**Banco de Dados – Procedures SQL - Executando código no MySQL– Micro-certificação - ALURA**

# Aula 01 – Preparando ambiente

Chegou a hora de você seguir todos os passos realizados por mim durante esta aula. Caso já tenha feito, excelente. Se ainda não, é importante que você execute o que foi visto nos vídeos para poder continuar com a próxima aula.

1) Se você está usando uma máquina limpa, você deve instalar o MYSQL. Por isso siga as instruções abaixo.

* Acesse o seu Browser e busque por MySQL Downloads. Acesse o link <http://www.mysql.com/downloads>
* Procure pelo link MySQL Community Edition (GPL) / Community (GPL) downloads.
* Vá para MySQL on Windows (Installer & Tools) / Downloads.
* Clique em MySQL Installer.
* Clique no botão de download ao lado da opção Windows (x86-32 Bits), MSI Installer (mysql-installer-web-community-8.0.15.0.msi).
* Faça um login no site da Oracle. Se você não possuir um login faça um cadastro.
* Após o Login clique em Download Now.
* Execute o programa que foi baixado.
* Clique em I Accept the license terms e depois em Next.
* Escolha a instalação Developer Default. Clique em Next duas vezes.
* Clique em Execute para o download e instalação do banco e seus componentes.
* Clique em Next duas vezes.
* Mantenha a escolha StandAlone MySQL Server / Classic MySQL Replication.
* Mantenha as propriedades padrões do serviço e da porta de comunicação. Clique Next.
* Mantenha a opção Use Strong Encryption for Authentication .... Clique em Next.
* Inclua a senha do usuário root duas vezes. Clique em Next.
* Mantenha as proprieades padrões. Clique em Next.
* Clique Execute para iniciar a instalação.
* Sempre selecione Next e Finish na medida que outras caixas de diálogo forem sendo exibidas. Se houver a pergunta sobre a senha do usuário root digite a senha configurada anteriormente na instalação.
* Automaticamente o Workbench será aberto. Clique na conexão que está configurada. Você então acessará o ambiente com o MySQL no ar.

Instalação no Ubuntu

* Comando para verificar se o Mysql está instalado
* dpkg -l mysql-server

mysql -V

* Comando para remover o Mysql
* sudo apt-get remove --purge mysql\*
* sudo apt-get purge mysql\*
* sudo apt-get autoremove

sudo apt-get autoclean

* Comando para instalar o Mysql
* sudo apt-get install mysql-server

sudo apt-get install mysql-workbench

* Configurando o Mysql ` sudo mysql\_secure\_installation Se você quiser fazer login como root através de programas externos: sudo mysql -u root ALTER USER 'root'@'localhost' IDENTIFIED WITH

2) Se você está usando uma máquina limpa, você deve recuperar a base de dados a ser usada neste curso. Por isso siga as instruções abaixo:

* Abra o MYSQL Workbench. Use a conexão LOCALHOST.
* Botão da direita do mouse sobre a lista das bases e escolha Create Schema.
* Insira o nome Sucos\_Vendas. Clique em Apply duas vezes.
* Faça Download do arquivo RecuperacaoAmbiente.zip.
* Descompacte o arquivo.
* Selecione, na área Navigator, em Administration.
* Selecione o link Data Import/Restore.
* Na opção Import From Dump Project Folder escolha o diretório DumpSucosVendas.
* Selecione Start Import.
* Verifique na base Sucos\_Vendas se as tabelas foram criadas.

# O que aprendemos?

Nesta aula, aprendemos:

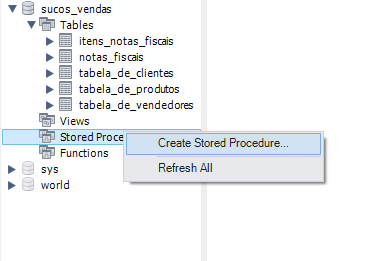
* A instalar o MySQL
* A recuperar da base de dados a ser usada neste treinamento

### Aula 02 – Stored Procedures Básico

Chegou a hora de você seguir todos os passos realizados por mim durante esta aula. Caso já tenha feito, excelente. Se ainda não, é importante que você execute o que foi visto nos vídeos para poder continuar com a próxima aula.

1) Abra o MYSQL Workbench e crie um novo script de comandos sql.

2) Vamos iniciar criando algumas Stored Procedures. Clique com o botão da direita do mouse sobre Stored Procedure, abaixo do banco sucos\_vendas.



3) Digite, após o comando CREATE PROCEDURE, entre as aspas, o nome da Stored Procedure (alo\_mundo).

4) Entre o BEGIN END digite:

select 'Alô Mundo !!!!';

5) Clique em Apply. Iremos ver o comando abaixo:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `alo\_mundo`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `alo\_mundo` ()

BEGIN

select 'Alô Mundo !!!!';

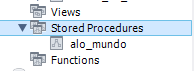
END$$

DELIMITER ;

6) Os comandos acima representam os comandos MYSQL para a criação da Stored Procedure. Se executarmos estes comandos direto no script MYSQL a SP (Stored Procedure) será criada.

7) Clique em Apply e depois Finish.

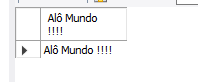
8) Podemos ver, abaixo do nó Stored Procedure, a mesma criada.



9) No Script digite e execute:

Call alo\_mundo;

10) Temos o resultado:



11) Digite, no script SQL:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `mostra\_numero`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `mostra\_numero` ()

BEGIN

select (1 + 9) - 5;

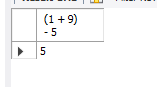
END$$

DELIMITER ;

Observação: Diferente um pouco dos vídeos onde as SPs foram criadas pelo assistente, aqui, no passo a passo, iremos apresentar o código MYSQL para cria a SP.

12) No Script digite e execute:

Call mostra\_numero;



13) Vimos, aqui, uma SP que retorna uma expressão numérica.

14) Digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `mostra\_numero\_alias`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `mostra\_numero\_alias` ()

BEGIN

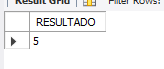
select (1 + 9) - 5 as RESULTADO;

END$$

DELIMITER ;

15) No Script digite e execute:

Call mostra\_numero\_alias;



Você pode ver que o resultado da SP trouxe o nome da coluna como um Alias.

16) Digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `sp\_com\_funcoes`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `sp\_com\_funcoes` ()

BEGIN

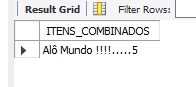
SELECT CONCAT('Alô Mundo !!!!', '.....', (1 + 9) - 5) as ITENS\_COMBINADOS;

END$$

DELIMITER ;

17) No Script digite e execute:

Call sp\_com\_funcoes;



Vimos como podemos usar funções MYSQL dentro das SPs.

18) Digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `sp\_com\_funcoes`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `sp\_com\_funcoes` ()

BEGIN

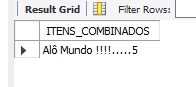
SELECT CONCAT('Alô Mundo !!!!', '.....', (1 + 9) - 5) as ITENS\_COMBINADOS;

END$$

DELIMITER ;

19) No Script digite e execute:

Call sp\_com\_funcoes;



Vimos como podemos usar funções MYSQL dentro das SPs.

20) Digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `sp\_com\_comentarios`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `sp\_com\_comentarios` ()

BEGIN

/\* Vamos exibir itens combinados

entre textos e números \*/

-- Usando a função CONCAT

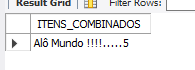
SELECT CONCAT('Alô Mundo !!!!', '.....', (1 + 9) - 5) as ITENS\_COMBINADOS;

END$$

DELIMITER ;

21) No Script digite e execute:

Call sp\_com\_comentarios;



No exemplo acima vimos como usamos comentários dentro do código da SP.

22) Digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `alo\_mundo\_2`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `alo\_mundo\_2` ()

BEGIN

SELECT 'Alô Mundo !!!!, tudo bem?' as RESULTADO;

END$$

DELIMITER ;

23) Vamos alterar uma SP existente. Para isso digite o comando abaixo:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `alo\_mundo\_3`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `alo\_mundo\_3`()

BEGIN

SELECT 'Alô Mundo !!!!, tudo bem? Versão 3' as RESULTADO;

END$$

DELIMITER ;

Note que o comando apaga a SP através de um DROP e a cria novamente com o novo código. É desta maneira que podemos alterar um SP já existente.

24) Como já mencionado no comando de alteração de uma SP, se quer apenas apagá-la, digite.

USE `sucos\_vendas`;

DROP procedure IF EXISTS `alo\_mundo\_2`;

25) Podemos usar variáveis para atribuir valores e usar as mesmas nos comandos da SP. Digite e execute o comando abaixo:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `exibe\_variavel`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `exibe\_variavel` ()

BEGIN

declare texto char(20) default 'Alô Mundo !!!!';

SELECT texto;

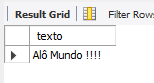
END$$

DELIMITER ;

Nesta SP declaramos uma variável (texto), atribuímos um valor inicial padrão e exibimos na saída.

26) Digite e execute:

call exibe\_variavel;



# O que aprendemos?

Nesta aula, aprendemos:

* A criar uma Stored Procedure (SP) que retorna um texto;
* Como a SP retorna um valor em sua saída;
* A usar funções do MYSQL e comentários na SP;
* Como alterar uma SP já existente;
* Como excluir uma SP.

### Aula 03 – Interações com o banco de dados

Chegou a hora de você seguir todos os passos realizados por mim durante esta aula. Caso já tenha feito, excelente. Se ainda não, é importante que você execute o que foi visto nos vídeos para poder continuar com a próxima aula.

1) Abra um novo script MYSQL.

2) Digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `manipulacao\_dados`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `manipulacao\_dados` ()

BEGIN

INSERT INTO tabela\_de\_produtos (CODIGO\_DO\_PRODUTO,NOME\_DO\_PRODUTO,SABOR,TAMANHO,EMBALAGEM,PRECO\_DE\_LISTA)

VALUES ('2001001','Sabor da Serra 700 ml - Manga','Manga','700 ml','Garrafa',7.50),

('2001000','Sabor da Serra 700 ml - Melão','Melão','700 ml','Garrafa',7.50),

('2001002','Sabor da Serra 700 ml - Graviola','Graviola','700 ml','Garrafa',7.50),

('2001003','Sabor da Serra 700 ml - Tangerina','Tangerina','700 ml','Garrafa',7.50),

('2001004','Sabor da Serra 700 ml - Abacate','Abacate','700 ml','Garrafa',7.50),

('2001005','Sabor da Serra 700 ml - Açai','Açai','700 ml','Garrafa',7.50),

('2001006','Sabor da Serra 1 Litro - Manga','Manga','1 Litro','Garrafa',7.50),

('2001007','Sabor da Serra 1 Litro - Melão','Melão','1 Litro','Garrafa',7.50),

('2001008','Sabor da Serra 1 Litro - Graviola','Graviola','1 Litro','Garrafa',7.50),

('2001009','Sabor da Serra 1 Litro - Tangerina','Tangerina','1 Litro','Garrafa',7.50),

('2001010','Sabor da Serra 1 Litro - Abacate','Abacate','1 Litro','Garrafa',7.50),

('2001011','Sabor da Serra 1 Litro - Açai','Açai','1 Litro','Garrafa',7.50);

SELECT \* FROM tabela\_de\_produtos WHERE NOME\_DO\_PRODUTO LIKE 'Sabor da Serra%';

UPDATE tabela\_de\_produtos SET PRECO\_DE\_LISTA = 8 WHERE NOME\_DO\_PRODUTO LIKE 'Sabor da Serra%';

SELECT \* FROM tabela\_de\_produtos WHERE NOME\_DO\_PRODUTO LIKE 'Sabor da Serra%';

DELETE FROM tabela\_de\_produtos WHERE NOME\_DO\_PRODUTO LIKE 'Sabor da Serra%';

SELECT \* FROM tabela\_de\_produtos WHERE NOME\_DO\_PRODUTO LIKE 'Sabor da Serra%';

