

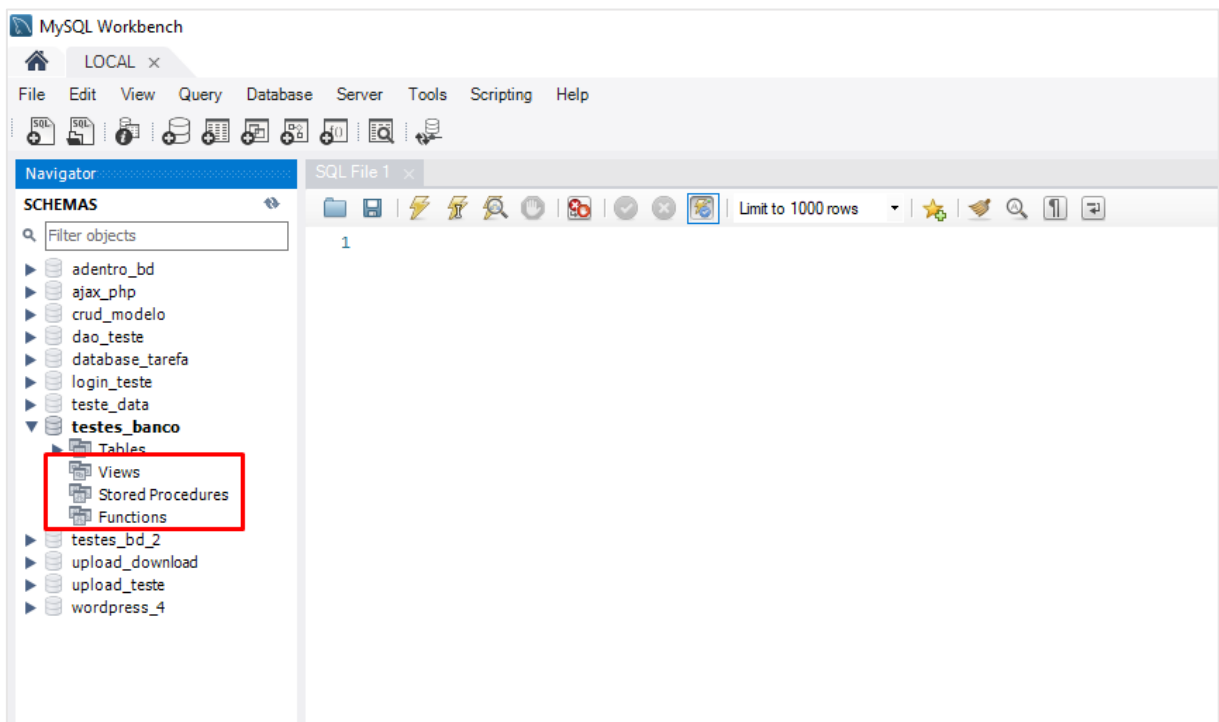
SUMÁRIO

1. ESTRUTURA DE UM BANCO DE DADOS MYSQL	1
2. RECURSOS AVANÇADOS DO MYSQL.....	1
2.1. SOBRE VIEW	1
2.2. SOBRE STORED PROCEDURE.....	2
2.2.1. COMANDOS AVANÇADOS	4
2.3. SOBRE FUNÇÕES	5
FONTES E CONTEÚDO COMPLEMENTAR	6

1. ESTRUTURA DE UM BANCO DE DADOS MYSQL

Pois bem, sabemos que como um SGBD relacional, o MySQL é uma ferramenta que oferece uma interface para criar e gerenciar bancos de dados.

Ainda, no MySQL, os bancos de dados são compostos de tabelas ou entidades, como bem sabemos, que são estruturas organizadas em linhas, ou registros, e colunas, atributos ou campos, e que entidades de um mesmo banco de dados podem estabelecer relações a partir do campo Foreign Key. Isto é o básico. Agora, falaremos sobre as guias View, Stored Procedures e Functions, existentes em cada banco de dados criado.



