Ciência da Computação
GBC043 Sistemas de Banco de Dados

Programando com SQL
Stored Procedures

Profa. Maria Camila Nardini Barioni
camila.barioni@ufu.br
Bloco B - sala 1B137

1° semestre de 2024

Avisos

- 25/10 – Entrega do projeto final
- Agendar as apresentações dos projetos até 11/10
  - O agendamento deve ser feito por meio de envio de mensagem privada no MS Teams informando dia/horário e os nomes de todos os integrantes do grupo

Agenda – Apresentações projetos

Data/Horário	Grupo
01/11 – 14:50h	
01/11 – 15:20h	
01/11 – 15:50h	
01/11 – 16:20h	

Todas as apresentações serão feitas no Campus Santa Mônica Lab04
Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para logar na máquina e deixar tudo preparado para a apresentação.

Agenda – Apresentações projetos

Data/Horário	Grupo
01/11 – 17:00h	
01/11 – 17:30h	
01/11 – 18:00h	

Todas as apresentações serão feitas no Campus Santa Mônica Lab04
Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para logar na máquina e deixar tudo preparado para a apresentação.

Agenda – Apresentações projetos

Data/Horário	Grupo
07/11 – 13:20h	
07/11 – 13:50h	

Todas as apresentações serão feitas no Campus Santa Mônica. **Nessa data as apresentações deverão ser feitas com computador próprio do grupo na minha sala 1B137.** Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para logar na máquina e deixar tudo preparado para a apresentação.

Agenda – Apresentações projetos

Data/Horário	Grupo
08/11 – 14:50h	
08/11 – 15:20h	
08/11 – 15:50h	
08/11 – 16:20h	

Todas as apresentações serão feitas no Campus Santa Mônica Lab04
Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para logar na máquina e deixar tudo preparado para a apresentação.

Agenda – Apresentações projetos

Data/Horário	Grupo
08/11 – 17:00h	
08/11 – 17:30h	
08/11 – 18:00h	

Todas as apresentações serão feitas no Campus Santa Mônica Lab04
Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para logar na máquina e deixar tudo preparado para a apresentação.

Agenda – Apresentações projetos

Data/Horário	Grupo
14/11 – 13:20h	
14/11 – 13:50h	
14/11 – 14:20h	

Todas as apresentações serão feitas no Campus Santa Mônica. **Nessa data as apresentações deverão ser feitas com computador próprio do grupo na minha sala 1B137.** Cada grupo deve chegar com 10 minutos de antecedência do horário agendado para logar na máquina e deixar tudo preparado para a apresentação.

Sumário

- Resolução exercícios SQL
- Procedimentos Armazenados

Resolução ex

b) Qual a sigla do partido p
cuja ocupação é ser 'GARIM
exibidos no resultado a sigl

```
PARTIDO (NR_PARTIDO,
OCUPACAO (CD_OCUPACAO
CANDIDATO (SG_UF, NM_
..., NR_PARTIDO (PART
(OCUPACAO.CD_OCUPACAO
```

Query | Query History

```
1  set search_path to eleicao_2024;
2
3  SELECT P.SG_PARTIDO, COUNT(*)
4  FROM PARTIDO P, CANDIDATO C, OCUPACAO O
5  WHERE P.NR_PARTIDO = C.NR_PARTIDO AND
6        C.CD_OCUPACAO = O.CD_OCUPACAO AND
7        DS_OCUPACAO = 'GARIMPEIRO'
8  GROUP BY P.SG_PARTIDO
9  ORDER BY COUNT(*) DESC
10 LIMIT 1;
```

Data Output | Messages | Notifications

	sg_partido text	count bigint
1	CIDADANIA	1
2	MDB	9
3	MOBILIZA	2
4	PDT	6
5	PL	10
6	PODE	4
7	PP	6
8	PRD	4
9	PSB	4
10	PSD	3
11	PSDB	2
12	PT	4
13	REPUBLICANOS	1
14	SOLIDARIEDADE	1
15	UNIÃO	1

Resolução e

b) Qual a sigla do partido
cuja ocupação é ser 'GARI
exibidos no resultado a sig

```
PARTIDO (NR_PARTIDO,
OCUPACAO (CD_OCUPACA
CANDIDATO (SG_UF, NM
..., NR_PARTIDO (PAR
(OCUPACAO.CD_OCUPACA
```

Query | Query History

```
1  set search_path to eleicao_2024;
2
3  SELECT P.SG_PARTIDO, COUNT(*)
4  FROM PARTIDO P, CANDIDATO C, OCUPACAO O
5  WHERE P.NR_PARTIDO = C.NR_PARTIDO AND
6        C.CD_OCUPACAO = O.CD_OCUPACAO AND
7        DS_OCUPACAO = 'GARIMPEIRO'
8  GROUP BY P.SG_PARTIDO
9  ORDER BY COUNT(*) DESC
10 LIMIT 1;
```

Data Output | Messages | Notifications

SQL

	sg_partido text	count bigint
1	PL	10
2	MDB	9
3	PP	6
4	PDT	6
5	PT	4
6	PODE	4
7	PRD	4
8	PSB	4
9	PSD	3
10	PSDB	2
11	MOBILIZA	2
12	UNIÃO	1
13	REPUBLICANOS	1
14	SOLIDARIEDADE	1
15	CIDADANIA	1

Resolução exercícios SQL

b) Qual a sigla do partido
cuja ocupação é ser 'GARI
exibidos no resultado a si

```
PARTIDO (NR_PARTIDO
OCUPACAO (CD_OCUPAC
CANDIDATO (SG_UF, N
..., NR_PARTIDO (PA
(OCUPACAO.CD_OCUPAC
```

Query | Query History

```
1  set search_path to eleicao_2024;
2
3  SELECT P.SG_PARTIDO, COUNT(*)
4  FROM PARTIDO P JOIN CANDIDATO C
5       ON P.NR_PARTIDO = C.NR_PARTIDO
6       JOIN OCUPACAO O ON C.CD_OCUPACAO = O.CD_OCUPACAO
7  WHERE DS_OCUPACAO = 'GARIMPEIRO'
8  GROUP BY P.SG_PARTIDO
9  ORDER BY COUNT(*) DESC
10 LIMIT 1;
```

Data Output | Messages | Notifications

	sg_partido text	count bigint
1	PL	10

Resolução ex

c) Qual a quantidade de candi
e o total de candidatos (resolv
ser exibidos o número do part
ordenados por número de par

```
GENERO (CD_GENERO, DS_
CANDIDATO (..., NR_PAR
CD_GENERO (GENERO.CD_G
```

