Criando Bancos e Tabelas no PostgreSQL

16.5

Aula 9

Introdução a linguagem SQL

Funções agregadas

Funções Windows

Funções agregadas

As funções agregadas são compostas por

- MAX,
- MIN,
- AVG,
- SUM e
- COUNT

e cada uma delas executa uma tarefa específica.

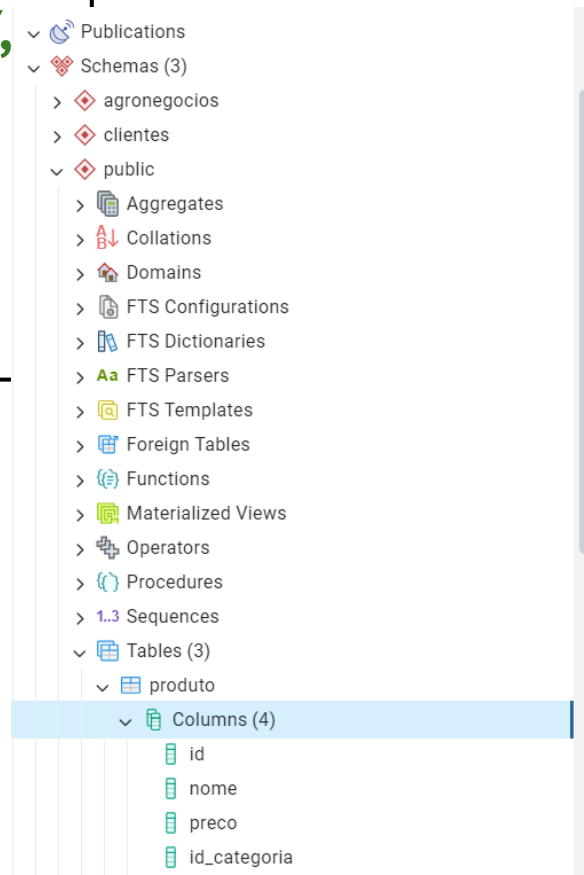
Funções agregadas

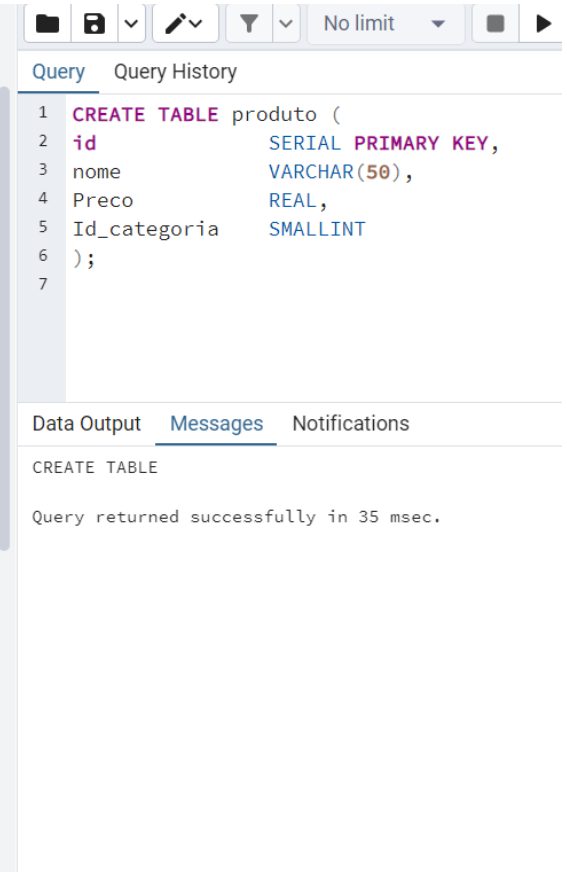
É possível utilizar as funções agregadoras em conjunto com o WHERE, lembrando que o resultado mudará de acordo com as sentenças definidas pelo filtro.

Criando tabelas para exemplos

Criando um exemplo

```
CREATE TABLE produto (
    id              SERIAL PRIMARY KEY,
    nome            VARCHAR(50),
    Preco           REAL,
    Id_categoria    SMALLINT
);
```



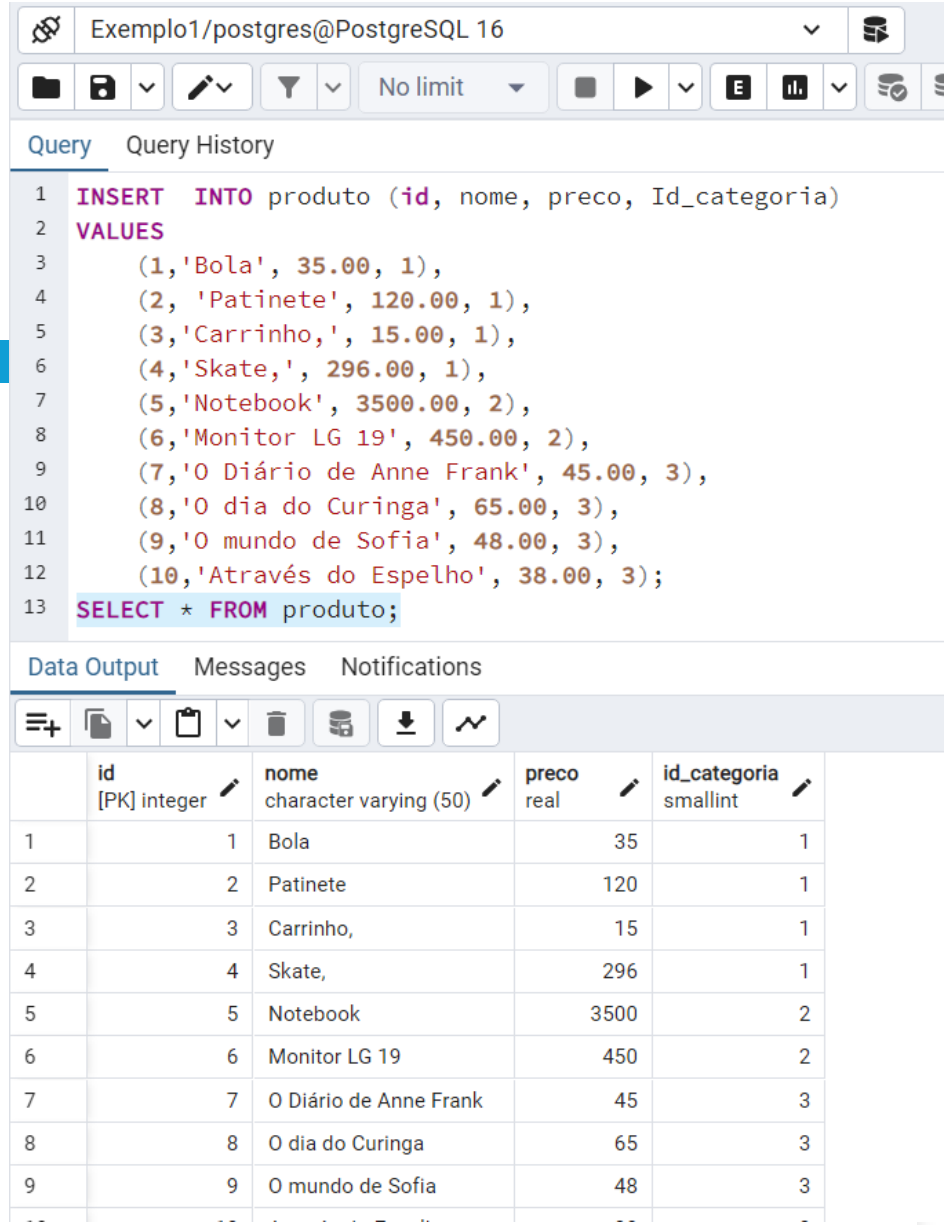


Dados da tabela produto:

1	Bola	35.00	1
2	Patinete	120.00	1
3	Carrinho	15.00	1
4	Skate	296.00	1
5	Notebook	3500.00	2
6	Monitor LG 19	450.00	2
7	O Diário de Anne Frank	45.00	3
8	O dia do Curinga	65.00	3
9	O mundo de Sofia	48.00	3
10	Através do Espelho	38.00	3

Inserindo

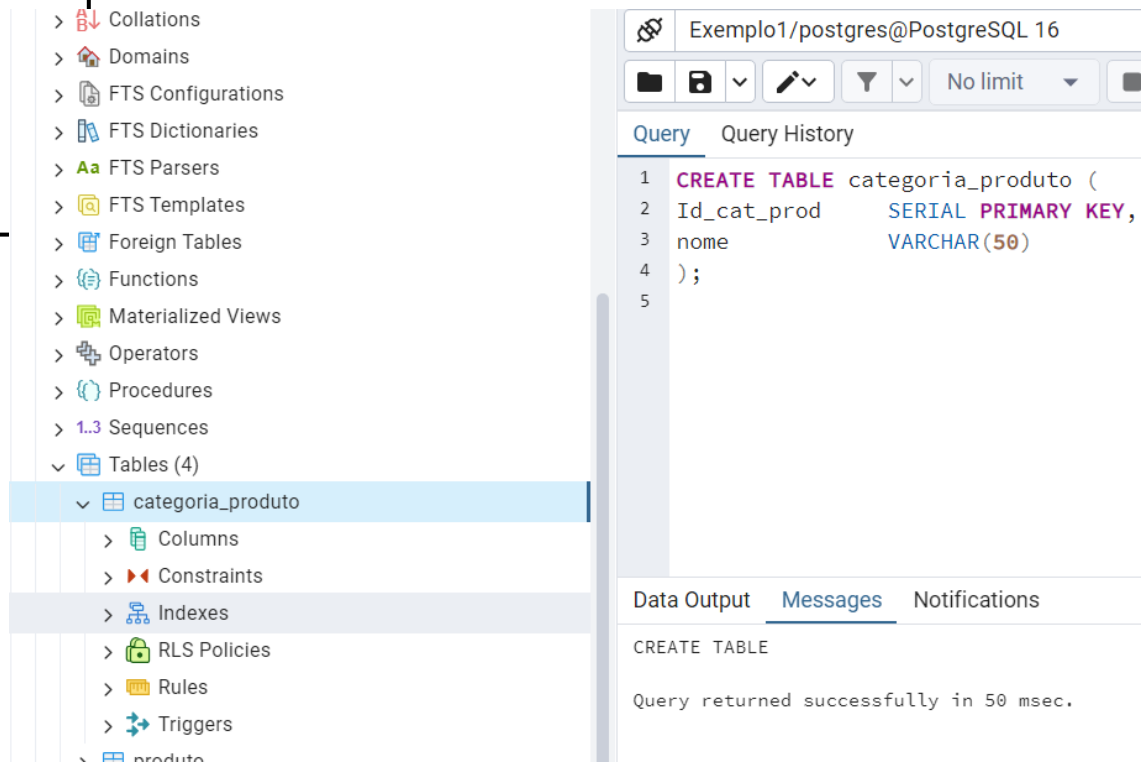
```
INSERT  INTO produto (id, nome, preco, Id_categoria)
VALUES
        (1,'Bola', 35.00, 1),
        (2, 'Patinete', 120.00, 1),
        (3,'Carrinho,', 15.00, 1),
        (4,'Skate,', 296.00, 1),
        (5,'Notebook', 3500.00, 2),
        (6,'Monitor LG 19', 450.00, 2),
        (7,'O Diário de Anne Frank', 45.00, 3),
        (8,'O dia do Curinga', 65.00, 3),
        (9,'O mundo de Sofia', 48.00, 3),
        (10,'Através do Espelho', 38.00, 3);
```



	id [PK] integer	nome character varying (50)	preco real	id_categoria smallint
1	1	Bola	35	1
2	2	Patinete	120	1
3	3	Carrinho,	15	1
4	4	Skate,	296	1
5	5	Notebook	3500	2
6	6	Monitor LG 19	450	2
7	7	O Diário de Anne Frank	45	3
8	8	O dia do Curinga	65	3
9	9	O mundo de Sofia	48	3

Criando um exemplo

```
CREATE TABLE categoria_produto (
    Id_cat_prod         SERIAL PRIMARY KEY,
    nome                VARCHAR(50),
);
```