END$$

DELIMITER ;

3) Execute:

Call manipulacao\_dados;

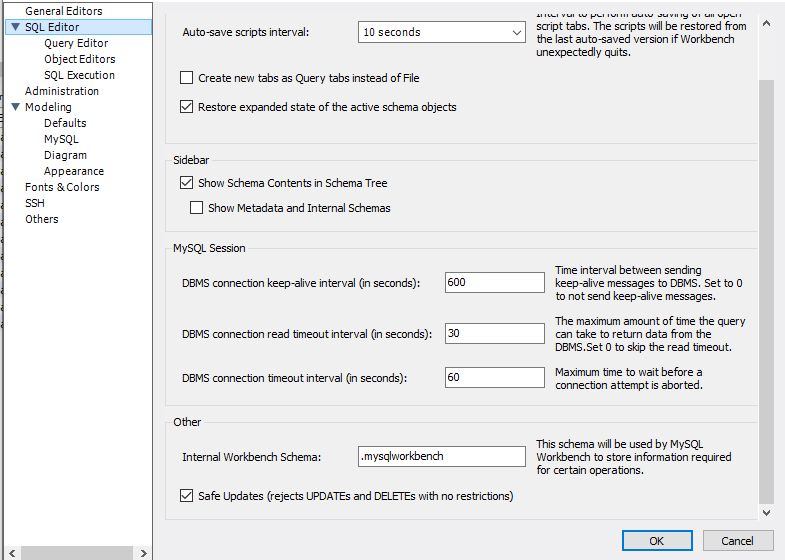
4) A SP acima manipula uma série de comandos no banco de dados MYSQL.

Observação: Se você observar o seguinte erro:

0.png

É porque você deve efetuar uma modificação na configuração do seu MYSQL.

Vá, no menu, em Edit / Preference e na seção SQL EDITOR vá até o final do formulário e desmarque a opção Safe Updates (Reject UPDATEs and DELETEs with no restrictions).



5) Podemos também usar a SP para inserir dados na tabela usando variáveis. Digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `inclui\_novo\_produto`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `inclui\_novo\_produto` ()

BEGIN

DECLARE vCodigo varchar(50) DEFAULT '3000001';

DECLARE vNome varchar(50) DEFAULT 'Sabor do Mar 700 ml - Manga';

DECLARE vSabor varchar(50) DEFAULT 'Manga';

DECLARE vTamanho varchar(50) DEFAULT '700 ml';

DECLARE vEmbalagem varchar(50) DEFAULT 'Garrafa';

DECLARE vPreco DECIMAL(10,2) DEFAULT 9.25;

INSERT INTO tabela\_de\_produtos

(CODIGO\_DO\_PRODUTO,NOME\_DO\_PRODUTO,SABOR,TAMANHO,EMBALAGEM,PRECO\_DE\_LISTA)

VALUES (vCodigo,

vNome,

vSabor,

vTamanho,

vEmbalagem,

vPreco);

END$$

DELIMITER ;

6) Execute:

Call inclui\_novo\_produto;

7) Veja que usamos referência direta as variáveis no comando INSERT, dentro da SP.

8) Podemos usar parâmetros para passar para a SP os dados que serão usados no comando de INSERT. Para isso digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `inclui\_novo\_produto\_parametro`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `inclui\_novo\_produto\_parametro`(vCodigo varchar(50),

vNome varchar(50), vSabor varchar(50), vTamanho varchar(50),

vEmbalagem varchar(50), vPreco DECIMAL(10,2))

BEGIN

INSERT INTO tabela\_de\_produtos

(CODIGO\_DO\_PRODUTO,NOME\_DO\_PRODUTO,SABOR,TAMANHO,EMBALAGEM,PRECO\_DE\_LISTA)

VALUES (vCodigo,

vNome,

vSabor,

vTamanho,

vEmbalagem,

vPreco);

END$$

DELIMITER ;

9) Execute a SP:

Call inclui\_novo\_produto\_parametro('4000001', 'Sabor do Pantanal 1 Litro - Melancia',

'Melancia', '1 Litro', 'PET', 4.76);

10) Veja que o comando acima passa os dados que serão incluídos dentro da tabela através dos parâmetros.

11) Repita a execução do comando novamente:

Call inclui\_novo\_produto\_parametro('4000004', 'Sabor do Pantanal 1 Litro - Melancia',

'Melancia', '1 Litro', 'PET', 4.76);

12) Note que houve um erro de chave primária porque o produto 4000004 já existe na tabela:

2.png

13) Podemos tratar este erro para que ele apresente uma mensagem amigável quando ocorrer. Execute e digite:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `inclui\_novo\_produto\_parametro\_2`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `inclui\_novo\_produto\_parametro\_2`(vCodigo varchar(50),

vNome varchar(50), vSabor varchar(50), vTamanho varchar(50),

vEmbalagem varchar(50), vPreco DECIMAL(10,2))

BEGIN

DECLARE mensagem VARCHAR(40);

DECLARE EXIT HANDLER FOR 1062

BEGIN

SET mensagem = 'Problema de Chave Primária !!!';

SELECT mensagem;

END;

INSERT INTO tabela\_de\_produtos

(CODIGO\_DO\_PRODUTO,NOME\_DO\_PRODUTO,SABOR,TAMANHO,EMBALAGEM,PRECO\_DE\_LISTA)

VALUES (vCodigo,

vNome,

vSabor,

vTamanho,

vEmbalagem,

vPreco);

SET mensagem = 'Produto incluido com sucesso !!!';

SELECT mensagem;

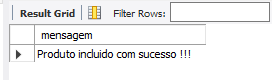
END$$

DELIMITER ;

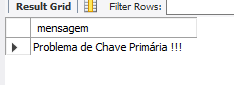
14) Execute e digite:

Call inclui\_novo\_produto\_parametro\_2('4000004', 'Sabor do Pantanal 1 Litro - Melancia',

'Melancia', '1 Litro', 'PET', 4.76);



15) Repita o comando. Teremos:



16) Vimos que o comando SET é usado para atribuir valores as variáveis. Más podemos atribuir valores a ela usando comandos SELECT através do SELECT INTO. Execute e digite:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `acha\_sabor\_produto`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `acha\_sabor\_produto`(vProduto VARCHAR(50))

BEGIN

DECLARE vSabor VARCHAR(50);

SELECT SABOR INTO vSabor FROM tabela\_de\_produtos WHERE codigo\_do\_produto = vProduto;

SELECT vSabor;

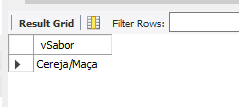
END$$

DELIMITER ;

17) Execute:

Call acha\_sabor\_produto('1013793');

18) Teremos como resposta o sabor do produto passado como parâmetro.



