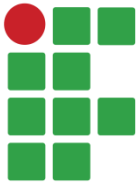
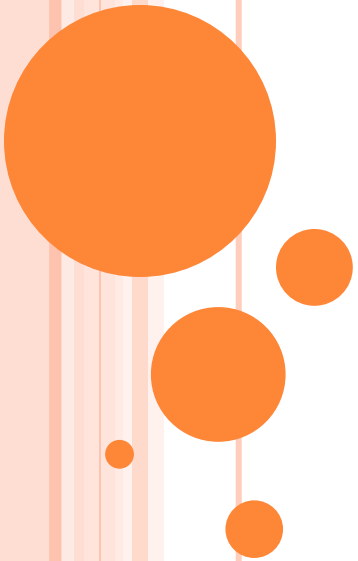


INSTITUTO FEDERAL
Rio Grande do Sul

# Banco de Dados II

Prof. Bruna Flor da Rosa



# Linguagem SQL
# Structured Query Language

# Roteiro

- Stored Procedure

# Stored procedure

- Stored procedures – procedimentos armazenados – são rotinas definidas no banco de dados, identificadas por um nome pelo qual podem ser invocadas. Um procedimento desses pode executar uma série de instruções, receber parâmetros e retornar valores, assim como funções, procedimentos ou métodos na programação.

# Stored procedure

- A utilização de Stored Procedures é uma técnica eficiente de executarmos operações repetitivas.

- Ao invés de digitar os comandos cada vez que determinada operação necessite ser executada, criamos um Stored Procedure e o chamamos. Em um Stored Procedure também podemos ter estruturas de controle e decisão, típicas das linguagens de programação.

# Stored procedure

- As Stored Procedures ajudam a reduzir o tráfego na rede, a melhorar o desempenho de consultas, a criar mecanismos de segurança e simplificar o código da aplicação, já que não haverá a necessidade de manter consultas SQL de várias linhas misturadas a toda lógica da sua aplicação.

# Criando Stored procedure

```
DELIMITER //
CREATE PROCEDURE mostratudo()
BEGIN
   SELECT * FROM clientes;

END //
DELIMITER ;
```

# Stored Procedures

Código:

```
DELIMITER //
CREATE PROCEDURE mostratudo()
BEGIN
   SELECT * FROM clientes;
END //
DELIMITER ;
```

O delimiter serve para marcar onde começa e onde termina a procedure, assim a procedure poderá possuir múltiplas linhas (separadas por ;). Quando a procedure termina, o delimiter padrão deve ser mudado para ; novamente.

# Stored Procedures

Código:

```
DELIMITER //
CREATE PROCEDURE mostratudo()
BEGIN
   SELECT * FROM clientes;
END //
DELIMITER ;
```

A sintaxe para criar um procedimento é CREATE PROCEDURE.

O que estiver entre o BEGIN e o END será executado quando o procedure for chamado.

Note que o END tem o delimitador.

# Stored Procedures

Código:

```
DELIMITER //
CREATE PROCEDURE mostratudo()
BEGIN
   SELECT * FROM clientes;
END //
DELIMITER ;

CALL mostratudo();
```

Note que o procedimento contém um comando simples de seleção. Este comando será executado quando chamarmos o procedimento.

CALL serva para chamar o procedimento previamente criado.

# Stored Procedures

Código:

```
DELIMITER //
DROP PROCEDURE IF EXISTS mostratudo //
CREATE PROCEDURE mostratudo()
BEGIN
   SELECT * FROM clientes;
END //
DELIMITER ;
```

Para rodar novamente o script, será necessário dar um DROP PROCEDURE IF EXISTS. Assim não dará erro dizendo que o procedimento já existe.

Note que como o comando fica dentro da nova definição de delimitador, deve obedecê-la.

# Considerações

- Em um Stored Procedure, podemos incluir qualquer comando SQL, com exceção dos seguintes: CREATE PROCEDURE, CREATE DEFAULT, CREATE RULE, CREATE TRIGGER E CREATE VIEW;

- Em um Stored Procedure podemos referenciar tabelas, Views, outras Stored Procedures.

# Variáveis

Analisemos o código abaixo:

```
DROP PROCEDURE IF EXISTS mostra_numero //
CREATE PROCEDURE mostra_numero()
BEGIN
   DECLARE num INT DEFAULT 0;
   SET num = 666;
   SELECT num;
END //
```

Para declarar uma variável usa-se a sintaxe DECLARE seguido do nome da variável e o tipo.

Caso queira desejemos atribuir um valor inicial usamos DEFAULT seguido do valor.

Para atribuir um valor à uma variável usamos SET.

Para mostrar o valor de uma variável simplesmente usamos um SELECT

# Variáveis

```
DROP PROCEDURE IF EXISTS conta_clientes //
CREATE PROCEDURE conta_clientes()
BEGIN
   DECLARE num_clientes INT DEFAULT 0;
   SELECT count(*) INTO num_clientes FROM clientes;
   SELECT num_clientes AS 'Número de clientes cadastrados';
END //
```

O comando count(*) retorna o número de linhas retornada pelo SELECT

SELECT ____ INTO *variável* diz que o resultado do SELECT será armazenado dentro da variável.

