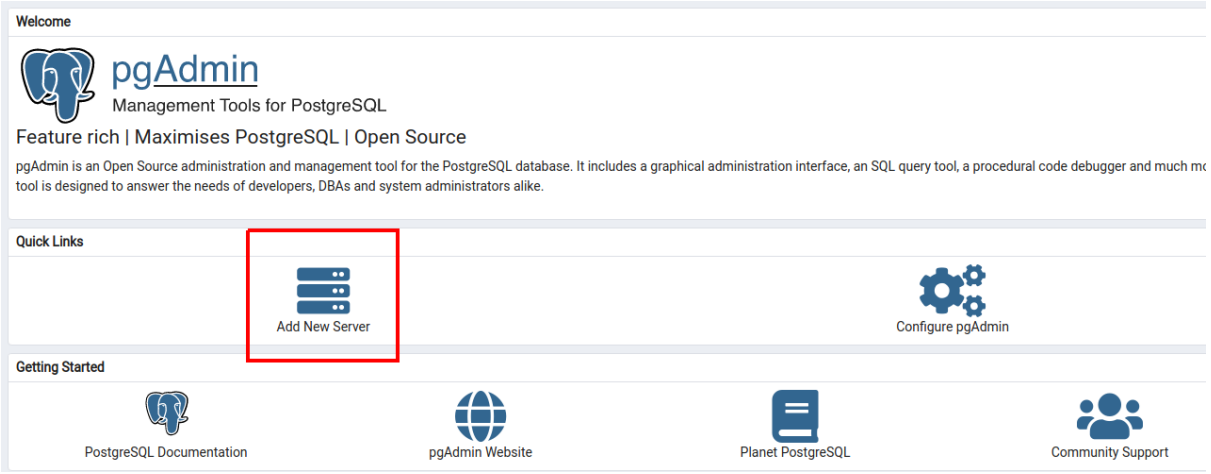
1 Introdução

Neste material, estudaremos os tipos de blocos que a linguagem **PL/pgSQL** permite utilizar.

2 Passo a passo

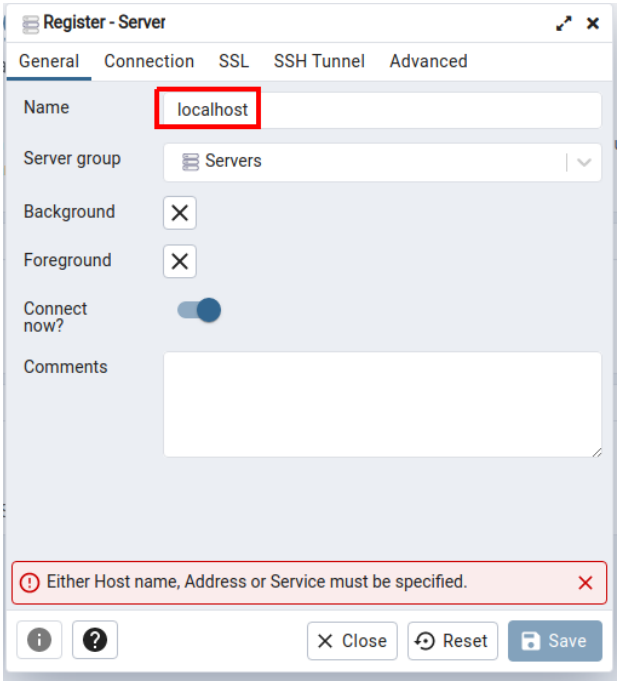
2.1 (Criando um servidor) Caso ainda não possua um servidor, abra o pgAdmin4 e clique em **Add New Server**, como mostra a Figura 2.1.1.

