

1 Introdução

Neste material, estudaremos as estruturas de repetição presentes na linguagem **PL/pgSQL**. Sua documentação pode ser encontrada no Link 1.1.

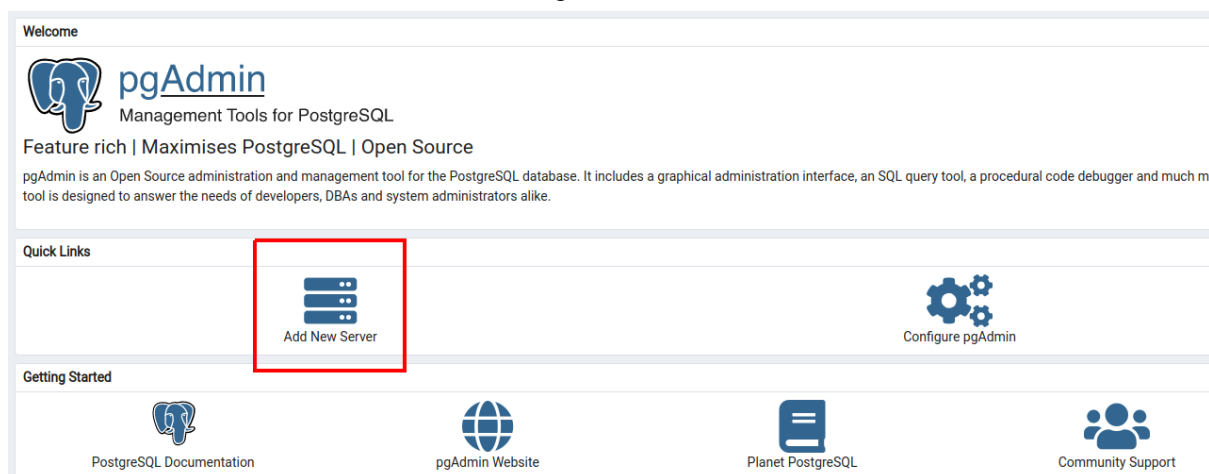
Link 1.1

<https://www.postgresql.org/docs/current/plpgsql-control-structures.html>

2 Passo a passo

2.1 (Criando um servidor) Caso ainda não possua um servidor, abra o pgAdmin4 e clique em **Add New Server**, como mostra a Figura 2.1.1.

Figura 2.1.1



O nome do servidor pode ser algo que lhe ajude a lembrar a razão de ser dele. Como é um servidor que está executando localmente, podemos chamá-lo de algo como **localhost**, como na Figura 2.1.2. Depois de preencher o nome, clique na aba **Connection**.

Figura 2.1.2

The screenshot shows the 'Register - Server' dialog box with the 'General' tab selected. The 'Name' field contains 'localhost' and is highlighted with a red box. The 'Server group' dropdown is set to 'Servers'. The 'Background' and 'Foreground' checkboxes are unchecked. The 'Connect now?' toggle is turned on. A red error message at the bottom states: 'Either Host name, Address or Service must be specified.' The 'Save' button is visible at the bottom right.

Agora clique na aba **Connection**, como na Figura 2.1.3. Preencha os campos como destacado e clique em **Save**.

