

Banco de Dados 2

02 - Funções PLPGSQL (PostgreSQL)

Funções PLPGSQL

- A linguagem procedural PL/pgSQL adiciona muitos elementos procedurais, como **estruturas de controle**, **loops** e **cálculos complexos**, para estender o SQL padrão. Ela permite desenvolver funções complexas e procedimentos armazenados em **PostgreSQL** que **talvez não fossem possíveis usando SQL simples**.
- A linguagem procedural **PL/pgSQL** é semelhante à **Oracle PL/SQL**.
- O **PL/pgSQL** foi projetado para:
 - Criar funções definidas pelo usuário, procedimentos armazenados (Stored Procedure) e gatilhos (triggers).
 - Amplia o SQL padrão adicionando **estruturas de controle** como instruções if-else, case e loop.
 - Herda todas as funções, operadores e tipos definidos pelo usuário.

Funções PL/PGSQL

- **Vantagens de usar PL/pgSQL:**

- **SQL** é uma linguagem de consulta que permite gerenciar dados no banco de dados de forma eficaz. No entanto, o **PostgreSQL** só pode executar instruções SQL individualmente.
- Isso significa que você tem várias instruções e precisa executá-las uma por uma, assim:
 - Primeiro, envie uma consulta ao servidor de banco de dados PostgreSQL.
 - Em seguida, aguarde o processamento.
 - Em seguida, processe o conjunto de resultados.
 - Depois disso, faça alguns cálculos.
 - Por fim, envie outra consulta ao servidor de banco de dados PostgreSQL e repita esse processo.
- Esse processo gera sobrecargas de comunicação entre processos e de rede.
- Para resolver esse problema, o **PostgreSQL** usa **PL/pgSQL**.

Funções PL/PGSQL

- Vantagens de usar PL/pgSQL:
 - PL/pgSQL **encapsula diversas instruções em um objeto e as armazena no servidor de banco de dados PostgreSQL.**
 - Em vez de enviar várias instruções ao servidor, uma por uma, você pode enviar uma única instrução para executar o objeto armazenado no servidor.
Isso permite:
 - Reduzir o número de viagens de ida e volta entre o aplicativo e o servidor de banco de dados PostgreSQL.
 - Evite transferir os resultados imediatos entre o aplicativo e o servidor.

Funções PL/PGSQL

- Desvantagens do PostgreSQL PL/pgSQL:
 - Além das vantagens de usar PL/pgSQL, há algumas ressalvas:
 - Mais lento no desenvolvimento de software porque PL/pgSQL requer habilidades especializadas que muitos desenvolvedores não possuem.
 - Versões difíceis de gerenciar e difíceis de depurar.
 - Pode não ser portátil para outros sistemas de gerenciamento de banco de dados.

Funções PLPGSQL

Constantes de string entre aspas

The screenshot shows a PostgreSQL query editor interface. The 'Query' tab is selected, displaying the following SQL code:

```
1
2 select 'String constant';
3
4
```

The 'Data Output' tab is selected, showing the results of the query:

	?column?	text
1		String constant

The toolbar below the tabs includes icons for new query, save, copy, paste, delete, refresh, and SQL.

- No **PostgreSQL**, constantes de string entre aspas simples permitem que você **construa strings que contêm aspas simples** em seu interior.
- Por exemplo, você pode colocar uma **constante de string** entre aspas simples, assim:

Funções PLPGSQL

Constantes de string entre aspas

- Mas quando uma constante de string contém uma aspa simples ('), você precisa escapá-la **dobrando a aspa simples**:

The screenshot shows a PostgreSQL query editor interface. At the top, there are tabs for "Query" and "Query History". Below the tabs, the query text is displayed:

```
1
2 select 'I''m a string constant';
3
4
```

Below the query text, there are tabs for "Data Output", "Messages", and "Notifications". Under "Data Output", there is a table with two rows of data:

	?column?	
1	text	🔒
1	I'm a string constant	

Funções PLPGSQL

Constantes de string entre aspas

- Para tornar o código mais legível, o PostgreSQL oferece uma sintaxe melhor chamada constante de string com aspas em dólar ou aspas em dólar:

The screenshot shows the pgAdmin 4 interface. The top bar has tabs for 'Query' (which is selected) and 'Query History'. Below the tabs, there are four numbered lines of code:
1
2 **select \$\$I'm a string constant\$\$;**
3
4

The 'Data Output' tab is selected at the bottom, showing a table with one row of data:

	?column?	
	text	🔒
1	I'm a string constant	

Funções PLPGSQL

Constantes de string entre aspas

The screenshot shows a PostgreSQL query editor interface. The top navigation bar has tabs for "Query" (which is selected) and "Query History". Below the editor area, there are tabs for "Data Output", "Messages", and "Notifications". A toolbar below the tabs contains various icons for file operations like new, open, save, and copy/paste, along with a "SQL" button.

```
1
2 select $$I'm a string constant$$ as message;
3
4
```

The main query editor area contains the following SQL code:

```
1
2 select $$I'm a string constant$$ as message;
3
4
```

The result pane shows a single row of data:

	message	
1	I'm a string constant	🔒

Funções PL/PGSQL

Estrutura de bloco PL/pgSQL

- PL/pgSQL é uma linguagem estruturada em blocos.
- Aqui está a sintaxe de um bloco em PL/pgSQL:

```
[ <<label>> ]
[ declare
    declarations ]
begin
    statements;
    ...
end [ label ];
```

Funções PL/PGSQL

Estrutura de bloco PL/pgSQL

- Cada bloco tem duas seções:
 - Declaração
 - Corpo
- A **seção de declaração** é opcional, enquanto a **seção do corpo** é obrigatória.
- Um **bloco** pode ter um opcional **Label** localizado no início e no fim do **bloco**. Um **bloco** termina com um ponto e vírgula (;) após a palavra-chave **END**.
- Normalmente, você usa o rótulo do bloco quando deseja especificá-lo na EXIT da declaração do corpo do bloco ou para qualificar os nomes das variáveis declaradas no bloco.

Funções PL/PGSQL

Estrutura de bloco PL/pgSQL

- A seção de declaração é onde você declara todas as variáveis usadas na seção do corpo. Cada instrução na seção de declaração é terminada com um ponto e vírgula (;).
- A sintaxe para declarar uma variável é a seguinte:
 - **variable_name type = initial_value;**
- Por exemplo, o seguinte declara uma variável chamada counter com o tipo int e valor inicial zero:
 - **counter int = 0;**

Funções PL/PGSQL

Estrutura de bloco PL/pgSQL

- Às vezes, você verá o operador `:=` em vez do operador `=`. Eles têm o mesmo significado:
 - `counter int := 0;`
- O valor inicial é opcional. Por exemplo, você pode declarar uma variável chamada `max` e do tipo inteiro.
 - `max int;`

Funções PL/PGSQL

Estrutura de bloco PL/pgSQL

Query History

```
1 do $$  
2 <<first_block>>  
3 declare  
4     professor_count integer := 0;  
5 begin  
6     -- get the number of films  
7     select count(*)  
8         into professor_count  
9     from Professor;  
10    -- display a message  
11    raise notice 'O total de professores eh %', professor_count;  
12 end first_block $$;  
13
```

Data Output Messages Notifications

NOTA: O total de professores eh 20
DO

Query returned successfully in 70 msec.