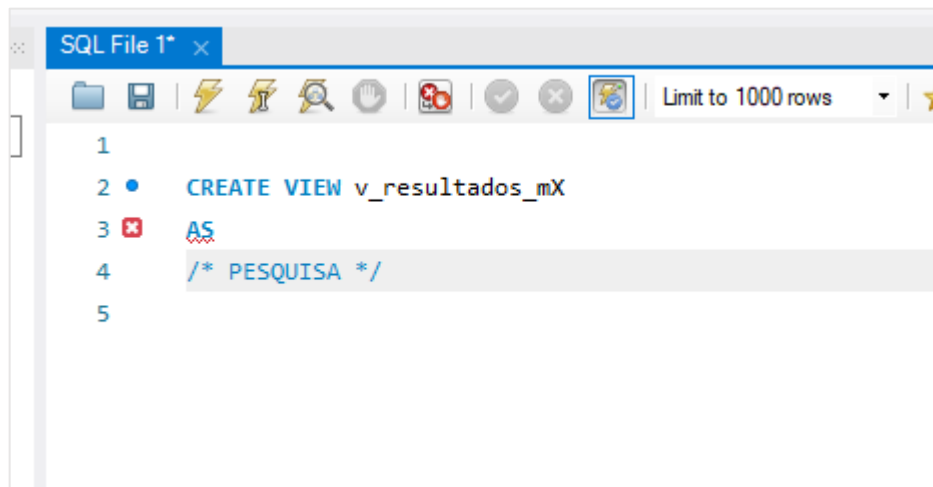
2. RECURSOS AVANÇADOS DO MYSQL

2.1. SOBRE VIEW

Se fossemos definir em uma palavra, poderíamos definir uma View como um “**Atalho de um SELECT**”, pois nada mais é do que uma funcionalidade que permite tornar o retorno de um SELECT em uma espécie de, como de fato formalmente chamam, “tabela virtual”, que pode ser usada posteriormente sem necessitar da reescrita da pesquisa.

A exemplo, se realizo uma pesquisa (com SELECT) que retorna os funcionários que realizaram vendas em um determinado mês X, seguido de outras informações, posso tornar o resultado da pesquisa um Objeto View chamado “**v_resultados_mX**”, de forma que se quiser

realizar novas pesquisas para extrair mais informações a partir dos mesmos dados, basta utilizar o nome do Objeto, ao invés de realizar uma nova pesquisa.



Assim, se entre os funcionários que venderam no mês X, existisse o de id igual a 2, poderíamos escrever:

```
SELECT * FROM v_resultados_mX WHERE id = 2;
```

2.2. SOBRE STORED PROCEDURE

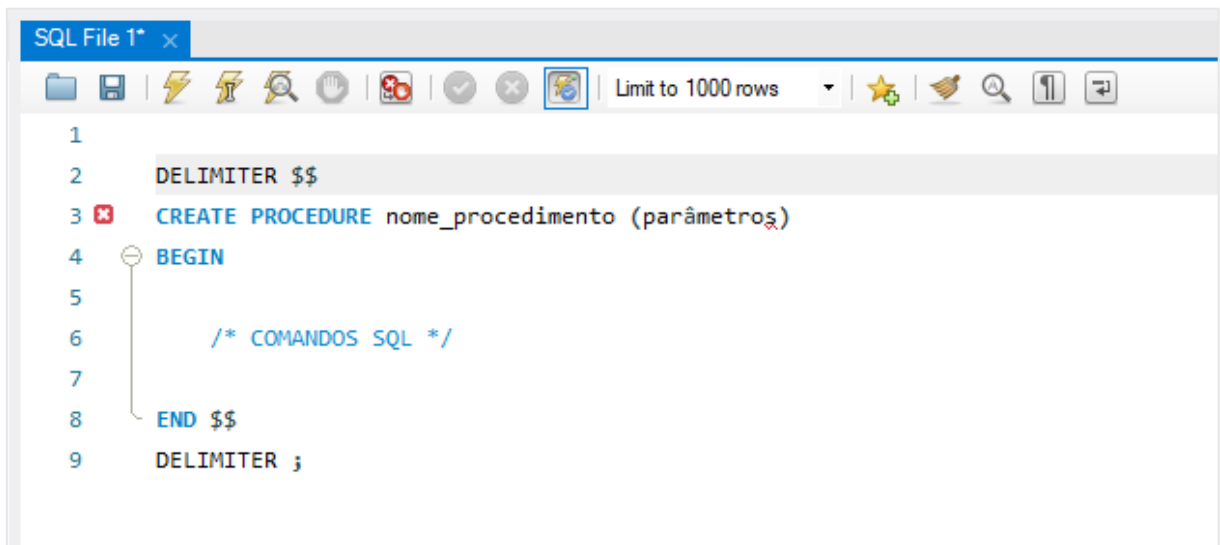
Como diz [neste artigo](#) da Devmedia, “quando desenvolvemos aplicações que acessam banco de dados (boa parte delas), é comum executarmos rotinas complexas de manipulação desses dados a partir da linguagem/ferramenta utilizada. Para isso, utilizamos várias instruções SQL em sequência para obter o resultado esperado”. Considere isto.

