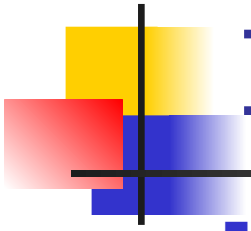
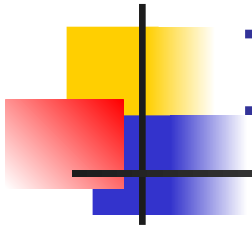
Procedimentos Armazenados (Stored Procedures)

Profa. Dra. Andréia R. Casare
E-mail: casareandreia@gmail.com
andreia.casare01@fatec.sp.gov.br



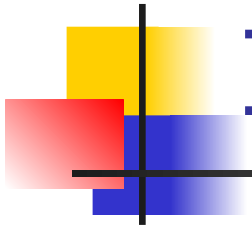
Introdução

- Procedimentos armazenados são pequenos pedaços de código que ficam armazenados no lado do servidor de banco de dados.
- É um recurso poderoso que pode ser explorado em um ambiente de banco de dados. Praticamente todos os gerenciadores de banco de dados oferecem a possibilidade de programação a nível de banco de dados.
- A programação de banco de dados é feita através da implementação de programas que são armazenados no banco de dados.



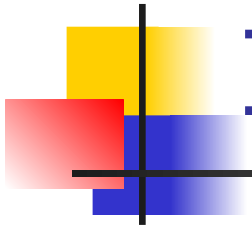
Introdução

- No PostgreSQL os procedimentos armazenados podem ser funções (Stored Functions) ou gatilhos (triggers).
- As stored procedures fornecem uma maneira poderosa de agrupar lógica de negócios e operações complexas em uma única unidade reutilizável. Elas podem ser usadas para automatizar tarefas, melhorar o desempenho e melhorar a segurança do seu banco de dados.



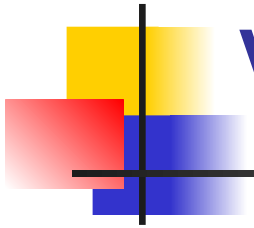
Introdução

- É possível usar a linguagem SQL e PL/pgSQL e ainda usar outras linguagens C, Python, Java para a implementação de procedimentos armazenados e gatilhos.



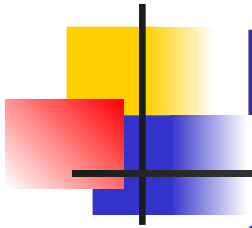
Introdução

- As não procedurais utilizam a SQL como linguagem. Essas são caracterizadas por não possuírem estruturas comuns às linguagens de programação, como por exemplo condição (if, else, case), etc.
- Entre as procedurais estão a PL/pgSQL e as externas.
- Entre as externas temos linguagens como Python, Java, C e C++.



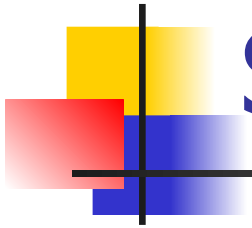
Vantagens

- Redução do tráfego na rede entre a aplicação e o servidor de BD. Ao invés de enviar vários comandos SQL ao servidor de banco de dados você pode embutir um grupo de comandos SQL em um procedimento armazenado e executar o procedimento que processará os comandos diretamente no servidor de banco de dados;
- Aumento da performance do sistema devido ao fato de que os procedimentos armazenados são compilados e armazenados no servidor de banco de dados.
- Reuso por várias aplicações que façam uso do banco de dados.



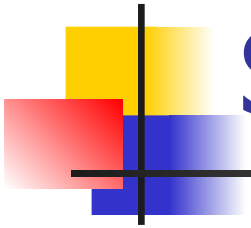
Desvantagens

- Requer profissionais com conhecimento mais especializado que nem todo desenvolvedor possui.
- Maior esforço para manter versões de código pois tem também código do servidor de BD para manter atualizado
- Dificuldade de depuração dos procedimentos armazenados pois requer ferramenta própria para isso e que são bastante caras.
- Perda de portabilidade caso precise migrar para outros gerenciadores de banco de dados. Por exemplo: DB2, SQL Server ou MYSQL.