# O que aprendemos?

Nesta aula, aprendemos:

* Como manipular comandos SQL dentro da SP;
* Como passar parâmetros para uma SP;
* Como tratamos erros;
* A atribuição de variáveis através de um comando SELECT.

### Aula 04 – Controle de Fluxo

Chegou a hora de você seguir todos os passos realizados por mim durante esta aula. Caso já tenha feito, excelente. Se ainda não, é importante que você execute o que foi visto nos vídeos para poder continuar com a próxima aula.

1) Abra um novo script MYSQL.

2) Digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `cliente\_novo\_velho`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `cliente\_novo\_velho`(vCPF VARCHAR(20))

BEGIN

DECLARE vResultado VARCHAR(20);

DECLARE vDataNascimento DATE;

SELECT DATA\_DE\_NASCIMENTO INTO vDataNascimento FROM

tabela\_de\_clientes WHERE CPF = vCPF;

IF vDataNascimento < '20000101' THEN

SET vResultado = 'CLIENTE VELHO';

ELSE

SET vResultado = 'CLIENTE NOVO';

END IF;

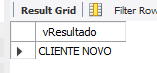
SELECT vResultado;

END$$

DELIMITER ;

2) Execute a SP.

Call cliente\_novo\_velho ('19290992743');



3) Esta SP usa a estrutura IF-THEN-ELSE para classificar um cliente como sendo novo ou velho, baseado em sua idade.

4) Vamos ver outra estrutura de controle de fluxo. Digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `acha\_status\_preco\_2`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `acha\_status\_preco\_2` (vProduto VARCHAR(50))

BEGIN

DECLARE vPreco FLOAT;

DECLARE vMensagem VARCHAR(30);

SELECT PRECO\_DE\_LISTA INTO vPreco FROM tabela\_de\_produtos

WHERE codigo\_do\_produto = vProduto;

IF vPreco >= 12 THEN

SET vMensagem = 'PRODUTO CARO.';

ELSEIF vPreco >= 7 AND vPreco < 12 THEN

SET vMensagem = 'PRODUTO EM CONTA.';

ELSE

SET vMensagem = 'PRODUTO BARATO.';

END IF;

SELECT vMensagem;

END$$

DELIMITER ;

5) Digite e execute:

Call acha\_status\_preco\_2 ('1000889');

6) Observe que a estrutura IF-THEN-ELSEIF permite encadear diversos testes.

7) O encadeamento de condições pode ser expresso, também, usando o comando CASE-END-CASE. Para isso digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `acha\_tipo\_sabor`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `acha\_tipo\_sabor`(vProduto VARCHAR(50))

BEGIN

DECLARE vSabor VARCHAR(50);

SELECT SABOR INTO vSabor FROM tabela\_de\_Produtos

WHERE codigo\_do\_produto = vProduto;

CASE vSabor

WHEN 'Lima/Limão' THEN SELECT 'Cítrico';

WHEN 'Laranja' THEN SELECT 'Cítrico';

WHEN 'Morango/Limão' THEN SELECT 'Cítrico';

WHEN 'Uva' THEN SELECT 'Neutro';

WHEN 'Morango' THEN SELECT 'Neutro';

ELSE SELECT 'Ácidos';

END CASE;

END$$

DELIMITER ;

8) Digite e execute:

CALL acha\_tipo\_sabor('1000889');

9) Uma estrutura derivada da mostrada é o CASE-NOT-FOUND. Para isso vamos criar a SP Acha\_Tipo\_Sabor\_Erro. Digite e execute:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `acha\_tipo\_sabor\_erro`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `acha\_tipo\_sabor\_erro`(vProduto VARCHAR(50))

BEGIN

DECLARE vSabor VARCHAR(50);

SELECT SABOR INTO vSabor FROM tabela\_de\_Produtos

WHERE codigo\_do\_produto = vProduto;

CASE vSabor

WHEN 'Lima/Limão' THEN SELECT 'Cítrico';

WHEN 'Laranja' THEN SELECT 'Cítrico';

WHEN 'Morango/Limão' THEN SELECT 'Cítrico';

WHEN 'Uva' THEN SELECT 'Neutro';

WHEN 'Morango' THEN SELECT 'Neutro';

END CASE;

END$$

DELIMITER ;

10) Esta SP difere da anterior apenas porque foi retirada a linha abaixo.

ELSE SELECT 'Ácidos';

Estamos criando uma situação onde nenhuma das condições podem ser satisfeitas.

11) Execute a SP:

CALL acha\_tipo\_sabor\_erro('1004327');

12) Temos o erro:

1.png

13) Para evitar de não incluirmos opções no CASE que contemplem todas as situações podemos acrescentar o tratamento de erro. Altere a SP executando:

USE `sucos\_vendas`;

DROP procedure IF EXISTS `acha\_tipo\_sabor\_erro`;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE `acha\_tipo\_sabor\_erro`(vProduto VARCHAR(50))

BEGIN

DECLARE vSabor VARCHAR(50);

DECLARE msgErro VARCHAR(30);

DECLARE CONTINUE HANDLER FOR 1339 SET msgErro = 'O case não está completo';

SELECT SABOR INTO vSabor FROM tabela\_de\_Produtos

WHERE codigo\_do\_produto = vProduto;

CASE vSabor

WHEN 'Lima/Limão' THEN SELECT 'Cítrico';

WHEN 'Laranja' THEN SELECT 'Cítrico';

WHEN 'Morango/Limão' THEN SELECT 'Cítrico';

WHEN 'Uva' THEN SELECT 'Neutro';

WHEN 'Morango' THEN SELECT 'Neutro';

END CASE;

SELECT msgErro;

END$$

DELIMITER ;

14) Execute a SP:

CALL acha\_tipo\_sabor\_erro('1004327');

Teremos o erro tratado.