Pois bem, veja: uma procedure serve para abstrair e agrupar a execução das diversas rotinas que devem ser realizadas. Em outras palavras, uma procedure permite que diversas rotinas sejam agrupadas em apenas uma...**exatamente como uma função**.

Uma procedure, assim como uma função, é uma espécie de bloco que pode receber parâmetros e que contém tarefas, para que assim, ao invés de escritas uma a uma, em todas as vezes que forem necessárias, possam ser executadas a partir da execução do bloco – e que pode ser infinitamente reutilizado.

Mas, diferentemente da função, neste caso o que chamamos de tarefas não são programas, e sim comandos SQL. Além disto, como blocos de comandos SQL, naturalmente as procedures não retornam dados à moda das funções de uma linguagem programável. Neste

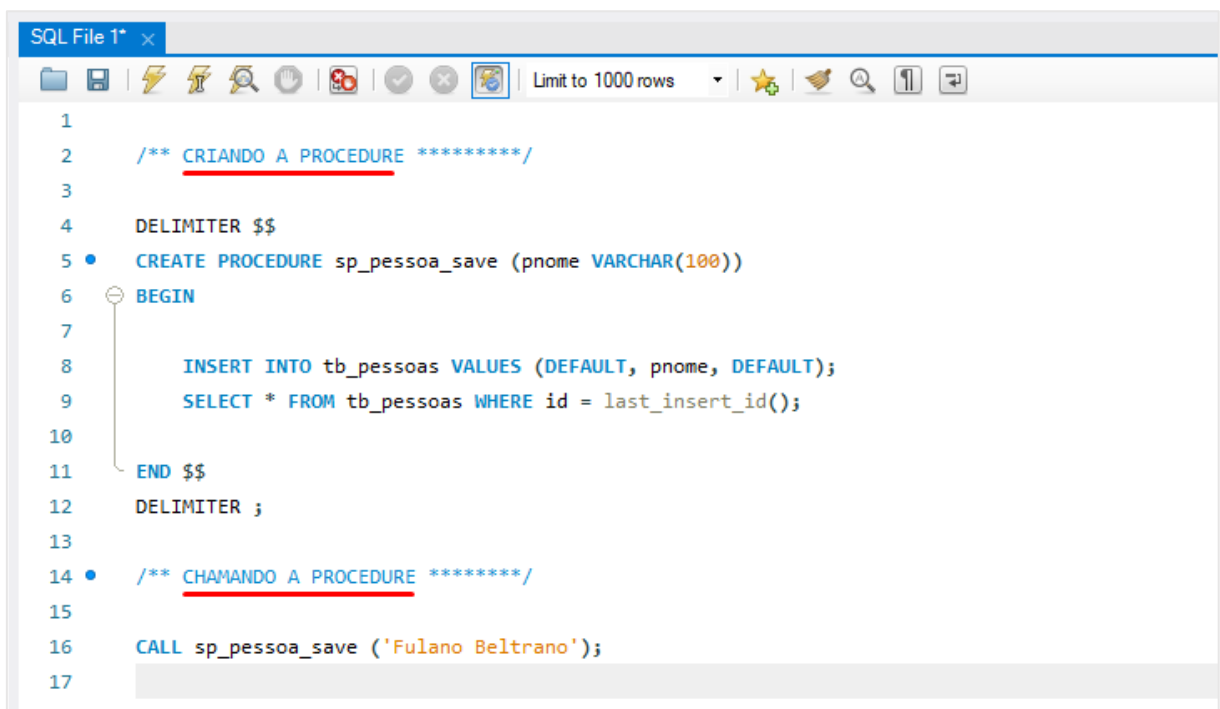
caso o único retorno que podemos desejar é a própria execução das queries, e seus efeitos no ambiente MySQL. **Veja abaixo a estrutura base de uma procedure:**



```

1
2 DELIMITER $$
3 CREATE PROCEDURE nome_procedimento (parâmetros)
4 BEGIN
5
6     /* COMANDOS SQL */
7
8 END $$
9 DELIMITER ;
  
```

Agora, na imagem abaixo veja um exemplo de criação de uma procedure cuja função é realizar um INSERT e em seguida um SELECT do último registro inserido que teve um campo auto incrementado (que por via de regra é o campo ID), isto é, necessariamente o registro que foi inserido a partir do INSERT da procedure.



```

1
2 /** CRIANDO A PROCEDURE *****/
3
4 DELIMITER $$
5 CREATE PROCEDURE sp_pessoa_save (pnome VARCHAR(100))
6 BEGIN
7
8     INSERT INTO tb_pessoas VALUES (DEFAULT, pnome, DEFAULT);
9     SELECT * FROM tb_pessoas WHERE id = last_insert_id();
10
11 END $$
12 DELIMITER ;
13
14 /** CHAMANDO A PROCEDURE *****/
15
16 CALL sp_pessoa_save ('Fulano Beltrano');
17
  