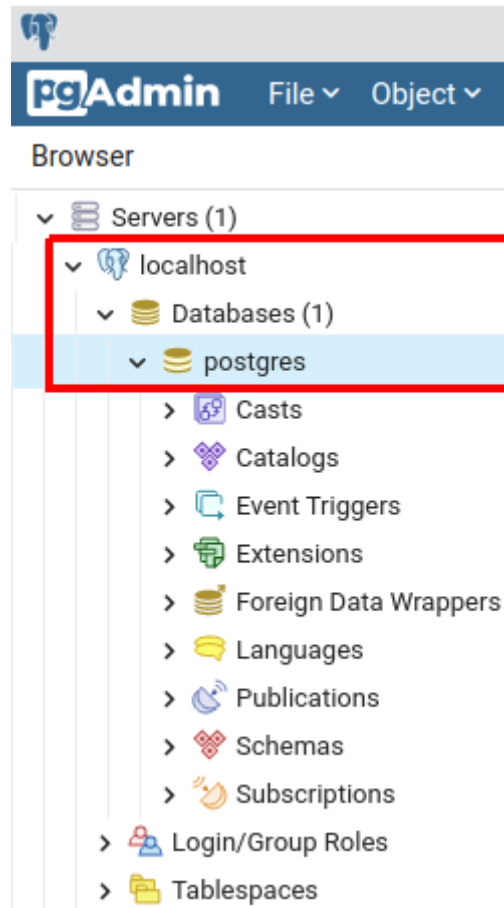
Figura 2.1.3

The screenshot shows the 'Register - Server' dialog box with the 'Connection' tab selected. The 'Host name/address' field contains 'localhost' and is highlighted with a red box. The 'Port' field contains '5432'. The 'Maintenance database' field contains 'postgres'. The 'Username' field contains 'rodrigo' and is highlighted with a red box. The 'Password' field contains '.....' and is highlighted with a red box. The 'Kerberos authentication?' and 'Save password?' toggles are turned off. The 'Role' and 'Service' fields are empty. The 'Save' button at the bottom right is highlighted with a red box.

Nota. Usuário e senha dependerão de suas configurações. No Windows, é comum a existência de um usuário chamado postgres com a senha também igual a postgres.

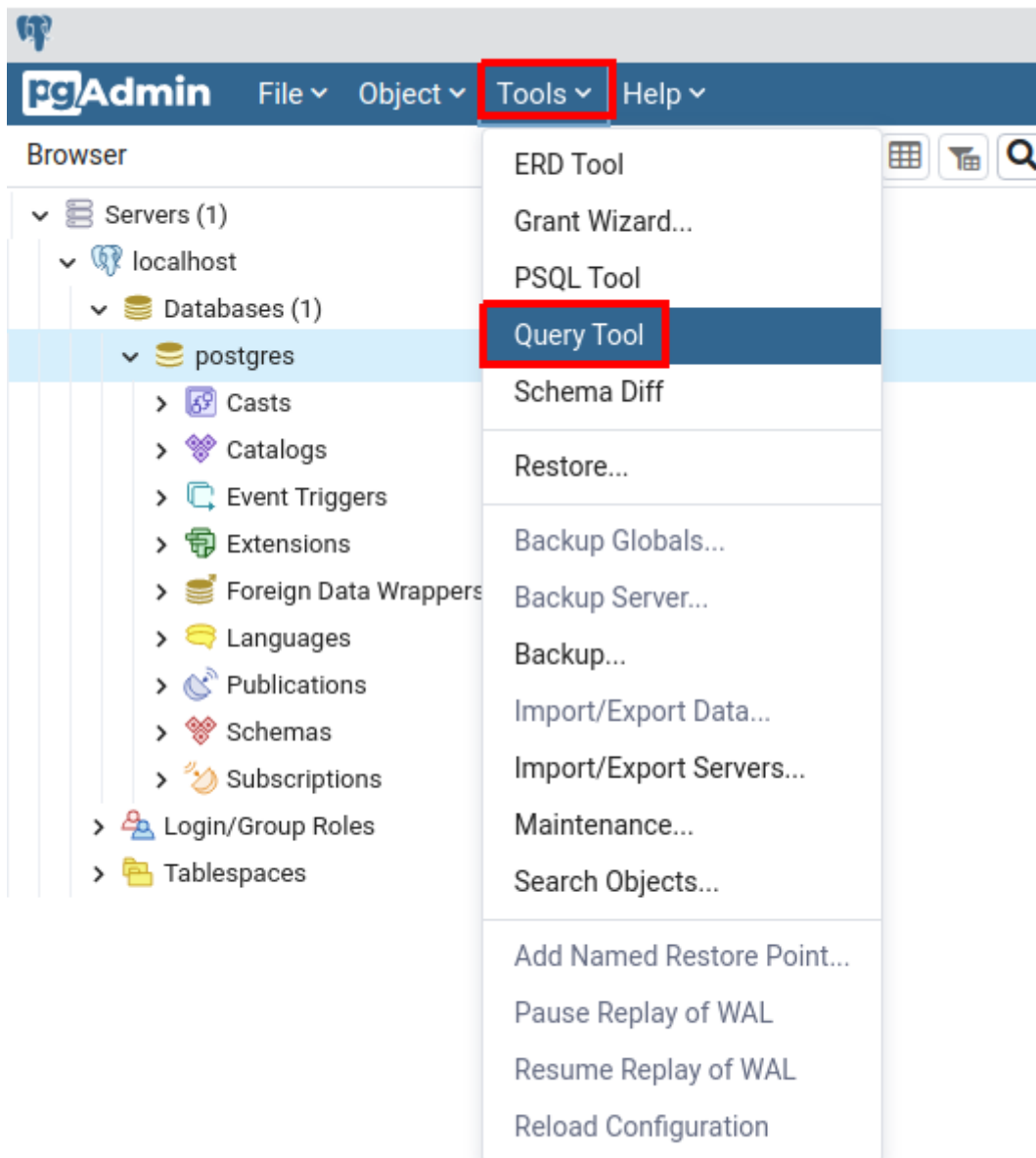
No canto superior esquerdo, encontre o seu servidor e clique sobre ele. Expanda **Databases** e encontre o database chamado **postgres**, cuja existência é muito comum. Veja a Figura 2.1.4.