Figura 2.1.1



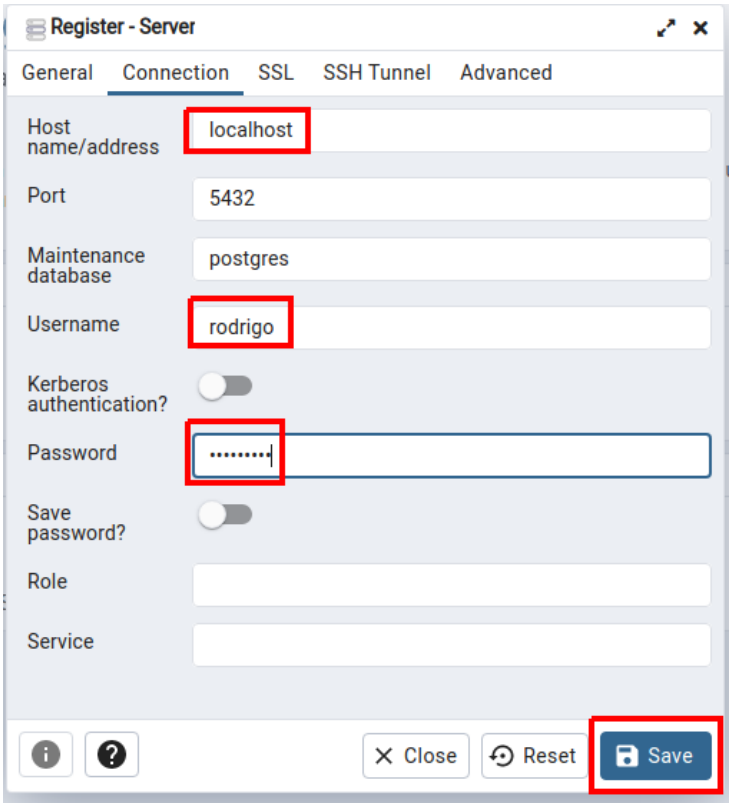
O nome do servidor pode ser algo que lhe ajude a lembrar a razão de ser dele. Como é um servidor que está executando localmente, podemos chamá-lo de algo como **localhost**, como na Figura 2.1.2. Depois de preencher o nome, clique na aba **Connection**.

Figura 2.1.2



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

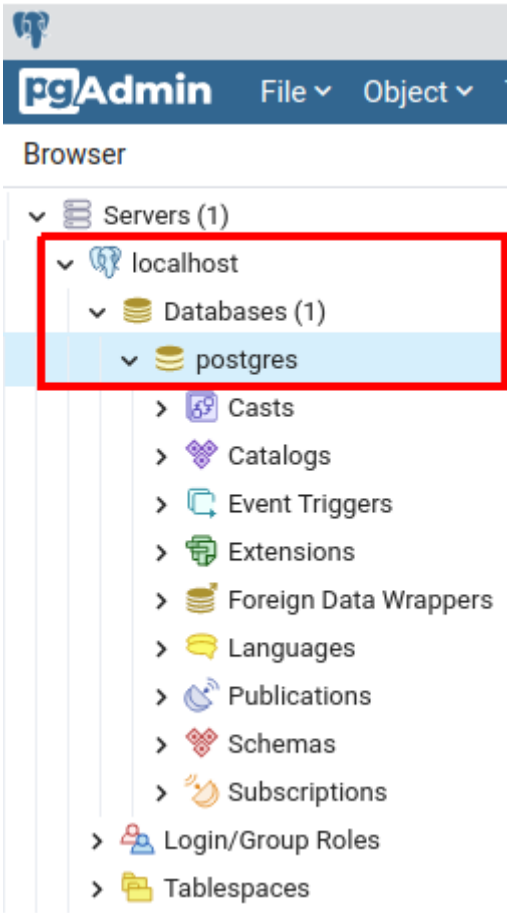
Figura 2.1.3



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.