Query | Query History

```
1  set search_path to eleicao_2024;
2
3  SELECT NR_PARTIDO, DS_GENERO, COUNT(*)
4  FROM CANDIDATO C JOIN GENERO O ON C.CD_GENERO = O.CD_GENER
5  GROUP BY ROLLUP (NR_PARTIDO, DS_GENERO)
6  ORDER BY NR_PARTIDO, DS_GENERO
7
8
```

Data Output | Messages | Notifications

	nr_partido integer	ds_genero text	count bigint
1	10	FEMININO	11488
2	10	MASCULINO	22462
3	10	NÃO DIVULGÁVEL	3
4	10	[null]	33953
5	11	FEMININO	13343
6	11	MASCULINO	26474
7	11	NÃO DIVULGÁVEL	1
8	11	[null]	39818
9	12	FEMININO	7858
10	12	MASCULINO	15003
11	12	NÃO DIVULGÁVEL	1
12	12	[null]	22862
13	13	FEMININO	10852
14	13	MASCULINO	19174
15	13	[null]	30026

Resolução exercícios SQL

2) b) Criar uma **visão ma**
para recuperar os nomes
políticos e a sigla do esta
quantidade de bens que e

GENERO (CD_GENERO,

CANDIDATO (..., NR_
CD_GENERO (GENERO.C

Query | Query History

```
1  set search_path to eleicao_2024;
2
3  CREATE MATERIALIZED VIEW DETALHE_BEM_CANDIDATO AS
4  SELECT NM_CANDIDATO, NR_PARTIDO, SG_UF, COUNT(*)
5  FROM CANDIDATO C JOIN BEM_CANDIDATO B
6       ON C.SQ_CANDIDATO = B.SQ_CANDIDATO
7  GROUP BY C.SQ_CANDIDATO
8  WITH NO DATA;
9
10
```

Data Output | Messages | Notifications

```
CREATE MATERIALIZED VIEW

Query returned successfully in 87 msec.
```

Resolução exercícios SQL

2) b) Criar uma **visão ma**
para recuperar os nomes
políticos e a sigla do esta
quantidade de bens que e

GENERO (CD_GENERO,

CANDIDATO (..., NR_
CD_GENERO (GENERO.C

Query | Query History

```
1  set search_path to eleicao_2024;
2
3  CREATE MATERIALIZED VIEW DETALHE_BEM_CANDIDATO AS
4  SELECT NM_CANDIDATO, NR_PARTIDO, SG_UF, COUNT(*)
5  FROM CANDIDATO C JOIN BEM_CANDIDATO B
6       ON C.SQ_CANDIDATO = B.SQ_CANDIDATO
7  GROUP BY C.SQ_CANDIDATO
8  WITH NO DATA;
9
10 SELECT * FROM DETALHE_BEM_CANDIDATO;
```

Data Output | Messages | Notifications

```
ERROR:  visão materializada "detalhe_bem_candidato" não foi preenchida
HINT:  Use o comando REFRESH MATERIALIZED VIEW.

ERRO:  visão materializada "detalhe_bem_candidato" não foi preenchida
SQL state: 55000
```

Resolução exercícios SQL

2) b) Criar uma **visão ma**
para recuperar os nomes
políticos e a sigla do estad
quantidade de bens que e

GENERO (CD_GENERO,

CANDIDATO (..., NR_
CD_GENERO (GENERO.C

Query | Query History

```
3  CREATE MATERIALIZED VIEW DETALHE_BEM_CANDIDATO AS
4  SELECT NM_CANDIDATO, NR_PARTIDO, SG_UF, COUNT(*)
5  FROM CANDIDATO C JOIN BEM_CANDIDATO B
6       ON C.SQ_CANDIDATO = B.SQ_CANDIDATO
7  GROUP BY C.SQ_CANDIDATO
8  WITH NO DATA;
9
10 SELECT * FROM DETALHE_BEM_CANDIDATO;
11
12 REFRESH MATERIALIZED VIEW DETALHE_BEM_CANDIDATO;
```

Data Output | Messages | Notifications

```
REFRESH MATERIALIZED VIEW

Query returned successfully in 4 secs 403 msec.
```

Resoluçã

2) b) Criar uma **visã**
para recuperar os n
políticos e a sigla do
quantidade de bens

GENERO (CD_GENE

CANDIDATO (...,
CD_GENERO (GENE

Query | Query History | Scratch Pad

```
3  CREATE MATERIALIZED VIEW DETALHE_BEM_CANDIDATO AS
4  SELECT NM_CANDIDATO, NR_PARTIDO, SG_UF, COUNT(*)
5  FROM CANDIDATO C JOIN BEM_CANDIDATO B
6       ON C.SQ_CANDIDATO = B.SQ_CANDIDATO
7  GROUP BY C.SQ_CANDIDATO
8  WITH NO DATA;
9
10 SELECT * FROM DETALHE_BEM_CANDIDATO;
11
12 REFRESH MATERIALIZED VIEW DETALHE_BEM_CANDIDATO;
```

Data Output | Messages | Notifications

SQL

	nm_candidato text	nr_partido integer	sg_uf text	count bigint
1	DANIEL LEITE BARCELOS	44	AC	2
2	FLAVIO SOUZA DE MESQUITA	10	AC	3
3	ANTONIO AUGUSTO GOMES MAIA	10	AC	4
4	HERIVELTO ALVES FONTENELE	10	AC	3
5	MARIA LUNARA MORENO VIANNA	10	AC	1
6	FRANCISCO DAS CHAGAS PEREIRA DE ALMEIDA	10	AC	2
7	EIRE FERREIRA DA SILVA	10	AC	1
8	JOSÉ ALDO DE CASTRO LUZ	10	AC	1
9	RAPHAEL SALES DA SILVA	10	AC	1
10	HELANE CHRISTINA DA ROCHA SILVA	10	AC	5
11	VANUSA DOS SANTOS ZAIRE	10	AC	2
12	ANTONIO CÉLIO SARAIVA DOS SANTOS	10	AC	1
13	JOSÉ LOPES JUNIOR	10	AC	47
14	NELSON DE FREITAS CORREIA	10	AC	2
15	LUCENILDO DA CRUZ RAMOS	10	AC	1

- Procedimentos Armazenados

STORED PROCEDURES

Stored Procedures

- Procedimentos Armazenados

- É um bloco de código que contém um conjunto de comandos que podem ser armazenados no servidor
  - Comandos padrão
  - Extensões procedurais de SQL
    - Procedimentos, funções, métodos