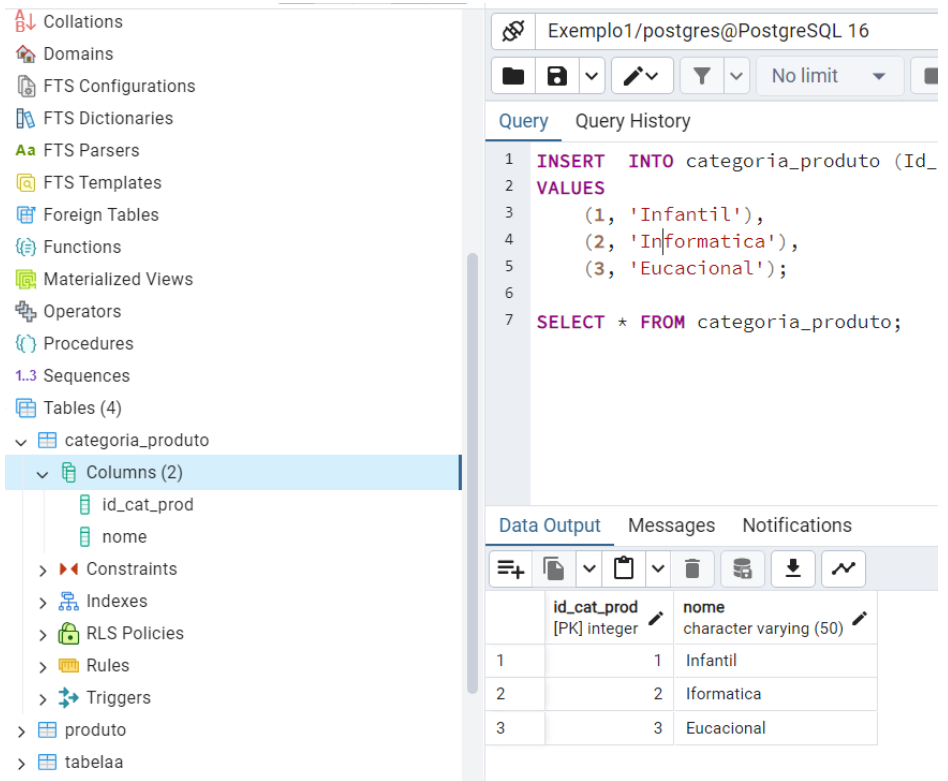


Dados da tabela categoria_produto :

id	nome
1	Infantil
2	Informatica
3	Educacional

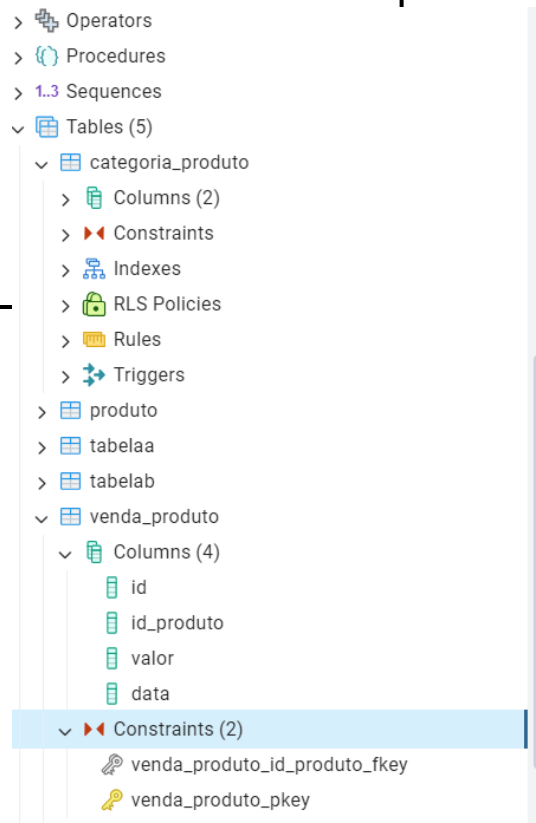
Inserindo

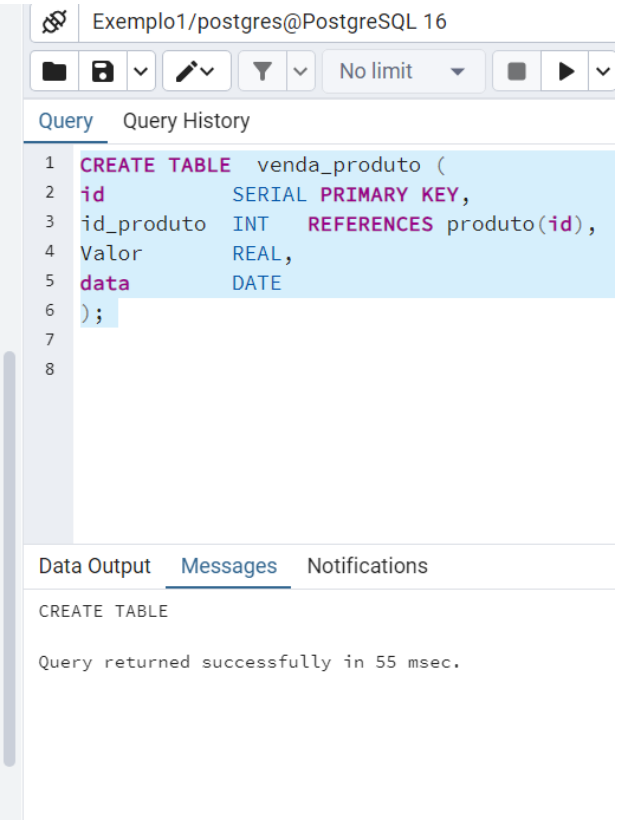
```
INSERT  INTO categoria_produto (id_cat_prod, nome)
VALUES
        (1, 'Infantil'),
        (2, 'Informatica'),
        (3, 'Eucacional');
SELECT * FROM categoria_produto;
```



Criando um exemplo

```
CREATE TABLE  venda_produto (
    id                  SERIAL PRIMARY KEY,
    id_produto          INT REFERENCES produto(id),
    Valor               REAL,
    data                DATE
);
```



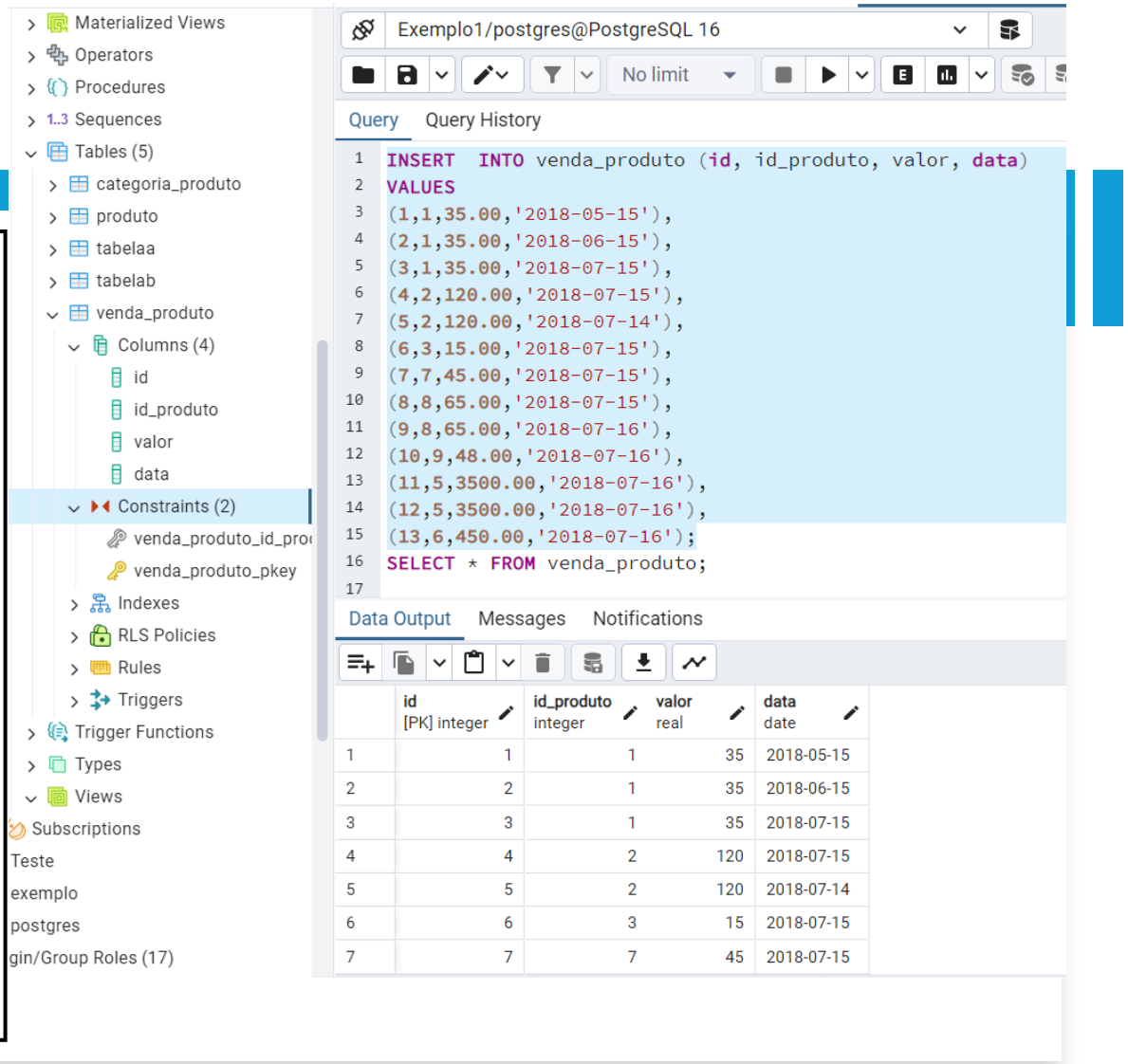


Dados da tabela venda_produto:

id	id_produto	valor	data
1	1	35.00	2018-05-15
2	1	35.00	2018-06-15
3	1	35.00	2018-07-15
4	2	120.00	2018-07-15
5	2	120.00	2018-07-14
6	3	15.00	2018-07-15
7	7	45.00	2018-07-15
8	8	65.00	2018-07-15
9	8	65.00	2018-07-16
10	9	48.00	2018-07-16
11	5	3500.00	2018-07-16
12	5	3500.00	2018-07-16
13	6	450.00	2018-07-16

Inserindo

```
INSERT  INTO venda_produto (id, id_produto, valor, data)
VALUES
    (1,1,35.00,'2018-05-15'),
    (2,1,35.00,'2018-06-15'),
    (3,1,35.00,'2018-07-15'),
    (4,2,120.00,'2018-07-15'),
    (5,2,120.00,'2018-07-14'),
    (6,3,15.00,'2018-07-15'),
    (7,7,45.00,'2018-07-15'),
    (8,8,65.00,'2018-07-15'),
    (9,8,65.00,'2018-07-16'),
    (10,9,48.00,'2018-07-16'),
    (11,5,3500.00,'2018-07-16'),
    (12,5,3500.00,'2018-07-16'),
    (13,6,450.00,'2018-07-16');
```



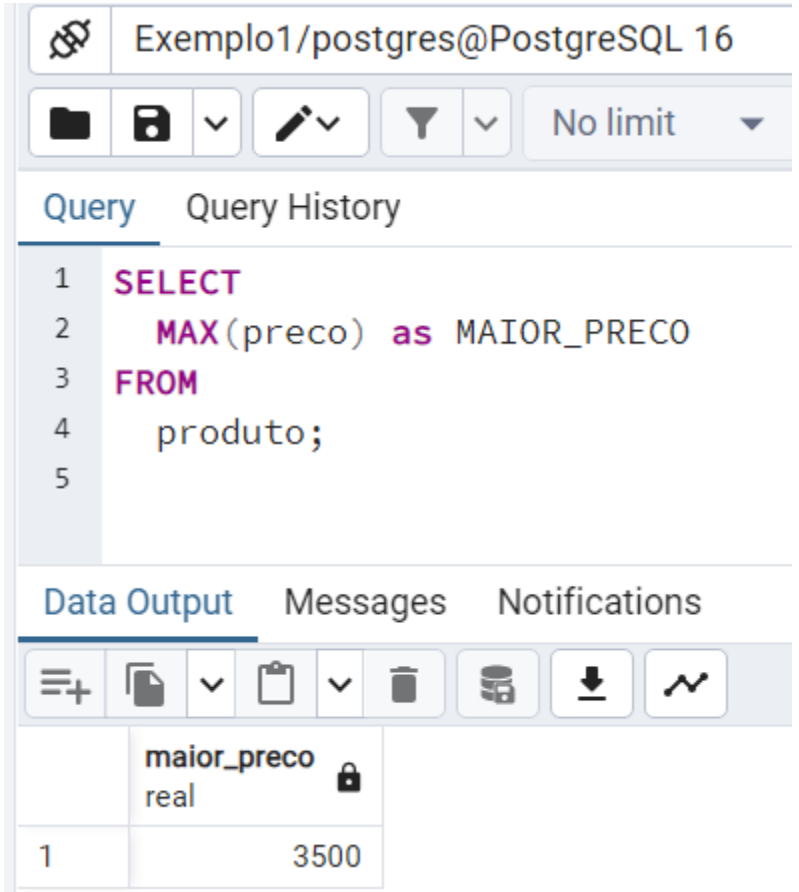
Criando exemplos

Usando MAX, MIN, AVG, SUM e COUNT

Query para MAX

Qual é o valor máximo da coluna preco da tabela produto?

```
SELECT
  MAX(preco) as MAIOR_PRECO
FROM
  produto;
```



Query para MAX

Qual seria a query que nos traga o preço maior por cada categoria cadastrada no banco de dados?

