar a estruturação do nosso banco e finalmente iniciar a criação das nossas rotinas, precisamos popular o banco de dados com alguns dados (**Listagem 7**), utilizando a mesma aba SQL com o banco de dados selecionado, e não esquecendo de clicar em executar para que o comando tenha efeito.

|  |
| --- |
| -- INSERINDO ALUNOS  INSERT INTO aluno (id, nome, matricula, data\_nascimento, data\_matricula) VALUES  (1,'MICHAEL JONH', '123A', STR\_TO\_DATE('23/08/1993', '%d/%m/%Y'), CURRENT\_DATE()),  (2,'WILLIANS JUNIOR', '400B', STR\_TO\_DATE('10/04/1993', '%d/%m/%Y'), CURRENT\_DATE()),  (3,'JOHN BILLBOARD', '420B', STR\_TO\_DATE('30/07/1993', '%d/%m/%Y'), CURRENT\_DATE()),  (4,'JENNY KILLY', '010A', NULL, STR\_TO\_DATE('25/01/2014', '%d/%m/%Y'))    -- INSERINDO PROVAS  INSERT INTO prova(id, data\_realizacao, descricao) VALUES  (1, STR\_TO\_DATE('30/03/2014', '%d/%m/%Y'), 'Prova A1'),  (2, STR\_TO\_DATE('30/04/2014', '%d/%m/%Y'), 'Prova B1'),  (3, STR\_TO\_DATE('30/05/2014', '%d/%m/%Y'), 'Prova C1'),  (4, STR\_TO\_DATE('30/07/2014', '%d/%m/%Y'), 'Prova A2'),  (5, STR\_TO\_DATE('30/08/2014', '%d/%m/%Y'), 'Prova B2'),  (6, STR\_TO\_DATE('30/09/2014', '%d/%m/%Y'), 'Prova C2')    INSERT INTO nota (aluno\_id, prova\_id, valor\_nota) VALUES  #ALUNO 1  (1, 1, 10),  (1, 2, 9.8),  (1, 3, 8),  (1, 4, 10),  (1, 5, 10),  (1, 6, 9),    #ALUNO 2  (2, 1, 7),  (2, 2, 7.5),  (2, 3, 6),  (2, 4, 8),  (2, 5, 8.5),  (2, 6, 9),    #ALUNO 3  (3, 1, 9),  (3, 2, 9),  (3, 3, 9),  (3, 4, 10),  (3, 5, 10),  (3, 6, 9.8),    #ALUNO 4  (4, 1, 3),  (4, 2, 6),  (4, 3, 7),  (4, 4, 8),  (4, 5, 7),  (4, 6, 7) |

**Listagem 7**. Populando o banco de dados

**Definindo nossas rotinas**

Baseado na nossa estrutura criada cima e os dados que foram populados, nós iremos definir algumas funções que ajudarão você a entender melhor o funcionamento na prática de funções e procedimentos no MySQL.

A criação de funções e procedimentos no PhpMyAdmin pode ser feita no mesmo console que usamos para população dos dados na **Listagem 7**, ou seja, você deve estar com o banco de dados escola\_abc selecionado e clicar na aba SQL (apenas relembrando o processo).

**Função para mostrar nota baseada em peso**

Suponha que a nota da prova de um determinado aluno deva ser calculada por um fator P (chamado de peso) para só então definir a nota final de determinada prova, sendo assim, criaremos uma função que receba como parâmetro a nota e retorne o valor calculado da nota. Observe a **Listagem 8**.

|  |
| --- |
| DELIMITER $  CREATE FUNCTION calc\_nota(nota NUMERIC(15,2)) RETURNS NUMERIC(15,2)  BEGIN   DECLARE peso INT;     #Se a nota do aluno for maior que 9.5 então sua nota terá um peso maior   IF nota > 9.5 THEN    SET peso = 2;   ELSE    SET peso = 1;   END IF;     RETURN (nota\*peso) / 20;  END |

**Listagem 8**. calcula\_nota()

Temos acima um script que recalcula a nota de um aluno baseado em um peso que pode ter o valor de 1 ou 2. Vamos ver na **Listagem 9** como utilizá-lo.

|  |
| --- |
| SELECT a.nome, p.descricao, calcula\_nota(n.valor\_nota)  AS nota\_calculada, n.valor\_nota AS nota\_original  FROM aluno a INNER JOIN nota n ON a.id = n.aluno\_id  INNER JOIN prova p ON n.prova\_id = p.id  ORDER BY a.id, nota\_calculada DESC |

**Listagem 9**. Usando o calcula\_nota()

No SELECT acima nós listamos todos os alunos com suas notas calculadas e as notas originais, ordenando o mesmo por id e valor da nota calculada. Vamos usar a mesma função para ordenar agora por nota média maior, ou seja, a soma de todas as notas calculadas. Observe a **Listagem 10**.

|  |
| --- |
| SELECT a.nome, (SUM(calcula\_nota(n.valor\_nota))/6) AS nota\_calculada\_media  FROM aluno a INNER JOIN nota n ON a.id = n.aluno\_id  INNER JOIN prova p ON n.prova\_id = p.id  GROUP BY a.nome  ORDER BY nota\_calculada\_media DESC; |

**Listagem 10**. Nota calculada média

**Procedimento para arredondar nota de alunos**

Vimos acima como usar uma função no MySQL e agora veremos como usar um procedimento. Iremos criar um procedimento que irá “varrer” todas as notas dos alunos e depois arredondar para mais ou para menos conforme necessidade.