Stored Procedures

- A partir da versão SQL-99
  - Definiu um padrão que adicionou funcionalidade procedural ao SQL
  - ***Persistent Storage Module (PSM)***
  - Vários SGBD possuem suas próprias extensões procedurais da SQL
    - PL/SQL no Oracle
    - TransactSQL no Microsoft SQL Server
    - PL/pgSQL no PostgreSQL

Stored Procedures: Vantagens

- Aumenta o desempenho do aplicativo:
  - Criado, **compilado** e armazenado no catálogo do BD
  - Mais rápido que comandos SQL **não compilados** enviados a partir de uma aplicação
- Reduz o tráfego de dados entre a aplicação e o servidor
  - Ao invés de enviar **vários** comandos SQL (longos) **não compilados** a aplicação envia apenas o nome da SP e obtém o resultado de volta

Stored Procedures: Vantagens

- É reutilizável e transparente para qualquer aplicação que pretende utilizá-la
- É seguro: DBA concede direitos a SP

Stored Procedures: Desvantagens

- O servidor de BD tem **carga maior** tanto em termos de memória quanto de processamento
- Ao invés de **focar apenas em armazenamento** e recuperação de dados o SGBD realizará uma série de operações lógicas, que em tese não é papel dele
- Não é possível "depurar" SP na maioria dos SGBDs
-
- Escrever e manter SP exige competências especializadas que não são tão comuns em desenvolvedores (da área de BD)

Extensões procedurais de SQL no PostgreSQL
FUNÇÕES

Preparação para os exemplos da aula

- Execute os seguintes comandos/scripts abaixo (aula25.zip)
  - CREATE SCHEMA LOJA2;
  - SET SEARCH_PATH TO LOJA2;
  - 01_CriaTabelas.sql
  - 02_DadosCadastrais.sql
  - InsVendas.sql

Função Simples

- CRIAÇÃO DA FUNÇÃO

```
CREATE OR REPLACE FUNCTION FUNC_DADOS_VENDEDOR
   (IN P_CODV VENDEDOR.CODVEND%TYPE,
    OUT P_NOME VENDEDOR.NOMEVEND%TYPE)
RETURNS VENDEDOR.NOMEVEND%TYPE AS '
BEGIN

   SELECT NOMEVEND
     INTO P_NOME
   FROM VENDEDOR
   WHERE CODVEND = P_CODV;

END' LANGUAGE 'plpgsql';
```

- EXECUÇÃO DA FUNÇÃO

```
…
SELECT FUNC_DADOS_VENDEDOR (240);
…
```

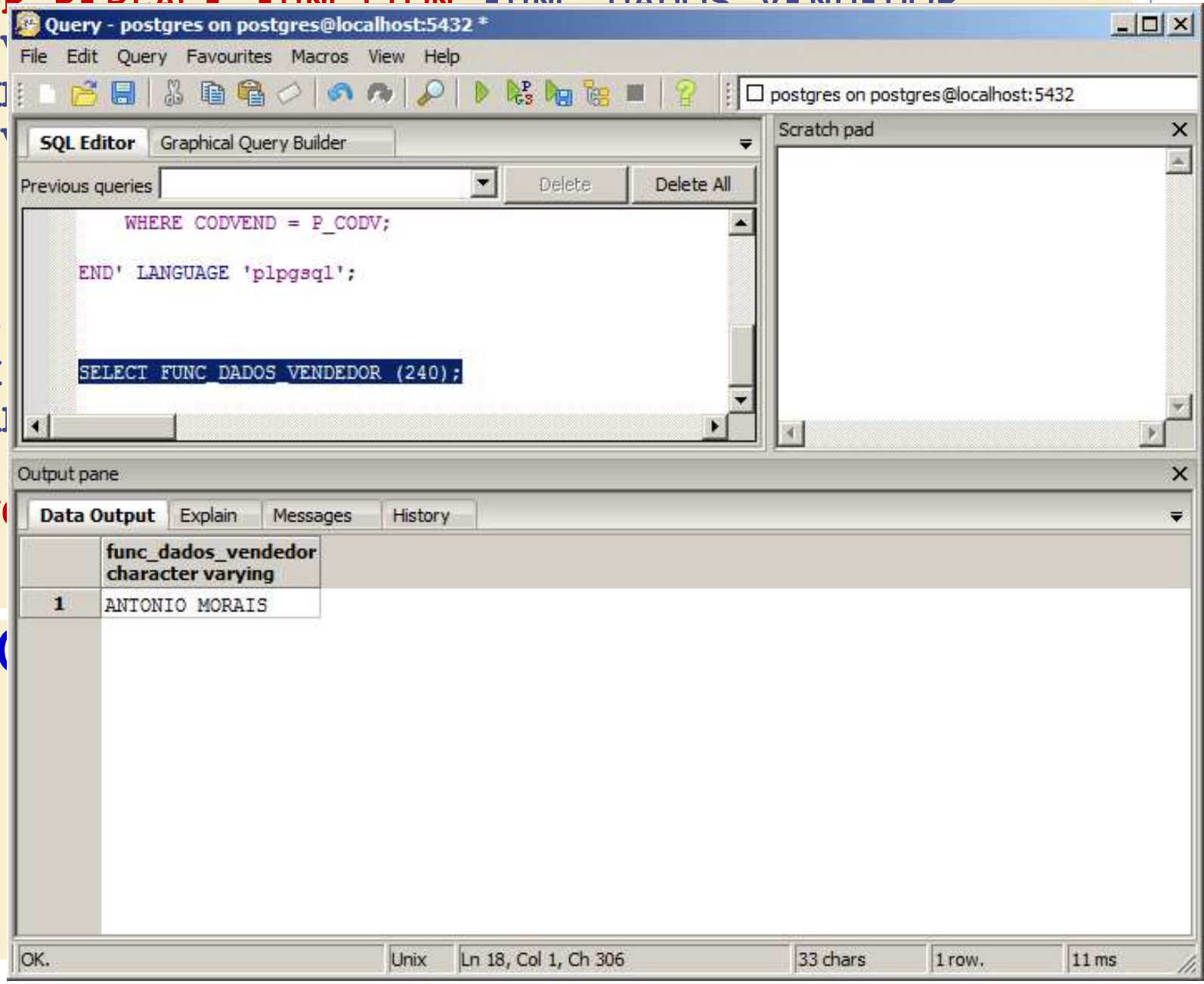
Função Simples

- CRIAÇÃO DA FUNÇÃO

```
CREATE OR REPLACE FUNCTION FUNC_DADOS_VENDEDOR
   (P_CODV
    P_NOM
RETURNS
BEGIN

   SELECT
      INTO
      FROM
      WHER

END' LAN
```