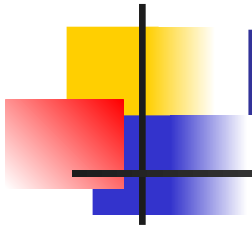
Sintaxe básica

- A criação de uma nova função é feita através da execução do comando CREATE FUNCTION.
- Sintaxe:
  - CREATE [OR REPLACE] FUNCTION nome (tipo [, ...]]) [RETURNS tipo_retorno] as
  - corpo da função
  - LANGUAGE nome_linguagem;



Sintaxe

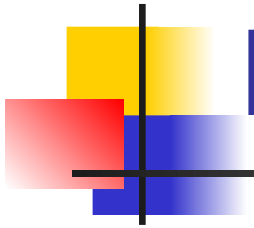
- onde:
  - nome - é o nome da função a ser criada.
  - tipo - define o tipo do argumento que será recebido.
  - tipo_retorno - é o tipo de dado que será retornado pela nossa função.
  - nome_linguagem - é o nome da linguagem que será utilizada para escrever a nossa função.
  - conteudo - é o corpo da nossa função



Exemplo

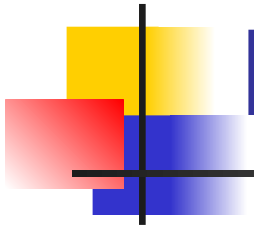
```
CREATE OR REPLACE FUNCTION
add_numeros(nr1 int, nr2 int) RETURNS int AS
$$
    SELECT $1 + $2;
$$
LANGUAGE SQL;
```

- Executando a função:
  - SELECT add_numeros(300, 700);



Exemplo

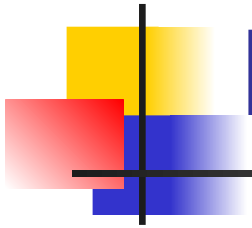
```
CREATE OR REPLACE FUNCTION inserir_dados(nome text, idade integer) RETURNS void AS  $$
INSERT INTO tabela_exemplo (nome, idade) VALUES (nome, idade);
$$ LANGUAGE SQL;

SELECT inserir_dados('Andreia',40);
```



Exemplo

```
CREATE FUNCTION add_ponto(disciplinas_aluno)
RETURNS numeric AS $$
    SELECT $1.nota + 0.5 AS nota;
$$ LANGUAGE SQL;
```

- select * from disciplinas_aluno where cod_aluno = 2013001
- Executando a função:
  - SELECT cod_aluno, add_ponto(disciplinas_aluno) AS nova_nota FROM disciplinas_aluno WHERE cod_aluno = 2013001;



Exemplo

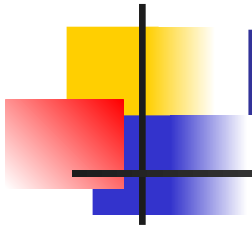
```
CREATE FUNCTION qtde_disciplinas_curso(integer)
RETURNS bigint  AS $$
    SELECT COUNT(cod_curso) FROM curso_disciplina
  WHERE cod_curso= $1
$$ LANGUAGE SQL;
```

- Executando:
  - SELECT qtde_disciplinas_curso(1);

Exemplo de função que retorna a idade

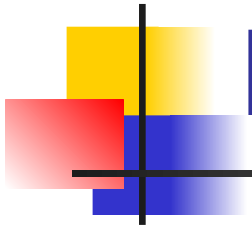
```
CREATE FUNCTION retorna_idade(int) RETURNS varchar AS
$$
SELECT TO_CHAR(AGE(CURRENT_DATE,data_nasc), 'YY') as intervalo FROM aluno WHERE ra = $1
$$
LANGUAGE SQL;
```

- Executando a função:
  - SELECT retorna_idade(2013001);



Exemplo

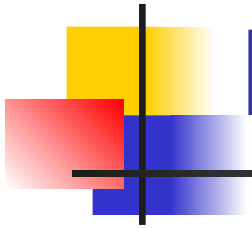
```
CREATE OR REPLACE FUNCTION
obter_dados_por_id(id integer) RETURNS TABLE
(nome text, idade integer) AS $$
    SELECT nome, idade FROM tabela_exemplo
    WHERE id = $1;
$$ LANGUAGE SQL;
```



Exemplo

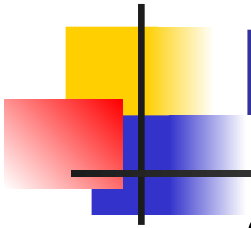
```
CREATE FUNCTION retorna_maior(anyelement,
anyelement) RETURNS boolean AS
$$
SELECT $1 > $2;
$$ LANGUAGE SQL;
```

- Executando
  - SELECT retorna_maior(2,1);



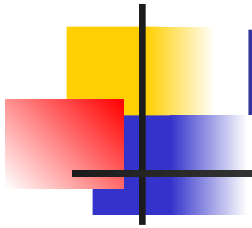
Linguagens Procedurais

- As funções em linguagens procedurais no PostgreSQL, como a PlpgSQL são correspondentes ao que se chama de Stored Procedures.
- A PlpgSQL é a linguagem de procedimentos armazenados mais utilizada no PostgreSQL, devido ser a mais madura e com mais recursos.