Nosso critério de arredondamento será o seguinte: caso a diferença entre a nota e o próximo inteiro seja menor ou igual a 0.2, então arredondamos para este. Caso contrário, a nota continuará intacta. Observe o código da **Listagem 11**.

|  |
| --- |
| DELIMITER $  CREATE PROCEDURE arredondamento\_nota()  BEGIN  #O DECLARE serve para declarar uma variável que será utilizada durante  o programa  DECLARE nota\_atual NUMERIC(15,2);  DECLARE id\_aluno, id\_prova  INT;    #Criamos um CURSOR que irá “guardar” o resultado do SELECT  DECLARE cur CURSOR FOR SELECT valor\_nota, aluno\_id, prova\_id FROM nota;    #Abrimos o CURSOR para utilizá-lo dentro do LOOP  OPEN cur;    #Inicamos o LOOP dando um nome ao mesmo para que este possa ser  referenciado caso #necessário  myloop:  LOOP    #Atribuímos o valor das colunas do cursor as variáveis que    criamos anteriormente    FETCH cur INTO nota\_atual, id\_aluno, id\_prova;      #Lógica principal da nossa rotina.    #Caso o próximo inteiro tenha um diferença de 0.2 ou menos    da nota atual    # então a nota atual será arredondada para este inteiro.    IF (CEIL(nota\_atual) - nota\_atual) <= 0.2 THEN      UPDATE nota SET valor\_nota = CEIL(nota\_atual)      WHERE aluno\_id = id\_aluno AND prova\_id = id\_prova;    END IF;    END LOOP;    #Fechamos o cursor  CLOSE cur;  END |

**Listagem 11**. arredonda\_nota()

No script acima realizamos toda a tarefa necessária sem necessidade de nenhum retorno, por isso, optamos por usar um Procedimento em vez de uma função. Para utilizá-lo, basta executar da mesma forma que a apresentada na **Listagem 12**.

|  |
| --- |
| CALL arredondamento\_nota(); |

**Listagem 12**. Executando o procedimento arredonda\_nota()

Você também pode excluir tanto o procedimento como a função usando os seguintes comandos da **Listagem 13**.

|  |
| --- |
| #EXCLUINDO FUNÇÃO calcula\_nota  DROP FUNCTION calcula\_nota;    #EXCLUINDO PROCEDIMENTO arredonda\_nota  DROP PROCEDURE arredonda\_nota; |

**Listagem 13**. Excluindo procedimentos e funções

Finalizando nosso artigo na **Listagem 14** temos a lista do script completo que pode ser executado a fim de criar toda estrutura, população, funções e procedimentos mostrados durante todo o artigo.