- EXECU

```
...
SELECT
...
```



Funções – Sintaxe

http://www.postgresql.org/docs/9.2/static/sql-createfunction.html

```
CREATE [OR REPLACE] FUNCTION function_name
[([IN | OUT | IN OUT] parameter_name  type [, ...])]

RETURNS type AS {' | $$}
[DECLARE declarations]
BEGIN
-- Corpo da função
function_body

END {' | $$}
LANGUAGE language_name;
```

Argumentos da função

- Uma função pode ter 0 ou mais parâmetros
- No corpo da função os parâmetros podem ser referidos usando o nome com os quais foram definidos ou usando a sintaxe $1, $2,..

```
--geom_avg
--Obtém a média geométrica de dois inteiros
CREATE FUNCTION geom_avg(int4, int4) RETURNS float8 AS
$$
BEGIN
RETURN SQRT($1 * $2::float8);
END;
$$ language plpgsql;
```

Argumentos da função

- Uma função pode ter 0 ou mais parâmetros
- No corpo da função os parâmetros podem ser referidos usando o nome com os quais foram definidos ou usando a sintaxe $1, $2,..

```
--geom_avg
--Obtém a média geométrica de dois inteiros
CREATE FUNCTION geom_avg(int4, int4) RETURNS float8 AS
$$
BEGIN
RETURN SQRT($1 * $2::float8);
END;
$$ language plpgsql;
```

Toda *stored procedure deve retornar um valor*
Mas pode retornar VOID

Declaração de Variáveis

Variáveis locais

- São declaradas no início da função
  - É necessário informar NOME e TIPO
  - Os tipos nas funções são similares aos tipos empregados na definição de atributos de tabelas. Para maiores detalhes, consulte o Manual do SGBD
- *nome[CONSTANT] tipo[NOT NULL] [{ DEFAULT | := } value];*
- Exemplo:

```
quant_vend  INTEGER;
```

Declaração de Variáveis

Variáveis locais

- Copiando tipos
- O tipo da variável pode ser especificado utilizando um tipo de outro item da base de dados

```
nome_variavel2 nome_variavel1%TYPE
quant_vend2 quant_vend%TYPE
-- mesmo tipo de quant_vend, ou seja, INTEGER
```

- O tipo da variável pode ser especificado utilizando um tipo já definido em um atributo de uma tabela

```
nome_variavel nome_tabela.nome_atributo%TYPE
quant_vend ITEMPED.QUANT%TYPE
  -- mesmo tipo do atributo QUANT da tabela ITEMPED
```

Declaração de Variáveis

Atribuição de Valores

```
quant_vend := 10;


SELECT COUNT(*) INTO quant_vend
FROM vendedor
/* - Variação do comando SELECT que permite
atribuir o resultado de uma consulta a uma variável
- Vale lembrar que essa variação é própria para
stored procedures
- Sintaxe:
- SELECT expressão INTO variável [FROM ..]; */
```

Declaração de Variáveis

- ESCOPO DAS VARIÁVEIS
  - Existem (são acessíveis) apenas dentro do bloco BEGIN – END em que foram declaradas

Declaração de Variáveis

A variável é estruturada em blocos (como em C++)

```
DECLARE
n1 integer;
n2 integer;
BEGIN
--pode-se usar n1 e n2 aqui
  n2 := 1;
      DECLARE
      n2 integer;  --esconde a variável n2 definida anteriormente
      n3 integer;
        BEGIN
              --n1, n2 e n3 estão disponíveis aqui
              n2 := 2;
        END;
      --n3 não está mais disponível
      --n2 ainda possui valor 1
END;
```

Exemplo – declaração de variáveis

```
CREATE OR REPLACE FUNCTION FUNC_NRO_CLIENTES
  (IN P_CEP CLIENTE.CEP%TYPE)
RETURNS NUMERIC AS $$
DECLARE V_CONTA_CEP NUMERIC;
BEGIN
   SELECT COUNT(CODCLI)
     INTO V_CONTA_CEP
     FROM CLIENTE
    WHERE CEP = P_CEP;

   RETURN V_CONTA_CEP;
END $$ LANGUAGE 'plpgsql';
```

Função que retorna o número de clientes que moram no mesmo CEP

iáveis

Query - postgres on postgres@localhost:5432 *

File Edit Query Favourites Macros View Help

postgres on postgres@localhost:5432

SQL Editor | Graphical Query Builder

Previous queries | Delete | Delete All

```
CREATE OR REPLACE FUNCTION FUNC_NRO_CLIENTES
 (IN P_CEP CLIENTE.CEP%TYPE)
RETURNS NUMERIC AS $$
DECLARE V_CONTA_CEP NUMERIC;
BEGIN
  SELECT COUNT(CODCLI)
    INTO V_CONTA_CEP
    FROM CLIENTE
   WHERE CEP = P_CEP;

  RETURN V_CONTA_CEP;
END $$ LANGUAGE 'plpgsql';

SELECT FUNC_NRO_CLIENTES ('22763931');
```

Output pane

Data Output | Explain | Messages | History

	func_nro_clientes numeric
1	2

OK. | Unix | Ln 14, Col 1, Ch 268 | 39 chars | 1 row. | 12 ms

Resultado

Parâmetros

- Três tipos:
  - IN (entrada) – são valores fornecidos para a SP
    - Especifica que se deve determinar um valor para o argumento quando a função é chamada
  - OUT (saída) – são modificados pela SP
    - Especifica que a função devolve um valor para esse argumento ao seu ambiente de chamada após a execução
  - IN OUT (entrada e saída): combinação de IN / OUT

Parâmetros de Entrada

- Exemplo de uma função que recebe (IN) a % de aumento para o SALÁRIO fixo dos vendedores e efetua o aumento (UPDATE).

