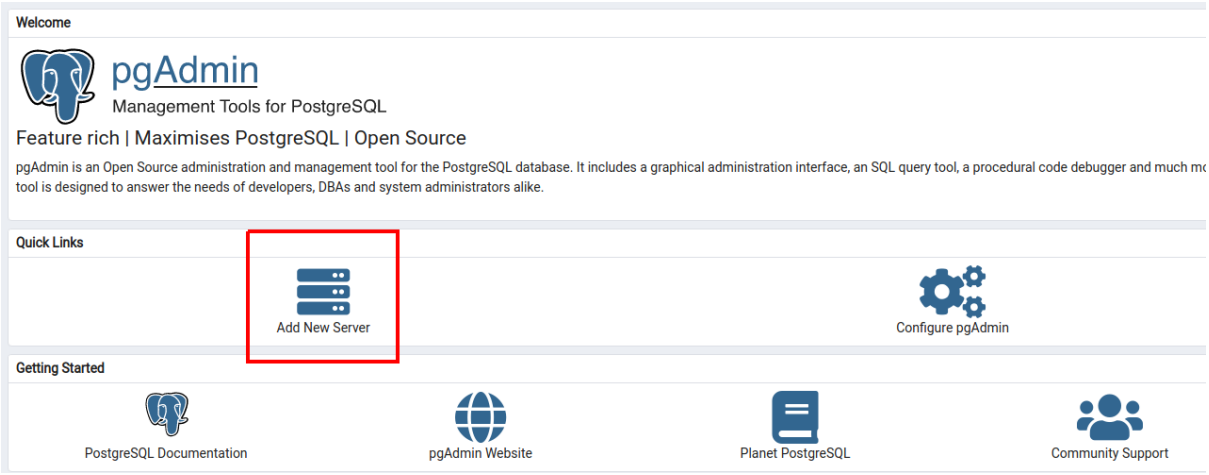
1 Introdução

Neste material, estudaremos as estruturas de seleção presentes na linguagem **PL/pgSQL**.

2 Passo a passo

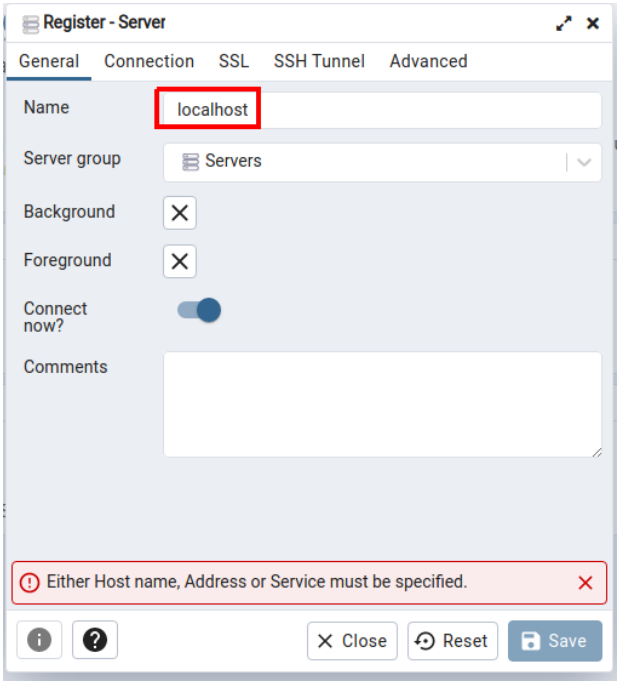
2.1 (Criando um servidor) Caso ainda não possua um servidor, abra o pgAdmin4 e clique em **Add New Server**, como mostra a Figura 2.1.1.