SELECT ____ AS *rótulo* diz para renomear a coluna que for retornada para o rótulo dado.

# Passagem de parâmetros

```
DROP PROCEDURE IF EXISTS cade_cliente //
CREATE PROCEDURE cade_cliente(IN nome_cli VARCHAR(100))
BEGIN
   SELECT * FROM clientes WHERE nome = nome_cli;
END //
```

Para passar uma variável como parâmetro para um procedimento, deve-se especificar se a variável é de entrada, saída, ou ambos (IN, OUT, INOUT), seu nome e seu tipo.

Neste exemplo, estamos usando uma variável de entrada, isto é, o parâmetro que for passado para o procedimento será utilizado dentro do mesmo, mas não poderá ser retornado.

- A utilização de parâmetros de entrada permite a criação de Stored Procedures mais flexíveis. Você pode inclusive criar Stored Procedures com comandos de Inserção (INSERT), Alteração (UPDATE) e Deleção (DELETE).

# Passagem de parâmetros

Quando queremos chamar um procedimento com algum parâmetro, devemos fazê-lo assim:

```
CALL cade_cliente('pablo');
```

Desta maneira, quando o procedimento for executado, o conteúdo passado como parâmetro vai ser atribuído à variável correspondente.

# Passagem de parâmetros

```
DROP PROCEDURE IF EXISTS pega_endereco//
CREATE PROCEDURE pega_endereco(IN nome_cli
VARCHAR(100), OUT end_cli VARCHAR(255))
BEGIN
   SELECT endereco INTO end_cli FROM clientes WHERE nome =
nome_cli;
END //
```

Acima temos um exemplo de uma variável de saída, declarada como OUT.

O resultado do SELECT (campo endereço do cliente dado como entrada) será colocado dentro da variável de saída end_cli através do parâmetro INTO do SELECT.

# Passagem de parâmetros

Chamamos assim um procedimento com um parâmetro de saída:

```
CALL pega_endereco('pablo',@end);
SELECT @end AS 'Endereço do cliente';
```

@end é uma variável que será usada fora do procedimento (note o uso de @ antes do nome da variável para variáveis da sessão).

Como de costume, podemos dar um SELECT para mostrar o conteúdo da variável.

# Passagem de parâmetros

```
DROP PROCEDURE IF EXISTS g_to_r//
CREATE PROCEDURE g_to_r(INOUT graus DECIMAL(16,4))
BEGIN
   SET graus = graus*pi()/180;
END //
```

Aqui podemos ver um uso de um tipo de variável de entrada e saída (INOUT). Com elas, a mesma variável que usamos como entrada será modificada dentro do procedimento. Esta modificação manterá seu efeito mesmo fora do escopo do procedimento.

Aqui vemos um exemplo de uso:

```
SET @ang = 31;
CALL g_to_r(@ang);
SELECT @ang AS 'Ângulo em radianos';
```

# PARA OS EXERCÍCIOS, CONSIDERE ESTE MODELO DE DADOS.

Areas (Acod, Nome, Descricao)

Especialidades (Ecod, Nome, CodArea#)

Planos (PLcod, Nome, Tipo)

Cidades (Ccod, Nome, Uf)

Pacientes (Pcod, Nome, Telefone, Endereco, DataNasc, CodPlano#, CodCid#)

Medico (Crm, Nome, Telefone, CodEspecialidade#)

Consultas (CodPaciente#, CrmMed#,Data, Hora, Valor)

# Exercícios: descreva o script para realizar as seguintes procedures e para suas chamadas também

1)Elabore uma stored procedure para mostrar na tela o nome de todos pacientes cadastrados.

2)Elabore uma stored procedure para exibir as informações dos pacientes de um determinado médico, no qual o crm é enviado por parâmetro.

3)Elabore uma stored procedure para inserir dados em uma cidade. Envie os dados por parâmetro.

4)Elabore uma stored procedure que receba o código de um paciente, o crm do médico e a data de uma consulta por parâmetro e retorne a hora da consulta.

5)Elabore uma stored procedure que exiba as informações dos pacientes que moram em uma determinada cidade.

# Exercícios

6)Elabore uma stored procedure que receba o crm de um médico e retorne o nome de sua especialidade.

7)Elabore uma stored procedure que receba o código de um plano de saúde e exiba na tela o nome dos pacientes que possuem este plano.

8) Elabore uma stored procedure que receba o código de uma área e seu novo nome e altere o nome para este.

9) Elabore uma stored procedure que altere o plano de saúde de um determinado paciente.

10) Elabore uma stored procedure que receba por parâmetro as informações de uma consulta e retorne o valor desta com 50% de desconto. Utilize a mesma variável como entrada e saída para o valor.