‘

MAX por categoria

id_cat_prod

	id [PK] integer	nome character varying (50)	preco real	id_categoria smallint
1	1	Bola	35	1
2	2	Patinete	120	1
3	3	Carrinho,	15	1
4	4	Skate,	296	1
5	5	Notebook	3500	2
6	6	Monitor LG 19	450	2
7	7	O Diário de Anne Frank	45	3
8	8	O dia do Curinga	65	3
9	9	O mundo de Sofia	48	3
10	10	Através do Espelho	38	3

P

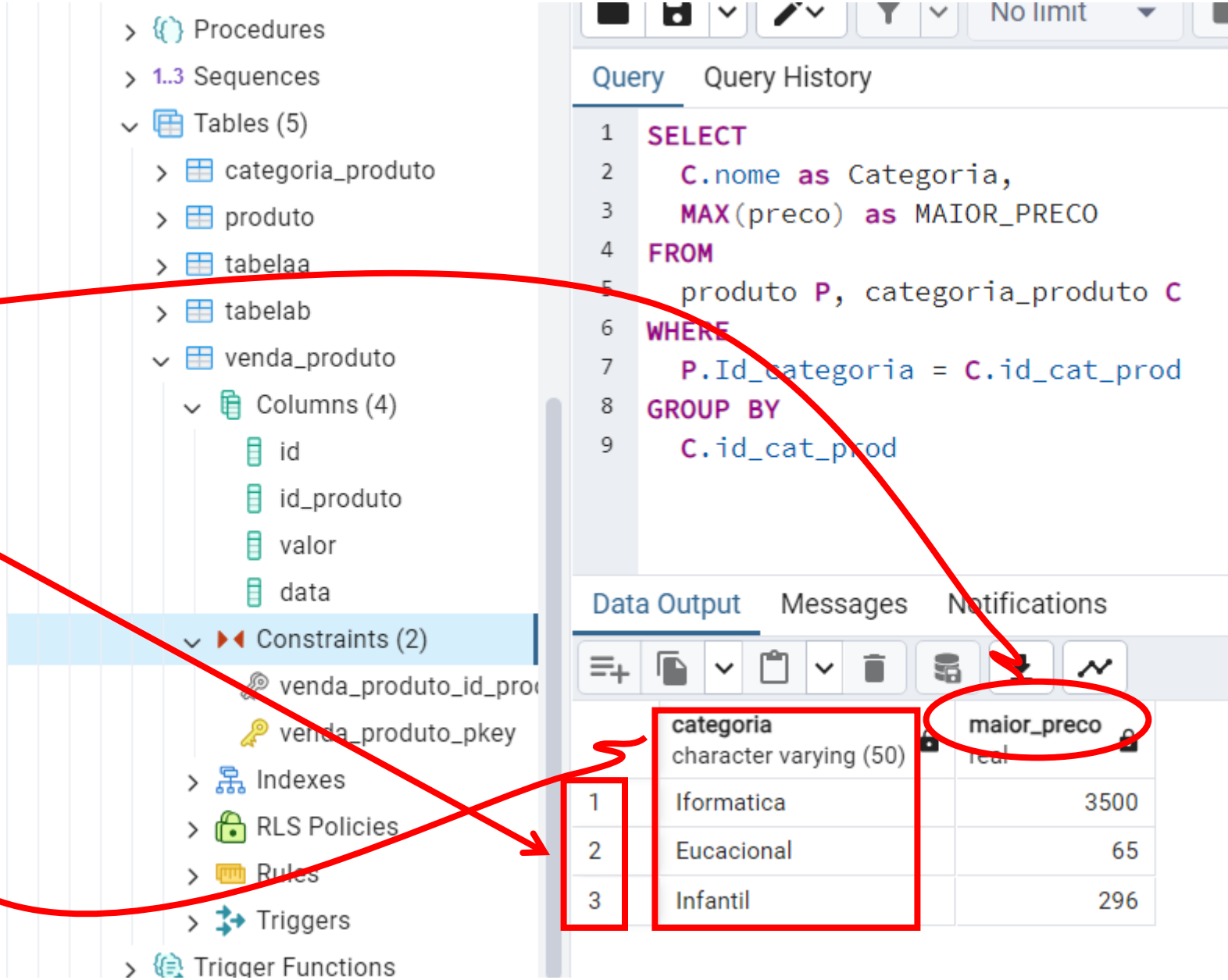
categoria

id	nome
1	Infantil
2	Informatica
3	Educacional

C

	categoria character varying (50)	maior_preco real
1	Iformatica	3500
2	Eucacional	65
3	Infantil	296

```
SELECT
 C.nome as Categoria,
 MAX(preco) as MAIOR_PRECO
FROM
 produto P, categoria_produto C
WHERE
 P.Id_categoria = C.id_cat_prod
GROUP BY
 C.id_cat_prod
```



Query para MIN

Qual seria a query que nos traga o preço menor por cada categoria cadastrada no banco de dados?

```
SELECT
  C.nome as Categoria,
  MIN(preco) as MENOR_PRECO
FROM
  Produto P, categoria_produto C
WHERE
  P.id_categoria = C.id_cat_prod
GROUP BY
  C.id_cat_prod
```

Query | Query History

```
1  SELECT
2    C.nome as Categoria,
3    MIN(preco) as MENOR_PRECO
4  FROM
5    Produto P, categoria_produto C
6  WHERE
7    P.id_categoria = C.id_cat_prod
8  GROUP BY
9    C.id_cat_prod
```

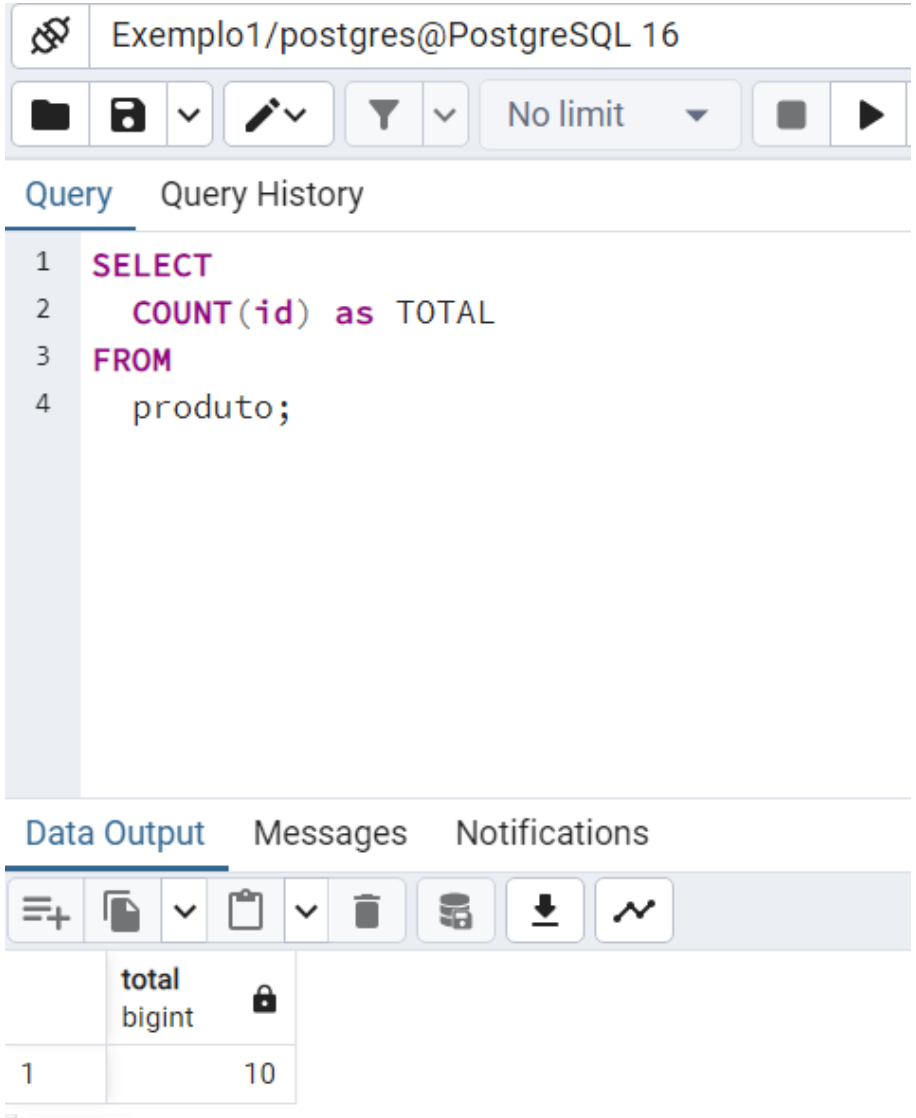
Data Output | Messages | Notifications

	categoria character varying (50)	menor_preco real
1	Iformatica	450
2	Eucacional	38
3	Infantil	15

Query para COUNT

Qual seria a query que nos traga o número de determinado elemento?

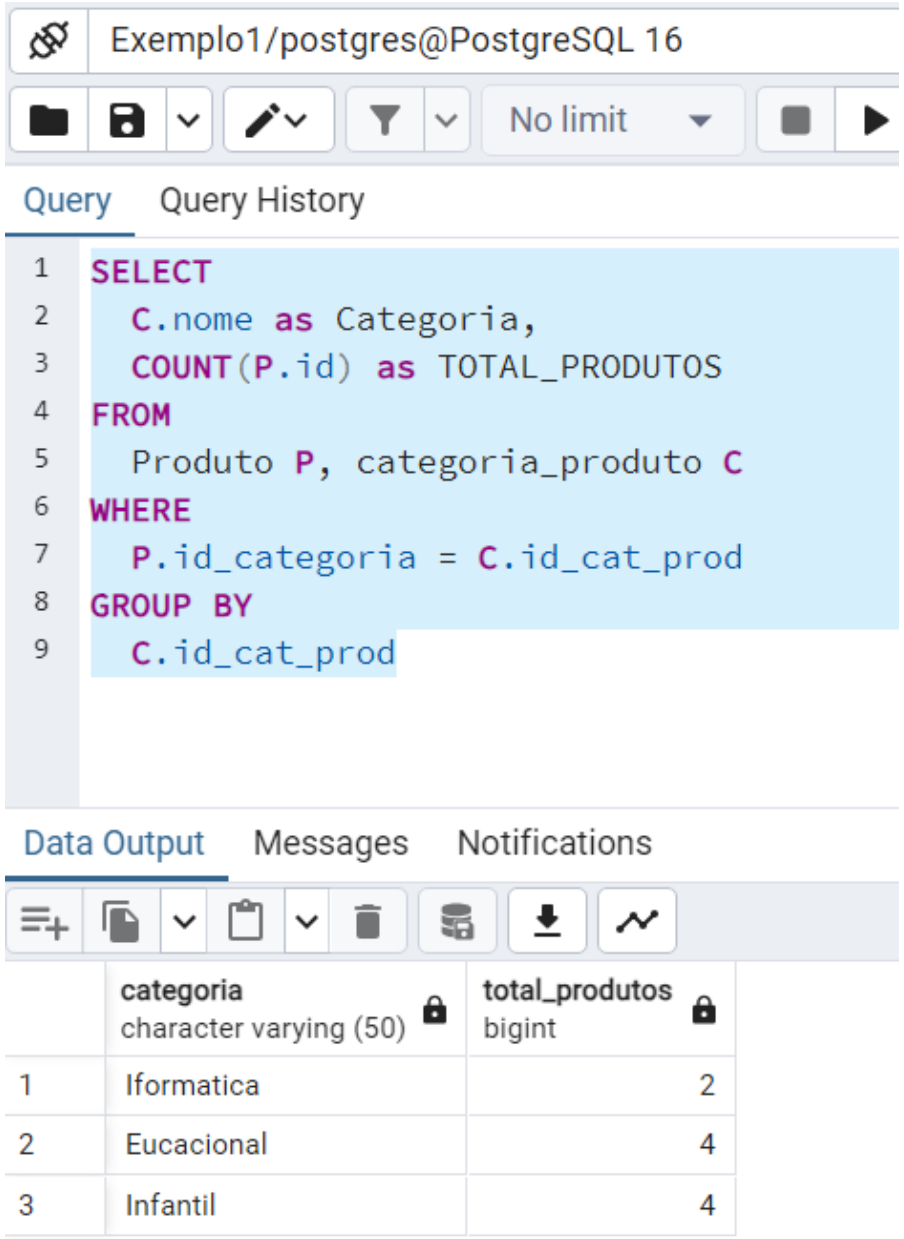
```
SELECT
  COUNT(id) as TOTAL
FROM
  produto;
```



Query para COUNT

Qual seria a query para a quantidade total de produtos, porém dessa vez agrupados por categoria, ou seja, saberemos exatamente qual a quantidade total de produtos por categoria?

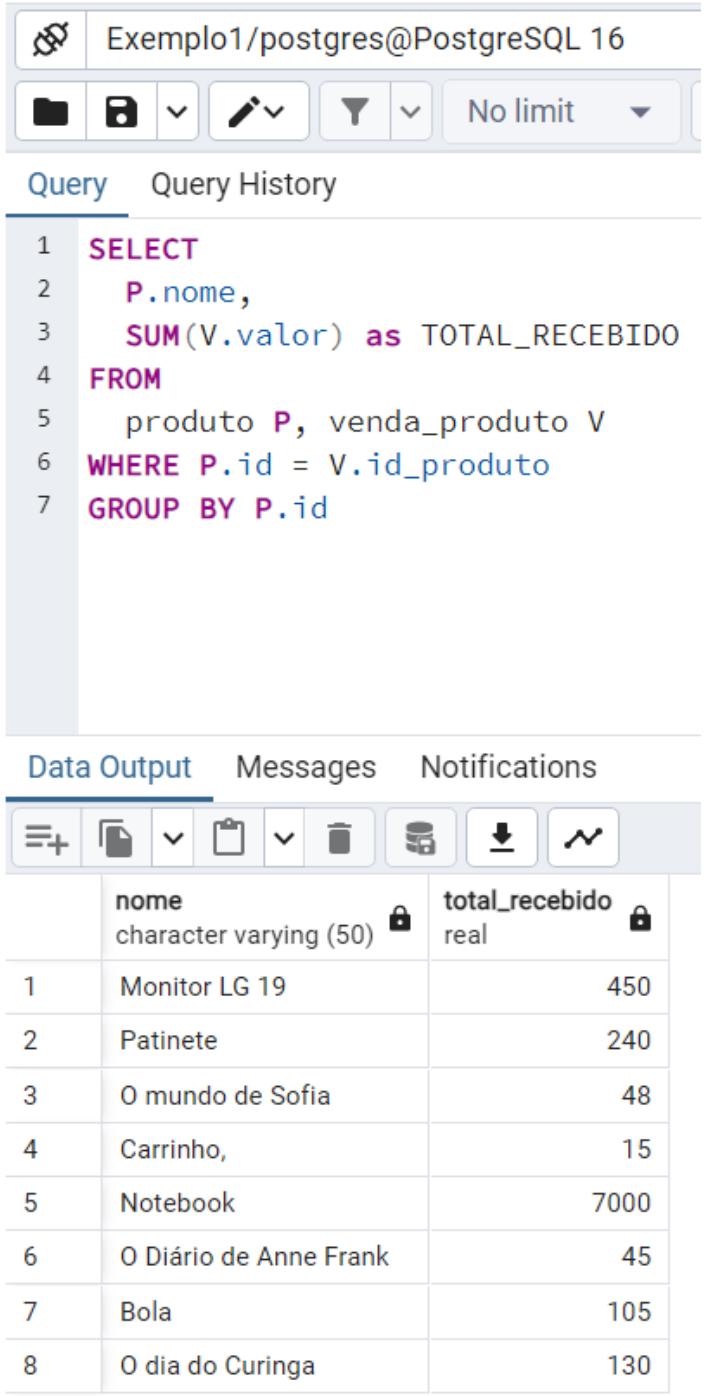
```
SELECT
  C.nome as Categoria,
  COUNT(P.id) as TOTAL_PRODUTOS
FROM
  Produto P, categoria_produto C
WHERE
  P.id_categoria = C.id_cat_prod
GROUP BY
  C.id_cat_prod
```