Funções PLPGSQL

Variáveis

The screenshot shows a PostgreSQL query editor interface. The top navigation bar has tabs for 'Query' and 'Query History', with 'Query' currently selected. The main area displays a PL/pgSQL script with numbered lines:

```
1 do $$  
2 declare  
3     counter    integer = 1;  
4     first_name varchar(50) = 'John';  
5     last_name   varchar(50) = 'Doe';  
6     payment     numeric(11,2) = 20.5;  
7 begin  
8     raise notice '% % % has been paid % USD',  
9             counter,  
10            first_name,  
11            last_name,  
12            payment;  
13 end $$;  
14  
15
```

Below the code, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is currently selected, showing the output of the 'raise notice' command:

NOTA: 1 John Doe has been paid 20.50 USD

DO

Query returned successfully in 72 msec.

Funções PLPGSQL

Variáveis

Query History

```
1 do $$  
2 declare  
3     first_name VARCHAR(50);  
4 begin  
5     first_name = split_part('John Doe', ' ', 1);  
6     raise notice 'The first name is %', first_name;  
7 end;  
8 $$;  
9  
10
```

Data Output Messages Notifications

NOTA: The first name is John
DO

Query returned successfully in 83 msec.

Funções PLPGSQL

Variáveis

```
Query  Query History
1 do $$  
2 declare  
3     created_at time = clock_timestamp();  
4 begin  
5     raise notice '%', created_at;  
6     perform pg_sleep(3);  
7     raise notice '%', created_at;  
8 end $$;  
9
```

Data Output Messages Notifications

NOTA: 14:49:36.384262

NOTA: 14:49:36.384262

DO

Query returned successfully in 3 secs 31 msec.

Funções PLPGSQL

Variáveis

- Copiando tipos de dados
- O **%type** fornece o tipo de dados de uma coluna de tabela ou outra variável. Normalmente, você usa o **%type** para declarar uma variável que contém um valor do banco de dados ou de outra variável.
- A seguir ilustramos como declarar uma variável com o tipo de dados de uma coluna de tabela: **A Disciplina (tabela de nosso Banco de Dados IES)**.
 - **variable_name table_name.column_name%type;**
 - Exemplo: **cargahoraria Disciplina.carga_horaria%type;**

Funções PLPGSQL

Variáveis

Query Query History

```
1 do
2 $$
3 declare
4     disciplina_nome Disciplina.nome%type;
5     cargahoraria Disciplina.carga_horaria%type;
6 begin
7
8     select nome, carga_horaria
9         from Disciplina
10        into disciplina_nome, cargahoraria
11       where id_disciplina = 1085;
12
13     raise notice 'Nome da Disciplina de ID 1085: % ... carga horaria: % horas', disciplina_nome, cargahoraria;
14 end; $$;
```

Data Output Messages Notifications

NOTA: Nome da Disciplina de ID 1085: BANCO DE DADOS ... carga horaria: 90 horas
DO

Query returned successfully in 133 msec.

Funções PLPGSQL

Escopo de Variáveis

- Quando você declara uma **variável** em um **sub-bloco** com o **mesmo nome** de **outra variável no bloco externo**, a **variável no bloco externo fica oculta dentro do sub-bloco**.
- Para acessar uma **variável no bloco externo**, **use o rótulo do bloco para qualificar seu nome**, conforme mostrado no exemplo a seguir:

Funções PLPGSQL

Escopo de Variáveis

Query History

```
1 do
2 $$ 
3 <<outer_block>>
4 declare
5     counter integer := 0;
6 begin
7     counter := counter + 1;
8     raise notice 'The current value of the counter is %', counter;
9     declare
10        counter integer := 0;
11    begin
12        counter := counter + 10;
13        raise notice 'Counter in the subblock is %', counter;
14        raise notice 'Counter in the outer block is %', outer_block.counter;
15    end;
16    raise notice 'Counter in the outer block is %', counter;
17 end outer_block $$;
```

Data Output Messages Notifications

NOTA: The current value of the counter is 1
NOTA: Counter in the subblock is 10
NOTA: Counter in the outer block is 1
NOTA: Counter in the outer block is 1
DO

Query returned successfully in 72 msec.

Funções PLPGSQL

Select Into

- A instrução SELECT INTO permite que você selecione dados do banco de dados e os atribua a uma variável .

The screenshot shows a PostgreSQL query editor interface. The top section is labeled "Query" and contains the following PL/pgSQL code:

```
1 do
2 $$ 
3 declare
4     contador integer;
5 begin
6
7     select count(*)
8     into contador
9     from Disciplina;
10
11    raise notice 'Total de Disciplinas cadastradas: %', contador;
12 end;
13 $$;
```

The bottom section is labeled "Messages" and displays the output of the query:

NOTA: Total de Disciplinas cadastradas: 8

DO

Query returned successfully in 73 msec.

Funções PLPGSQL

Row-Type

- **Variáveis de linha (Row-Type)** ou **variáveis de tipo de linha** são **variáveis de tipos compostos** que podem **armazenar linhas inteiras** de um conjunto de resultados.
- Essas **variáveis de linha** podem conter a linha inteira retornada pela instrução SELECT INTO ou FOR.

Funções PLPGSQL

Row-Type

Query Query History

```
1 do
2 $$
3 declare
4     selected_professor professor%rowtype;
5 begin
6
7     select *
8         from professor
9         into selected_professor
10        where id_professor = 10;
11
12    raise notice 'Dados do Professor de ID = 10: % %',
13          selected_professor.nome,
14          selected_professor.id_professor;
15 end;
16 $$;
```

Data Output Messages Notifications

NOTA: Dados do Professor de ID = 10: ASDRUBAL SOARES RIBEIRO 10
DO

Query returned successfully in 94 msec.

Funções PLPGSQL

Tipo RECORD (Registro)

- O PostgreSQL fornece um “**tipo**” chamado RECORD que é semelhante ao tipo de linha.
- É importante observar que um **registro não é um tipo verdadeiro**, mas sim um espaço reservado. Além disso, a estrutura de uma variável de registro mudará quando você a reatribuir a outro valor.
- Para declarar uma variável RECORD , basta usar o nome da variável seguido pela palavra-chave RECORD, como esta:
 - **variable_name record;**

Funções PLPGSQL

Tipo RECORD (Registro)

- Uma variável **RECORD** é semelhante a uma **variável do tipo linha** , que **pode conter apenas uma linha de um conjunto de resultados**.
- Ao contrário de uma **variável do tipo linha**, uma **variável RECORD** **não possui uma estrutura predefinida**. Em vez disso, a estrutura de uma variável **RECORD** é determinada quando uma linha real é atribuída a ela por meio da **instrução SELECT ou FOR**.

Funções PLPGSQL

Tipo RECORD (Registro)

Query History

```
1 do
2 $$
3 declare
4     rec record;
5 begin
6     -- seleção Disciplina
7     select id_disciplina, nome, carga_horaria
8     into rec
9     from Disciplina
10    where id_disciplina = 1086;
11    raise notice '% % %', rec.id_disciplina, rec.nome, rec.carga_horaria;
12 end;
13 $$
14 language plpgsql;
```

Data Output Messages Notifications

NOTA: 1086 BANCO DE DADOS 2 120
DO

Query returned successfully in 75 msec.

- Primeiro, declare uma variável de registro chamada **rec** na seção de declaração.
- Em segundo lugar, use a instrução **SELECT INTO** para selecionar uma linha cujo valor é um **registro da tabela Disciplina**.
- Terceiro, imprima as informações da **Disciplina** por meio da **variável record**.