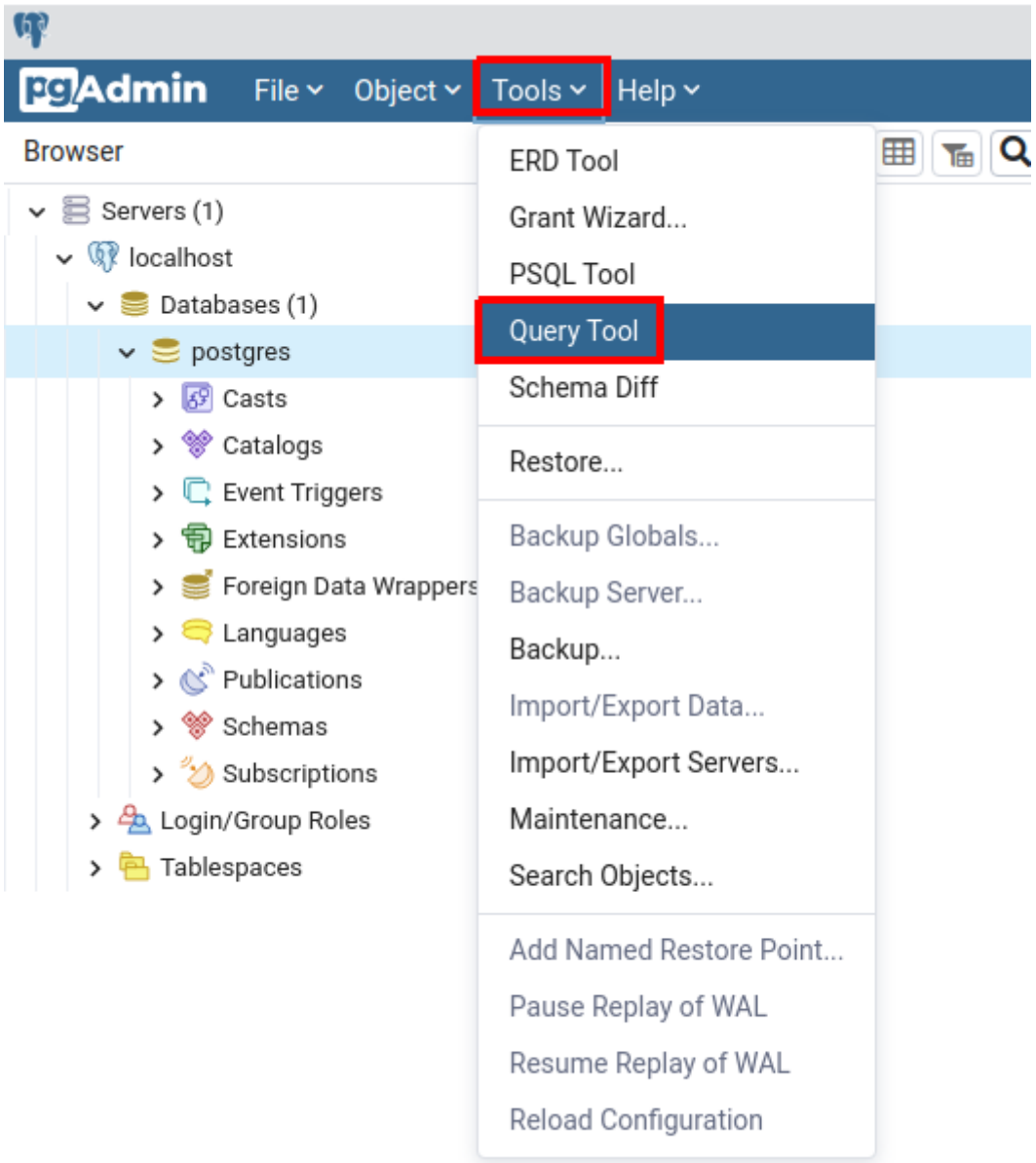
No canto superior esquerdo, encontre o seu servidor e clique sobre ele. Expanda **Databases** e encontre o database chamado **postgres**, cuja existência é muito comum. Veja a Figura 2.1.4.

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 (Blocos anônimos) A linguagem **PL/pgSQL** permite que criemos blocos de código que nada mais são do que pequenos programas de computador. Além de utilizar o padrão SQL, também é possível utilizar diversos recursos que a linguagem oferece que não fazem parte do padrão. Um bloco anônimo tem a estrutura exibida pelo Bloco de Código 2.2.1.

Bloco de Código 2.2.1

```
-- colchetes indicam que a região é opcional
[ rótulo ] -- o rótulo pode ser usado para qualificar variáveis, por exemplo
[
  DECLARE declarações de variáveis aqui
]

BEGIN
  comandos aqui;
END [ rótulo ];
```

O Bloco de Código 2.2.2 exibe um primeiro bloco de código anônimo. Observe que ele ainda não está pronto para ser executado.

Bloco de Código 2.2.2

```
BEGIN
   --para exibir valores no console
   RAISE NOTICE 'Meu primeiro Bloco anônimo!!';
END;
```

(Delimitando com aspas simples) Blocos anônimos devem ser delimitados por aspas simples. Assim, aspas internas devem ser “duplicadas”. Veja o Bloco de Código 2.2.3.

Bloco de Código 2.2.3

```
'
BEGIN
   --para exibir valores no console
   RAISE NOTICE ''Meu primeiro Bloco anônimo!!'';
END;
'
```