Query para SUM

Suponhamos agora que seja necessário saber exatamente quanto ganhamos com cada produto até o momento, ou seja, olharemos na tabela vendas_produto o valor total recebido com cada produto?

```
SELECT
  P.nome,
  SUM(V.valor) as TOTAL_RECEBIDO
FROM
    produto P, venda_produto V
WHERE P.id = V.id_produto
GROUP BY P.id
```

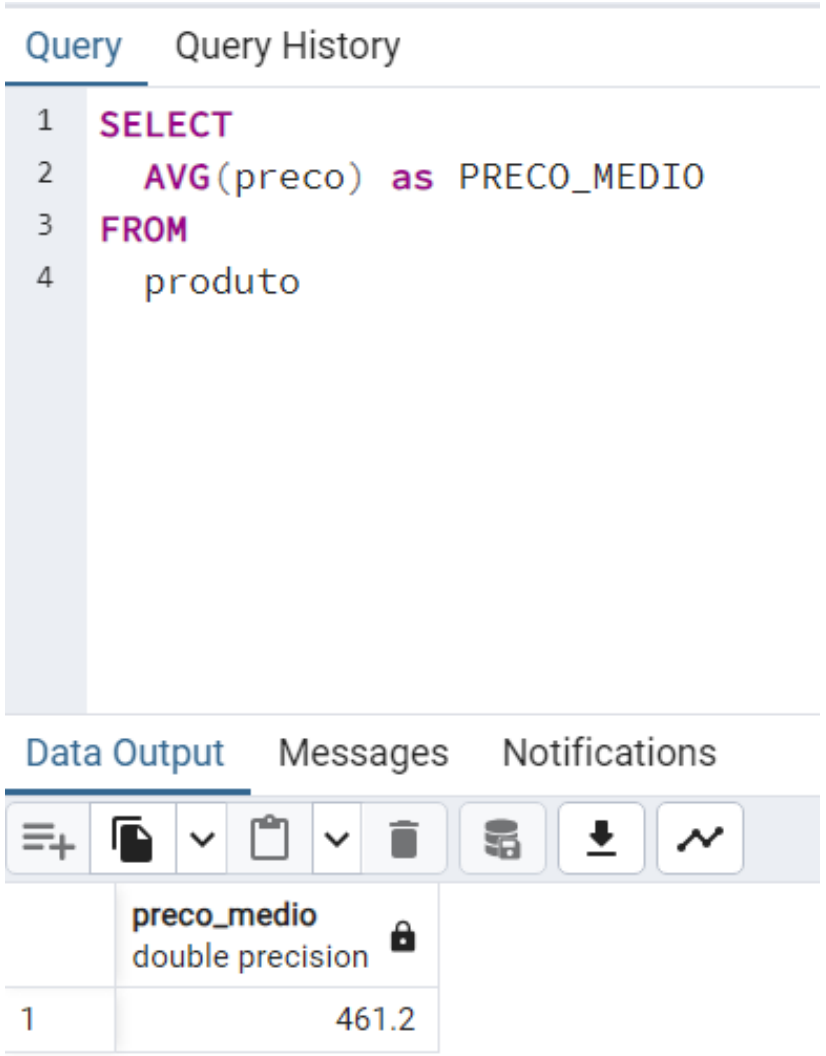


	nome character varying (50)	total_recebido real
1	Monitor LG 19	450
2	Patinete	240
3	O mundo de Sofia	48
4	Carrinho,	15
5	Notebook	7000
6	O Diário de Anne Frank	45
7	Bola	105
8	O dia do Curinga	130

Query para AVG

Como fica a query quando precisamos da média de valores de uma determinada coluna informada na consulta?

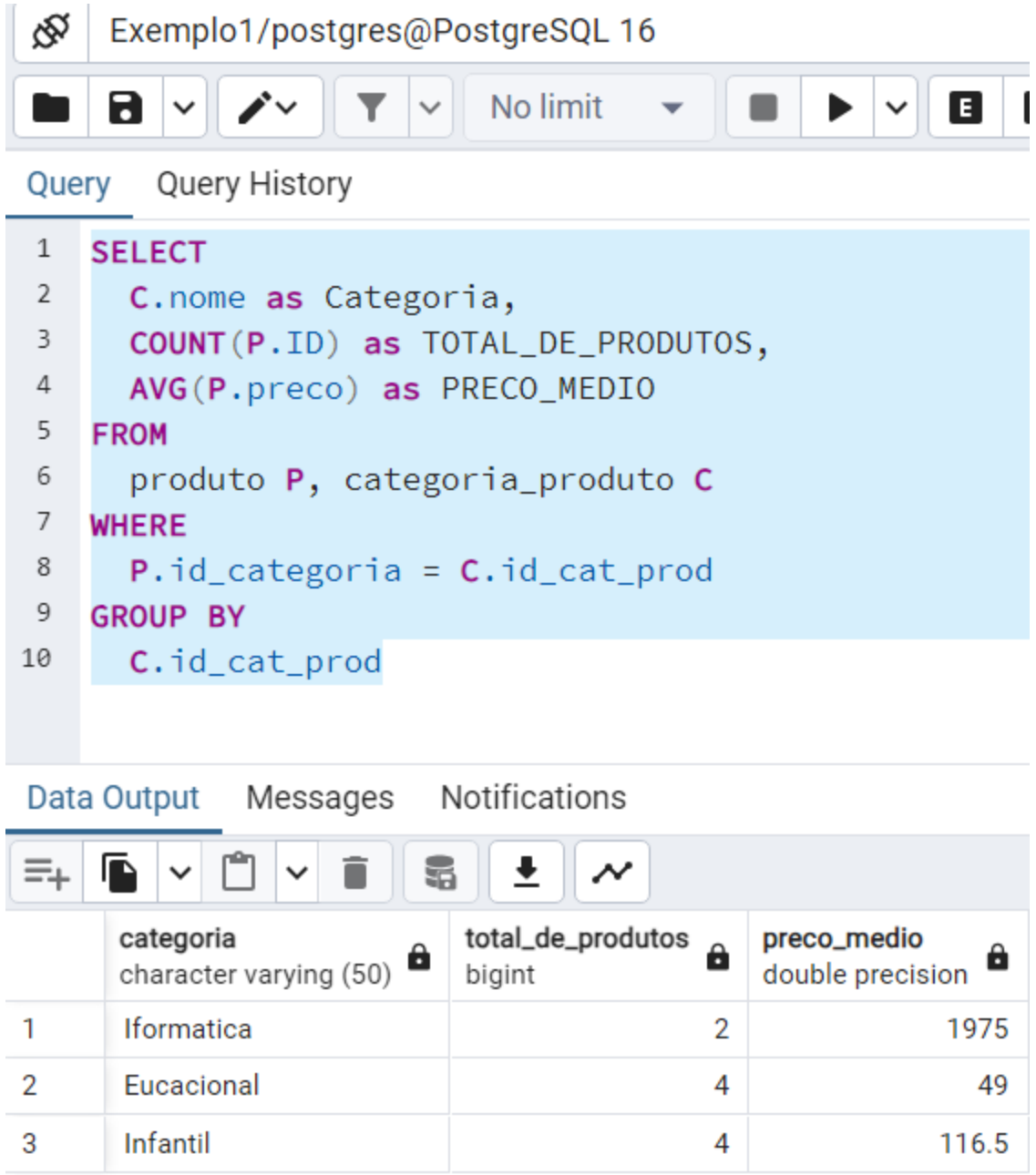
```
SELECT
  AVG(preco) as PRECO_MEDIO
FROM
  produto;
```



Query para AVG

Agora precisamos obter a média de preço de todos os produtos, agrupados por categoria, trazendo também a quantidade de produtos em cada uma delas?

```
SELECT
  C.nome as Categoria,
  COUNT(P.ID) as
TOTAL_DE_PRODUTOS,
  AVG(P.preco) as PRECO_MEDIO
FROM
  produto P, categoria_produto C
WHERE
  P.id_categoria = C.id_cat_prod
GROUP BY
  C.id_cat_prod
```



Window Functions em SQL

Window functions (funções de janela)

As em SQL são uma ferramenta poderosa e versátil para analisar e processar dados em conjuntos de dados complexos.

Em Ciência de dados, utilizar essas funções pode elevar as habilidades de manipulação e análise de dados do programador.

O que são Window Functions em SQL?

Em termos simples, as window functions em SQL permitem que se realize cálculos ou agregações em um conjunto específico de linhas relacionadas a uma linha de dados específica.

Essas funções operam sobre uma "janela" de dados que é definida com base em condições específicas, como uma partição ou ordenação.

Window Functions

As Windows Functions são trabalhadas com um conjunto de linhas definidas por uma cláusula OVER, que permite trabalhar com totais, agrupamentos, ordenações, cálculos complexos dentre outros.

Assim é possível melhorar a performance com ordenações avançadas, além de limitarmos o número de linhas que serão retornadas em um subconjunto de dados associados a uma determinada tabela.

As funções de agregação que são definidas pelo usuário também podem atuar como Windows Functions quando estas possuem uma chamada com a palavra-chave OVER.

Windows Functions PostgreSQL são 11

Existem várias funções de janela essenciais, incluindo

- ROW_NUMBER()
- RANK()
- DENSE_RANK()
- LEAD()
- LAG()
- CUME_DIST()
- PRESENT_RANK()
- FIRST_VALUE()
- LAST_VALUE()
- NTH_VALUE()
- NTILE()

Construindo um exemplo

Criando uma tabela

```
CREATE TABLE funcionarios_windows_function
    (
      codigo_func integer NOT NULL,
      nome_func character varying(100) NOT NULL,
      profissao character varying(100) NOT NULL,
      nome_departamento character varying(100) NOT NULL,
      departamento_cod integer NOT NULL,
      salario real,
      CONSTRAINT funcionarios_windows_function_pkey PRIMARY KEY (codigo_func)
    )
    WITH (
      OIDS=FALSE
    );
```

Populando com dados

```
INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, departamento_cod, SALARIO) VALUES (1, 'Edson Dionisio', 'Desenvolvedor Web', 'Desenvolvimento web', 10, 2000.00);

INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, departamento_cod, SALARIO) VALUES (2, 'Marília Késsia', 'Scrum Master', 'Desenvolvimento', 10, 6000.00);

INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, departamento_cod, SALARIO) VALUES (3, 'Caroline França', 'Desenvolvedor Android', 'Mobile', 30, 2500.00);

INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, departamento_cod, SALARIO) VALUES (4, 'Gustavo França', 'Desenvolvedor IOS', 'Mobile', 30, 2800.00);

INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, departamento_cod, SALARIO) VALUES (5, 'Renato silva', 'Desenvolvedor de Sistemas', 'Desenvolvimento de Sistemas', 10, 2000.00);

INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, departamento_cod, SALARIO) VALUES (6, 'João dos testes', 'Analista de Testes', 'Testes', 16, 2000.00);

INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, departamento_cod, SALARIO) VALUES (7, 'Maria das dores', 'Análista de Software', 'Engenharia de software', 12, 3000.00);

INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, departamento_cod, SALARIO) VALUES (8, 'Rodrigo Sampaio', 'Desenvolvedor Windows Phone', 'Mobile', 30, 2600.00)
```

Conferindo...