Funções PLPGSQL

Tipo RECORD (Registro)

The screenshot shows a PostgreSQL PL/pgSQL editor interface. The top section is labeled "Query" and contains the following code:

```
1 do
2 $$ 
3 declare
4     rec record;
5 begin
6     for rec in select nome, carga_horaria
7         from Disciplina
8             where id_disciplina > 1080
9                 order by nome
10    loop
11        raise notice '% (%)', rec.nome, rec.carga_horaria;
12    end loop;
13 end;
14 $$;
```

The bottom section is labeled "Messages" and displays the output of the "raise notice" statements:

- NOTA: BANCO DE DADOS (90)
- NOTA: BANCO DE DADOS 2 (120)
- NOTA: ESTRUTURA DE DADOS (120)
- NOTA: LOGICA (120)
- NOTA: MATEMATICA DISCRETA (90)
- NOTA: PROGRAMACAO (120)

Below the messages, the text "DO" is displayed. At the very bottom, the message "Query returned successfully in 87 msec." is shown.

Funções PLPGSQL

Tipo RECORD (Registro)

- Primeiro, declare uma **variável** chamada **rec** com o tipo **RECORD**.
- Em segundo lugar, use a instrução **FOR LOOP** para buscar linhas da tabela **DISCIPLINA** (no banco de dados IES). A instrução **FOR LOOP** atribui a linha que consiste nas colunas **NOME** e **CARGA_HORARIA** à variável **rec** em cada iteração.
- Terceiro, mostre o **conteúdo dos campos** da variável de registro usando a **notação de ponto** (**rec.nome** e **rec.carga_horaria**).

Funções PLPGSQL

Constantes

- Em PL/pgSQL, constantes são identificadores cujos valores não podem ser alterados durante a execução do código.
- Normalmente, você usa constantes para atribuir nomes significativos a valores que permanecem constantes durante a execução de um bloco , uma função ou um procedimento armazenado .
- A seguir estão os motivos para usar constantes:
- Primeiro, as constantes tornam o código mais legível e fácil de manter. Suponha que você tenha a seguinte fórmula:
 - **Preco_de_venda = preco_rede + preco_rede * 0.1;**

Funções PLPGSQL

Constantes

- Nesta fórmula, o **valor mágico 0,1** não transmite nenhum significado.
- Entretanto, ao utilizar a fórmula a seguir, o significado para determinar o **preço de venda** fica claro:
 - **Preco_de_venda = preco_rede + preco_rede * imposto;**
- Segundo, as **constantes** ajudam a reduzir o esforço de manutenção.
- Suponha que você tenha uma fórmula que calcula o preço de venda por meio de uma função. Quando o **imposto** muda de **0,1** para **0,12**, você precisa atualizar todos esses valores codificados.
- Ao usar uma **constante**, você só precisa modificar seu valor no local onde você define a **constante**.
- Então, como você define uma **constante** em **PL/pgSQL**?

Funções PL/PGSQL

Constantes

- Para definir **uma constante** em PL/pgSQL, use a seguinte sintaxe:
 - **constant_name constant data_type = expression;**
- Nesta **sintaxe**:
- Primeiro, especifique o **nome da constante**. O nome deve ser o mais descritivo possível.
- Segundo, adicione a palavra-chave **constant** após o nome e especifique o **tipo de dados** da **constante**.
- Terceiro, **inicialize um valor para a constante** após o **operador de atribuição** (=).

Funções PLPGSQL

Constantes

Query History

```
1 do $$  
2 declare  
3     imposto constant numeric = 0.1;  
4     preco_rede    numeric = 20.5;  
5 begin  
6     raise notice 'O preço de venda eh %', preco_rede * ( 1 + imposto );  
7 end $$;  
8  
9
```

Data Output Messages Notifications

NOTA: O preço de venda eh 22.55
DO

Query returned successfully in 164 msec.

Funções PLPGSQL

Constantes

Query Query History

```
1 do $$  
2 declare  
3     imposto constant numeric = 0.1;  
4     preco_rede    numeric = 20.5;  
5 begin  
6     raise notice 'O preço de venda eh %', preco_rede * ( 1 + imposto );  
7     imposto = 0.12;  
8 end $$;  
9
```

Data Output Messages Notifications

ERROR: variável "imposto" está declarada CONSTANT
LINE 7: imposto = 0.12;
 ^

ERRO: variável "imposto" está declarada CONSTANT
SQL state: 22005
Character: 163

Funções PL/PGSQL

Instrução Assert (Asserção)

- A instrução **ASSERT** é uma abreviação útil para inserir verificações de depuração no código **PL/pgSQL**.
- Aqui está a sintaxe básica da instrução **ASSERT**:
 - **assert condition [, message];**
- Esta **CONDITION** é uma **expressão booleana** que sempre deve retornar **TRUE**.
- Quando a **CONDITION** é avaliada como **TRUE** a instrução **ASSERT** **não fará nada**.

Funções PLPGSQL

Instrução Assert (Asserção)

- Caso **CONDITION** for avaliado como **FALSE** ou **NULL** o PostgreSQL gera uma exceção **ASSERT_FAILURE**.
- A **MESSAGE** é opcional.

Funções PLPGSQL

Instrução Assert (Asserção)

Query Query History

```
1 do $$  
2 declare  
3     contador integer;  
4 begin  
5     select count(*)  
6         into contador  
7     from professor;  
8     assert contador >= 1000, 'Menos de 1000 Professores foram encontrados! Verifique a tabela Professor.';  
9 end$$;  
10
```

Data Output Messages Notifications

ERROR: Menos de 1000 Professores foram encontrados! Verifique a tabela Professor.
CONTEXT: função PL/pgSQL inline_code_block linha 8 em ASSERT

ERRO: Menos de 1000 Professores foram encontrados! Verifique a tabela Professor.
SQL state: P0004

Funções PLPGSQL

Instrução Assert (Asserção)

The screenshot shows a PostgreSQL query editor interface. The top navigation bar has tabs for "Query" and "Query History", with "Query" being the active tab. Below the tabs is a code editor area containing PL/pgSQL code. The code is numbered from 1 to 10. It declares a variable "contador" as integer, performs a select count(*) into "contador" from the "professor" table, and then uses an assert statement to check if the value is less than 1000. If not, it prints a message: "Mais de 1000 Professores foram encontrados! Verifique a tabela Professor.". The code ends with an end \$\$; command. Below the code editor are three tabs: "Data Output", "Messages" (which is active), and "Notifications". The "Messages" tab displays the output "DO" and the success message "Query returned successfully in 81 msec."

```
1 do $$  
2 declare  
3     contador integer;  
4 begin  
5     select count(*)  
6     into contador  
7     from professor;  
8     assert contador < 1000, 'Mais de 1000 Professores foram encontrados! Verifique a tabela Professor.';  
9 end$$;  
10
```

Data Output Messages Notifications

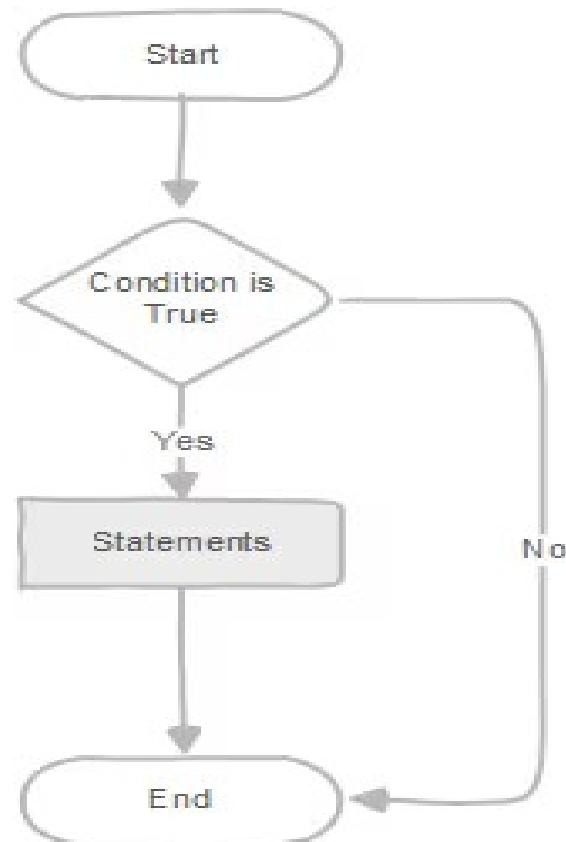
DO

Query returned successfully in 81 msec.