Figura 2.1.4



Para abrir um editor em que possa digitar seus comandos SQL, clique em **Tools >> Query Tool**, como mostra a Figura 2.1.5.

Figura 2.1.5



2.2 (Estruturas de repetição: Quais são?) A linguagem **PL/pgSQL** possui as seguintes estruturas de repetição.

- LOOP
- WHILE
- FOR
- FOREACH

como veremos, seu uso pode ser combinado com

- rótulos
- EXIT
- CONTINUE

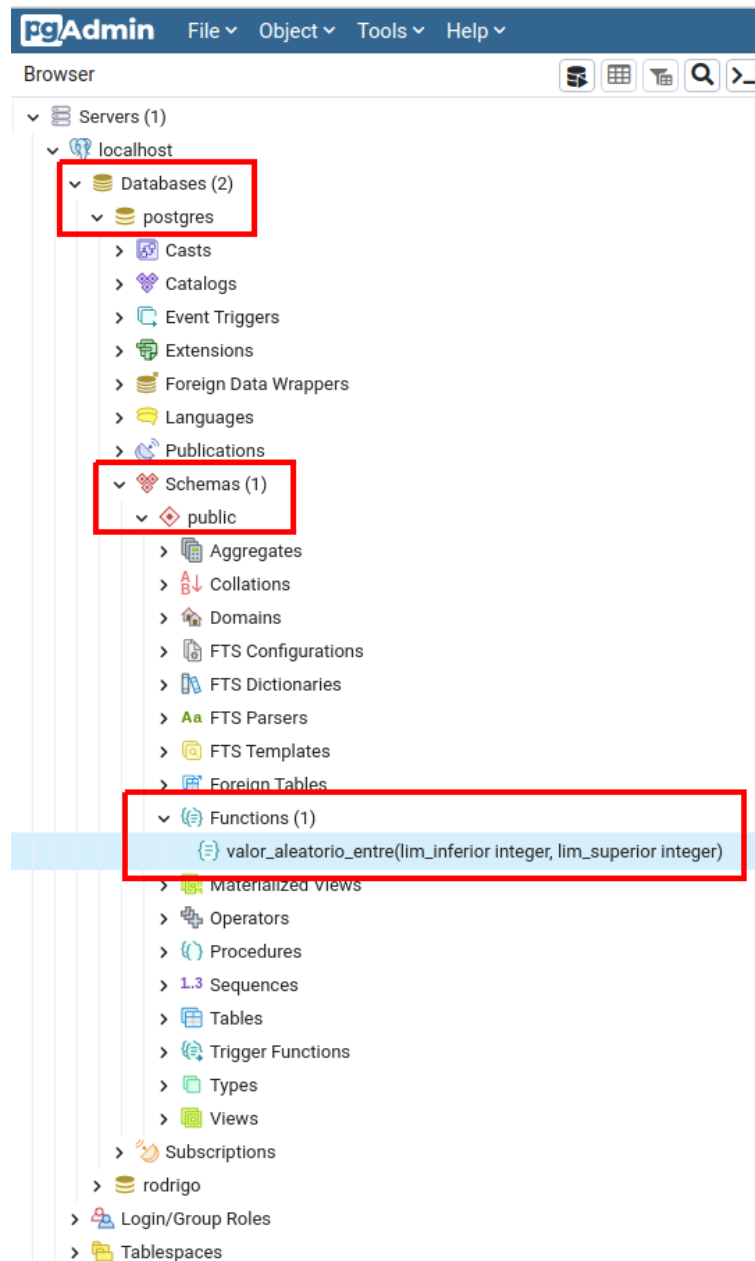
2.3 (Função para geração de valores aleatórios) Ao longo do material, faremos a geração de valores aleatórios em alguns exemplos. Para tal, utilizaremos a função dada no Bloco de Código 2.3.1.