- SELECT * FROM funcionarios_windows_function

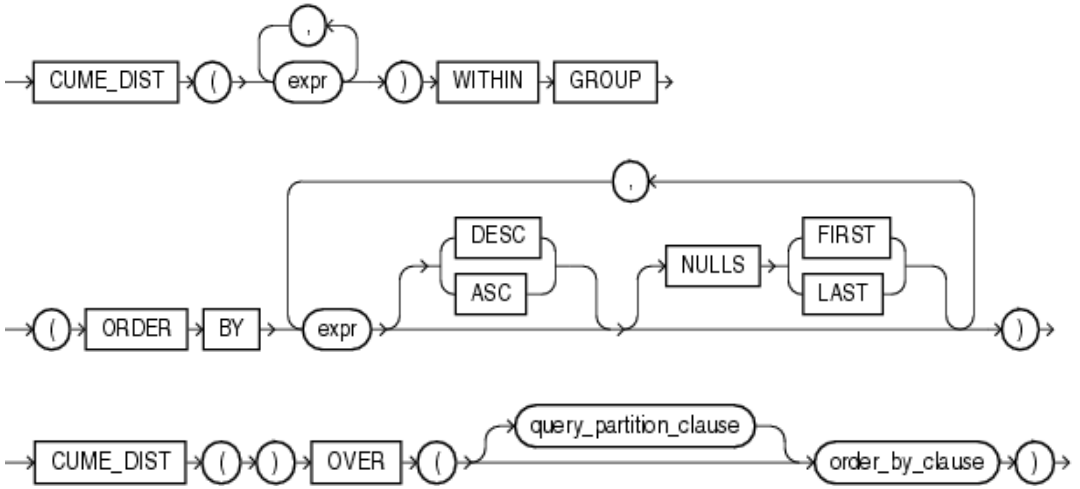
Query | Query History

```
14
15 INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, depa
16 INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, depa
17 INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, depa
18 INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, depa
19 INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, depa
20 INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, depa
21 INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, depa
22 INSERT INTO FUNCIONARIOS_WINDOWS_FUNCTION (codigo_func, nome_func, profissao, nome_departamento, depa
23
24 SELECT * FROM funcionarios_windows_function
25
```

Data Output | Messages | Notifications

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real
1	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000
2	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000
3	3	Caroline França	Desenvolvedor Android	Mobile	30	2500
4	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800
5	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000
6	6	João dos testes	Analista de Testes	Testes	16	2000
7	7	Maria das dores	Análista de Software	Engenharia de software	12	3000
8	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600

CUME_DIST



CUME_DIST()

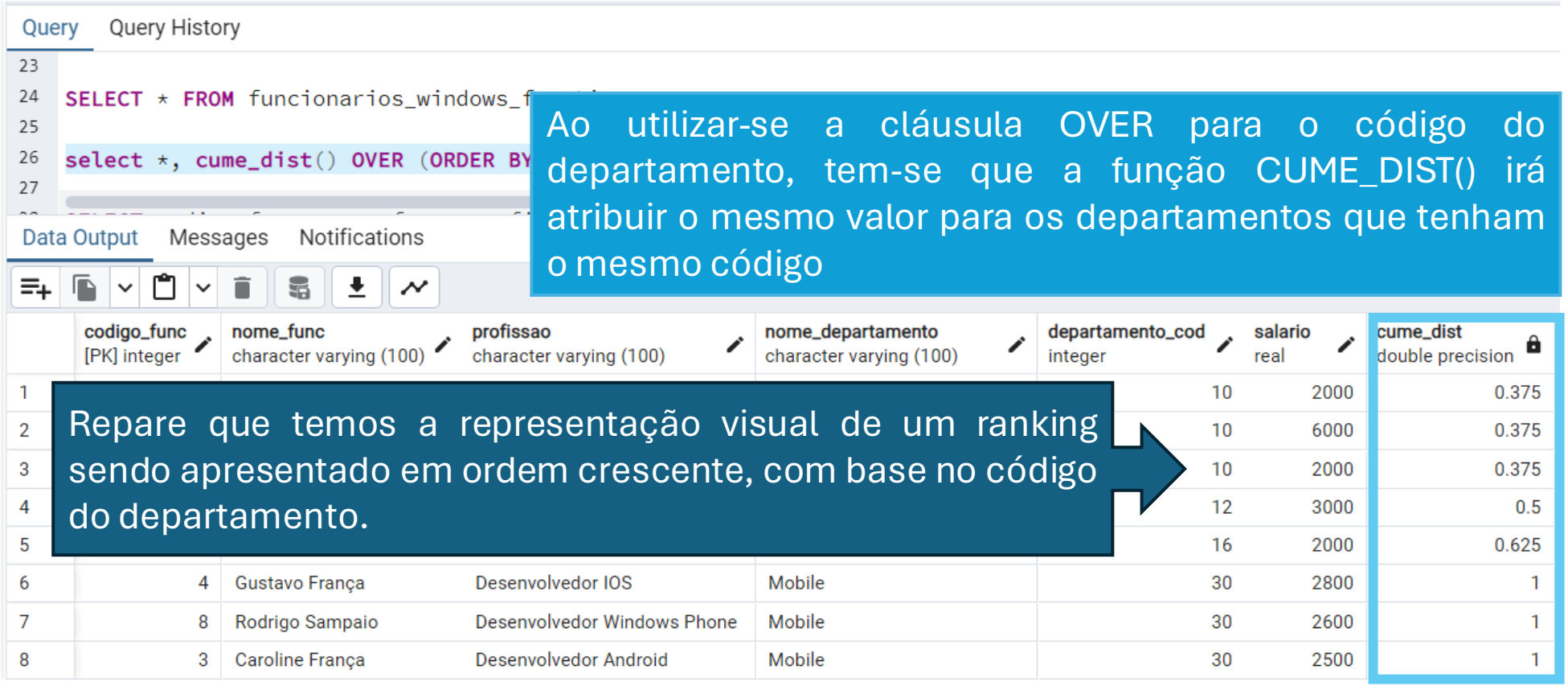
Descrição: Esta é utilizada com o intuito de obtermos a classificação da linha atual. Para que este resultado seja obtido é realizado um cálculo no qual ocorre a divisão do número de linhas anteriores a linha atual pelo total de linhas encontradas. A

Fórmula a seguir mostra essa relação, onde o tipo de retorno é o double precision:

Linha atual = (Número de linhas anteriores a linha atual)/(número total de linhas)

- Após isso, utilizaremos a consulta com o SELECT anterior, dessa vez adicionando a função CUME_DIST(), para ver o resultado da operação, conforme a seguinte instrução:

```
select *, CUME_DIST() OVER (ORDER BY departamento_cod) from FUNCIONARIOS_WINDOWS_FUNCTION;
```



	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	cume_dist double precision
1					10	2000	0.375
2					10	6000	0.375
3					10	2000	0.375
4					12	3000	0.5
5					16	2000	0.625
6	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	1
7	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	1
8	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	1

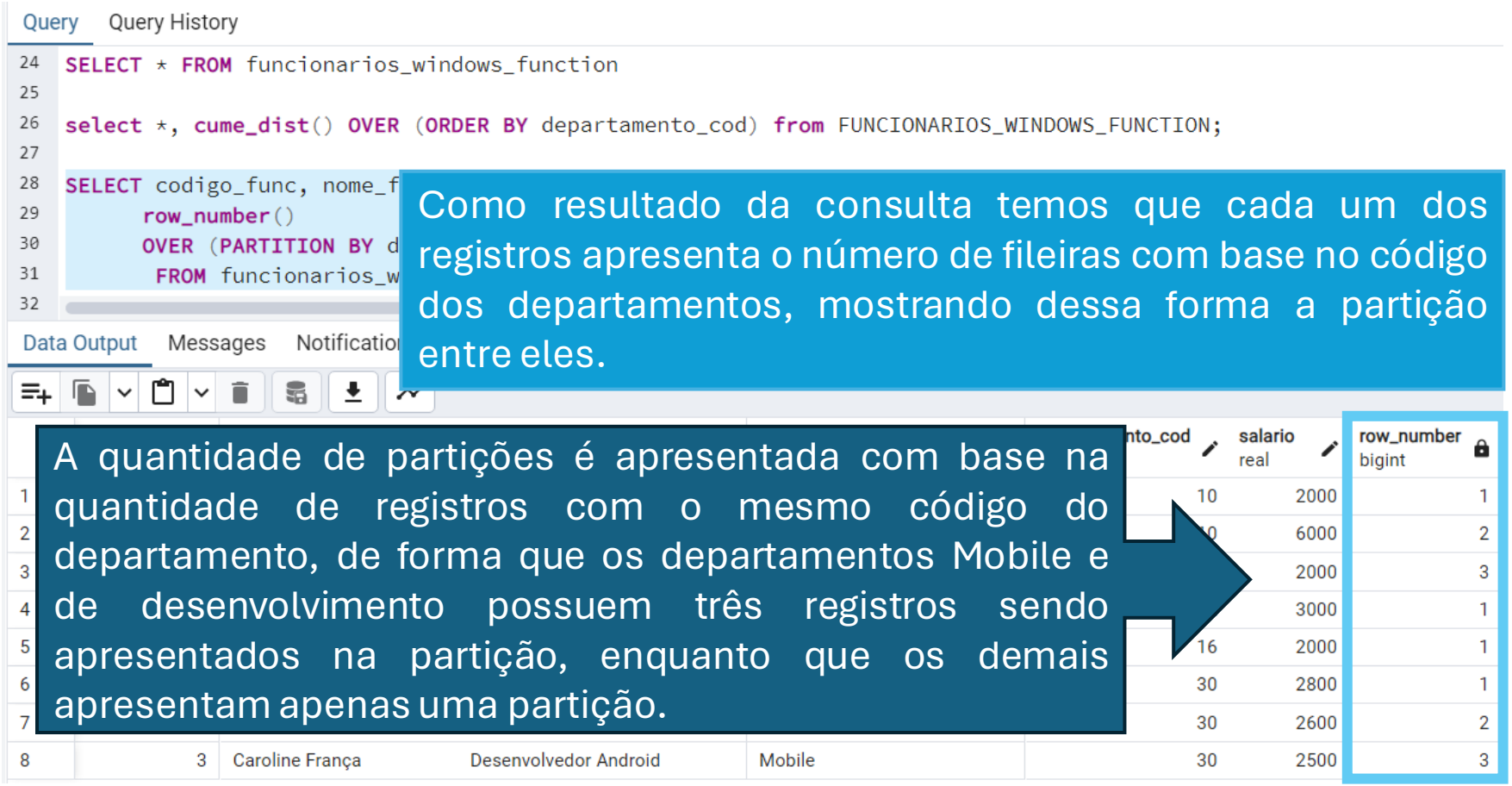
ROW_NUMBER

ROW_NUMBER():

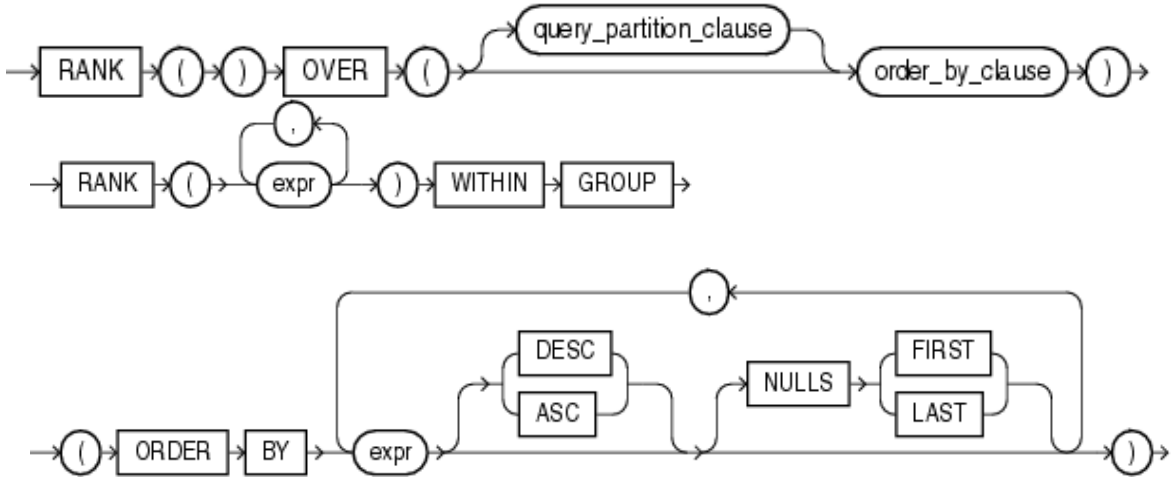
- **Descrição**: A função ROW_NUMBER() atribui um número sequencial único para cada linha em um conjunto de resultados, baseado na ordem especificada.
- **Uso Prático**: Útil quando você precisa de uma identificação única para cada linha.