![2.png](https://caelum-online-public.s3.amazonaws.com/1223-mysqlproceduresefuncions/04/2.png)

15) O Case Condicional utiliza uma estrutura de Case semelhante a usada quando executamos um comando SELECT. Digite e execute:

USE sucos\_vendas; DROP procedure IF EXISTS acha\_status\_preco\_case; DELIMITER $$

USE sucos\_vendas$$ CREATE PROCEDURE acha\_status\_preco\_case(vProduto VARCHAR(50)) BEGIN DECLARE vPreco FLOAT; DECLARE vMensagem VARCHAR(30); SELECT PRECO\_DE\_LISTA INTO vPreco FROM tabela\_de\_produtos WHERE codigo\_do\_produto = vProduto; CASE WHEN vPreco >= 12 THEN SET vMensagem = 'PRODUTO CARO.'; WHEN vPreco >= 7 AND vPreco < 12 THEN SET vMensagem = 'PRODUTO EM CONTA.'; WHEN vPreco < 7 THEN SET vMensagem = 'PRODUTO BARATO'; END CASE; SELECT vMensagem; END$$ DELIMITER ;

16) Digite e execute:

CALL acha\_status\_preco\_case('1004327');

17) A estrutura de Loop permite repetir um conjunto de comandos até que determinada condição aconteça. Digite e execute:

CREATE TABLE TAB\_LOOPING (ID INT);

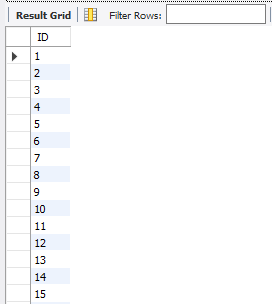
18) Após a criação de uma tabela auxiliar digite e execute

USE sucos\_vendas; DROP procedure IF EXISTS looping\_while; DELIMITER $$

USE sucos\_vendas$$ CREATE PROCEDURE looping\_while(vNumInicial INT, vNumFinal INT) BEGIN DECLARE vContador INT; DELETE FROM TAB\_LOOPING; SET vContador = vNumInicial; WHILE vContador <= vNumFinal DO INSERT INTO TAB\_LOOPING (ID) VALUES (vContador); SET vContador = vContador + 1; END WHILE; SELECT \* FROM TAB\_LOOPING; END$$ DELIMITER ;

19) Vamos executar a SP passando como parâmetro um número inicial e um número final para criação de uma sequência numérica na tabela temporária. Digite e execute:

call looping\_while (1, 1000); `



Teremos uma sequência numérica que vai de 1 a 1000.

# O que aprendemos?

Nesta aula, aprendemos:

* O control de fluxo IF-THEN-ELSE.
* Uma variação do controle anterior chamado IF-THEN-ELSEIF.
* A estrutura CASE;
* Como tratar erros no CASE quando nem todas as opções são contempladas;
* o CASE condicional, semelhante ao usado nos comandos SQL;
* O uso de Loops para repetir um conjunto de comandos até uma condição ser satisfeita.

### Aula 05 – Cursor e Função

Chegou a hora de você seguir todos os passos realizados por mim durante esta aula. Caso já tenha feito, excelente. Se ainda não, é importante que você execute o que foi visto nos vídeos para poder continuar com a próxima aula.

1) Quando o resultado de um SELECT possui mais de uma linha não podemos atribuí-lo a uma variável usando o SELECT INTO. Para isso temos que usar o Cursor para receber valores vindos de uma tabela, com uma ou mais colunas.

2) Vamos criar uma SP que utilize Cursor. Digite e execute:

USE sucos\_vendas;

DROP procedure IF EXISTS cursor\_primeiro\_contato;

DELIMITER $$

USE sucos\_vendas$$

CREATE PROCEDURE cursor\_primeiro\_contato()

BEGIN

DECLARE vNome VARCHAR(50);

DECLARE c CURSOR FOR SELECT NOME FROM tabela\_de\_clientes limit 4;

OPEN c

FETCH c INTO vNome;

SELECT vNome;

FETCH c INTO vNome;

SELECT vNome;

FETCH c INTO vNome;

SELECT vNome;

FETCH c INTO vNome;

SELECT vNome;

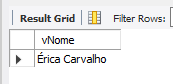
CLOSE c;

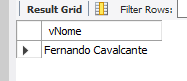
END$$

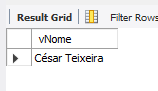
DELIMITER ;

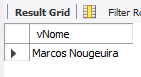
3) Note que cada linha da seleção é atribuída a variável vNome, uma de cada vez. Execute:

call cursor\_primeiro\_contato;









4) Na SP acima executamos um Cursos controlado onde sabíamos quantos elementos o mesmo tinha. Mas, normalmente, não sabemos esta informação. Por isso usamos sempre o Cursos combinados com um Looping. Digite e execute a criação da SP abaixo:

USE sucos\_vendas;

DROP procedure IF EXISTS cursor\_looping;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE PROCEDURE cursor\_looping()

BEGIN

DECLARE fim\_do\_cursor INT DEFAULT 0;

DECLARE vNome VARCHAR(50);

DECLARE c CURSOR FOR SELECT NOME FROM tabela\_de\_clientes;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET fim\_do\_cursor = 1;

OPEN c;

WHILE fim\_do\_cursor = 0

DO

FETCH c INTO vNome;

IF fim\_do\_cursor = 0 THEN

SELECT vNome;

END IF;

END WHILE;

CLOSE c;

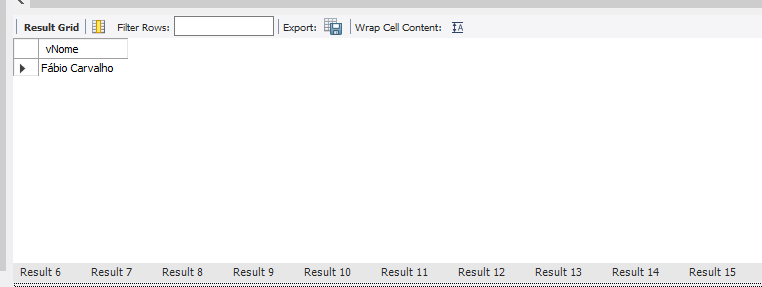
END$$

DELIMITER ;

5) Executando a SP:

call cursor\_looping;

Teremos diversos resultados em diferentes consultas.