Bloco de Código 2.3.1

```
CREATE OR REPLACE FUNCTION valor_aleatorio_entre (lim_inferior INT, lim_superior
INT) RETURNS INT AS
$$
BEGIN
    RETURN FLOOR(RANDOM() * (lim_superior - lim_inferior + 1) + lim_inferior)::INT;
END;
$$ LANGUAGE plpgsql;
```

Basta executar o bloco de código para que a função seja criada. No pgAdmin, verifique a sua existência, como mostra a Figura 2.3.1.

Figura 2.3.1



Depois de criar a função, você pode testá-la como mostra o Bloco de Código 2.3.2.

Bloco de Código 2.3.2

```
CREATE OR REPLACE FUNCTION valor_aleatorio_entre (lim_inferior INT, lim_superior
INT) RETURNS INT AS
$$
BEGIN
    RETURN FLOOR(RANDOM() * (lim_superior - lim_inferior + 1) + lim_inferior)::INT;
END;
$$ LANGUAGE plpgsql;

SELECT valor_aleatorio_entre (2, 10);
```

2.4 (Exemplos básicos de estruturas de repetição) Nesta seção, veremos alguns exemplos básicos de uso de cada estrutura de repetição de **PL/pgSQL**.

(LOOP: sem teste de continuidade) O Bloco de Código 2.4.1 mostra um exemplo de loop sem condição de continuidade especificada.

Bloco de Código 2.4.1

```
-- Observe como não é condição de continuidade
-- Estamos diante de um loop infinito
DO
$$
BEGIN
    LOOP
        RAISE NOTICE 'Teste loop simples...';
    END LOOP;
END;
$$
```

(LOOP: teste de continuidade com IF/EXIT) O Bloco de Código 2.4.2 mostra um exemplo em que controlamos o número de repetições com um contador. Seu encerramento é feito combinando-se IF e EXIT.

Bloco de Código 2.4.2

```
-- Contando de 1 a 10
-- Saída com IF/EXIT
DO
$$
DECLARE
    contador INT := 1;
BEGIN
    LOOP
        RAISE NOTICE '%', contador;
        contador := contador + 1;
        IF contador > 10 THEN
            EXIT;
        END IF;
    END LOOP;
END;$$
```

(LOOP: teste de continuidade com EXIT/WHEN) Observe, no Bloco de Código 2.4.3, como a condição de continuidade pode ser expressa combinando-se EXIT/WHEN, sem necessidade de uso do bloco IF.

Bloco de Código 2.4.3

```
-- Contando de 1 a 10
-- Saída com EXIT/WHEN
DO
$$
DECLARE
    contador INT := 1;
BEGIN
    LOOP
        RAISE NOTICE '%', contador;
        contador := contador + 1;
        EXIT WHEN contador > 10;
    END LOOP;
END;
$$
```

(LOOP: Ignorando iterações com CONTINUE) A construção CONTINUE nos possibilita ignorar iterações que tenham uma característica que podemos especificar. Veja dois exemplos no Bloco de Código 2.4.4.

Bloco de Código 2.4.4

```
DO
$$
DECLARE
    contador INT := 0;
BEGIN
    LOOP
        contador := contador + 1;
        EXIT WHEN contador > 100;
        -- ignorando iteração da vez quando contador for múltiplo de 7 com
    IF/CONTINUE
        IF contador % 7 = 0 THEN
            CONTINUE;
        END IF;
        --ignorando iteração da vez quando contador for múltiplo de 11 com
    CONTINUE WHEN
        CONTINUE WHEN contador % 11 = 0;
        RAISE NOTICE '%', contador;
    END LOOP;
END;
$$
```

(LOOP: rótulos e loops aninhados) Uma construção LOOP pode ter um rótulo associado. Ele pode ser usado pelas construções EXIT e CONTINUE, as quais funcionam da seguinte forma.