Figura 2.1.1



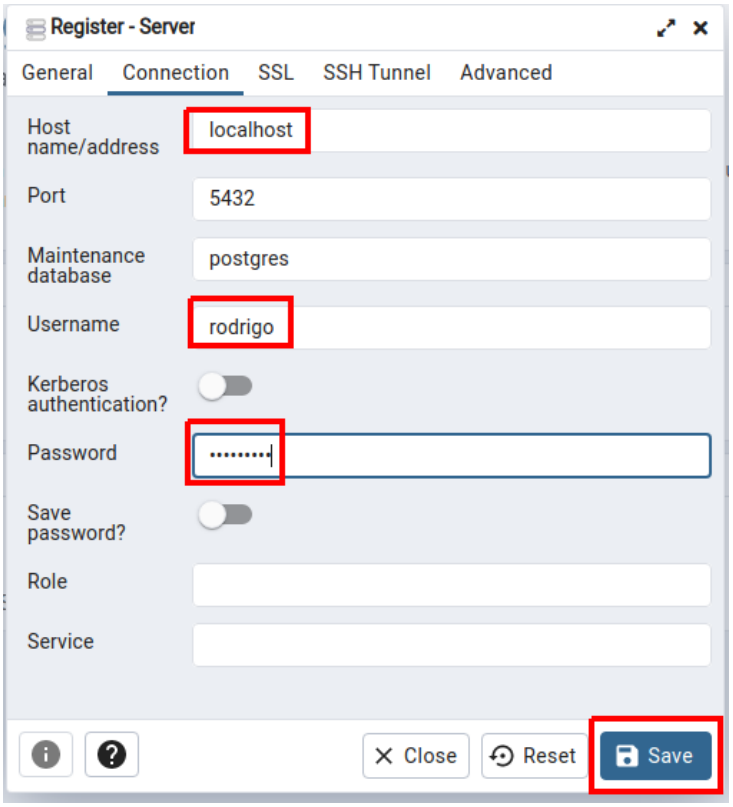
O nome do servidor pode ser algo que lhe ajude a lembrar a razão de ser dele. Como é um servidor que está executando localmente, podemos chamá-lo de algo como **localhost**, como na Figura 2.1.2. Depois de preencher o nome, clique na aba **Connection**.

Figura 2.1.2



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

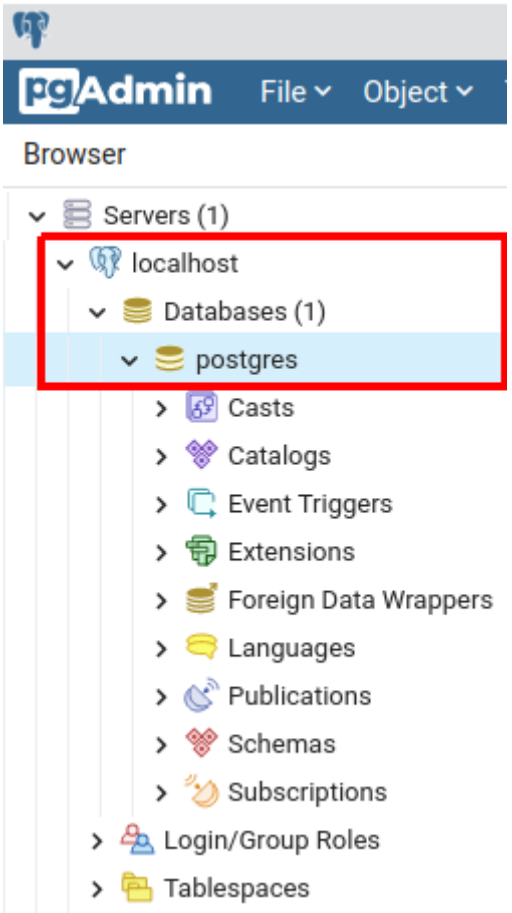
Figura 2.1.3



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.