6) O Cursor pode suportar mais de uma coluna. Crie a SP abaixo:

USE sucos\_vendas;

DROP procedure IF EXISTS looping\_cursor\_multiplas\_colunas;

DELIMITER $$

USE sucos\_vendas$$

CREATE PROCEDURE looping\_cursor\_multiplas\_colunas()

BEGIN

DECLARE fim\_do\_cursor INT DEFAULT 0;

DECLARE vCidade, vEstado, vCep VARCHAR(50);

DECLARE vNome, vEndereco VARCHAR(150);

DECLARE c CURSOR FOR

SELECT nome, endereco\_1, cidade, estado, cep FROM tabela\_de\_clientes;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET fim\_do\_cursor = 1;

OPEN c;

WHILE fim\_do\_cursor = 0

DO

FETCH c INTO vNome, vEndereco, vCidade, vEstado, vCep;

IF fim\_do\_cursor = 0 THEN

SELECT CONCAT(vNome, ' Endereço: ',

vEndereco, ', ', vCidade , ' - ', vEstado, ' CEP: ' , vCep);

END IF;

END WHILE;

CLOSE c;

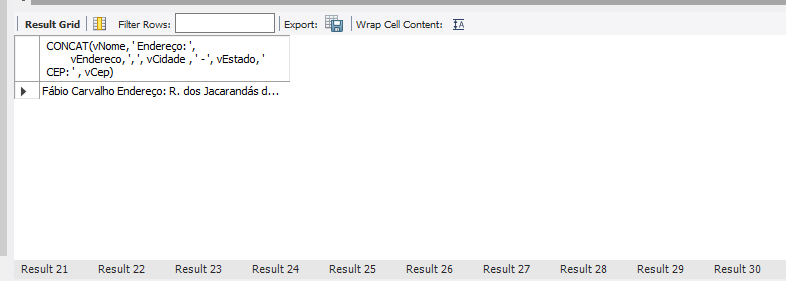
END$$

DELIMITER ;

7) Execute-a:

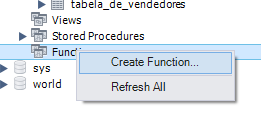
call looping\_cursor\_multiplas\_colunas;

Esta SP também retorna múltiplas consultas.



8) Também podemos criar uma função. A diferença de uma função e uma SP é que a função retorna um valor e pode ser usada dentro de um comando SELECT, INSERT, UPDATE e condições de DELETE.

9) Para criar uma função vá com o botão da direita do mouse sobre Function e selecione Create Function:



10) Acrescente o código abaixo:

CREATE FUNCTION f\_acha\_tipo\_sabor(vSabor VARCHAR(50)) RETURNS varchar(20) CHARSET utf8mb4

BEGIN

DECLARE vRetorno VARCHAR(20) default "";

CASE vSabor

WHEN 'Lima/Limão' THEN SET vRetorno = 'Cítrico';

WHEN 'Laranja' THEN SET vRetorno = 'Cítrico';

WHEN 'Morango/Limão' THEN SET vRetorno = 'Cítrico';

WHEN 'Uva' THEN SET vRetorno = 'Neutro';

WHEN 'Morango' THEN SET vRetorno = 'Neutro';

ELSE SET vRetorno = 'Ácidos';

END CASE;

RETURN vRetorno;

END

11) Clique em Apply. Teremos o código que poderá ser eecutado diretamente do script MYSQL.

USE sucos\_vendas;

DROP function IF EXISTS f\_acha\_tipo\_sabor;

DELIMITER $$

USE `sucos\_vendas`$$

CREATE FUNCTION f\_acha\_tipo\_sabor(vSabor VARCHAR(50)) RETURNS varchar(20) CHARSET utf8mb4

BEGIN

DECLARE vRetorno VARCHAR(20) default "";

CASE vSabor

WHEN 'Lima/Limão' THEN SET vRetorno = 'Cítrico';

WHEN 'Laranja' THEN SET vRetorno = 'Cítrico';

WHEN 'Morango/Limão' THEN SET vRetorno = 'Cítrico';

WHEN 'Uva' THEN SET vRetorno = 'Neutro';

WHEN 'Morango' THEN SET vRetorno = 'Neutro';

ELSE SET vRetorno = 'Ácidos';

END CASE;

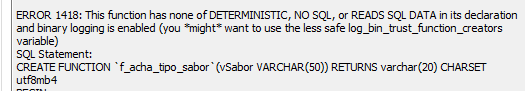
RETURN vRetorno;

END$$

DELIMITER ;

12) Clique em Apply. Teremos o código que poderá ser executado diretamente do script MYSQL.

Observação: Se você verificar um erro como mostrado abaixo:



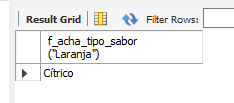
É porque o MYSQL não permite a construção de Funções. Para permitir, execute o comando:

SET GLOBAL log\_bin\_trust\_function\_creators = 1;

E crie a função novamente.