- Note que PERC é um parâmetro FLOAT do tipo IN (entrada): este valor será passado por parâmetro para esta função e será usado para ALTERAR (update) o salário fixo dos vendedores.

```
CREATE OR REPLACE FUNCTION FUNC_ATUALIZA_SAL_FIXO
    (IN PERC FLOAT)
RETURNS VOID AS $$
BEGIN

  UPDATE VENDEDOR
    SET SALFIXO = SALFIXO + (SALFIXO * PERC / 100);


END $$ LANGUAGE 'plpgsql';
```

Parâmetros de Entrada

- Veja os salários (SALFIXO) dos vendedores ANTES de executar o ATUALIZA_SAL_FIXO

	CODVEND	NOMEVEND	SALFIXO	FAIXACOMIS
1	11	JOAO SANTOS OLIVEIRA	2780	C
2	101	JOAO SANTOS SILVA	2650	C
3	111	CARLOS VIEIRA	2490	A
4	209	JOSE FELISBERTO	1800	C
5	213	JONAS SILVA	2300	A
6	240	ANTONIO MORAIS	9500	C
7	250	MAURICIO SERRA	2930	B
8	310	JOSIAS DA SILVA	870	B
9	720	FELIPE NASCIMENTO	4600	A
10	100	MARCOS FARIAS	800	D
11	200	MONICA DA SILVA	1600	D
12	280	SAMANTA DE OLIVEIRA	3600	C

- Executa-se o ATUALIZA_SAL_FIXO com o parâmetro 50, que significa aumentar em 50% o SALFIXO

Parâmetros : Resultados

ANTES

	CODVEND	NOMEVEND	SALFIXO	FAIXACOMIS
1	11	JOAO SANTOS OLIVEIRA	2780	C
2	101	JOAO SANTOS SILVA	2650	C
3	111	CARLOS VIEIRA	2490	A
4	209	JOSE FELISBERTO	1800	C
5	213	JONAS SILVA	2300	A
6	240	ANTONIO MORAIS	9500	C
7	250	MAURICIO SERRA	2930	B
8	310	JOSIAS DA SILVA	870	B
9	720	FELIPE NASCIMENTO	4600	A
10	100	MARCOS FARIAS	800	D
11	200	MONICA DA SILVA	1600	D
12	280	SAMANTA DE OLIVEIRA	3600	C

Parâmetros !

	CODVEND	NOMEVEND	SALFIXO	FAIXACOMIS
1	11	JOAO SANTOS OLIVEIRA	2780	C
2	101	JOAO SANTOS SILVA		
3	111	CARLOS VIEIRA		
4	209	JOSE FELISBERTO		
5	213	JONAS SILVA		
6	240	ANTONIO MORAIS		
7	250	MAURICIO SERRA		
8	310	JOSIAS DA SILVA		
9	720	FELIPE NASCIMENTO		
10	100	MARCOS FARIAS		
11	200	MONICA DA SILVA		
12	280	SAMANTA DE OLIVEI		

Query - postgres on postgres@localhost:5432 *

File Edit Query Favourites Macros View Help

postgres on postgres@localhost:5432

SQL Editor | Graphical Query Builder

Previous queries | Delete | Delete All

```
CREATE OR REPLACE FUNCTION FUNC_ATUALIZA_SAL_FIXO
   (IN PERC FLOAT)
RETURNS VOID AS $$
BEGIN

  UPDATE VENDEDOR
    SET SALFIXO = SALFIXO + (SALFIXO * PERC / 100);

END $$ LANGUAGE 'plpgsql';

SELECT * FROM VENDEDOR

SELECT FUNC_ATUALIZA_SAL_FIXO(50)
```

Output pane

Data Output | Explain | Messages | History

	codvend integer	nomevend character varying(50)	salfixo double precision	faixacomis character(1)
1	11	JOAO SANTOS OLIVEI	4170	C
2	101	JOAO SANTOS SILVA	3975	C
3	111	CARLOS VIEIRA	3735	A
4	209	JOSE FELISBERTO	2700	C
5	213	JONAS SILVA	3450	A
6	240	ANTONIO MORAIS	14250	C
7	250	MAURICIO SERRA	4395	B
8	310	JOSIAS DA SILVA	1305	B
9	720	FELIPE NASCIMENTO	6900	A
10	100	MARCOS FARIAS	1200	D
11	200	MONICA DA SILVA	2400	D
12	280	SAMANTA DE OLIVEIR	5400	C

OK. | Unix | Ln 28, Col 1, Ch 537 | 33 chars | 12 rows. | 13 ms

DEPOIS

Parâmetros de Saída

- Exemplo de uma função que CONTA quantos VENDEDORES tem no cadastro e "RETORNA" (OUT)

  - Note que P_QUANT é um parâmetro INTEGER do tipo **OUT** (saída): este valor será calculado dentro da função e poderá ser usado FORA da função

```
CREATE OR REPLACE FUNCTION
FUNC_CONTA_VENDEDOR (OUT P_QUANT INTEGER)
RETURNS INTEGER AS $$
BEGIN

  SELECT COUNT(*) INTO P_QUANT FROM CLIENTE;

END $$ LANGUAGE 'plpgsql';
```

Pa

- E
  V

-

CRE
FUN
RET
BEG

SI

END

T)

T;

Query - postgres on postgres@localhost:5432 *

File Edit Query Favourites Macros View Help

postgres on postgres@localhost:5432

SQL Editor | Graphical Query Builder

Previous queries | Delete | Delete All

```
CREATE OR REPLACE FUNCTION FUNC_CONTA_VENDEDOR (OUT P_QUANT INTEGER)
RETURNS INTEGER AS $$
BEGIN

  SELECT COUNT(*) INTO P_QUANT FROM CLIENTE;

END $$ LANGUAGE 'plpgsql';

SELECT FUNC_CONTA_VENDEDOR()
```

Output pane

Data Output | Explain | Messages | History

	func_conta_vendedor integer
1	15

OK. | Unix | Ln 38, Col 1, Ch 745 | 28 chars | 1 row. | 11 ms

Parâmetros de Saída

- Exemplo de uma função que possui múltiplos parâmetros OUT

  - Note que a função retorna uma tupla contendo os múltiplos valores dos parâmetros OUT