ROW_NUMBER

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    ROW_NUMBER()
    OVER (PARTITION BY departamento_cod)
     FROM funcionarios_windows_function;
```



RANK

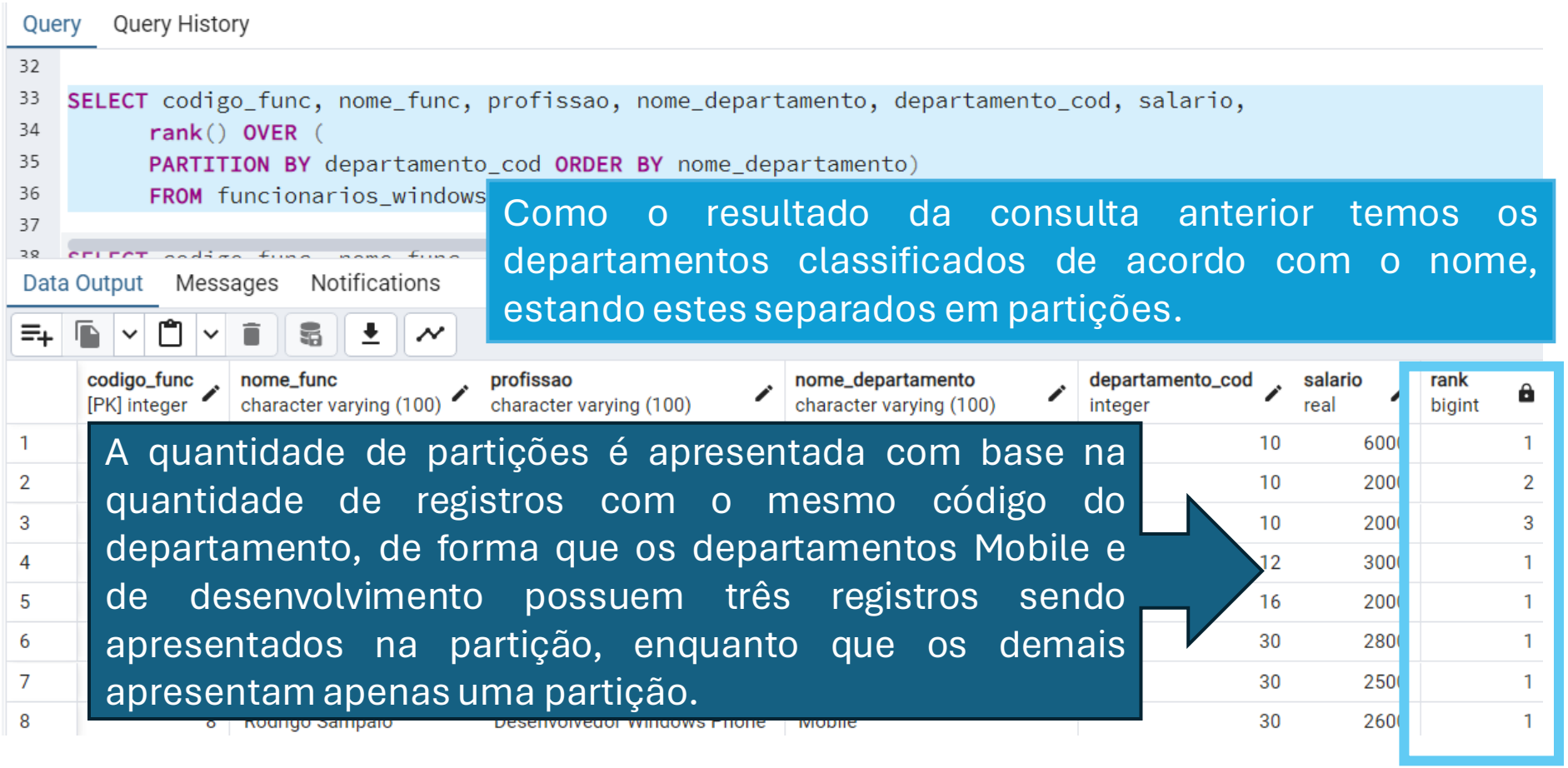


RANK():

- **Descrição:** A função **RANK()** atribui uma classificação única para cada linha com base no valor especificado. Valores iguais recebem a mesma classificação, e o próximo valor recebe a classificação subsequente.
- **Uso Prático:** Útil para identificar a posição relativa de valores em uma ordem específica.

RANK

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    RANK() OVER (
    PARTITION BY departamento_cod ORDER BY nome_departamento)
    FROM funcionarios_windows_function;
```



RANK

Agora vamos considerar o seguinte caso: ao invés de utilizarmos a função para classificarmos os registros com base no código do departamento, vamos fazer a consulta baseada no nome do departamento.

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    RANK() OVER (ORDER BY nome_departamento)
     FROM funcionarios_windows_function;
```

Query | Query History

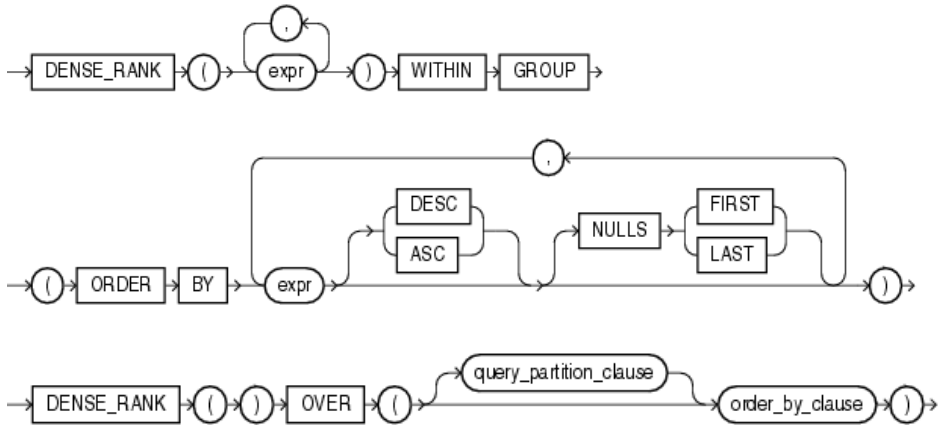
```
37
38  SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
39        rank() OVER (ORDER BY nome_departamento)
40          FROM funcionarios_windows_f
41
```

A nova consulta muda o resultado, assim a classificação será com base no nome do departamento, atribuindo o mesmo valor classificatório para os itens repetidos

Data Output | Messages | Notifications

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	rank bigint
1	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	1
2	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	2
3	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	3
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	4
5	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	5
6	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	5
7	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	5
8	6	João dos testes	Analista de Testes	Testes	16	2000	8

DENSE_RANK



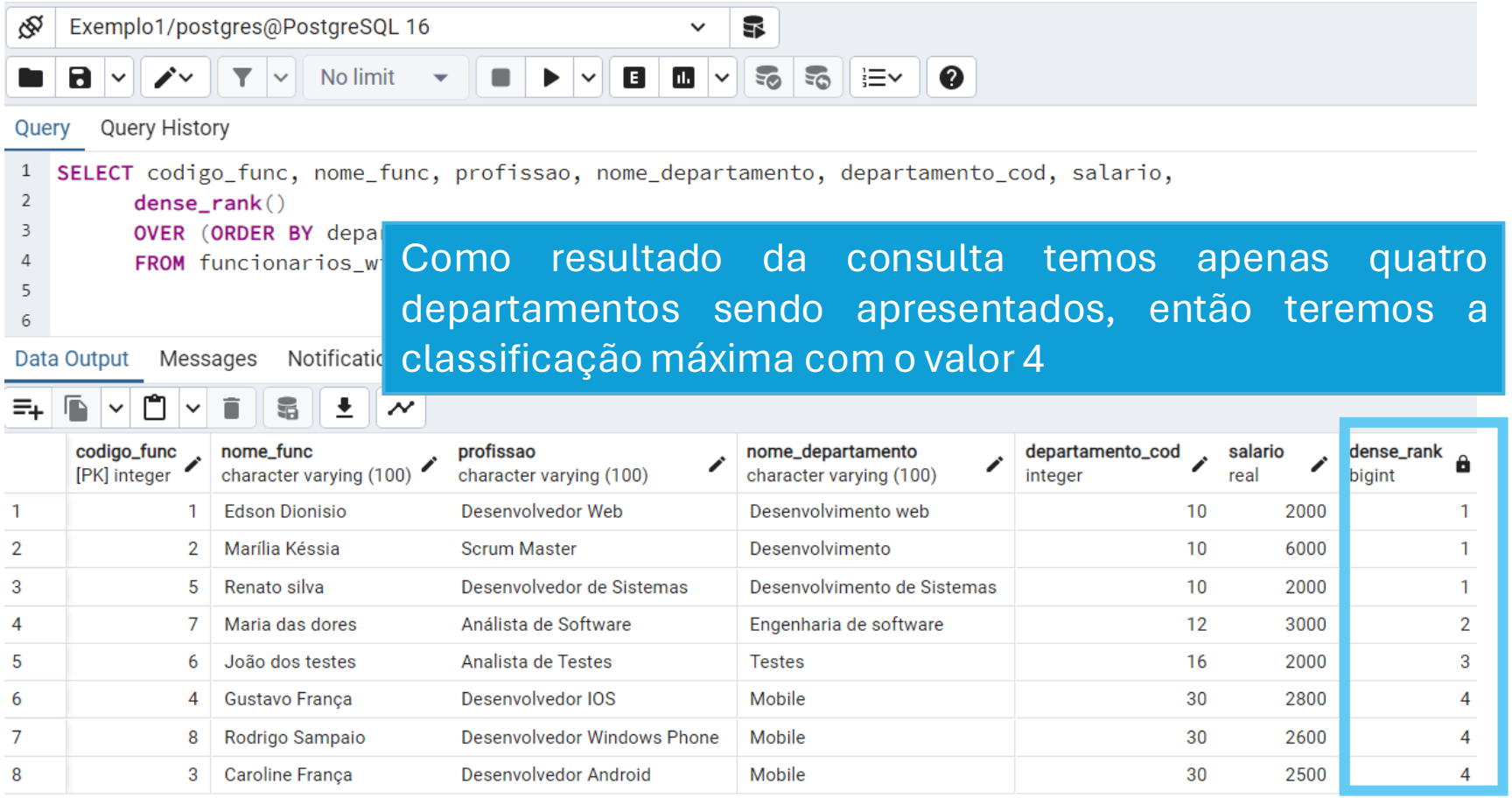
DENSE_RANK():

- **Descrição:** Similar ao **RANK()**, a função **DENSE_RANK()** também atribui classificações únicas, mas sem pular classificações para valores iguais.
- **Uso Prático:** Útil quando você deseja evitar lacunas nas classificações para valores iguais.

DENSE_RANK

Utilização da coluna “departamento_cod”.

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    dense_rank()
    OVER (ORDER BY departamento_cod)
    FROM funcionarios_windows_function;
```



	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	dense_rank bigint
1	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	1
2	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	1
3	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	1
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	2
5	6	João dos testes	Analista de Testes	Testes	16	2000	3
6	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	4
7	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	4
8	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	4

DENSE_RANK

Utilização da coluna “nome_departamento”.

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    dense_rank()
    OVER (ORDER BY nome_departamento)
    FROM funcionarios_windows_function;
```

Query | Query History

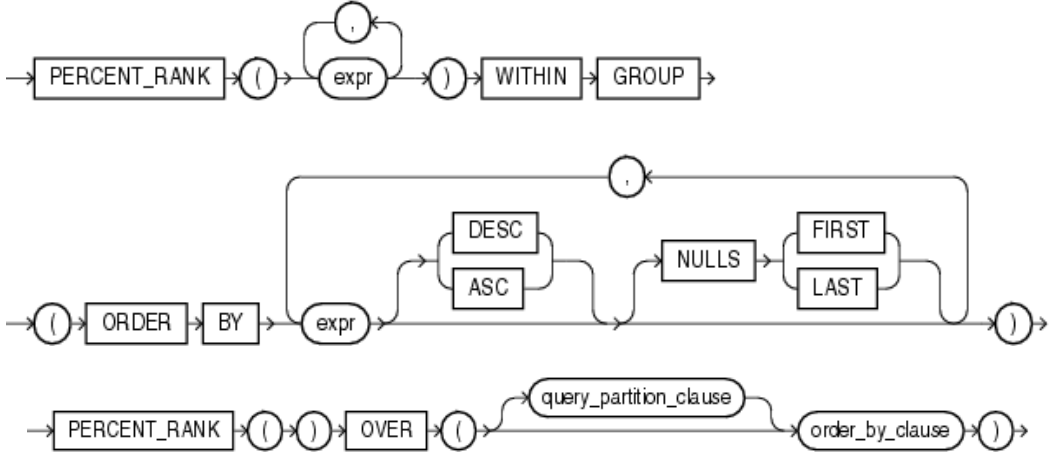
```
1
2  SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
3         dense_rank()
4         OVER (ORDER BY nome_
5         FROM funcionarios_wi
6
```

Como resultado desta nova consulta obtivemos o rank gerado até a sexta posição, pois temos seis departamentos distintos sendo apresentados.

Data Output | Messages | Notificatio

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	dense_rank bigint
1	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	1
2	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	2
3	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	3
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	4
5	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	5
6	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	5
7	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	5
8	6	João dos testes	Analista de Testes	Testes	16	2000	6

PERCENT_RANK()



PERCENT_RANK

- **Descrição:** Semelhante à função CUME_DIST (distribuição cumulativa). O intervalo de valores retornado por PERCENT_RANK é de 0 a 1, inclusive. A primeira linha em qualquer conjunto tem PERCENT_RANK igual a 0. O valor de retorno é NUMBER.

PERCENT_RANK()

PERCENT_RANK

- Quando precisamos obter uma classificação relativa das classificações, podemos utilizar a função percent_rank(), que é utilizada para obtermos a classificação relativa da linha atual. Para que tenhamos a posição relativa da linha atual, realizamos o cálculo com base na seguinte fórmula:

Posição relativa da linha atual = (rank - 1) / (número total de linhas - 1)

PERCENT_RANK

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    percent_rank()
    OVER (PARTITION BY departamento_cod ORDER BY profissao)
    FROM funcionarios_windows_function;
```

Query | Query History

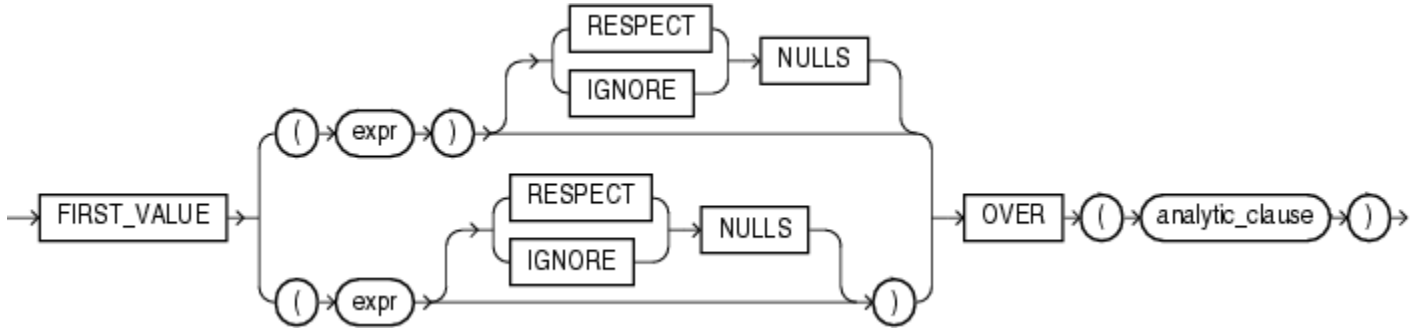
```
1  SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
2         percent_rank()
3         OVER (PARTITION BY departamento_cod ORDER BY profissao)
4         FROM funcionarios_wi
5
6
```

Neste caso, para os departamentos com o mesmo código, obtivemos os valores de 0, 0.5 e 1. Enquanto que os demais registros obtiveram o valor 0..