Funções PLPGSQL

IF

```
if condition then  
    statements;  
end if;
```



Funções PLPGSQL

IF

Query Query History

```
1 -- O exemplo a seguir usa uma ifinstrução para verificar se uma consulta retorna alguma linha:  
2 do $$  
3 declare  
4     Professor_selecionado professor%rowtype;  
5     input_id_professor professor.id_professor%type = 0;  
6 begin  
7     select * from Professor  
8     into Professor_selecionado  
9     where id_professor = input_id_professor;  
10  
11    if not found then  
12        raise notice'E Professor % NAO foi ENCONTRADO!',  
13            input_id_professor;  
14    end if;  
15 end $$;
```

Data Output Messages Notifications

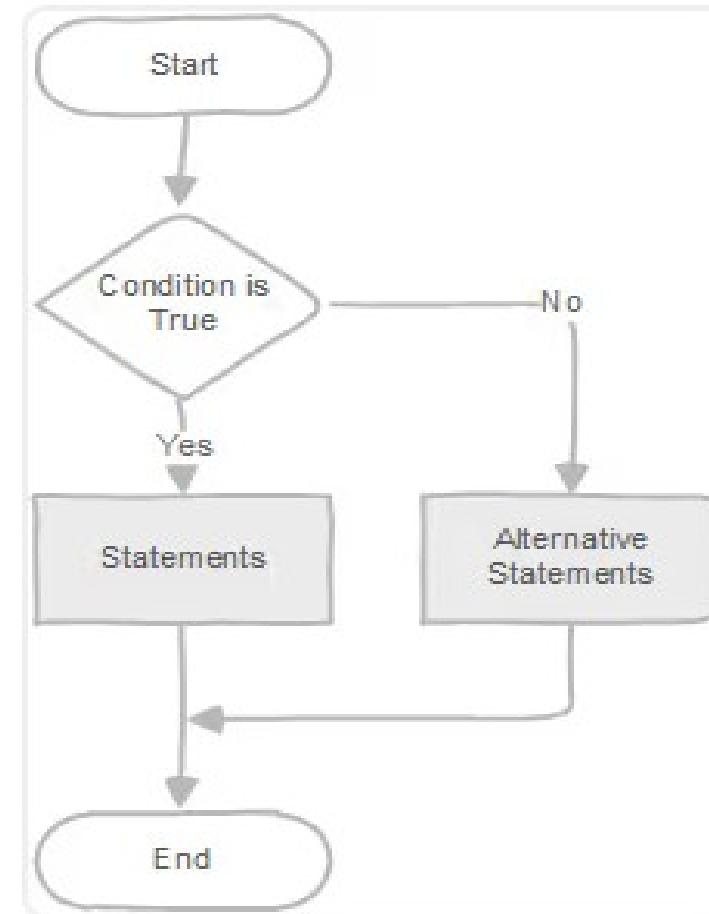
NOTA: O Professor 0 NAO foi ENCONTRADO!
DO

Query returned successfully in 124 msec.

Funções PLPGSQL

IF ELSE

```
if condition then  
    statements;  
  
else  
    alternative-statements;  
  
end if;
```



Funções PLPGSQL

IF ELSE

Query Query History

```
1 -- O exemplo a seguir usa uma instrução if else para verificar se uma consulta retorna alguma linha:
2 do $$ 
3 declare
4   Professor_selecionado professor%rowtype;
5   input_id_professor professor.id_professor%type = 35;
6 begin
7   select * from Professor
8   into Professor_selecionado
9   where id_professor = input_id_professor;
10
11  if not found then
12    raise notice 'O professor % NAO ENCONTRADO!', 
13      input_id_professor;
14  else
15    raise notice 'O nome do PROFESSOR eh %', Professor_selecionado.Nome;
16  end if;
17 end $$;
```

Data Output Messages Notifications

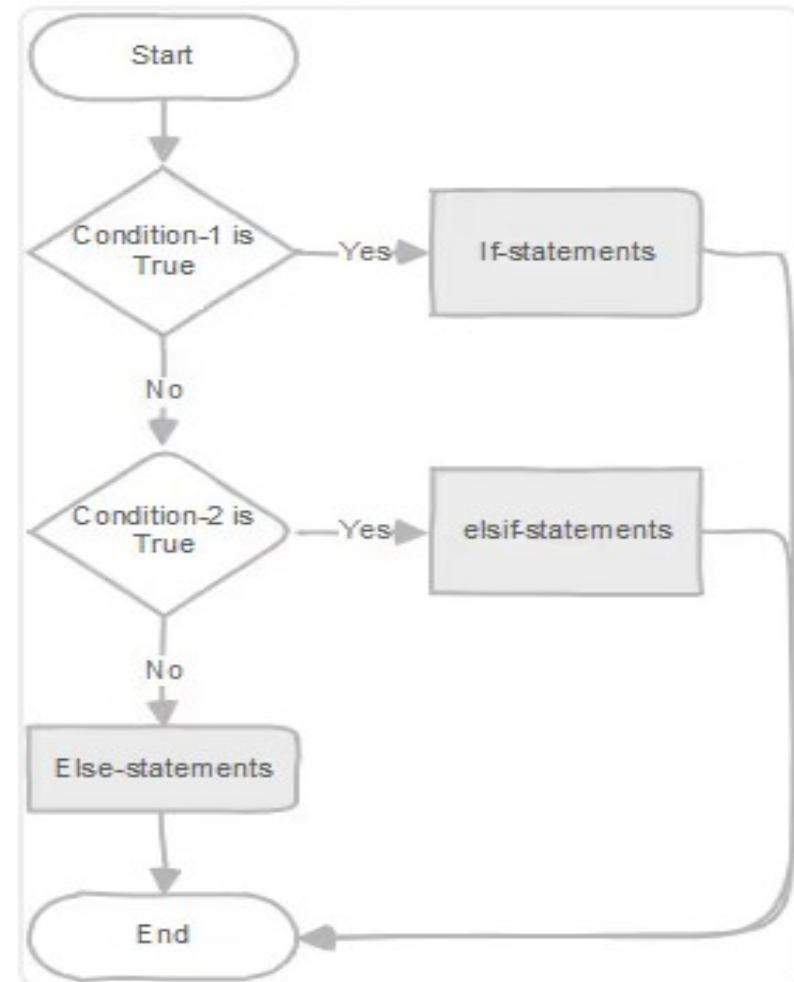
NOTA: O nome do PROFESSOR eh CINTHIA RIBEIRO
DO

Query returned successfully in 96 msec.