- Quando há um único LOOP, elas se referem a ele e o uso de rótulos é opcional.
- Quando há um LOOP aninhado, as construções EXIT e CONTINUE se referem ao mais interno por padrão. Podemos alterar esse funcionamento usando um rótulo.

O Bloco de Código 2.4.5 mostra um exemplo de LOOP aninhado que usa rótulos e a construção EXIT.

Bloco de Código 2.4.5

```
DO
$$
DECLARE
    i INT;
    j INT;
BEGIN
    i := 0;
    <<externo>>
    LOOP
        i := i + 1;
        EXIT WHEN i > 10;
        j := 1;
        <<interno>>
        LOOP
            RAISE NOTICE '% %', i, j;
            j := j + 1;
            EXIT WHEN j > 10;
            -- j vai contar até 5, o loop externo vai ser interrompido e o
            programa acaba
            EXIT externo WHEN j > 5;
        END LOOP;
    END LOOP;
END;
$$
```

O Bloco de Código 2.4.6 mostra um exemplo de LOOP aninhado em que rótulos e a construção CONTINUE são usados.

Bloco de Código 2.4.6

```
DO
$$
DECLARE
    i INT;
    j INT;
BEGIN
    i := 0;
    <<externo>>
    LOOP
        i := i + 1;
        EXIT WHEN i > 10;
        j := 1;
        <<interno>>
        LOOP
            RAISE NOTICE '% %', i, j;
            j := j + 1;
            EXIT WHEN j > 10;
            -- j vai contar até 5, o loop interno vai ser interrompido e
            -- prosseguimos para a próxima iteração do loop externo
            CONTINUE externo WHEN j > 5;
        END LOOP;
    END LOOP;
END;
$$
```

(WHILE: calculando a média de uma coleção de valores) A estrutura WHILE nos permite especificar uma condição de continuidade num campo apropriado para isso, sem a necessidade de se utilizar a construção EXIT. No Bloco de Código 2.4.7, fazemos o cálculo de uma coleção de notas geradas aleatoriamente. A repetição termina quando o valor gerado for igual a zero.

Bloco de Código 2.4.7

```
DO
$$
DECLARE
    nota INT;
    media NUMERIC(10, 2) := 0;
    contador INT := 0;
BEGIN
    -- inicialmente, valores de 0 a 11
    -- com o -1, temos valores de -1 a 10
    SELECT valor_aleatorio_entre (0, 11) - 1 INTO nota;
    WHILE nota >= 0 LOOP
        RAISE NOTICE '%', nota;
        media := media + nota;
        contador := contador + 1;
        SELECT valor_aleatorio_entre (0, 11) - 1 INTO nota;
    END LOOP;
    IF contador > 0 THEN
        RAISE NOTICE 'Média %.', media / contador;
    ELSE
        RAISE NOTICE 'Nenhuma nota gerada.';
    END IF;
END;
$$
```

O uso de rótulos e das construções CONTINUE e EXIT funcionam para a estrutura WHILE da mesma forma como funcionam para a estrutura LOOP.

(FOR: iterando sobre um intervalo inteiros) Esta versão da estrutura FOR nos permite especificar um intervalo de inteiros sobre o qual iterar. Veja alguns exemplos no Bloco de Código 2.4.8.

Bloco de Código 2.4.8

```
DO
$$
BEGIN
    --repare como não precisamos declarar a variável i
    -- de 1 a 10, pulando de um em um
    RAISE NOTICE 'de 1 a 10, pulando de um em um';
    FOR i IN 1..10 LOOP
        RAISE NOTICE '%', i;
    END LOOP;

    -- E agora? -- não mostra nada
    RAISE NOTICE 'E agora?';
    FOR i IN 10..1 LOOP
        RAISE NOTICE '%', i;
    END LOOP;

    -- de 10 a 1, pulando de um em um
    --repare que, usando reverse, é preciso escrever 10..1 em vez de 1..10.
    RAISE NOTICE 'de 10 a 1, pulando de um em um';
    FOR i IN REVERSE 10..1 LOOP
        RAISE NOTICE '%', i;
    END LOOP;

    -- de 1 a 50, pulando de dois em dois
    RAISE NOTICE 'de 1 a 50, pulando de dois em dois';
    FOR i IN 1..50 BY 2 LOOP
        RAISE NOTICE '%', i;
    END LOOP;

    -- de 50 a 1, pulando de dois em dois
    RAISE NOTICE 'de 50 a 1, pulando de dois em dois';
    FOR i IN REVERSE 50..1 BY 2 LOOP
        RAISE NOTICE '%', i;
    END LOOP;

END;
$$
```