Data Output | Messages | Notification

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	percent_rank
1	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	0
2	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	0.5
3	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	1
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	0
5	6	João dos testes	Analista de Testes	Testes	16	2000	0
6	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	0
7	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	0.5
8	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	1

FIRST_VALUE()

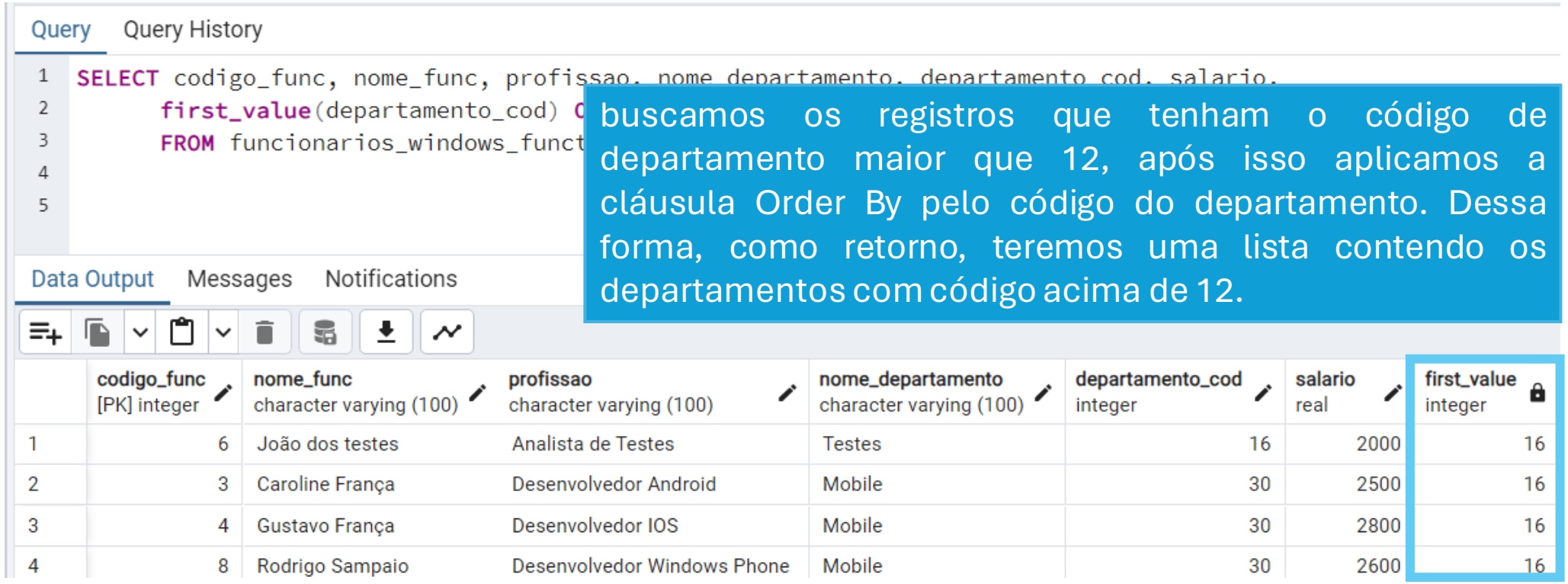


FIRST_VALUE

- **Descrição:** FIRST_VALUE é uma função analítica. Ele retorna o primeiro valor em um conjunto ordenado de valores. Se o primeiro valor do conjunto for nulo, a função retornará NULL, a menos que você especifique IGNORE NULLS.
- **Uso Prático:** Esta configuração é útil para densificação de dados.

FIRST_VALUE

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    first_value(departamento_cod) OVER (ORDER BY departamento_cod)
    FROM funcionarios_windows_function WHERE departamento_cod > 12;
```



buscamos os registros que tenham o código de departamento maior que 12, após isso aplicamos a cláusula Order By pelo código do departamento. Dessa forma, como retorno, teremos uma lista contendo os departamentos com código acima de 12.

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	first_value integer
1	6	João dos testes	Analista de Testes	Testes	16	2000	16
2	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	16
3	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	16
4	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	16

FIRST_VALUE

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    first_value(departamento_cod) OVER (ORDER BY nome_departamento)
    FROM funcionarios_windows_function WHERE departamento_cod > 12;
```

Query | Query History

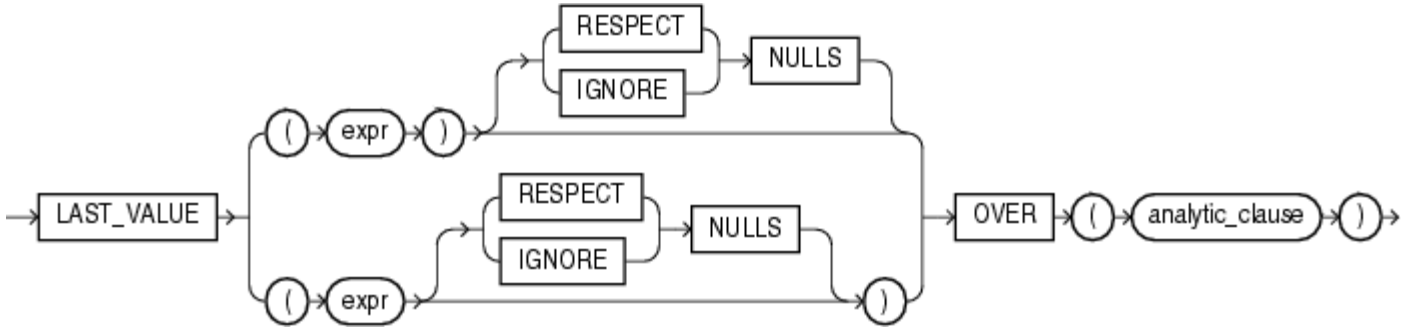
```
1  SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
2        first_value(departamento_cod) OVER (ORDER BY nome_departamento)
3        FROM funcionarios_windows_function WHERE departamento_cod > 12;
4
5
```

Num segundo exemplo, utiliza-se a cláusula order by para ordenarmos os registros por nome do departamento, onde com isso, obteremos um valor diferente para o valor da função first_value()

Data Output | Messages | Notifications

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	first_value integer
1	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	30
2	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	30
3	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	30
4	6	João dos testes	Analista de Testes	Testes	16	2000	30

LAST_VALUE()



LAST_VALUE

- **Descrição:** Ao contrário da função first_value(), a função last_value() é utilizada para a obtenção do valor presente na última linha de registro presente na tabela, onde utilizamos o nome da coluna como argumento, de igual forma a função anterior.

LAST_VALUE

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    last_value(departamento_cod) OVER (ORDER BY nome_departamento)
    FROM funcionarios_windows_function;
```

Query | Query History

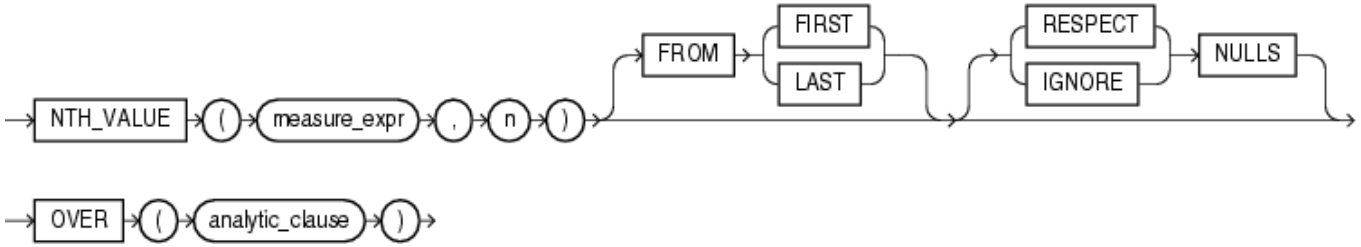
```
1  SELECT codigo_func, nome_func,
2         last_value(departamento_c
3         FROM funcionarios_windows
4
5
```

Após a consulta, obtivemos o resultado presente na figura a seguir, onde o nosso último registro, com base na ordenação por nome de departamento, é o departamento de testes, que corresponde ao código do departamento 16.

Data Output | Messages | Notifications

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	last_value integer
1	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	10
2	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	10
3	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	10
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	12
5	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	30
6	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	30
7	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	30
8	6	João dos testes	Analista de Testes	Testes	16	2000	16

NTH_VALUE()



NTH_VALUE

- **Descrição:** A função nth_value(), possibilita receber um valor diferente do inicial e do final, obtendo assim um valor presente na enésima linha da tabela. Para utilizar esta função passamos o nome da coluna desejada e o enésimo número como argumentos de entrada. Caso o valor informado não seja encontrado na tabela, o valor apresentado pela função será nulo.

NTH_VALUE

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    last_value(departamento_cod) OVER (ORDER BY nome_departamento)
    FROM funcionarios_windows_function;
```

Query | Query History

```
1  SELECT codigo_func, nome_func, prof
2         nth_value(nome_departamento,
3         OVER (PARTITION BY departamen
4         FROM funcionarios_windows_fun
5
6
```

Veja que estamos utilizando a cláusula PARTITION BY para dividirmos os registros com base no código do departamento, onde cada partição criada terá um número de saída, que será o valor que utilizaremos para a função nth_value.

Data Output | Messages | Notifications

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	nth_value character varying
1	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	[null]
2	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	Desenvolvimento de Sistemas
3	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	Desenvolvimento de Sistemas
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	[null]
5	6	João dos testes	Analista de Testes	Testes	16	2000	[null]
6	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	Mobile
7	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	Mobile
8	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	Mobile

NTILE()

NTILE

- **Descrição:** Esta função permite atribuir valores para grupos de resultados, ou seja, um número inteiro a eles. Para melhor entendermos a sua utilização veremos o exemplo com 2 e 3 partições.

NTILE

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    ntile(2)
    OVER (ORDER BY departamento_cod)
    FROM funcionarios_windows_function;
```

Query | Query History

```
1  SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
2        ntile(2)
3        OVER (ORDER BY departamento_cod)
4        FROM funcionarios_windows_function;
5
6
```

Data Output | Messages | Notifications

A tabela está dividida em duas partições pela função ntile()

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	ntile integer
1	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	1
2	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	1
3	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	1
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	1
5	6	João dos testes	Analista de Testes	Testes	16	2000	2
6	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	2
7	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	2
8	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	2

NTILE

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    ntile(3)
    OVER (ORDER BY departamento_cod)
    FROM funcionarios_windows_function;
```

Query | Query History

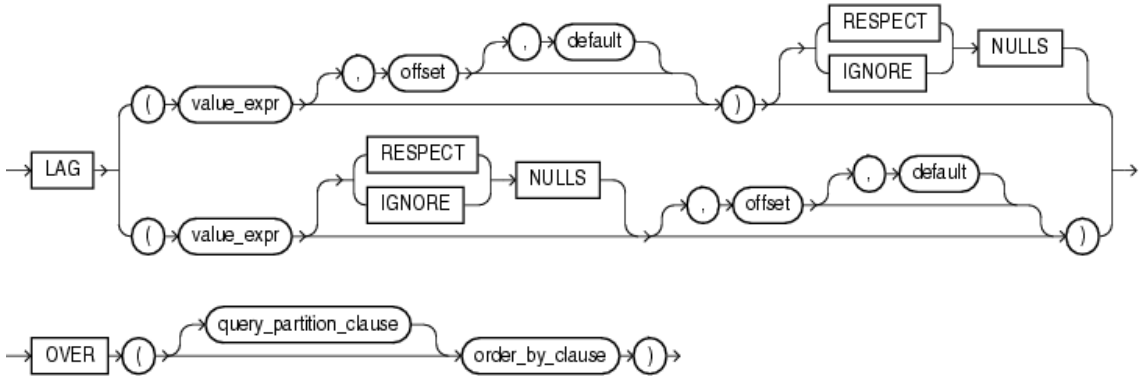
```
1  SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
2         ntile(3)
3         OVER (ORDER BY departamento_cod)
4         FROM funcionarios_windows_function;
5
6
```

Data Output | Messages | Notifications

A tabela está dividida em três partições pela função ntile()

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	tile integer
1	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	1
2	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	1
3	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	1
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	2
5	6	João dos testes	Analista de Testes	Testes	16	2000	2
6	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	2
7	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	3
8	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	3

LAG



LAG():

- **Descrição:** A função **LAG()** fornece o valor da linha anterior em relação à linha atual, com base na ordem especificada pelas cláusulas **OVER**.
- **Uso Prático:** Similar ao **LEAD()**, útil para análise de séries temporais e comparação de valores consecutivos.