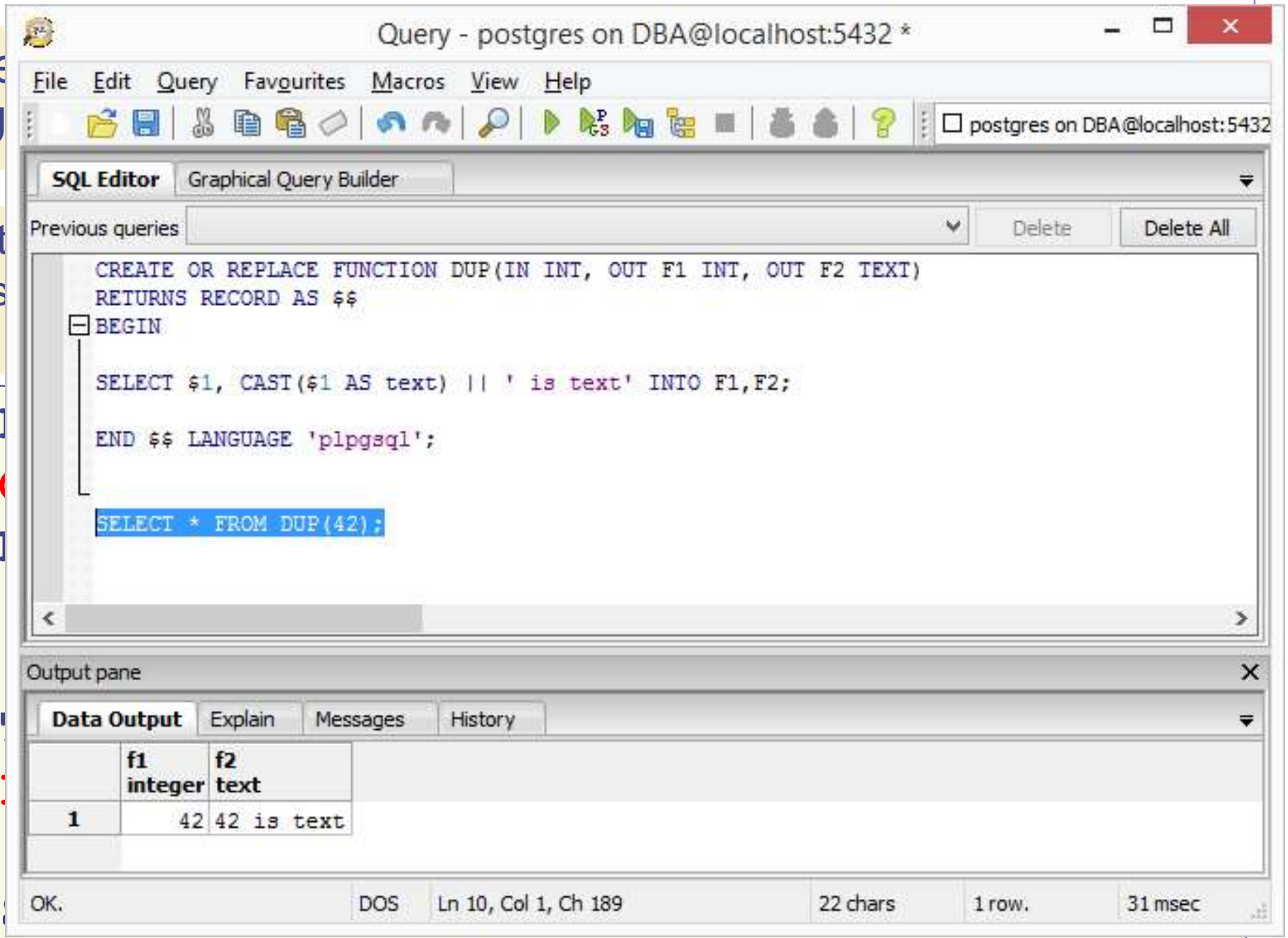
```
CREATE OR REPLACE FUNCTION DUP(IN INT, OUT F1
INT, OUT F2 TEXT)
RETURNS RECORD AS $$
BEGIN

SELECT $1, CAST($1 AS text) || ' is text'
INTO F1,F2;

END $$ LANGUAGE 'plpgsql';
```

Parâmetros de Saída

- Exe
  OU

  - Not
    dos

```
CREATI
INT, (
RETURI
BEGIN

SELEC
INTO

END $
```



Condição

SINTAXE DO IF

```
IF condição
THEN comandos
    [ELSEIF condição
        THEN comandos]
        [ELSE comandos]
END IF;
```

Condição

```
CREATE OR REPLACE FUNCTION FUNC_CADASTRA_PRODUTO
(IN P_COD PRODUTO.CODPROD%TYPE,
 IN P_NOME PRODUTO.DESCRI%TYPE,
 IN P_UN PRODUTO.UNIDADE%TYPE,
 IN P_VALOR PRODUTO.VALUNIT%TYPE)
RETURNS VOID AS $$
DECLARE V_CONT INTEGER;
BEGIN

 SELECT COUNT(*)
   INTO V_CONT
   FROM PRODUTO WHERE DESCRI = P_NOME;


  IF V_CONT = 0 THEN
    INSERT INTO PRODUTO VALUES (P_COD, P_UN, P_NOME,
P_VALOR);
  ELSE
     RAISE WARNING 'NOME de Produto Duplicado: %', P_NOME;
  END IF;

END $$ LANGUAGE 'plpgsql';
```

função para impedir o cadastramento de NOME DE PRODUTO duplicado !

ondição

Query - postgres on postgres@localhost:5432 *

File Edit Query Favourites Macros View Help

postgres on postgres@localhost:5432

SQL Editor | Graphical Query Builder

Previous queries | Delete | Delete All

```
    INSERT INTO PRODUTO VALUES (P_COD, P_UN, P_NOME, P_VALOR);
  ELSE
     RAISE WARNING 'NOME de Produto Duplicado: %', P_NOME;
  END IF;

END $$ LANGUAGE 'plpgsql';

SELECT FUNC_CADASTRA_PRODUTO(60, 'LEITE', 'L', 1.60);
```

2ª tentativa

Output pane

Data Output | Explain | Messages | History

```
AVISO:  NOME de Produto Duplicado: LEITE

Total query runtime: 14 ms.
1 row retrieved.
```

OK. | Unix | Ln 64, Col 1, Ch 1330 | 1 row. | 14 ms

Exceções e Mensagens

- Ao encontrar uma condição que impossibilita continuar a execução de uma *stored procedure* pode-se usar exceções

- `RAISE level 'format' [, variable...];`
  - https://www.postgresql.org/docs/16/plpgsql-errors-and-messages.html

Table 10-11. *PostgreSQL Exception Levels*

Level	Behavior
`DEBUG, LOG, INFO`	Writes a message in the log (usually suppressed)
`NOTICE, WARNING`	Writes a message in the log and sends it to the application
`EXCEPTION`	Writes a message in the log and terminates the stored procedure

Condição

SINTAXE - WHEN

- CASE
  WHEN condição THEN comandos ...