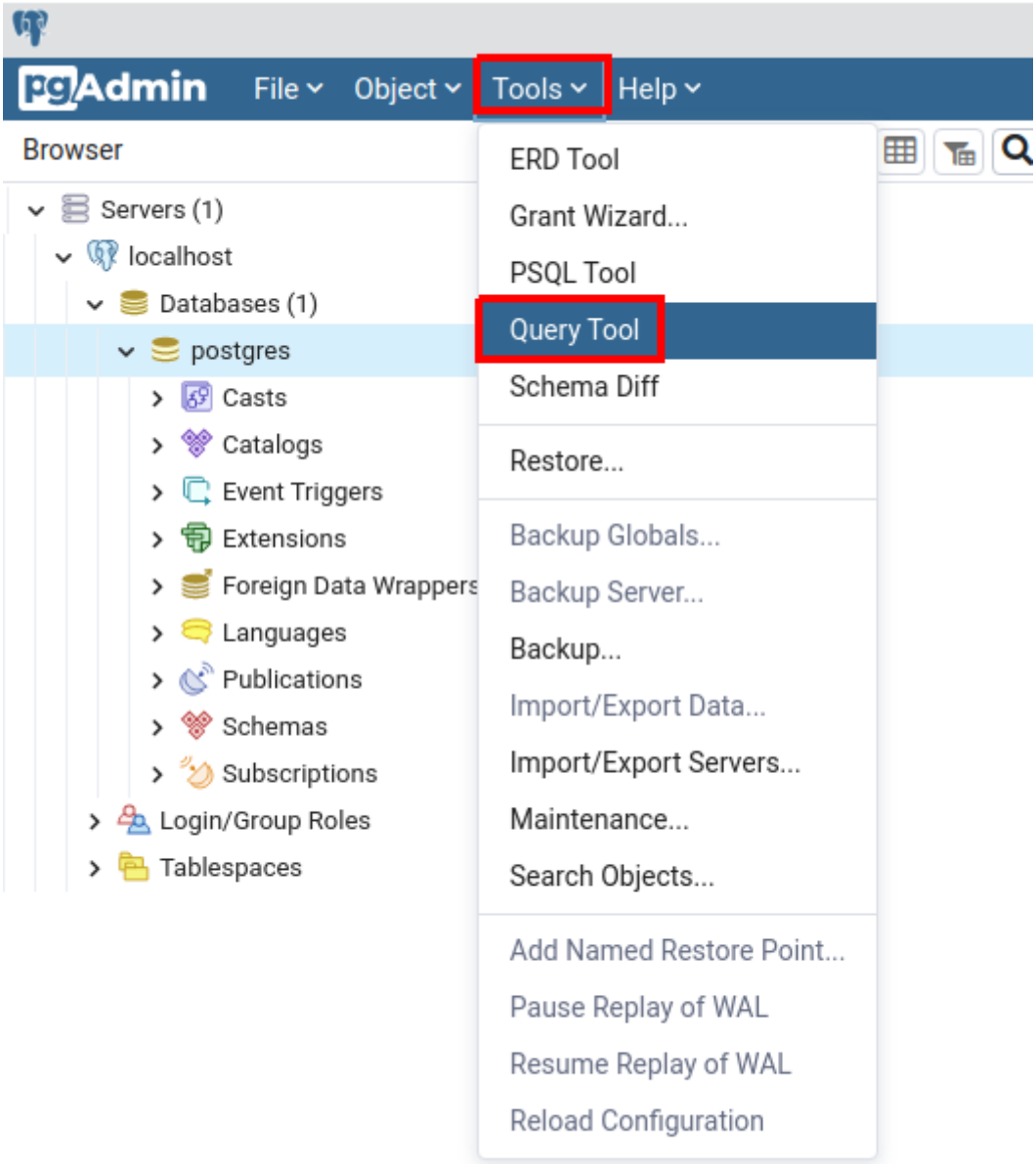
No canto superior esquerdo, encontre o seu servidor e clique sobre ele. Expanda **Databases** e encontre o database chamado **postgres**, cuja existência é muito comum. Veja a Figura 2.1.4.

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 (Operadores lógicos) Os operadores lógicos presentes na linguagem PL/pgSQL são os seguintes.

- AND
- OR
- NOT

A Tabela 2.2.1, que é uma tabela-verdade, relembra o funcionamento deles.

Tabela 2.2.1

A	B	A AND B	A OR B	NOT A	NOT B
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE

2.3 (Operadores relacionais) A Tabela 2.3.1 mostra os operadores relacionais de PL/pgSQL e alguns exemplos.