LAG

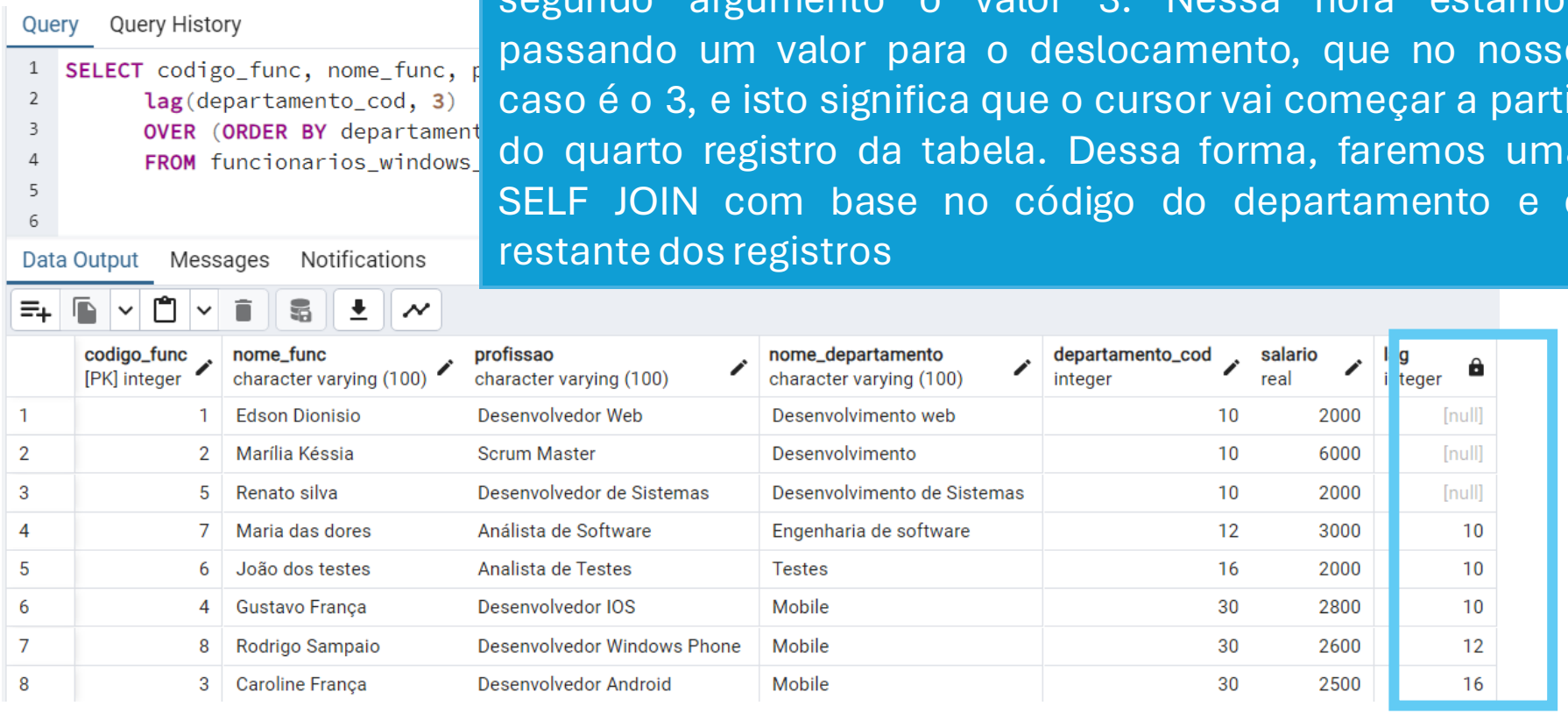
LAG():

- A função lag() é utilizada para acessarmos mais de uma linha presente na tabela ao mesmo tempo, sem a necessidade de utilizarmos o SELF JOIN.

LAG

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    lag(departamento_cod, 3)
    OVER (ORDER BY departamento_cod)
    FROM funcionarios_windows_function;
```

Passamos para a função o código do departamento e como segundo argumento o valor 3. Nessa hora estamos passando um valor para o deslocamento, que no nosso caso é o 3, e isto significa que o cursor vai começar a partir do quarto registro da tabela. Dessa forma, faremos uma SELF JOIN com base no código do departamento e o restante dos registros



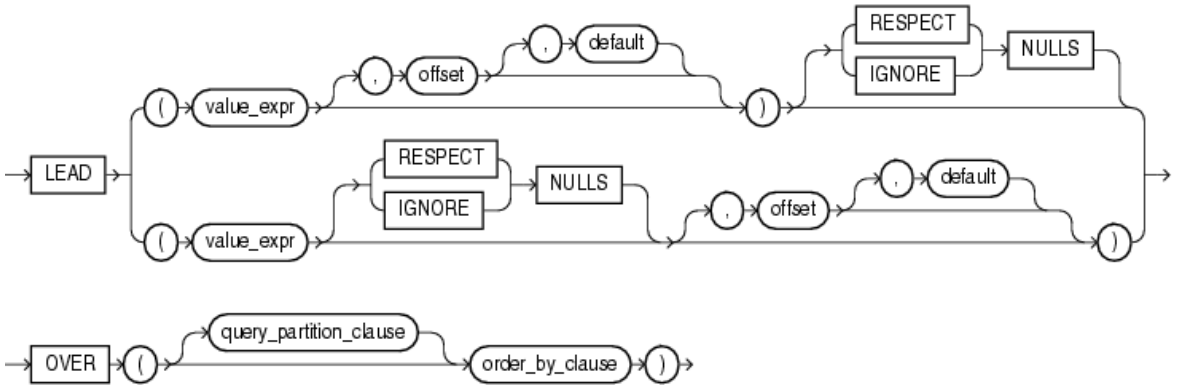
Query | Query History

```
1  SELECT codigo_func, nome_func, p
2         lag(departamento_cod, 3)
3         OVER (ORDER BY departament
4         FROM funcionarios_windows_
5
6
```

Data Output | Messages | Notifications

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	lag integer
1	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	[null]
2	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	[null]
3	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	[null]
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	10
5	6	João dos testes	Analista de Testes	Testes	16	2000	10
6	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	10
7	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	12
8	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	16

LEAD



LEAD():

- **Descrição:** A função **LEAD()** fornece o valor da próxima linha em relação à linha atual, com base na ordem especificada pelas cláusulas **OVER**.
- **Uso Prático:** Útil para comparar valores consecutivos em séries temporais.

LEAD

LEAD():

- A função lead(), a qual é utilizada para obtermos os valores retornados para linhas de registro com base no deslocamento abaixo da linha atual da partição. Se o argumento de deslocamento não é informado no momento de chamarmos a função, ela será definida como um, por padrão.

LEAD

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    lead(nome_departamento, 1)
    OVER (PARTITION BY departamento_cod ORDER BY departamento_cod)
    FROM funcionarios_windows_function;
```

Query | Query History

```
1  SELECT codigo_func, nome_fur
2       lead(nome_departamento
3       OVER (PARTITION BY dep
4       FROM funcionarios_winc
5
6
```

Como resultado da consulta temos que cada um dos registros apresenta o número de fileiras com base no código dos departamentos, mostrando dessa forma a partição entre eles.

Data Output | Messages | Notifications

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	lead character varying
1	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	Desenvolvimento
2	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	Desenvolvimento de Sistemas
3	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	[null]
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	[null]
5	6	João dos testes	Analista de Testes	Testes	16	2000	[null]
6	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	Mobile
7	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	Mobile
8	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	[null]

Uso combinado ROW_NUMBER, RANK e DENSE_RANK

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
        ROW_NUMBER() OVER (ORDER BY departamento_cod) AS RowNum,
        RANK() OVER (ORDER BY departamento_cod) AS Rank,
        DENSE_RANK() OVER (ORDER BY departamento_cod) AS DenseRank
        FROM funcionarios_windows_function;
```

Query | Query History

```
1  SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
2      ROW_NUMBER() OVER (ORDER BY departamento_cod) AS RowNum,
3      RANK() OVER (ORDER BY dep
4      DENSE_RANK() OVER (ORDER
5      FROM funcionarios_windows
6
```

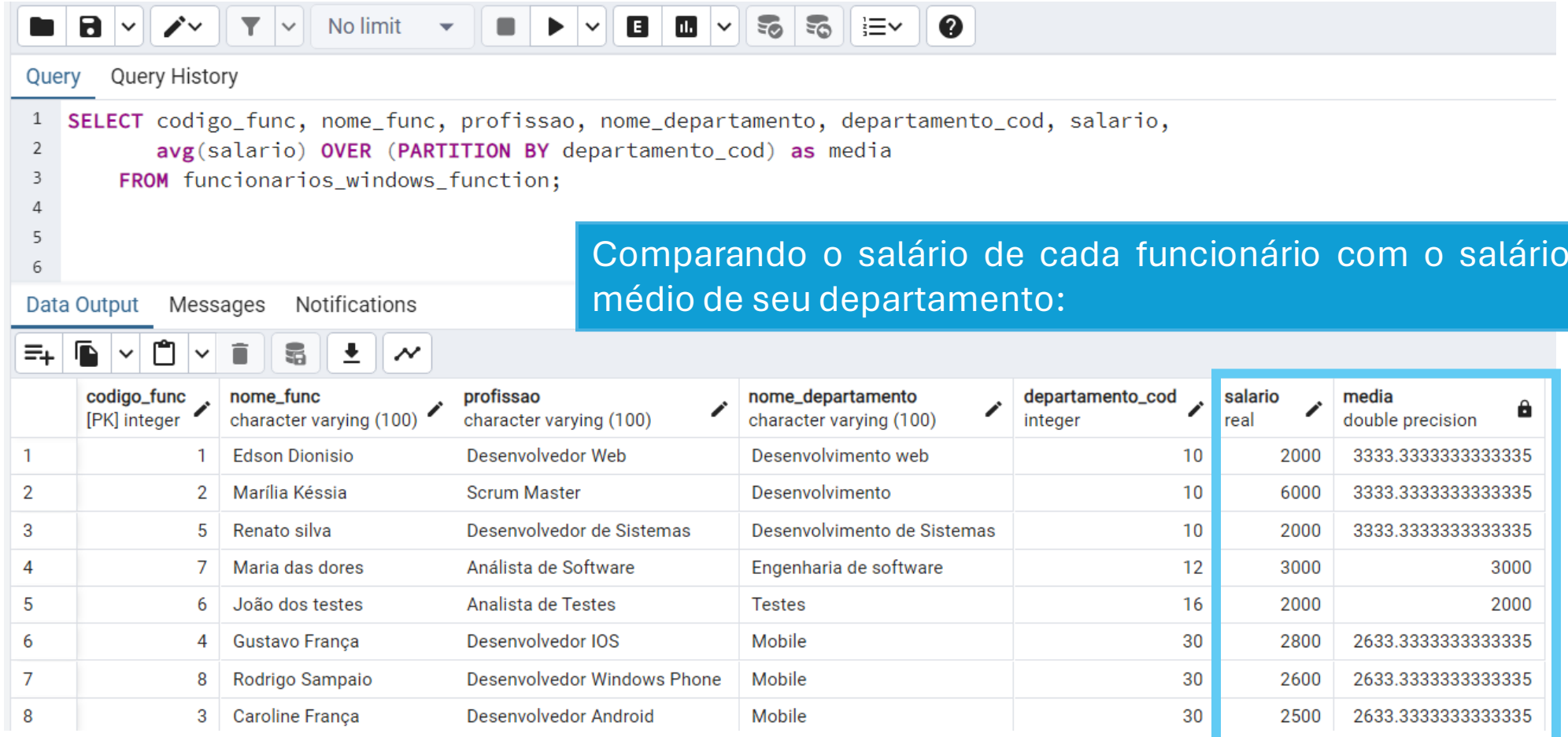
Essas funções são comumente usadas para atribuir um número de linha a cada registro em um conjunto de dados, permitindo ordenação e classificação eficientes.

Data Output | Messages | Notifications

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento integer	salario real	rownum bigint	rank bigint	denserank bigint
1	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	1	1	1
2	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	2	1	1
3	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	3	1	1
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	4	4	2
5	6	João dos testes	Analista de Testes	Testes	16	2000	5	5	3
6	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	6	6	4
7	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	7	6	4
8	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	8	6	4

Uso combinado SUM() e AVG()

```
SELECT codigo_func, nome_func, profissao, nome_departamento, departamento_cod, salario,
    avg(salario) OVER (PARTITION BY departamento_cod) as media
    FROM funcionarios_windows_function;
```



Comparando o salário de cada funcionário com o salário médio de seu departamento:

	codigo_func [PK] integer	nome_func character varying (100)	profissao character varying (100)	nome_departamento character varying (100)	departamento_cod integer	salario real	media double precision
1	1	Edson Dionisio	Desenvolvedor Web	Desenvolvimento web	10	2000	3333.3333333333335
2	2	Marília Késsia	Scrum Master	Desenvolvimento	10	6000	3333.3333333333335
3	5	Renato silva	Desenvolvedor de Sistemas	Desenvolvimento de Sistemas	10	2000	3333.3333333333335
4	7	Maria das dores	Análista de Software	Engenharia de software	12	3000	3000
5	6	João dos testes	Analista de Testes	Testes	16	2000	2000
6	4	Gustavo França	Desenvolvedor IOS	Mobile	30	2800	2633.3333333333335
7	8	Rodrigo Sampaio	Desenvolvedor Windows Phone	Mobile	30	2600	2633.3333333333335
8	3	Caroline França	Desenvolvedor Android	Mobile	30	2500	2633.3333333333335

Referencias

https://www.postgresql.org/docs/16/tutorial-window.html

https://halleyoliv.gitlab.io/pgdocptbr/functions.html

https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlrf/Functions.html

https://www.devmedia.com.br/trabalhando-com-windows-functions-no-postgresql/33707

https://medium.com/lets-data/desvendando-o-poder-das-window-functions-em-sql-para-estudantes-de-ci%C3%AAncia-de-dados-71b65ec9962d