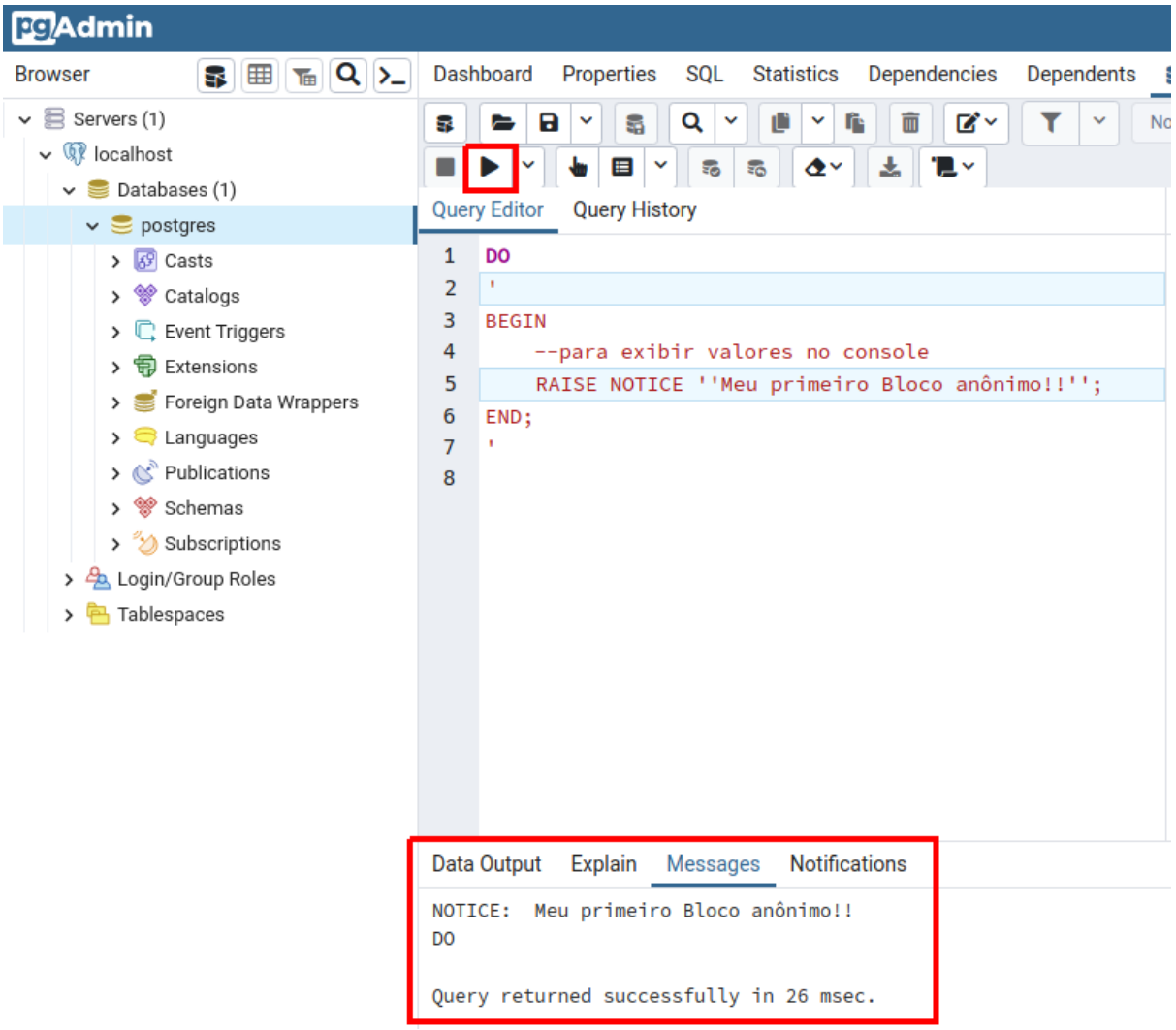
(Executando com DO) Repare que o bloco ainda não pode ser executado. A sua execução pode ser feita aplicando-se o comando **DO**. Veja o Bloco de Código 2.2.4.

Bloco de Código 2.2.4

```
DO
'
BEGIN
   --para exibir valores no console
   RAISE NOTICE ''Meu primeiro Bloco anônimo!!'';
END;
'
```

No pgAdmin, clique no botão de execução para ver o resultado. Veja a Figura 2.2.1.

Figura 2.2.1



Ter de fazer o “escape” das aspas pode ser trabalhoso e inconveniente. Em geral, é mais comum utilizar o símbolo $$ para delimitar strings que representam blocos de código. O programa anterior pode ser, portanto, escrito como mostra o Bloco de Código 2.2.5.

Bloco de Código 2.2.5

```
DO
$$
BEGIN
   --para exibir valores no console
   RAISE NOTICE 'Meu primeiro Bloco anônimo!!';
END;
$$
```

2.3 (Placeholder de expressões em strings) É possível "montar" strings escrevendo modelos que consistem de partes fixas de interesse e do símbolo **%** em cada ponto em que desejamos encaixar um valor resultante de uma expressão. Veja o Bloco de Código 2.3.1.