Operador	Signicado	Exemplo	Resultado
<	Menor	1 < 2	TRUE
>	Maior	1 > 2	FALSE
<=	Menor ou igual	1 <= 2	TRUE
>=	Maior ou igual	2 >= 2	TRUE
=	Igual	2 = 3	FALSE
<>	Diferente	2 <> 3	TRUE
!=	Diferente	2 != 3	TRUE

Nota. O operador “!=” é um **pseudônimo** (nome fictício) de <>. Durante a compilação, ele é substituído por <>.

Nota. Todos os operadores relacionais são binários e devolvem um valor booleano. Como não é possível comparar um valor booleano com um valor inteiro, expressões como

1 < 2 < 3

não são válidas.

Quando envolvem NULL, todos os operadores relacionais devolvem NULL. Veja a Tabela 2.3.2.

Tabela 2.3.2

Operador	Exemplo	Resultado
<	1 < NULL	NULL
>	NULL > 2	NULL
<=	NULL <= 2	NULL
>=	NULL >= 3	NULL
=	NULL = 1	NULL
<>	NULL <> 1	NULL
!=	NULL != 3	NULL
<	NULL < NULL	NULL
>	NULL > NULL	NULL
<=	NULL <= NULL	NULL
>=	NULL >= NULL	NULL
=	NULL = NULL	NULL
<>	NULL <> NULL	NULL
!=	NULL != NULL	NULL

Nota. NULL representa um valor desconhecido. Por isso o resultado é também desconhecido.

Também é possível comparar **valores booleanos**. Veja a Tabela 2.3.3.

Tabela 2.3.3

A	B	A < B	A > B	A <= B	A >= B	A = B	A <> B	A != B
TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE

A Tabela 2.3.4 mostra o funcionamento da **comparação entre strings**.

A	B	A < B	A > B	A <= B	A >= B	A = B	A <> B	A != B
bola	carro	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
a10	a9	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
a	A	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
a	a	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE

2.4 (Predicados de comparação) A linguagem PL/pgSQL conta com os predicados de comparação ilustrados na Tabela 2.4.1.

Tabela 2.4.1

Predicado	Exemplo	Resultado	Observação
BETWEEN	2 BETWEEN 1 AND 3	TRUE	
BETWEEN	2 BETWEEN 1 AND 2	TRUE	Intervalo fechado
NOT BETWEEN	2 NOT BETWEEN 1 AND 3	FALSE	
NOT BETWEEN	2 NOT BETWEEN 1 AND 2	FALSE	
BETWEEN	2 BETWEEN 3 AND 1	FALSE	
BETWEEN SYMMETRIC	2 BETWEEN SYMMETRIC 3 AND 1	TRUE	ordena os dois limites
NOT BETWEEN SYMMETRIC	2 NOT BETWEEN SYMMETRIC 3 AND 1	FALSE	
IS DISTINCT FROM	2 IS DISTINCT FROM NULL	TRUE	Tratando null como algo comparável. Resultado é booleano e não NULL.
IS DISTINCT FROM	NULL IS DISTINCT FROM NULL	FALSE	
IS NOT DISTINCT FROM	2 IS NOT DISTINCT FROM NULL	FALSE	

IS NOT DISTINCT FROM	NULL IS NOT DISTINCT FROM NULL	TRUE	
IS NULL	2 IS NULL	FALSE	
IS NULL	NULL IS NULL	TRUE	
IS NOT NULL	2 IS NOT NULL	TRUE	
IS NOT NULL	NULL IS NOT NULL	FALSE	
ISNULL	2 ISNULL	FALSE	
ISNULL	NULL ISNULL	TRUE	
NOTNULL	2 NOTNULL	TRUE	
NULL	NULL NOTNULL	FALSE	