```

2.2.1. COMANDOS AVANÇADOS

Podemos ainda ampliar as procedures introduzindo comandos DTL, e deixa-las mais parecidas com linguagens programáveis, declarando variáveis, e determinando condicionais para execução de determinadas queries. Veja abaixo uma procedure com essas funcionalidades:

```

1  DELIMITER $$
2  • CREATE PROCEDURE sp_pessoa_save (pnome VARCHAR(100), psalario DECIMAL(10,2), dt_admissao DATETIME)
3  BEGIN
4
5      DECLARE ID_pessoa INT;
6
7      START TRANSACTION;
8
9      IF NOT EXISTS(select id_pessoa from tb_pessoas where nome = pnome) THEN
10
11          INSERT INTO tb_pessoas VALUES (DEFAULT, pnome, DEFAULT);
12
13          SET ID_pessoa = last_insert_id();
14
15      ELSE
16
17          SELECT 'Usuário já cadastrado!' AS resultado;
18          ROLLBACK;
19
20      END IF;
21
22      COMMIT;
23
24      SELECT 'Operação realizada com sucesso!' AS resultado;
25
26  END $$
27  DELIMITER ;
28
29  CALL sp_pessoa_save ('Fulano Beltrano', 50000, current_date());

```

Vamos entender: (i) **primeiro** criamos a procedure, cujo nome é `sp_pessoa_save`, e cujos parâmetros são um campo nome do tipo varchar, um campo salário do tipo decimal, e um campo de data do tipo datetime; (ii) **segundo**, no corpo da procedure, primeiramente declaramos uma variável `ID_pessoa`; (iii) **terceiro**, em seguida começamos uma transação que irá conter o conjunto de comandos SQL e suas condições de execução.

Continuando: a condicional utilizada é uma de negação de existência, isto é, “se tal coisa não existir, faça”, e neste caso, se o retorno da pesquisa, que é o argumento da condição, retornar false, a própria condicional será verdadeira, e então será realizado um INSERT, e a atribuição do valor do campo ID auto incrementado à variável `ID_pessoa`. Caso contrário, se a pesquisa retornar true, isto é, se existir o registro pesquisado, a próprio condicional será false, e o código executado será um SELECT que retornará a mensagem “Usuário já cadastrado” em um campo virtual.

Assim, veja, o SELECT utilizado para retornar uma mensagem em um campo virtual, caso exista um registro com os valores informados, é como um retorno de uma função de uma linguagem de programação. Não existe “return” no SQL, mas podemos realizar uma saída de dados para o usuário com o próprio SELECT.

Ainda, após este SELECT, escrevemos o comando ROLLBACK, que faz parte do conjunto DTL, e serve para negar a transação.

Por fim, fechamos o bloco de condicional com END IF, e em seguida escrevemos COMMIT, que é o comando DTL para executar a transação. Assim, a transação só é aceita se a primeira condicional for verdadeira, se não, será impressa uma mensagem, para informar que o registro já existe, e a transação será negada.

Agora, para que serviu a variável declarada? Em termos de participação da tarefa da procedure, não teve utilidade. A ideia era apenas mostrar como variáveis são declaradas no SQL, e como podem receber valores – e se necessário, serem utilizadas livremente em queries dentro da mesma procedure.

2.3. SOBRE FUNÇÕES

Além das “Built-In Functions” que já utilizamos, e que são nativas o MySQL, podemos criar nossas próprias funções, as chamadas “User Defined Functions”, que ficam disponíveis para consulta na guia de functions, dentro de cada banco de dados.

Além disto, e diferentemente das procedures, as funções naturalmente, como já sabemos e conhecemos, possuem um retorno obtido a partir do seu chamado em uma query. Também, para finalizar, as funções não são chamadas a partir do comando CALL, como são as procedures, mas sim da mesma forma que são aquelas que são nativas, como SUM(), AVG(), MAX(), MIN(), ou seja, nome_da_função(parâmetros).

Para entender como criar e utilizar funções, clique [aqui](#).

FONTES E CONTEÚDO COMPLEMENTAR

<https://www.devmedia.com.br/stored-procedures-no-mysql/29030>

<https://www.devmedia.com.br/procedures-e-funcoes-no-mysql/2550>

Mais sobre condicionais: <https://www.youtube.com/watch?v=xikU6R7vZrs>