13) Execute a função:

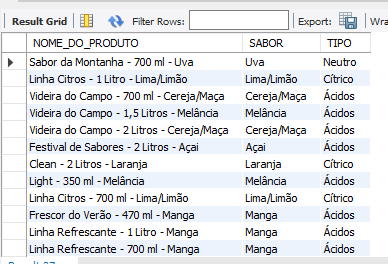
SELECT f\_acha\_tipo\_sabor ("Laranja");



14) Podemos usar a função num comando SELECT. Digite e execute:

SELECT NOME\_DO\_PRODUTO, SABOR, f\_acha\_tipo\_sabor (SABOR) as TIPO

FROM tabela\_de\_produtos;



# O que aprendemos?

Nesta aula, aprendemos:

* Conhecemos a estrutura de Cursor onde podemos atribuir valores resultados de um SELECT com múltiplas linhas.
* Vimos que podemos atribuir ao Cursos mais de uma coluna.
* Aprendemos como usar o Cursos em conjunto com um Looping.
* Foi mostrado como criamos e usamos uma função.

### Aula 06 – Problema prático

Chegou a hora de você seguir todos os passos realizados por mim durante esta aula. Caso já tenha feito, excelente. Se ainda não, é importante que você execute o que foi visto nos vídeos para poder continuar com os próximos cursos que tenham este como pré-requisito.

1) Vamos colocar nossos conhecimentos em prática. Iremos criar uma SP que irá criar uma venda aleatória.

2) Iniciamos com uma função que retorna um número aleatório entre dois valores. Para isso crie a função abaixo:

USE sucos\_vendas;

DROP function IF EXISTS f\_numero\_aleatorio;

DELIMITER $$

USE sucos\_vendas$$

CREATE FUNCTION f\_numero\_aleatorio(min INT, max INT) RETURNS int(11)

BEGIN

DECLARE vRetorno INT;

SELECT FLOOR((RAND() \* (max-min+1)) + min) INTO vRetorno;

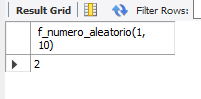
RETURN vRetorno;

END$$

DELIMITER ;

3) Teste a função várias vezes para verificar que os números aleatórios são criados:

SELECT f\_numero\_aleatorio(1, 10);



4) Usaremos a função abaixo para escolher um cliente de forma aleatória da tabela de clientes. Iremos escolher uma posição entre 1 e o número de registros da tabela e retornar o cliente escolhido conforme o número aleatório determinado. Digite e execute:

USE sucos\_vendas;

DROP function IF EXISTS f\_cliente\_aleatorio;

DELIMITER $$

USE sucos\_vendas$$

CREATE FUNCTION f\_cliente\_aleatorio() RETURNS varchar(11) CHARSET utf8mb4

BEGIN

DECLARE vRetorno VARCHAR(11);

DECLARE num\_max\_tabela INT;

DECLARE num\_aleatorio INT;

SELECT COUNT(\*) INTO num\_max\_tabela FROM tabela\_de\_clientes;

SET num\_aleatorio = f\_numero\_aleatorio(1, num\_max\_tabela);

SET num\_aleatorio = num\_aleatorio - 1;

SELECT CPF INTO vRetorno FROM tabela\_de\_clientes

LIMIT num\_aleatorio, 1;

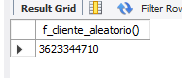
RETURN vRetorno;

END$$

DELIMITER ;

5) Teste a função para determinar um cliente aleatoriamente:

SELECT f\_cliente\_aleatorio();



6) Nos exercícios associados a esta aula criamos a função para buscar, de forma aleatória, um produto e um vendedor. Mas caso não tenha sido feito o exercício proceda criando as duas funções conforme o código abaixo:

DELIMITER $$

USE `sucos\_vendas`$$

CREATE DEFINER=`root`@`localhost` FUNCTION `f\_produto\_aleatorio`() RETURNS varchar(10) CHARSET utf8mb4

BEGIN

DECLARE vRetorno VARCHAR(10);

DECLARE num\_max\_tabela INT;

DECLARE num\_aleatorio INT;

SELECT COUNT(\*) INTO num\_max\_tabela FROM tabela\_de\_produtos;

SET num\_aleatorio = f\_numero\_aleatorio(1, num\_max\_tabela);

SET num\_aleatorio = num\_aleatorio - 1;

SELECT CODIGO\_DO\_PRODUTO INTO vRetorno FROM tabela\_de\_produtos

LIMIT num\_aleatorio, 1;

RETURN vRetorno;

END$$

DELIMITER ;

;

DELIMITER $$

USE sucos\_vendas$$

CREATE DEFINER=root@localhost FUNCTION f\_vendedor\_aleatorio() RETURNS varchar(5) CHARSET utf8mb4

BEGIN

DECLARE vRetorno VARCHAR(5);

DECLARE num\_max\_tabela INT;

DECLARE num\_aleatorio INT;

SELECT COUNT(\*) INTO num\_max\_tabela FROM tabela\_de\_vendedores;

SET num\_aleatorio = f\_numero\_aleatorio(1, num\_max\_tabela);

SET num\_aleatorio = num\_aleatorio - 1;

SELECT MATRICULA INTO vRetorno FROM tabela\_de\_vendedores

LIMIT num\_aleatorio, 1;

RETURN vRetorno;

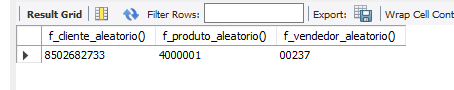
END$$

DELIMITER ;

;

7) Podemos testar as funções num único SELECT:

SELECT f\_cliente\_aleatorio(), f\_produto\_aleatorio(), f\_vendedor\_aleatorio();



8) Vamos, finalmente, criar a SP para incluir uma venda de forma aleatória. Digite e execute a criação da SP abaixo:

USE sucos\_vendas;

DROP procedure IF EXISTS p\_inserir\_venda;

DELIMITER $$

USE sucos\_vendas$$

CREATE PROCEDURE p\_inserir\_venda(vData DATE, max\_itens INT,

max\_quantidade INT)

BEGIN

DECLARE vCliente VARCHAR(11);

DECLARE vProduto VARCHAR(10);

DECLARE vVendedor VARCHAR(5);

DECLARE vQuantidade INT;

DECLARE vPreco FLOAT;

DECLARE vItens INT;