  WHEN condição THEN comandos

  ELSE comandos
  END CASE;

```
CREATE OR REPLACE FUNCTION FUNC_CADASTRA_PRODUTO_WHEN
(IN P_COD PRODUTO.CODPROD%TYPE, IN P_NOME PRODUTO.DESCRI%TYPE,
 IN P_UN PRODUTO.UNIDADE%TYPE, IN P_VALOR PRODUTO.VALUNIT%TYPE)
RETURNS VOID AS $$
DECLARE V_CONT INTEGER;
BEGIN

   SELECT COUNT(*) INTO V_CONT FROM PRODUTO WHERE DESCRI = P_NOME;

   CASE
     WHEN P_NOME = ' ' OR P_NOME IS NULL THEN
      RAISE WARNING 'NOME de Produto não pode ser vazio';
     WHEN P_UN = ' ' OR P_UN IS NULL THEN
      RAISE WARNING 'Unidade de Medida não pode ser vazio';
     WHEN V_CONT = 0 THEN
       INSERT INTO PRODUTO VALUES (P_COD, P_UN, P_NOME, P_VALOR);
     ELSE
      RAISE WARNING 'NOME de Produto Duplicado';
    END CASE;

END $$ LANGUAGE 'plpgsql';
```

função para impedir o cadastramento de valores nulos!

```
CREATE OR REPLACE FUNCTION FUNC_CADASTRA_PRODUTO_WHEN
(IN P_COD_PRODUTO CODPROD%TYPE, IN P_NOME_PRODUTO DESCRI%TYPE,
 IN P_
RETURI
DECLAI
BEGIN
   SELE
   CASI
      WI
      WI
      WI
      EI
   ENI
END $$
```

Query - postgres on postgres@localhost:5432 *

File Edit Query Favourites Macros View Help

postgres on postgres@localhost:5432

SQL Editor | Graphical Query Builder

Previous queries

```
CASE
  WHEN P_NOME = ' ' OR P_NOME IS NULL THEN
       RAISE WARNING 'NOME de Produto não pode ser vazio';
  WHEN P_UN = ' ' OR P_UN IS NULL THEN
       RAISE WARNING 'Unidade de Medida não pode ser vazio';
  WHEN V_CONT = 0 THEN
       INSERT INTO PRODUTO VALUES (P_COD, P_UN, P_NOME, P_VALOR);
  ELSE
       RAISE WARNING 'NOME de Produto Duplicado';
 END CASE;

END $$ LANGUAGE 'plpgsql';


SELECT FUNC_CADASTRA_PRODUTO_WHEN(86, 'BANANA PRATA', NULL , 1.50);
```

Output pane

Data Output | Explain | **Messages** | History

```
AVISO:  Unidade de Medida não pode ser vazio

Total query runtime: 13 ms.
1 row retrieved.
```

os!

Repetição

- **LOOP**
  comandos
  END LOOP

- **WHILE** condição **LOOP**
  comandos
  END LOOP

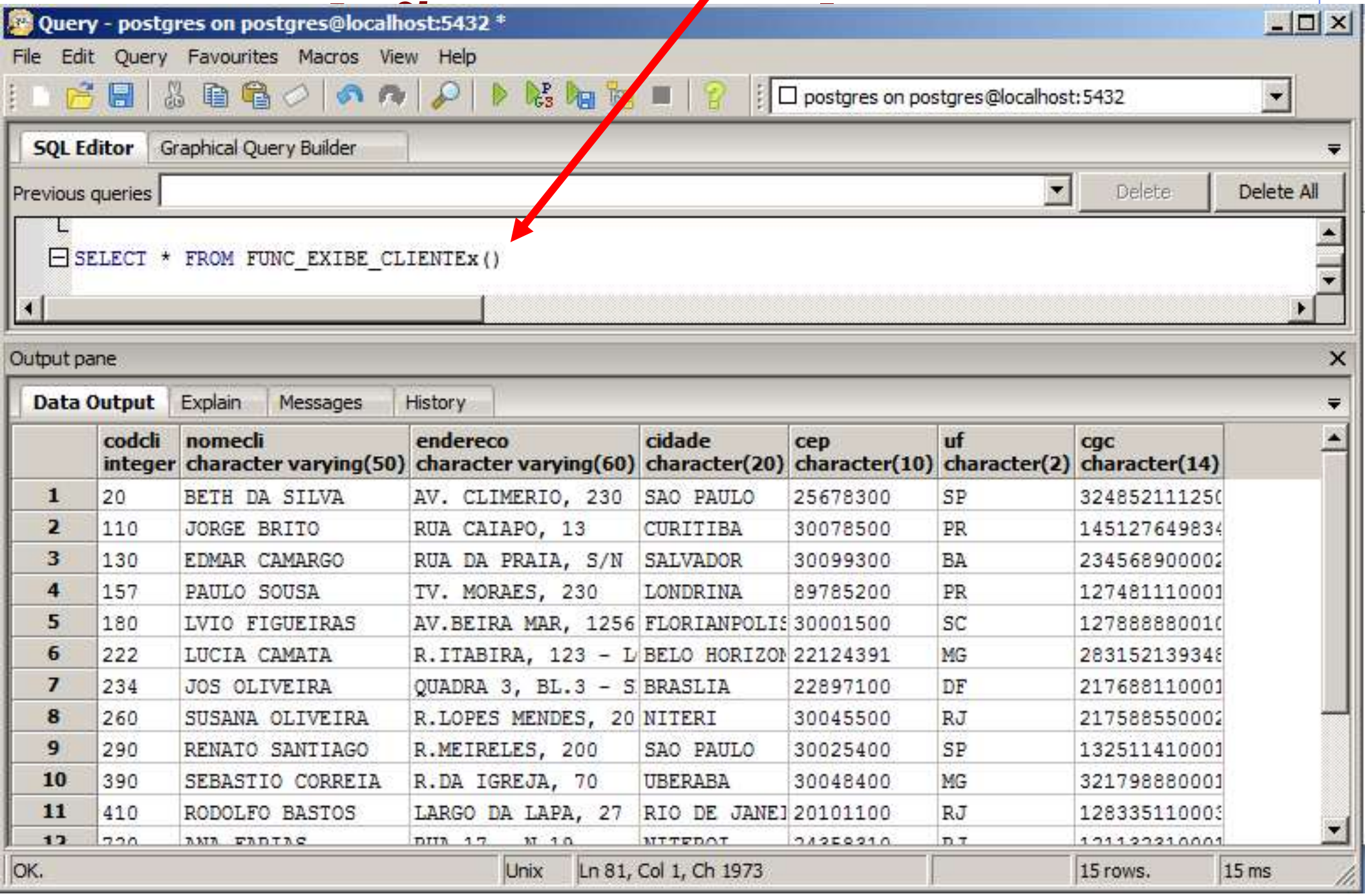
- **FOR** loop_variable **IN [REVERSE] from** lower_bound **to** upper_bound **LOOP**
  comandos
  END LOOP;