(FOR: iterando sobre resultados de um select) A estrutura FOR também pode ser utilizada para iterar sobre resultados obtidos a partir de um SELECT. No Bloco de Código 2.4.9 ilustramos isso da seguinte forma.

- criamos uma tabela capaz de armazenar médias de alunos
- criamos um bloco anônimo que popula a tabela com dez valores aleatórios
- criamos um bloco anônimo que ilustra o FOR iterando sobre os resultados de um SELECT

Observe como cada linha é do tipo RECORD.

Bloco de Código 2.4.9

```
--criando a tabela
CREATE TABLE tb_aluno (
    cod_aluno SERIAL PRIMARY KEY,
    nota INT
);

-- gerando dez notas e inserindo na tabela
DO
$$
BEGIN

    ---geramos notas para 10 alunos
    FOR i in 1..10 LOOP
        INSERT INTO tb_aluno (nota) VALUES (valor_aleatorio_entre(0, 10));
    END LOOP;
END;
$$

--verificando se tudo deu certo até agora
SELECT * FROM tb_aluno;

--calculando a média com um FOR
DO
$$
DECLARE
    aluno RECORD;
    media NUMERIC(10, 2) := 0;
    total INT;
BEGIN
    FOR aluno IN
        SELECT * FROM tb_aluno
    LOOP
        RAISE NOTICE 'Nota: %', aluno.nota;
        media := media + aluno.nota;
    END LOOP;
    SELECT COUNT(*) FROM tb_aluno INTO total;
    RAISE NOTICE 'Média: %', media / total;
END;
$$
```

(FOREACH: iterando sobre valores de um array) A estrutura FOREACH nos permite iterar sobre valores de uma coleção. Para ilustrar o seu funcionamento, vamos calcular a soma dos valores armazenados em um array de 5 posições. Veja o Bloco de Código 2.4.10.

Bloco de Código 2.4.10

```
DO
$$
DECLARE
    valores INT[] := ARRAY[
        valor_aleatorio_entre(1, 10),
        valor_aleatorio_entre(1, 10),
        valor_aleatorio_entre(1, 10),
        valor_aleatorio_entre(1, 10),
        valor_aleatorio_entre(1, 10)
    ];
    valor INT;
    soma INT := 0;
BEGIN
    FOREACH valor IN ARRAY valores LOOP
        RAISE NOTICE 'Valor da vez: %', valor;
        soma := soma + valor;
    END LOOP;
    RAISE NOTICE 'Soma: %', soma;
END;
$$
```

(FOREACH: Fatias com SLICE) Usando a construção SLICE, podemos percorrer “fatias” de um array em PL/pgSQL. O valor de SLICE especificado representa o número de dimensões que desejamos que o objeto resultante tenha, a cada iteração. Ele não pode ser maior do que o número de dimensões do objeto original. Veja alguns exemplos no Bloco de Código 2.4.11.