DECLARE vNumeroNota INT;

DECLARE vContador INT DEFAULT 1;

SELECT MAX(numero) + 1 INTO vNumeroNota from notas\_fiscais;

SET vCliente = f\_cliente\_aleatorio();

SET vVendedor = f\_vendedor\_aleatorio();

INSERT INTO notas\_fiscais (CPF, MATRICULA, DATA\_VENDA, NUMERO, IMPOSTO)

VALUES (vCliente, vVendedor, vData, vNumeroNota, 0.18);

SET vItens = f\_numero\_aleatorio(1, max\_itens);

WHILE vContador <= vItens

DO

SET vProduto = f\_produto\_aleatorio();

SET vQuantidade = f\_numero\_aleatorio(10, max\_quantidade);

SELECT PRECO\_DE\_LISTA INTO vPreco FROM tabela\_de\_produtos

WHERE CODIGO\_DO\_PRODUTO = vProduto;

INSERT INTO itens\_notas\_fiscais (NUMERO, CODIGO\_DO\_PRODUTO,

QUANTIDADE, PRECO) VALUES (vNumeroNota, vProduto, vQuantidade, vPreco);

SET vContador = vContador + 1;

END WHILE;

END$$

DELIMITER ;

9) Execute a SP para criar uma venda:

Call p\_inserir\_venda('20190517', 3, 100);

10) Se executarmos várias vezes a criação da venda teremos, num determinado momento, erro de PK.

4.png

11) Para resolver este problema precisamos testar se o produto aleatório determinado pela função já existe na tabela de vendas. Modifique a SP com o código abaixo:

USE sucos\_vendas;

DROP procedure IF EXISTS p\_inserir\_venda;

DELIMITER $$

USE sucos\_vendas$$

CREATE DEFINER=root@localhost PROCEDURE p\_inserir\_venda(vData DATE, max\_itens INT,

max\_quantidade INT)

BEGIN

DECLARE vCliente VARCHAR(11);

DECLARE vProduto VARCHAR(10);

DECLARE vVendedor VARCHAR(5);

DECLARE vQuantidade INT;

DECLARE vPreco FLOAT;

DECLARE vItens INT;

DECLARE vNumeroNota INT;

DECLARE vContador INT DEFAULT 1;

DECLARE vNumItensNota INT;

SELECT MAX(numero) + 1 INTO vNumeroNota from notas\_fiscais;

SET vCliente = f\_cliente\_aleatorio();

SET vVendedor = f\_vendedor\_aleatorio();

INSERT INTO notas\_fiscais (CPF, MATRICULA, DATA\_VENDA, NUMERO, IMPOSTO)

VALUES (vCliente, vVendedor, vData, vNumeroNota, 0.18);

SET vItens = f\_numero\_aleatorio(1, max\_itens);

WHILE vContador <= vItens

DO

SET vProduto = f\_produto\_aleatorio();

SELECT COUNT(\*) INTO vNumItensNota FROM itens\_notas\_fiscais

WHERE NUMERO = vNumeroNota AND CODIGO\_DO\_PRODUTO = vProduto;

IF vNumItensNota = 0 THEN

SET vQuantidade = f\_numero\_aleatorio(10, max\_quantidade);

SELECT PRECO\_DE\_LISTA INTO vPreco FROM tabela\_de\_produtos

WHERE CODIGO\_DO\_PRODUTO = vProduto;

INSERT INTO itens\_notas\_fiscais (NUMERO, CODIGO\_DO\_PRODUTO,

QUANTIDADE, PRECO) VALUES (vNumeroNota, vProduto, vQuantidade, vPreco);

END IF;

SET vContador = vContador + 1;

END WHILE;

END$$

DELIMITER ;

12) No curso de manipulação de dados vimos, quando aprendemos TRIGGERs, que devemos criar uma para inclusão, uma para alteração e outra para exclusão. Se, dentro delas, tivemos que executar os mesmos comandos a manutenção será problemática porque qualquer mudança na regra de negócio deverá ser feita nas três TRIGGERs. Mas, se usarmos SP, as mudanças deverão ser feita apenas na SP e não em cada uma das TRIGGERs.

13) Criando a SP:

USE sucos\_vendas;

DROP procedure IF EXISTS p\_calculo\_faturamento;

DELIMITER $$

USE sucos\_vendas$$

CREATE PROCEDURE p\_calculo\_faturamento()

BEGIN

DELETE FROM TAB\_FATURAMENTO;

INSERT INTO TAB\_FATURAMENTO

SELECT A.DATA\_VENDA, SUM(B.QUANTIDADE \* B.PRECO) AS TOTAL\_VENDA FROM

NOTAS\_FISCAIS A INNER JOIN ITENS\_NOTAS\_FISCAIS B

ON A.NUMERO = B.NUMERO

GROUP BY A.DATA\_VENDA;

END$$

DELIMITER ;

14) Criando as TRIGGERs usando a SP:

CREATE TABLE TAB\_FATURAMENTO

(DATA\_VENDA DATE NULL, TOTAL\_VENDA FLOAT);

DELIMITER //

CREATE TRIGGER TG\_CALCULA\_FATURAMENTO\_INSERT AFTER INSERT ON ITENS\_NOTAS\_FISCAIS

FOR EACH ROW BEGIN

Call p\_calculo\_faturamento;

END//

DELIMITER //

CREATE TRIGGER TG\_CALCULA\_FATURAMENTO\_UPDATE AFTER UPDATE ON ITENS\_NOTAS\_FISCAIS

FOR EACH ROW BEGIN

Call p\_calculo\_faturamento;

END//

DELIMITER //

CREATE TRIGGER TG\_CALCULA\_FATURAMENTO\_DELETE AFTER DELETE ON ITENS\_NOTAS\_FISCAIS

FOR EACH ROW BEGIN

Call p\_calculo\_faturamento;

END//

# O que aprendemos?

Nesta aula, aprendemos:

* Usamos nosso conhecimento prático para criação de uma SP que cria uma venda aleatória;
* Vimos como a SP pode melhorar o gerenciamento de TRIGGERs.