Repetição - Exemplo

```
CREATE OR REPLACE FUNCTION FUNC_EXIBE_CLIENTE ()
RETURNS void AS $$
/* listar o nome e cidade de clientes no programa aplicativo
*/
DECLARE tupla record;
BEGIN
FOR tupla IN SELECT * FROM cliente
LOOP

  --processamento sobre o cliente
  RAISE NOTICE 'NOME CLIENTE: %, CIDADE: %', tupla.nomecli,
tupla.cidade;

END LOOP;
END $$ LANGUAGE 'plpgsql';
```

Repetição - Exemplo

```
CREATE OR REPLACE FUNCTION FUNC_EXIBE_CLIENTEx ()
RETURNS SETOF CLIENTE AS $$
DECLARE tupla record;
BEGIN
FOR tupla IN SELECT * FROM cliente
LOOP

  --processamento sobre o cliente
  RETURN NEXT tupla;

END LOOP;
END $$ LANGUAGE 'plpgsql';
```


Query - postgres on postgres@localhost:5432 *
File Edit Query Favourites Macros View Help
postgres on postgres@localhost:5432
SQL Editor
Graphical Query Builder
Previous queries
Delete
Delete All
SELECT * FROM FUNC_EXIBE_CLIENTEx()
Output pane
Data Output
Explain
Messages
History
codcli integer
nomecli character varying(50)
endereco character varying(60)
cidade character(20)
cep character(10)
uf character(2)
cgc character(14)
1 20 BETH DA SILVA AV. CLIMERIO, 230 SAO PAULO 25678300 SP
2 110 JORGE BRITO RUA CAIAPO, 13 CURITIBA 30078500 PR
3 130 EDMAR CAMARGO RUA DA PRAIA, S/N SALVADOR 30099300 BA
4 157 PAULO SOUSA TV. MORAES, 230 LONDRINA 89785200 PR
5 180 LVIO FIGUEIRAS AV.BEIRA MAR, 1256 FLORIANPOLIS 30001500 SC
6 222 LUCIA CAMATA R.ITABIRA, 123 - L BELO HORIZON 22124391 MG
7 234 JOS OLIVEIRA QUADRA 3, BL.3 - S BRASLIA 22897100 DF
8 260 SUSANA OLIVEIRA R.LOPES MENDES, 20 NITERI 30045500 RJ
9 290 RENATO SANTIAGO R.MEIRELES, 200 SAO PAULO 30025400 SP
10 390 SEBASTIO CORREIA R.DA IGREJA, 70 UBERABA 30048400 MG
11 410 RODOLFO BASTOS LARGO DA LAPA, 27 RIO DE JANEI 20101100 RJ
OK.
Unix
Ln 81, Col 1, Ch 1973
15 rows.
15 ms

Comandos de Remoção

- Funções
  - **DROP FUNCTION** NomeFunção;

ESTUDO DE CASO: A LOJA NA INTERNET

Estudo de caso: A loja na internet
Análise de consultas e atualizações

- Relembrando o esquema relacional:

Livro (isbn, titulo, autor, qtde-estoque, preço-a, preço-v, ano-pub)

Cliente(id-cliente, nome, endereço)

Pedido(nro-pedido, id-cliente(Cliente.id-cliente), nro-cartão, data-pedido)

Lista-Pedido(nro-pedido, isbn (Livro.isbn), qtde, data-remessa)

Estudo de caso: A loja na internet
Análise de consultas e atualizações

- Dentre as tarefas executadas pelos clientes
  - Os clientes procuram livros pelo nome do autor, título ou ISBN
  - Os clientes se registram no website. Os clientes registrados podem desejar alterar suas informações de contato.
  - Os clientes conferem a cesta de compras e somam o total para completar a compra
  - Os clientes verificam o status de pedidos existentes e examinam pedidos antigos
  - ...

Estudo de caso: A loja na internet
Análise de consultas e atualizações

- Dentre as tarefas executadas pelos funcionários da livraria
  - Os funcionários examinam as informações de contato dos clientes
  - Os funcionários acrescentam novos livros ao inventário
  - Os funcionários atualizam o estoque de livros cadastrados
  - Os funcionários atualizam as datas de remessa dos livros individuais
  - Os funcionários analisam os dados para encontrar os clientes lucrativos
  - ...

Estudo de caso: A loja na internet
Análise de consultas e atualizações

- Dentre as tarefas executadas pelos funcionários da livraria
  - Os funcionários examinam as informações de contato dos clientes
  - Os funcionários acrescentam novos livros ao inventário
  - **Os funcionários atualizam o estoque de livros cadastrados**
  - Os funcionários atualizam as datas de remessa dos livros individuais
  - **Os funcionários analisam os dados para encontrar os clientes lucrativos**
  - ...

Estudo de caso: A loja na internet
Exercícios

- Crie procedimentos armazenados para:
  - **Os funcionários atualizam o estoque de livros cadastrados**

```
CREATE OR REPLACE FUNCTION FUNC_ATUALIZA_INVENTARIO
    (?)
RETURNS ? AS $$
BEGIN


?


END $$ LANGUAGE 'plpgsql';
```

Estudo de caso: A loja na internet
Exercícios

- Crie procedimentos armazenados para
  - **Os funcionários analisam os dados para encontrar os clientes lucrativos**
  - **Definição de cliente lucrativo (avaliado anualmente)**
    - Cliente que compra mais do que 10 livros em um mês deve ser avaliado como categoria 2
    - Cliente que compra mais do que 5 livros em um mês deve ser avaliado como categoria 1
    - Cliente que compra 5 ou menos livros em um mês deve ser avaliado como categoria 0

Estudo de caso: A loja na internet
Exercícios

- Crie procedimentos armazenados para
  - **Os funcionários analisam os dados para encontrar os clientes lucrativos**

```
CREATE OR REPLACE FUNCTION FUNC_AVALIA_CLIENTE ( ? )
RETURNS ? AS $$
DECLARE ?

BEGIN




?



END $$ LANGUAGE 'plpgsql';
```

Leitura Adicional para Casa

- SILBERSCHATZ-KHORH-SUDARSHAN – **Sistema de Banco de Dados (5ª edição)**
  - **Capítulo 4 – SQL Avançada**
- ELMASRI-NAVATHE **– Sistemas de Banco de Dados (6ª edição)**
  - **Seção 13.4 – Procedimentos Armazenados em Banco de Dados**
- Manual de SQL do PostgreSQL
  - https://www.postgresql.org/docs/16/sql-createfunction.html