Bloco de Código 2.4.11

```
DO
$$
DECLARE
    vetor INT[] := ARRAY[1, 2, 3];
    matriz INT[] := ARRAY[
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ];
    var_aux INT;
    vet_aux INT[];
BEGIN
    RAISE NOTICE 'SLICE %, vetor', 0;
    -- exemplo sem slice com vetor
    FOREACH var_aux IN ARRAY vetor LOOP
        RAISE NOTICE '%', var_aux;
    END LOOP;

    --exemplo com slice igual a 1, com vetor
    --observe que a variável deve ser um vetor
    --com slice igual a 1, pegamos o vetor inteiro
    RAISE NOTICE 'SLICE %, vetor', 1;
    FOREACH vet_aux SLICE 1 IN ARRAY vetor LOOP
        RAISE NOTICE '%', vet_aux;
        --podemos percorrer vet_aux
        FOREACH var_aux IN ARRAY vet_aux LOOP
            RAISE NOTICE '%', var_aux;
        END LOOP;
    END LOOP;

    --exemplo com slice igual a 0, com matriz
    RAISE NOTICE 'SLICE %, matriz', 0;
    FOREACH var_aux IN ARRAY matriz LOOP
        RAISE NOTICE '%', var_aux;
    END LOOP;

    --exemplo com slice igual a 1, com matriz
    --com slice igual a 1, pegamos um vetor (linha) por vez
    RAISE NOTICE 'SLICE %, matriz', 1;
    FOREACH vet_aux SLICE 1 IN ARRAY matriz LOOP
        RAISE NOTICE '%', vet_aux;
    END LOOP;

    --exemplo com slice igual a 2, com matriz
    --com slice igual a 2, pegamos a matriz inteira numa única iteração
    RAISE NOTICE 'SLICE %, matriz', 2;
    FOREACH vet_aux SLICE 2 IN ARRAY matriz LOOP
        RAISE NOTICE '%', vet_aux;
    END LOOP;
END;
$$
```


(2.5 Manipulação de erros) Por padrão, quando um erro acontece, o processamento da função atual é interrompido. Podemos especificar um bloco de código a ser executado quando um erro acontecer com a construção `EXCEPTION`.

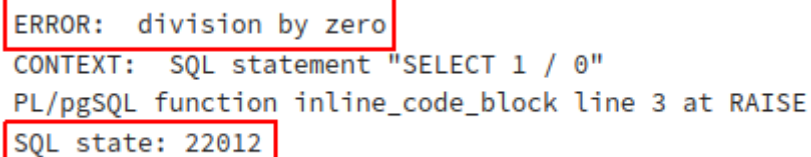
(EXCEPTION: Exemplo de divisão por zero) A divisão com divisor igual a zero causa um erro do tipo `division_by_zero`. Veja um programa que faz com que ele aconteça no Bloco de Código 2.5.1.

Bloco de Código 2.5.1

```
DO
$$
BEGIN
    RAISE NOTICE '%', 1 / 0;
END;
$$
```

O resultado esperado aparece na Figura 2.5.1.

Figura 2.5.1



The screenshot shows a PostgreSQL error message with the following text: `ERROR: division by zero`, `CONTEXT: SQL statement "SELECT 1 / 0"`, `PL/pgSQL function inline_code_block line 3 at RAISE`, and `SQL state: 22012`. The first line and the last line are highlighted with red boxes.

Observe como cada erro possui um código associado. Muitos deles são previstos no padrão SQL. Outros são específicos do PostgreSQL. Veja uma lista completa no Link 2.5.1.

Link 2.5.1

<https://www.postgresql.org/docs/current/errcodes-appendix.html>

(EXCEPTION: Tratando explicitamente) O Bloco de Código 2.5.2 mostra como tratar erros especificando um bloco próprio para isso.

Bloco de Código 2.5.2

```
DO
$$
BEGIN
    RAISE NOTICE '%', 1 / 0;
EXCEPTION
    WHEN division_by_zero THEN
        RAISE NOTICE 'Não é possível fazer divisão por zero.';
END;
$$
```

(EXCEPTION: Gerando exceções explicitamente) Observe, no Bloco de Código 2.5.3, como podemos gerar exceções explicitamente.

Bloco de Código 2.5.3

```
DO
$$
DECLARE
    a INT := valor_aleatorio_entre(0, 5);
BEGIN
    IF a = 0 THEN
        RAISE 'a não pode ser zero';
    ELSE
        RAISE NOTICE 'Valor de a: %', a;
    END IF;
EXCEPTION WHEN OTHERS THEN
    --SQLState é o código da Exceção
    --SQLERRM é a mensagem (SQLERRM: SQL Error Message)
    RAISE NOTICE 'exceção: %, %', SQLSTATE, SQLERRM;
END;
```

Bibliografia

LOPES, A.; GARCIA, G..**Introdução à Programação - 500 Algoritmos Resolvidos**. 1a Ed., Elsevier, 2002.

PostgreSQL: Documentation: 14: PostgreSQL 14.2 Documentation. PostgreSQL, 2022. Disponível em <<https://www.postgresql.org/docs/current/index.html>>. Acesso em abril de 2022.