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

- IF THEN
- IF THEN ELSE
- IF THEN ELSIF THEN ELSE
- CASE valor WHEN valor THEN ELSE
- CASE WHEN THEN ELSE

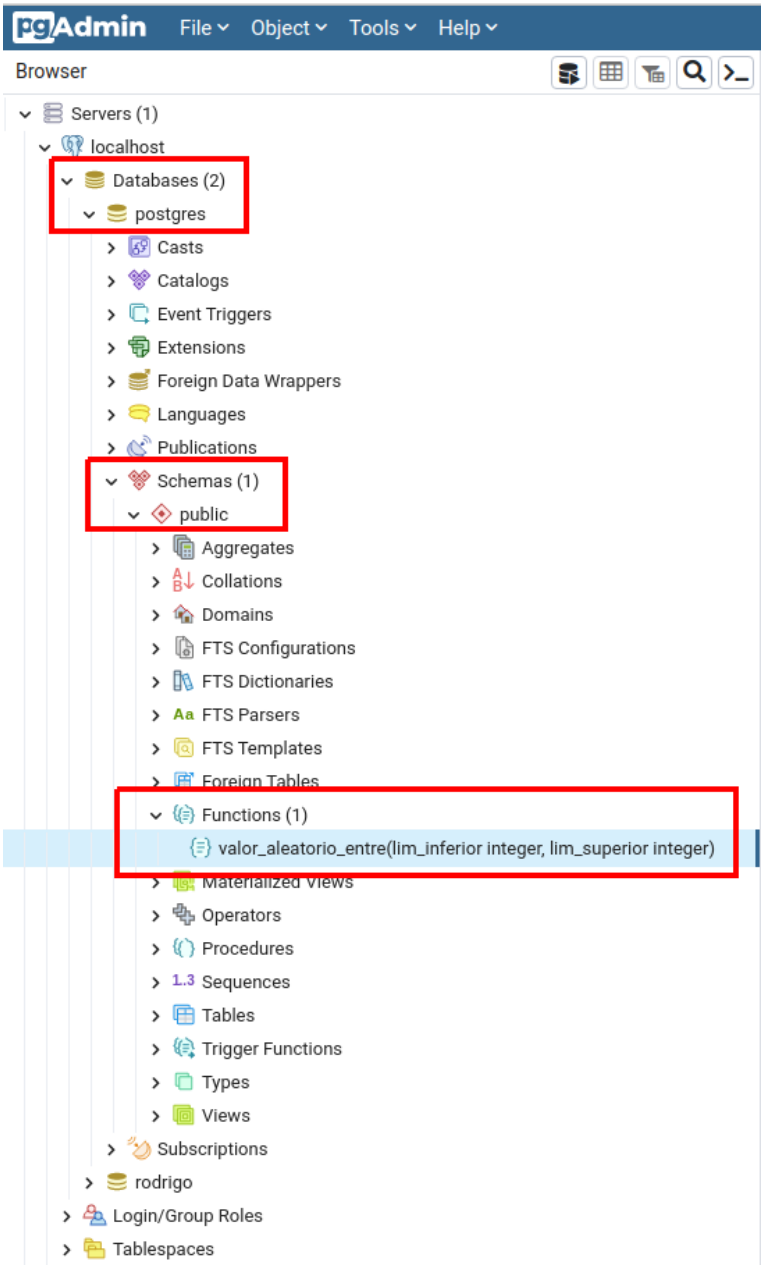
Vejamos alguns exercícios resolvidos. Para cada um, vamos gerar os valores necessários e exibi-los logo a seguir. Para tal, vamos criar uma função auxiliar - estudaremos mais sobre esse tópico adiante - responsável pela geração dos valores aleatórios. Veja o Bloco de Código 2.4.1.

Bloco de Código 2.4.1

```
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 2.4.1.

Figura 2.4.1



Depois de criar a função, você pode testá-la como mostra o Bloco de Código 2.4.2.

Bloco de Código 2.4.2

```
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);
```

(IF: Exercício resolvido) Dado um número inteiro, exiba metade de seu valor caso seja maior do que 20. Os blocos de código 2.4.3 e 2.4.4 mostram duas possíveis soluções.

Bloco de Código 2.4.3