Bloco de Código 2.3.1

```
DO
$$
BEGIN
   RAISE NOTICE '% + % = %', 2, 2, 2 + 2;
END;
$$
```

2.4 (Variáveis) A declaração de variáveis envolve duas partes: tipo e nome. Veja alguns exemplos no Bloco de Código 2.4.1.

Bloco de Código 2.4.1

```
DO $$
DECLARE
   codigo INTEGER := 1;
   nome_completo VARCHAR(200) := 'João Santos';
   -- 11 digitos no total, dois para valores decimais
   salario numeric (11, 2) := 20.5 ;
BEGIN
   RAISE NOTICE 'Meu código é %, me chamo % e meu salário é R$%',
codigo, nome_completo, salario;
END $$;
```

2.5 (Operadores aritméticos) A linguagem PL/pgSQL define diversos operadores aritméticos. Veja alguns dos principais na Tabela 2.5.1.

Tabela 2.5.1

Operação	Operador	Exemplo	Resultado
Soma	+	2 + 2	4
+ unário	+	+ 2	2
Subtração	-	2 - 2	0
- unitário	-	-5	-5
Multiplicação	*	2 * 2	4
Divisão inteira (trunca)	/	5 / 2	2
Divisão real	/	5 / 2	2.5
Módulo	%	5 % 2	1
Exponenciação	^	2 ^ 3	8
Raiz quadrada	|/	|/25	5
Raiz cúbica	||/	||/8	2
Valor absoluto	@	@-5	5

O Bloco de Código 2.5.1 mostra alguns exemplos.

Bloco de Código 2.5.1

```
DO $$
DECLARE
   n1 INTEGER := 5;
   n2 INTEGER := 2;
   n3 NUMERIC(5, 2) := 5;
   n4 INTEGER := -5;
BEGIN
   -- adição
   RAISE NOTICE '% + % = %', n1, n2, n1 + n2;
   -- + unário: sem efeito
   RAISE NOTICE '%', +n1;
   -- subtração
   RAISE NOTICE '% - % = %', n1, n2, n1 - n2;
   -- - uniário: negação
   RAISE NOTICE '%', -n1;
   -- multiplicação
   RAISE NOTICE '% * % = %', n1, n2, n1 * n2;
   -- divisão (para inteiros, trunca o resultado em direção ao zero)
   RAISE NOTICE '% / % = %', n1, n2, n1 / n2;
   -- divisão (se envolve um real, a divisão é real)
   RAISE NOTICE '% / % = %', n3, n2, n3 / n2;
   -- divisão (formatando) Veja:
https://www.postgresql.org/docs/current/functions-formatting.html
   RAISE NOTICE '% / % = %', n3, n2, to_char(n3 / n2, '99.99');
   -- resto da divisão
   -- usamos %% para escapar um %
   RAISE NOTICE '% %% % = %', n1, n2, n1 % n2;
   -- exponenciação
   RAISE NOTICE '% ^ % = %', n1, n2, n1 ^ n2;
   -- raiz quadrada
   RAISE NOTICE '|/ % = %', n1, |/ n1;
   -- raiz cubica
   RAISE NOTICE '||/ % = %', n1, ||/ n1;
   -- valor absoluto
   RAISE NOTICE '@% = % e @% = %', n1, @n1, n4, @n4;
END $$;
```

2.6 (Valores aleatórios) A função **random** produz um valor real 0 <= **n** < 1. Com algumas operações aritméticas, podemos obter resultados interessantes. Veja o Bloco de Código 2.6.1.

Bloco de Código 2.6.1

```
DO $$
DECLARE
   n1 NUMERIC (5, 2);
   n2 INTEGER;
   limite_inferior INTEGER := 5;
   limite_superior INTEGER := 17;
BEGIN
   -- 0 <= n1 < 1 (real)
   n1 := random();
   RAISE NOTICE '%', n1;
   -- 1 <= n1 < 10 (real)
   n1 := random() * 10 + 1;
   RAISE NOTICE '%', n1;
   -- 1 <= n2 <10 (:: faz type cast) (floor arredonda para baixo)
   n2 := floor(random() * 10 + 1)::int;
   RAISE NOTICE '%', n2;
   -- limite_inferior <= n2 <= limite_superior
   n2 := floor(random() * (limite_superior - limite_inferior + 1) +
limite_inferior)::int;
   RAISE NOTICE '%', n2;
END $$;
```