|  |
| --- |
| #Criação da Base  CREATE DATABASE escola\_abc CHARACTER SET = 'utf8' COLLATE  = 'utf8\_general\_ci'    #Criação das tabelas  -- -----------------------------------------------------  -- Table aluno  -- -----------------------------------------------------  CREATE  TABLE IF NOT EXISTS aluno (     id  INT NOT NULL ,     nome  VARCHAR(100) NOT NULL ,     matricula  VARCHAR(45) NOT NULL ,     data\_nascimento  DATE NULL ,     data\_matricula  DATE NOT NULL ,    PRIMARY KEY ( id ) ,    UNIQUE INDEX  matricula\_UNIQUE  ( matricula  ASC) )  ENGINE = InnoDB;      -- -----------------------------------------------------  -- Table  prova  -- -----------------------------------------------------  CREATE  TABLE IF NOT EXISTS prova (     id  INT NOT NULL ,     data\_realizacao  DATE NOT NULL ,     descricao  VARCHAR(255) NOT NULL ,    PRIMARY KEY ( id ) )  ENGINE = InnoDB;      -- -----------------------------------------------------  -- Table nota  -- -----------------------------------------------------  CREATE  TABLE IF NOT EXISTS nota (     aluno\_id  INT NOT NULL ,     prova\_id  INT NOT NULL ,     valor\_nota  DECIMAL(15,2) NULL ,    PRIMARY KEY ( aluno\_id ,  prova\_id ) ,    INDEX  fk\_aluno\_has\_prova\_prova1  ( prova\_id  ASC) ,    INDEX  fk\_aluno\_has\_prova\_aluno1  ( aluno\_id  ASC) ,    CONSTRAINT  fk\_aluno\_has\_prova\_aluno1      FOREIGN KEY ( aluno\_id  )      REFERENCES aluno ( id  )      ON DELETE NO ACTION      ON UPDATE NO ACTION,    CONSTRAINT  fk\_aluno\_has\_prova\_prova1      FOREIGN KEY ( prova\_id  )      REFERENCES prova  ( id  )      ON DELETE NO ACTION      ON UPDATE NO ACTION)  ENGINE = InnoDB;    #População das tabelas  -- INSERINDO ALUNOS  INSERT INTO aluno (id, nome, matricula, data\_nascimento,  data\_matricula) VALUES  (1,'MICHAEL JONH', '123A', STR\_TO\_DATE('23/08/1993',  '%d/%m/%Y'), CURRENT\_DATE()),  (2,'WILLIANS JUNIOR', '400B', STR\_TO\_DATE('10/04/1993',  '%d/%m/%Y'), CURRENT\_DATE()),  (3,'JOHN BILLBOARD', '420B', STR\_TO\_DATE('30/07/1993',  '%d/%m/%Y'), CURRENT\_DATE()),  (4,'JENNY KILLY', '010A', NULL, STR\_TO\_DATE('25/01/2014',  '%d/%m/%Y'))    -- INSERINDO PROVAS  INSERT INTO prova(id, data\_realizacao, descricao) VALUES  (1, STR\_TO\_DATE('30/03/2014', '%d/%m/%Y'), 'Prova A1'),  (2, STR\_TO\_DATE('30/04/2014', '%d/%m/%Y'), 'Prova B1'),  (3, STR\_TO\_DATE('30/05/2014', '%d/%m/%Y'), 'Prova C1'),  (4, STR\_TO\_DATE('30/07/2014', '%d/%m/%Y'), 'Prova A2'),  (5, STR\_TO\_DATE('30/08/2014', '%d/%m/%Y'), 'Prova B2'),  (6, STR\_TO\_DATE('30/09/2014', '%d/%m/%Y'), 'Prova C2')    INSERT INTO nota (aluno\_id, prova\_id, valor\_nota) VALUES  #ALUNO 1  (1, 1, 10),  (1, 2, 9.8),  (1, 3, 8),  (1, 4, 10),  (1, 5, 10),  (1, 6, 9),    #ALUNO 2  (2, 1, 7),  (2, 2, 7.5),  (2, 3, 6),  (2, 4, 8),  (2, 5, 8.5),  (2, 6, 9),    #ALUNO 3  (3, 1, 9),  (3, 2, 9),  (3, 3, 9),  (3, 4, 10),  (3, 5, 10),  (3, 6, 9.8),    #ALUNO 4  (4, 1, 3),  (4, 2, 6),  (4, 3, 7),  (4, 4, 8),  (4, 5, 7),  (4, 6, 7)    #Criação da função calcula\_nota()  DELIMITER $  CREATE FUNCTION calc\_nota(nota NUMERIC(15,2)) RETURNS NUMERIC(15,2)  BEGIN   DECLARE peso INT;     #Se a nota do aluno for maior que 9.5 então sua nota terá um peso maior   IF nota > 9.5 THEN    SET peso = 2;   ELSE    SET peso = 1;   END IF;     RETURN (nota\*peso) / 20;  END  $    #Criação do procedimento arredonda\_nota  DELIMITER $  CREATE PROCEDURE arredondamento\_nota()  BEGIN   #O DECLARE serve para declarar uma variável que será utilizada   durante o programa   DECLARE nota\_atual NUMERIC(15,2);   DECLARE id\_aluno, id\_prova  INT;     #Criamos um CURSOR que irá “guardar” o resultado do SELECT   DECLARE cur CURSOR FOR SELECT valor\_nota, aluno\_id,   prova\_id FROM nota;     #Abrimos o CURSOR para utilizá-lo dentro do LOOP   OPEN cur;    #Inicamos o LOOP dando um nome ao mesmo para que este possa ser  referenciado caso #necessário   myloop:   LOOP      #Atribuímos o valor das colunas do cursor as variáveis que      criamos anteriormente      FETCH cur INTO nota\_atual, id\_aluno, id\_prova;        #Lógica principal da nossa rotina.      #Caso o próximo inteiro tenha um diferença de 0.2 ou menos da      nota atual      # então a nota atual será arredondada para este inteiro.      IF (CEIL(nota\_atual) - nota\_atual) <= 0.2 THEN        UPDATE nota SET valor\_nota = CEIL(nota\_atual) WHERE aluno\_id =        id\_aluno AND prova\_id = id\_prova;      END IF;     END LOOP;      #Fechamos o cursor    CLOSE cur;  END |

**Listagem 14**. Script final

O objetivo deste artigo não foi apenas apresentar teorias e conceitos que podem ser encontrados no próprio manual do MySQL a respeito de funções e procedimentos. Mostramos como aplicar estes na prática com um cenário real, criando toda a estrutura do banco até a população do mesmo.

Usamos como ferramenta o PhpMyAdmin apenas para dar uma melhor noção para aqueles que estão começando agora, mas nada impede que sejam utilizadas outras ferramentas como, por exemplo, o [**MySQL Workbench**](https://www.devmedia.com.br/introducao-ao-novo-mysql-workbench/25939), que é uma poderosa ferramenta para realizar até a [**modelagem do banco de dados**](https://www.devmedia.com.br/mysql-workbench-modelagem-de-dados-passo-a-passo/30021) e fazer engenharia reserva do modelo para o script.