```
DO $$
DECLARE
        valor INT;
BEGIN
        valor := valor_aleatorio_entre(1, 100);
        RAISE NOTICE 'O valor gerado é: %', valor;
        IF valor <= 20 THEN
                RAISE NOTICE 'A metade do valor % é %', valor, valor / 2::FLOAT;
        END IF;
END;
$$
```

Bloco de Código 2.4.4

```
DO $$
DECLARE
        valor INT;
BEGIN
        SELECT valor_aleatorio_entre(1, 100) INTO valor;
        RAISE NOTICE 'O valor gerado é: %', valor;
        IF valor BETWEEN 1 AND 20 THEN
                RAISE NOTICE 'A metade do valor % é %', valor, valor / 2.;
        END IF;
END;
$$
```

(IF/ELSE: Exercício resolvido) Dado um número inteiro, exiba se ele é par ou ímpar. Veja uma solução no Bloco de Código 2.4.5.

Bloco de Código 2.4.5

```
DO $$
DECLARE
	valor INT := valor_aleatorio_entre(1, 100);
BEGIN
	RAISE NOTICE 'O valor gerado é: %', valor;
	IF valor % 2 = 0 THEN
		RAISE NOTICE '% é par', valor;
	ELSE
		RAISE NOTICE '% é ímpar', valor;
	END IF;
END;
$$
```

(IF/ELSIF/ELSE: Exercício resolvido) Dados valores a, b e c desempenhando o papel de coeficientes de uma potencial equação do segundo grau, calcule as potenciais raízes. Considere que qualquer um dos coeficientes pode ser igual a zero. O Bloco de Código 2.4.6 mostra uma possível solução.

Bloco de Código 2.4.6

```
DO $$
DECLARE
	a INT := valor_aleatorio_entre(0, 20);
	b INT := valor_aleatorio_entre(0, 20);
	c INT := valor_aleatorio_entre(0, 20);
	delta NUMERIC(10,2);
	raizUm NUMERIC(10, 2);
	raizDois NUMERIC(10, 2);
BEGIN
	--U& precedendo uma string indica que podemos especificar símbolos unicode
	RAISE NOTICE 'Equação: %x% + %x + % = 0', a, U&'\00B2', b, c;
	IF a = 0 THEN
		RAISE NOTICE 'Não é uma equação do segundo grau';
	ELSE
		-- observe que podemos omitir * para multiplicação
		delta := b ^ 2 - 4 * a * c;
		RAISE NOTICE 'Valor de delta: %', delta;
		IF delta < 0 THEN
			RAISE NOTICE 'Nenhum raiz.';
		-- ELSIF pode ser ELSEIF também
		ELSIF delta = 0 THEN
			raizUm := (-b + |/delta) / 2a;
			RAISE NOTICE 'Uma raiz: %', raizUm;
		ELSE
			raizUm := (-b + |/delta) / 2a;
			raizDois := (-b - |/delta) / 2a;
			RAISE NOTICE 'Duas raizes: % e %', raizUm, raizDois;
		END IF;
	END IF;
END;
$$
```

(CASE valor WHEN valor THEN ELSE: Exercício resolvido) Dado um valor entre 1 e 10, decidir se ele é par ou ímpar. Para tal, use a estrutura CASE valor WHEN valor THEN ELSE. O Bloco de Código 2.4.7 mostra uma solução. É claro que podemos fazer algoritmos bem mais inteligentes. Esta primeira versão tem como finalidade estruturas as características básicas do CASE.

Bloco de Código 2.4.7

```
DO $$
DECLARE
	valor INT;
	mensagem VARCHAR(200);
BEGIN
	--vamos admitir alguns valores fora do intervalo para ver o que acontece quando
não há case previsto
	valor := valor_aleatorio_entre (1, 12);
	RAISE NOTICE 'O valor gerado é: %', valor;
	CASE valor
		WHEN 1 THEN
			mensagem := 'Ímpar';
		WHEN 3 THEN
			mensagem := 'Ímpar';
		WHEN 5 THEN
			mensagem := 'Ímpar';
		WHEN 7 THEN
			mensagem := 'Ímpar';
		WHEN 9 THEN
			mensagem := 'Ímpar';
		WHEN 2 THEN
			mensagem := 'Par';
		WHEN 4 THEN
			mensagem := 'Par';
		WHEN 6 THEN
			mensagem := 'Par';
		WHEN 8 THEN
			mensagem := 'Par';
		WHEN 10 THEN
			mensagem := 'Par';
		--comente o ELSE e veja o resultado quando não houver case para o valor:
Exceção CASE_NOT_FOUND
		ELSE
			mensagem := 'Valor fora do intervalo';
	END CASE;
	RAISE NOTICE '%', mensagem;
END;
$$
```

O Bloco de Código 2.4.8 mostra outra possibilidade, em que agrupamos os valores que têm tratamento igual.

Bloco de Código 2.4.8