Funções PLPGSQL

IF ELSEIF

```
if condition_1 then  
    statement_1;  
elsif condition_2 then  
    statement_2  
...  
elsif condition_n then  
    statement_n;  
else  
    else-statement;  
end if;
```



Funções PLPGSQL

IF ELSEIF

Query History

```
1 CREATE TABLE FILM (
2     FILM_ID INTEGER NOT NULL PRIMARY KEY,
3     TITLE VARCHAR(100) NOT NULL,
4     PRODUCTION_YEAR INTEGER NOT NULL,
5     LENGTH INTEGER NOT NULL
6 );
7
8 INSERT INTO FILM VALUES (100,'EL CID', 1961, 206), (200,'THE KID', 1921,90), (300,'LIMELIGHT', 1956,110);
9
```

Data Output Messages Notifications

INSERT 0 3

Query returned successfully in 134 msec.

Desculpem pela grafia equivocada de LENGTH.

Por preguiça de corrigir continuaremos vendo LENGTH daqui até o final deste SLIDE.

Funções PLPGSQL

IF ELSEI

Query History

```
7
8   INSERT INTO FILM VALUES (100,'EL CID', 1961, 206), (200,'THE KID', 1921,90), (300,'LIMELIGHT', 1956,110);
9
10  do $$ 
11    declare
12      v_film film%rowtype;
13      len_description varchar(100);
14    begin
15      select * from film
16      into v_film
17      where film_id = 100;
18      if not found then
19          raise notice 'Film not found';
20      else
21          if v_film.length >0 and v_film.length <= 50 then
22              len_description := 'Short';
23          elsif v_film.length > 50 and v_film.length < 120 then
24              len_description := 'Medium';
25          elsif v_film.length > 120 then
26              len_description := 'Long';
27          else
28              len_description := 'N/A';
29          end if;
30          raise notice 'The % film is %.',
31                  v_film.title,
32                  len_description;
33      end if;
34    end $$;
```

Data Output Messages Notifications

NOTA: The EL CID film is Long.
DO

Query returned successfully in 142 msec.

Funções PLPGSQL

CASE

- Além da **instrução if** , o PostgreSQL fornece a **instrução CASE que permitem executar um bloco de código com base em condições.**
- A instrução CASE seleciona uma seção WHEN conforme uma dada condição:

```
case search-expression  
      when expression_1 [, expression_2, ...] then  
            when-statements  
            [...]  
      [else  
            else-statements ]  
END case;
```

Funções PLPGSQL CASE

Query Query History

```
1 ALTER TABLE FILM ADD COLUMN RENTAL_RATE NUMERIC(4,2);
2
3 SELECT * FROM FILM;
```

Data Output Messages Notifications

SQL

	film_id [PK] integer	title character varying (100)	production_year integer	length integer	rental_rate numeric (4,2)
1	100	EL CID	1961	206	[null]
2	200	THE KID	1921	90	[null]
3	300	LIMELIGHT	1956	110	[null]

Query Query History

```
1 ALTER TABLE FILM ADD COLUMN RENTAL_RATE NUMERIC(4,2);
2
3 SELECT * FROM FILM;
4
5 UPDATE FILM SET RENTAL_RATE = 4.99 WHERE FILM_ID = 100;
6 UPDATE FILM SET RENTAL_RATE = 2.99 WHERE FILM_ID = 200;
7 UPDATE FILM SET RENTAL_RATE = 0.99 WHERE FILM_ID = 300;
```

Data Output Messages Notifications

UPDATE 1

Query returned successfully in 94 msec.

Funções PLPGSQL

CASE

Query Query History

```
1 ALTER TABLE FILM ADD COLUMN RENTAL_RATE NUMERIC(4,2);
2
3 SELECT * FROM FILM;
4
5 UPDATE FILM SET RENTAL_RATE = 4.99 WHERE FILM_ID = 100;
6 UPDATE FILM SET RENTAL_RATE = 2.99 WHERE FILM_ID = 200;
7 UPDATE FILM SET RENTAL_RATE = 0.99 WHERE FILM_ID = 300;
8
9 INSERT INTO FILM VALUES (400, 'SCARAMOUCHE', 1954, 140, 4.99);
```

Data Output Messages Notifications

≡+

	film_id [PK] integer	title character varying (100)	production_year integer	length integer	rental_rate numeric (4,2)
1	100	EL CID	1961	206	4.99
2	200	THE KID	1921	90	2.99
3	300	LIMELIGHT	1956	110	0.99
4	400	SCARAMOUCHE	1954	140	4.99

Funções PLPGSQL CASE

```
10
11 do $$
12 declare
13     rate film.rental_rate%type;
14     price_segment varchar(50);
15 begin
16     -- get the rental rate
17     select rental_rate into rate
18     from film
19     where film_id = 100;
20     -- assign the price segment
21     if found then
22         case rate
23             when 0.99 then
24                 price_segment = 'Mass';
25             when 2.99 then
26                 price_segment = 'Mainstream';
27             when 4.99 then
28                 price_segment = 'High End';
29             else
30                 price_segment = 'Unspecified';
31             end case;
32             raise notice '%', price_segment;
33         else
34             raise notice 'film not found';
35         end if;
36     end; $$
```

Data Output Messages Notifications

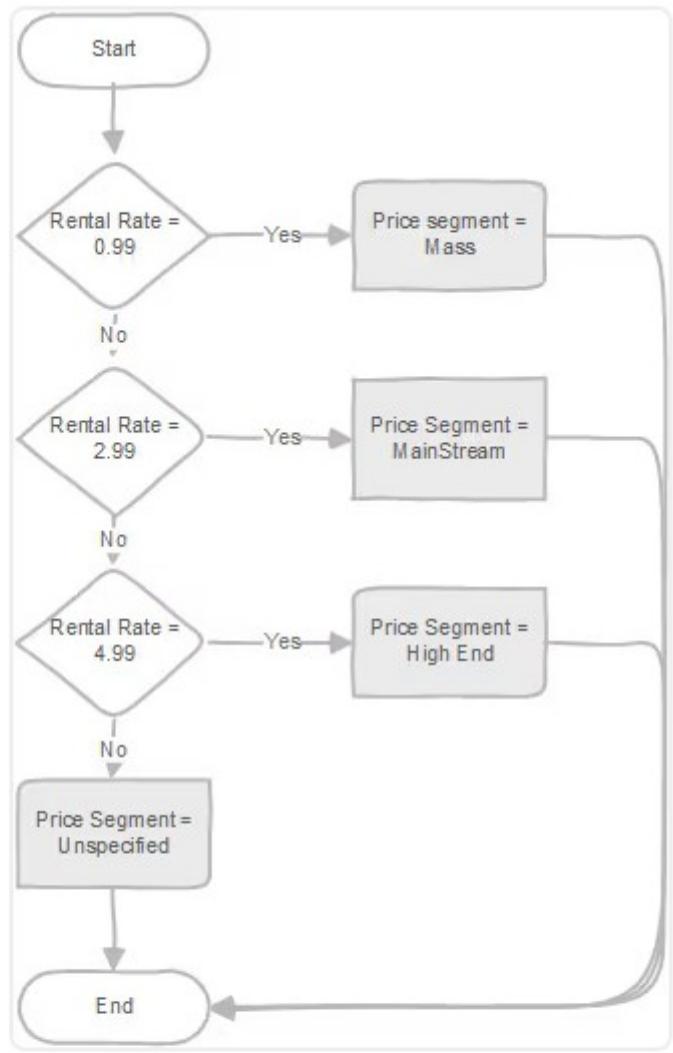
NOTA: High End

DO

Query returned successfully in 109 msec.

