

## Consolidando o seu conhecimento

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;

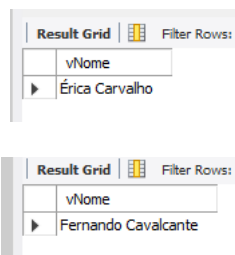
    SELECT vNome;
    FETCH c INTO vNome;

    SELECT vNome;
    CLOSE c;
END$$

DELIMITER ;
```

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

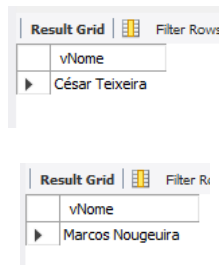
```
call cursor_primeiro_contato;
```



Result Grid	Filter Rows:
vNome	
▶ Érica Carvalho	

Result Grid	Filter Rows:
vNome	
▶ Fernando Cavalcante	



Result Grid		Filter Rows
	vNome	
▶	César Teixeira	

Result Grid		Filter Ru
	vNome	
▶	Marcos Nogueira	

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;

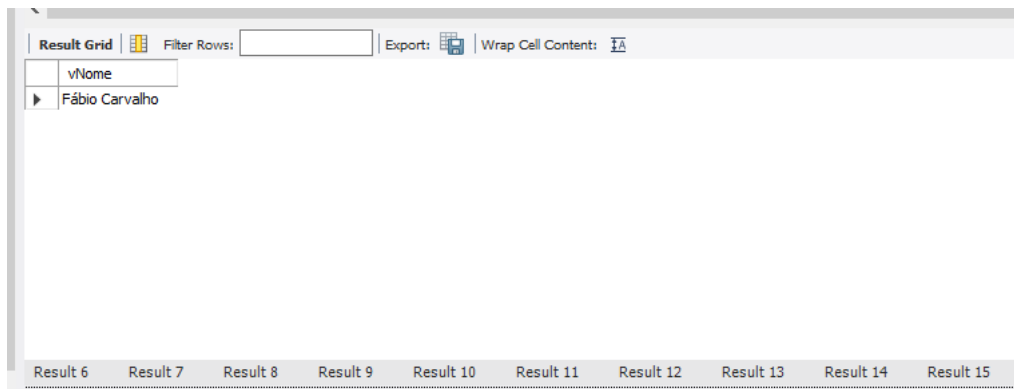
END$$

DELIMITER ;
```

5) Executando a SP:

```
call cursor_looping;
```

Teremos diversos resultados em diferentes consultas.



The screenshot shows a SQL IDE interface. At the top, there is a toolbar with buttons for 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. Below the toolbar, a table is displayed with one column labeled 'vNome' and one row containing the value 'Fábio Carvalho'. At the bottom of the IDE, there is a tab bar with labels for 'Result 6' through 'Result 15'.

vNome
Fábio Carvalho

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;
```

```
CLOSE C;
```

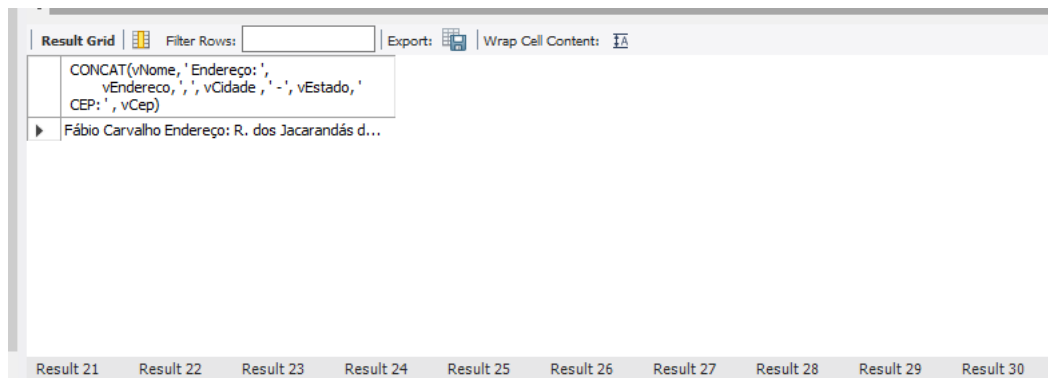
```
END$$
```

```
DELIMITER ;
```

7) Execute-a:

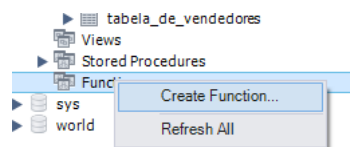
```
call looping_cursor_multiplas_colunas;
```

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 executado 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:

```
ERROR 1418: This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
SQL Statement:
CREATE FUNCTION `f_acha_tipo_sabor`(vSabor VARCHAR(50)) RETURNS varchar(20) CHARSET utf8mb4
BEGIN
```

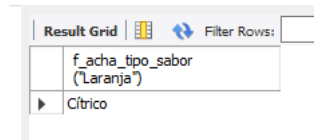
É porque o MYSQL não permite a construção de Funções. Para permitir, execute o comando:

```
SET GLOBAL log_bin_trust_function_creators = 1;
```

E crie a função novamente.

13) Execute a função:

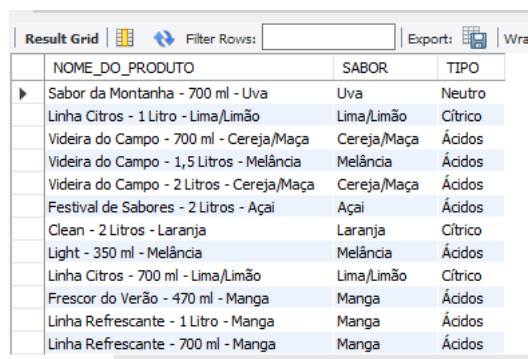
```
SELECT f_acha_tipo_sabor ("Laranja");
```



Result Grid	Filter Rows:
f_acha_tipo_sabor ('Laranja')	
▶ Cítrico	

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;
```



NOME_DO_PRODUTO	SABOR	TIPO
▶ Sabor da Montanha - 700 ml - Uva	Uva	Neutro
Linha Citros - 1 Litro - Lima/Limão	Lima/Limão	Cítrico
Videira do Campo - 700 ml - Cereja/Maça	Cereja/Maça	Ácidos
Videira do Campo - 1,5 Litros - Melância	Melância	Ácidos
Videira do Campo - 2 Litros - Cereja/Maça	Cereja/Maça	Ácidos
Festival de Sabores - 2 Litros - Açaí	Açaí	Ácidos
Clean - 2 Litros - Laranja	Laranja	Cítrico
Light - 350 ml - Melância	Melância	Ácidos
Linha Citros - 700 ml - Lima/Limão	Lima/Limão	Cítrico
Frescor do Verão - 470 ml - Manga	Manga	Ácidos
Linha Refrescante - 1 Litro - Manga	Manga	Ácidos
Linha Refrescante - 700 ml - Manga	Manga	Ácidos