Exemplo 1

```
CREATE FUNCTION somar_tres_valores(v1 integer, v2
    integer, v3 integer) RETURNS integer AS $$
DECLARE
    resultado integer;
BEGIN
    resultado := v1 + v2 + v3;
    RETURN resultado;
END;
$$ LANGUAGE plpgsql;
```

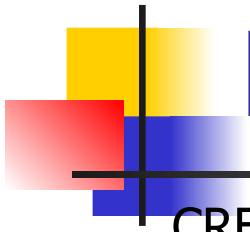


Exemplo 1

- Para executar a função:
  - SELECT somar_tres_valores(10,20,30);

Exemplo 2

```
CREATE OR REPLACE FUNCTION verificar_idade(idade integer)
RETURNS text AS $$
DECLARE
        mensagem text;
BEGIN
        IF idade >= 18 THEN
            mensagem := 'Pessoa é maior de idade’;
        ELSE
            mensagem := 'Pessoa é menor de idade’;
        END IF;
      RETURN mensagem;
END;
$$ LANGUAGE plpgsql;
```

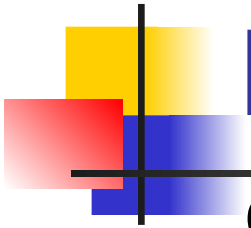


Exemplo 3

```
CREATE FUNCTION maior(num1 integer, num2 integer) RETURNS integer
    AS $$
DECLARE
  num1 ALIAS FOR $1;
  num2 ALIAS FOR $2;
BEGIN
    IF $1 > $2 THEN
                return $1;
    ELSE
                 return $2;
    END IF;
END;
$$ LANGUAGE 'plpgsql'
```

Exemplo 4

```
CREATE FUNCTION reajuste(valor numeric) RETURNS numeric AS $$
DECLARE
  valor ALIAS FOR $1;
  novoval numeric;
BEGIN
   IF $1 > 5000 THEN
       novoval:= $1 + ($1 * 0.10);
   ELSE
       novoval:= $1 + ($1 * 0.05);
   END IF;
   return novoval;
END;
$$ LANGUAGE 'plpgsql'
```



Exemplo 5

```
CREATE FUNCTION verificar_par(num int) RETURNS
varchar AS $$
BEGIN
  IF num % 2 = 0 THEN
    RAISE NOTICE 'O número % é par.', num;
  ELSE
    RAISE NOTICE 'O número % é ímpar.', num;
  END IF;
END;
$$ LANGUAGE plpgsql;
```

Exemplo 6

```
CREATE OR REPLACE FUNCTION consultar_nota(ra INTEGER, cod_dis INTEGER) RETURNS NUMERIC(4,2) AS $$
DECLARE
    v_nota NUMERIC(4,2);
BEGIN
    SELECT nota
    INTO v_nota
    FROM disciplinas_aluno    WHERE cod_aluno = ra AND cod_discip = cod_dis;
    IF v_nota IS NULL THEN
        RAISE NOTICE 'Não foi encontrada nota para o aluno % na disciplina %',    ra, cod_dis;
    ELSE
        RETURN v_nota;
    END IF;
END;
$$
LANGUAGE plpgsql;

SELECT consultar_nota(2014010,2);
```

Exemplo 7

```
CREATE OR REPLACE FUNCTION contar_turmas_por_curso(cod_cur INTEGER) RETURNS INTEGER
    AS $$
DECLARE
   qtd INTEGER;
BEGIN
   -- Conta quantas turmas existem para o curso informado
   SELECT COUNT(*)
   INTO qtd
   FROM turma
   WHERE cod_curso = cod_cur;
   RAISE NOTICE 'O curso % possui % turma(s) cadastrada(s)', cod_cur, qtd;
END;
$$
LANGUAGE plpgsql;
```