Funções PLPGSQL

CASE

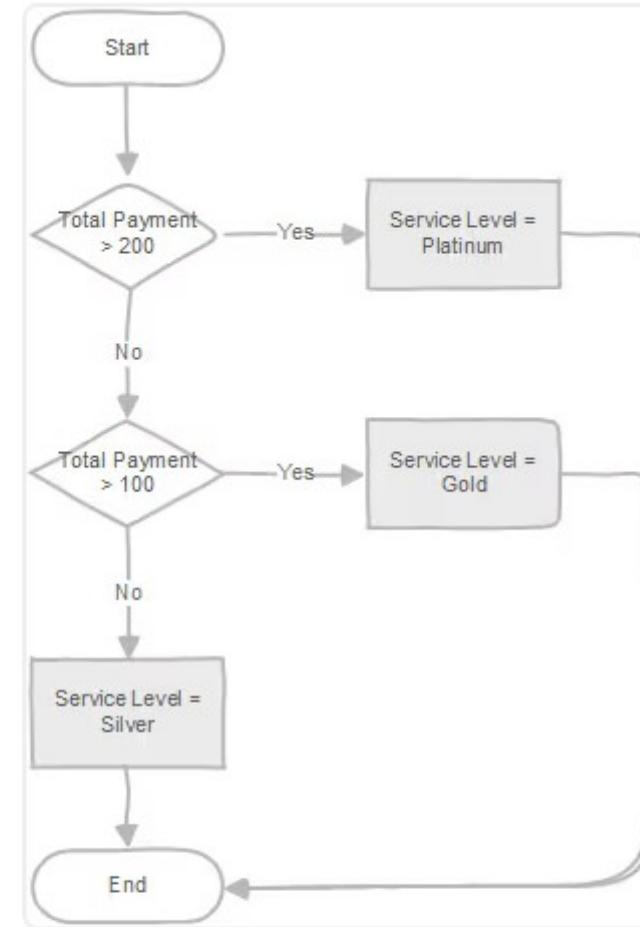


- Como funciona.
- Primeiro, selecione a taxa de aluguel do filme com id 100.
- Segundo, atribua o segmento de preço à variável `price_segment` se o ID do filme 100 existir ou uma mensagem caso contrário.
- Com base nas taxas de aluguel de 0,99, 2,99 ou 4,99, a instrução `case` atribui à variável `PRICE_SEGMENT` "mass", "mainstream" ou "high-end". Se a taxa de aluguel não for um desses valores, a instrução `CASE` atribui a string "Unspecified".

Funções PLPGSQL

CASE

```
case  
    when boolean-expression-1 then  
        statements  
    [ when boolean-expression-2 then  
        statements  
    ... ]  
    [ else  
        statements ]  
end case;
```

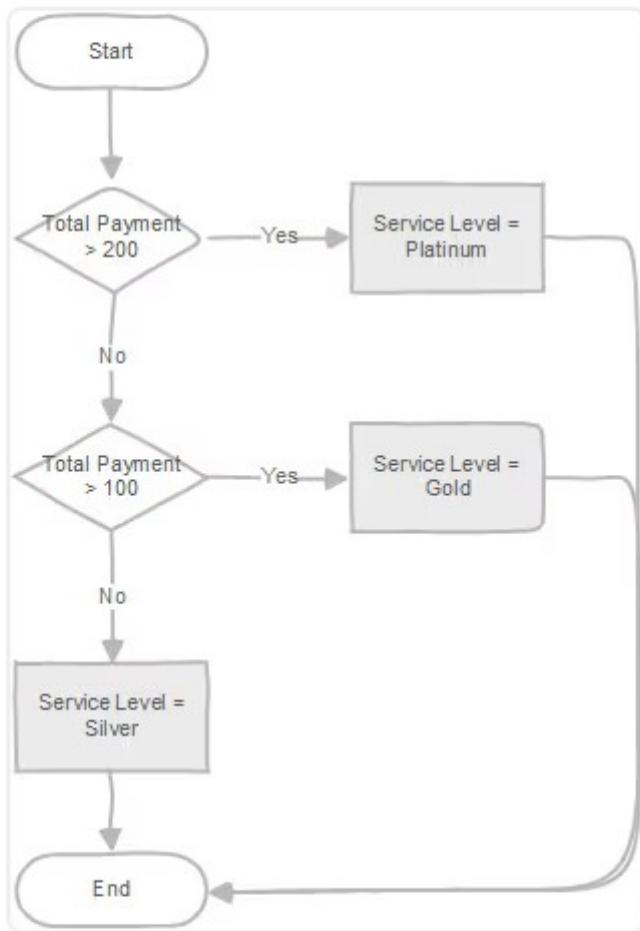


Funções PLPGSQL CASE

```
do $$  
declare  
    total_payment numeric;  
    service_level varchar(25) ;  
begin  
    select sum(amount) into total_payment  
    from Payment  
    where customer_id = 100;  
  
    if found then  
        case  
            when total_payment > 200 then  
                service_level = 'Platinum' ;  
            when total_payment > 100 then  
                service_level = 'Gold' ;  
            else  
                service_level = 'Silver' ;  
        end case;  
        raise notice 'Service Level: %', service_level;  
    else  
        raise notice 'Customer not found';  
    end if;  
end; $$
```

Funções PLPGSQL

CASE



Como funciona:

- Primeiro, selecione o pagamento total pago pelo cliente id 100 na payment tabela.
- Em seguida, atribua o nível de serviço ao cliente com base no pagamento total

Funções PLPGSQL

LOOP

- O comando **LOOP** define um loop incondicional que executa um bloco de código repetidamente até ser encerrado por uma instrução **EXIT** ou **RETURN**.

```
<<label>>
loop
    statements;
end loop;
```

```
<<label>>
loop
    statements;
    if condition then
        exit;
    end if;
end loop;
```

Funções PLPGSQL

LOOP

```
<<outer>>
loop
    statements;
<<inner>>
loop
    /* ... */
    exit <<inner>>
end loop;
end loop;
```

- Quando você tem loops aninhados, é necessário usar rótulos de loop.
- Os rótulos de loop permitem especificar o loop nas instruções **EXIT** e **CONTINUE**, **indicando a qual loop essas instruções se referem**.

Funções PLPGSQL

LOOP

Query History

```
1
2 do $$ 
3 declare
4     counter int := 0;
5 begin
6     loop
7         counter = counter + 1;
8         raise notice '%', counter;
9         if counter = 5 then
10             exit;
11         end if;
12     end loop;
13 end;
14 $$;
```

Data Output Messages Notifications

```
NOTA: 1
NOTA: 2
NOTA: 3
NOTA: 4
NOTA: 5
DO
```

Query returned successfully in 82 msec.

Query History

```
1 do $$ 
2 declare
3     counter int := 0;
4 begin
5     loop
6         counter = counter + 1;
7         raise notice '%', counter;
8         exit when counter = 5;
9     end loop;
10 end;
11 $$;
```

Data Output Messages Notifications

```
NOTA: 1
NOTA: 2
NOTA: 3
NOTA: 4
NOTA: 5
DO
```

Query returned successfully in 73 msec.