```
DO $$
DECLARE
	valor INT := valor_aleatorio_entre(1, 12);
	mensagem VARCHAR(200);
BEGIN
	RAISE NOTICE 'O valor gerado é: %', valor;
	CASE valor
		WHEN 1, 3, 5, 7, 9 THEN
			mensagem := 'Ímpar';
		WHEN 2, 4, 6, 8, 10 THEN
			mensagem := 'Par';
		ELSE
			mensagem := 'Fora do intervalo';
	END CASE;
	RAISE NOTICE '%', mensagem;
END;
$$
```

(CASE WHEN THEN ELSE: Exercício resolvido) Dado um valor entre 1 e 10, decidir se ele é par ou ímpar. Use CASE WHEN THEN ELSE. O Bloco de Código 2.4.9 mostra um exemplo. Observe que estamos usando um CASE aninhado.

Bloco de Código 2.4.9

```
DO $$
DECLARE
	valor INT := valor_aleatorio_entre (1, 12);
BEGIN
	RAISE NOTICE 'O valor gerado é: %', valor;
	CASE
		WHEN valor BETWEEN 1 AND 10 THEN
			CASE
				WHEN valor % 2 = 0 THEN
					RAISE NOTICE 'Par';
				ELSE
					RAISE NOTICE 'Ímpar';
			END CASE;
		ELSE
			RAISE NOTICE 'Fora do intervalo';
	END CASE;
END;
$$
```

(Exercício resolvido) Dado valor no formato **ddmmaaaa**, verificar se ele representa uma data válida. O Bloco de Código 2.4.10 mostra uma solução.

Bloco de Código 2.4.10

```
DO $$
DECLARE
	--testar
	--22/10/2022: valida
	--29/02/2020: 2020 é bissexto, válida
	--29/02/2021: inválida
	--28/02/2021: válida
	--31/06/2021: inválida

	data INT := 31062021;
	dia INT;
	mes INT;
	ano INT;
	data_valida BOOL := TRUE;
BEGIN
	dia := data / 1000000;
	mes := data % 1000000 / 10000;
	ano := data % 10000;
	RAISE NOTICE 'A data é %/%/%', dia, mes, ano;
	RAISE NOTICE 'Vejamos se é ela é válida...';
	IF ano >= 1 THEN
		CASE
			WHEN mes > 12 OR mes < 1 OR dia < 1 OR dia > 31 THEN
				data_valida := FALSE;
			ELSE
				--abril, junho, setembro e novembro não podem ter mais de 30 dias
				IF ((mes = 4 OR mes = 6 OR mes = 9 OR mes = 11) AND dia > 30) THEN
					data_valida := FALSE;
				ELSE
					--fevereiro
					IF mes = 2 THEN
						CASE
							--se o ano for bissexto
							WHEN ((ano % 4 = 0 AND ano % 100 <> 0)
OR ANO % 400 = 0) THEN
								IF dia > 29 THEN
									data_valida := FALSE;
								END IF;

							ELSE
								IF dia > 28 THEN
									data_valida := FALSE;
								END IF;
						END CASE;
					END IF;

				END IF;
		END CASE;

	ELSE
		data_valida := FALSE;
	END IF;

	CASE
		WHEN data_valida THEN
			RAISE NOTICE 'Data válida';
		ELSE
			RAISE NOTICE 'Data inválida';
	END CASE;
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.