Funções PLPGSQL

LOOP

Query	Query History
1 do \$\$ 2 declare 3 counter int := 0; 4 begin 5 <<my_loop>> 6 loop 7 counter = counter + 1; 8 raise notice '%', counter; 9 exit my_loop when counter = 5; -- USO DO RÓTULO 10 end loop; 11 end; 12 \$\$; 13	

Data Output Messages Notifications

NOTA: 1
NOTA: 2
NOTA: 3
NOTA: 4
NOTA: 5
DO

Query returned successfully in 90 msec.

Funções PLPGSQL LOOP

Query History

```
1 do $$  
2 declare  
3     row_var int := 0;  
4     col_var int := 0;  
5 begin  
6     <<outer_loop>>  
7     loop  
8         row_var = row_var + 1;  
9         <<inner_loop>>  
10        loop  
11            col_var = col_var + 1;  
12            raise notice '(%, %)', row_var, col_var;  
13            -- terminate the inner loop  
14            exit inner_loop when col_var = 3;  
15        end loop;  
16        -- reset the column  
17        col_var = 0;  
18        -- terminate the outer loop  
19        exit outer_loop when row_var = 3;  
20    end loop;  
21 end;  
22 $$;
```

Data Output Messages Notifications

NOTA: (1, 1)
NOTA: (1, 2)
NOTA: (1, 3)
NOTA: (2, 1)
NOTA: (2, 2)
NOTA: (2, 3)
NOTA: (3, 1)
NOTA: (3, 2)
NOTA: (3, 3)
DO

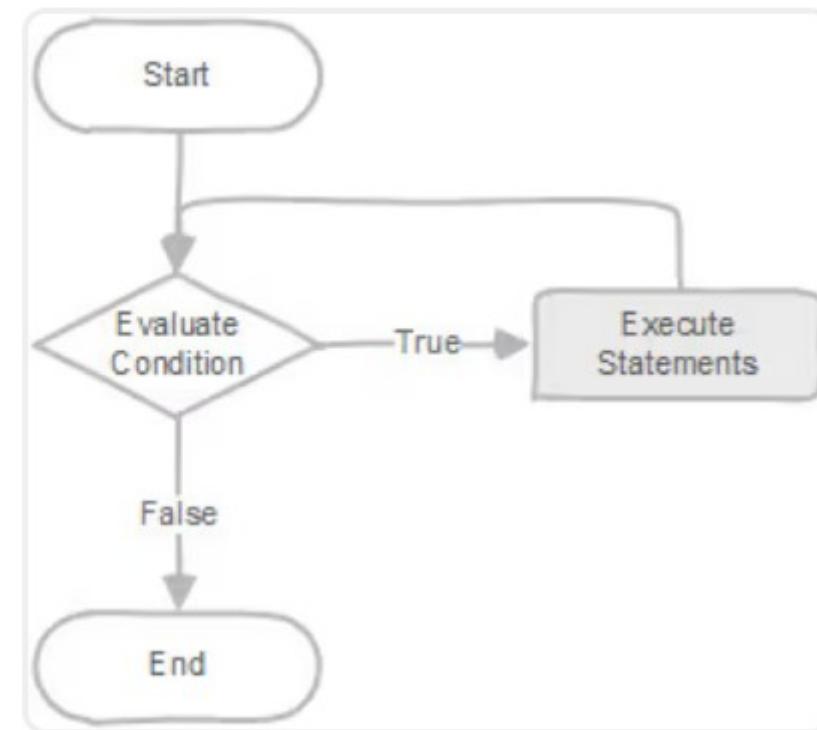
Query returned successfully in 83 msec.

Funções PLPGSQL

WHILE LOOP

- A instrução WHILE LOOP executa uma ou mais instruções, desde que uma condição especificada seja verdadeira.

```
[ <<label>> ]  
while condition loop  
  statements;  
end loop;
```



Funções PLPGSQL

WHILE LOOP

Query Query History

```
1 do $$  
2 declare  
3     counter integer := 0;  
4 begin  
5     while counter < 5 loop  
6         raise notice 'Counter %', counter;  
7         counter := counter + 1;  
8     end loop;  
9 end;  
10 $$;
```

Data Output Messages Notifications

NOTA: Counter 0
NOTA: Counter 1
NOTA: Counter 2
NOTA: Counter 3
NOTA: Counter 4
DO

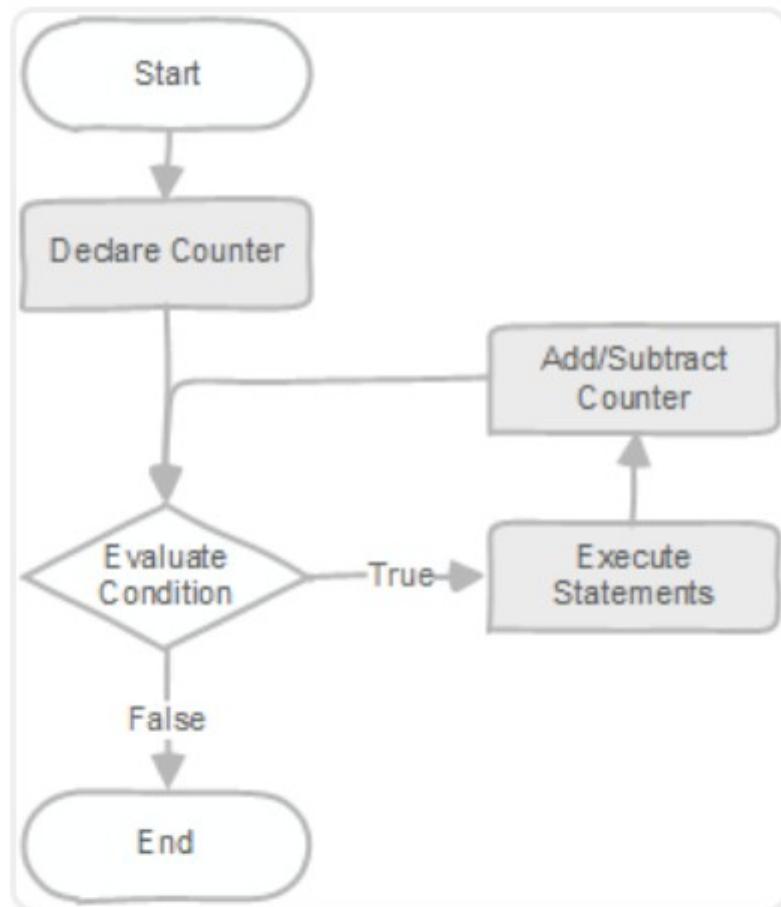
Query returned successfully in 93 msec.

Funções PLPGSQL

FOR LOOP

```
[ <<label>> ]  
for loop_counter in [ reverse ] from.. to [ by step ] loop  
    statements  
end loop [ label ];
```

Funções PLPGSQL FOR LOOP



Funções PLPGSQL FOR LOOP

Query Query History

```
1 do
2 $$ 
3 begin
4   for counter in 1..5 loop
5     raise notice 'counter: %', counter;
6   end loop;
7 end;
8 $$;
```

Data Output Messages Notifications

NOTA: counter: 1
NOTA: counter: 2
NOTA: counter: 3
NOTA: counter: 4
NOTA: counter: 5
DO

Query returned successfully in 101 msec.

Query Query History

```
1 do $$ 
2 begin
3   for counter in reverse 5..1 loop
4     raise notice 'counter: %', counter;
5   end loop;
6 end; $$
```

Data Output Messages Notifications

NOTA: counter: 5
NOTA: counter: 4
NOTA: counter: 3
NOTA: counter: 2
NOTA: counter: 1
DO

Query returned successfully in 81 msec.

Funções PLPGSQL FOR LOOP

Query Query History

```
1 do
2 $$
3 declare
4     f record;
5 begin
6     for f in select title, lenght
7         from film
8             order by lenght desc, title
9                 limit 10
10    loop
11        raise notice '%(% mins)', f.title, f.lenght;
12    end loop;
13 end;
14 $$
```

Data Output Messages Notifications

NOTA: EL CID(206 mins)
NOTA: SCARAMOUCHE(140 mins)
NOTA: LIMELIGHT(110 mins)
NOTA: THE KID(90 mins)
DO

Query returned successfully in 76 msec.

Funções PLPGSQL FOR LOOP

Query Query History

```
1 do $$  
2 declare  
3   -- sort by 1: title, 2: release year  
4   sort_type smallint := 1;  
5   -- return the number of films  
6   rec_count int := 10;  
7   -- use to iterate over the film  
8   rec record;  
9   -- dynamic query  
10  query text;  
11 begin  
12  query := 'select title, production_year from film ';  
13  if sort_type = 1 then  
14    query := query || 'order by title';  
15  elsif sort_type = 2 then  
16    query := query || 'order by production_year';  
17  else  
18    raise 'invalid sort type %s', sort_type;  
19  end if;  
20  query := query || ' limit $1';  
21  for rec in execute query using rec_count  
22    loop  
23      raise notice '% - %', rec.production_year, rec.title;  
24    end loop;  
25  end;  
$$
```

Data Output Messages Notifications

NOTA: 1961 - EL CID
NOTA: 1956 - LIMELIGHT
NOTA: 1954 - SCARAMOUCHE
NOTA: 1921 - THE KID
DO

Query returned successfully in 97 msec.

Funções PLPGSQL FOR LOOP

Query Query History

```
1 do $$  
2 declare  
3     -- sort by 1: title, 2: release year  
4     sort_type smallint := 2;  
5     -- return the number of films  
6     rec_count int := 10;  
7     -- use to iterate over the film  
8     rec record;  
9     -- dynamic query  
10    query text;  
11 begin  
12     query := 'select title, production_year from film ';  
13     if sort_type = 1 then  
14         query := query || 'order by title';  
15     elsif sort_type = 2 then  
16         query := query || 'order by production_year';  
17     else  
18         raise 'invalid sort type %s', sort_type;  
19     end if;  
20     query := query || ' limit $1';  
21     for rec in execute query using rec_count  
22         loop  
23             raise notice '% - %', rec.production_year, rec.title;  
24         end loop;  
25     end;  
26 $$
```

Data Output Messages Notifications

NOTA: 1921 - THE KID
NOTA: 1954 - SCARAMOUCHE
NOTA: 1956 - LIMELIGHT
NOTA: 1961 - EL CID
DO

Query returned successfully in 79 msec.

Funções PLPGSQL

```
create [or replace] function function_name(param_list)
       returns return_type
       language plpgsql
       as
$$
declare
    -- variable declaration
begin
    -- logic
end;
$$;
```

Funções PLPGSQL

Query Query History

```
1  create function get_film_count(len_from int, len_to int)
2  returns int
3  language plpgsql
4  as
5  $$
6  declare
7      film_count integer;
8  begin
9      select count(*)
10     into film_count
11    from film
12   where lenght between len_from and len_to;
13   return film_count;
14 end;
15 $$;
```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 78 msec.

Funções PLPGSQL

Query Query History

```
1  create function get_film_count(len_from int, len_to int)
2    returns int
3    language plpgsql
4    as
5    $$
6    declare
7      film_count integer;
8    begin
9      select count(*)
10     into film_count
11     from film
12     where length between len_from and len_to;
13     return film_count;
14   end;
15   $$;
16
17  select get_film_count(40,90);
```

Data Output Messages Notifications

≡+ SQL

	get_film_count	integer
1		1

Funções PLPGSQL

Query Query History

```
1  create function get_film_count(len_from int, len_to int)
2  returns int
3  language plpgsql
4  as
5  $$
6  declare
7      film_count integer;
8  begin
9      select count(*)
10     into film_count
11    from film
12   where lenght between len_from and len_to;
13   return film_count;
14 end;
15 $$;
16
17 select get_film_count(40,90);
18
19 select * from film;
```

Data Output Messages Notifications

≡+ SQL

	film_id [PK] integer	title character varying (100)	production_year integer	lenght integer	rental_rate numeric (4,2)
1	100	EL CID	1961	206	4.99
2	200	THE KID	1921	90	2.99
3	300	LIMELIGHT	1956	110	0.99
4	400	SCARAMOUCHE	1954	140	4.99

Funções PLPGSQL

The screenshot shows a PostgreSQL query editor interface. The top bar has tabs for "Query" and "Query History", with "Query" selected. The main area contains the following PL/pgSQL code:

```
1  create function get_film_count(len_from int, len_to int)
2    returns int
3    language plpgsql
4    as
5    $$
6    declare
7      film_count integer;
8    begin
9      select count(*)
10     into film_count
11     from film
12    where lenght between len_from and len_to;
13    return film_count;
14  end;
15  $$;
16
17  select get_film_count(90,140);
18
19  select * from film;
```

Below the code, there are tabs for "Data Output", "Messages", and "Notifications", with "Data Output" selected. The data output pane shows a table with one row:

	get_film_count	integer
1		3

At the bottom of the editor are several toolbar icons.

Funções PLPGSQL

```
...
16
17     select get_film_count(90,140);
18
19     select * from film;
20
21     select get_film_count(
22         len_from => 90,
23         len_to => 140
24 );
```

Data Output Messages Notifications

The screenshot shows the pgAdmin interface. At the top, there's a toolbar with various icons. Below it is a table showing the definition of the 'get_film_count' function:

	get_film_count	integer	🔒
1		3	

Usando notação nomeada !

Forma alternativa de invocar a função.

Funções PLPGSQL

```
10
17 select get_film_count(90,140);
18
19 select * from film;
20
21 select get_film_count(
22     len_from => 90,
23     len_to => 140
24 );
25
26 -- mantendo a compatibilidade com velhas versões do PostgreSQL
27 select get_film_count(
28     len_from := 40,
29     len_to := 90
30 );
```

Data Output Messages Notifications

get_film_count integer

	get_film_count	integer
1		1

Funções PLPGSQL

```
20
21 select get_film_count(
22     len_from => 90,
23     len_to => 140
24 );
25
26 -- mantendo a compatibilidade com velhas versões do PostgreSQL
27 select get_film_count(
28     len_from := 40,
29     len_to := 90
30 );
31
32 -- chamada mista
33 select get_film_count(40, len_to => 90);
34
```

Data Output Messages Notifications

The screenshot shows a PostgreSQL client interface with the following elements:

- A code editor window containing PL/pgSQL code for defining a function named `get_film_count`.
- Below the code editor are three tabs: "Data Output", "Messages", and "Notifications".
- An icon bar below the tabs contains icons for file operations (New, Open, Save, Print, Find, Copy, Paste, Cut, Delete, Undo, Redo) and a SQL tab.
- A table with two rows and two columns is displayed. The first row has a header cell and a cell containing the function name `get_film_count`. The second row has a cell with the value `1` and a cell with the value `1`.

Fonte

- <https://neon.com/postgresql/postgresql-plpgsql>