

# PROGRAMAÇÃO BACK-END

PostgreSQL



# PROGRAMAÇÃO BACK-END

---

**Postgresql**

# PROGRAMAÇÃO BACK-END

---

**Consultando Dados**

# PROGRAMAÇÃO BACK-END

## PostgreSQL SELECT

Uma das tarefas mais comuns, quando se trabalha com banco de dados, é recuperar dados de tabelas usando a instrução **SELECT**.

A instrução **SELECT** é uma das instruções **mais complexas** do PostgreSQL. Possui muitas cláusulas que pode-se usar para formar uma consulta flexível.

# PROGRAMAÇÃO BACK-END

**A declaração SELECT possui as seguintes cláusulas:**

- **Selecione linhas distintas usando o operador **DISTINCT**.**
- **Classifique as linhas usando a cláusula **ORDER BY**.**
- **Filtre as linhas usando a cláusula **WHERE**.**
- **Selecione um subconjunto de linhas de uma tabela usando a cláusula **LIMIT** ou **FETCH**.**
- **Agrupe linhas em grupos usando a cláusula **GROUP BY**.**

# PROGRAMAÇÃO BACK-END

**A declaração SELECT possui as seguintes cláusulas (continuação):**

- **Filtre grupos usando a cláusula HAVING.**
- **Junte-se a outras tabelas usando junções como cláusulas INNER JOIN, LEFT JOIN, FULL OUTER JOIN, CROSS JOIN.**
- **Execute operações de conjunto usando UNION, INTERSECT e EXCEPT.**

# PROGRAMAÇÃO BACK-END

## Sintaxe da instrução SELECT do PostgreSQL

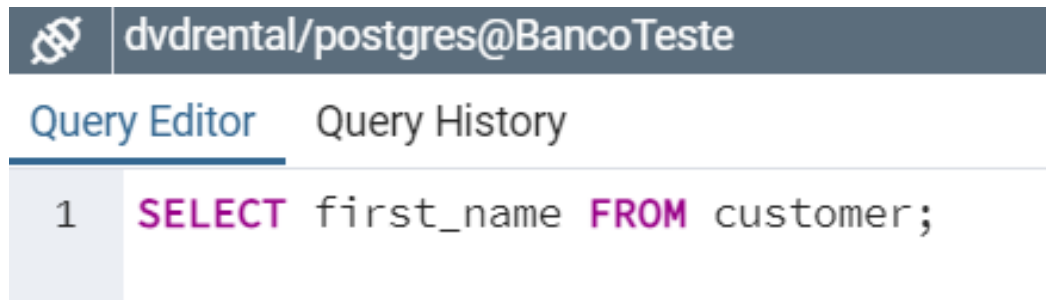
A forma básica da instrução **SELECT** que recupera dados de uma única tabela.

O seguinte ilustra a sintaxe da instrução **SELECT**:

```
SELECT  
    select_list  
FROM  
    table_name;
```

# PROGRAMAÇÃO BACK-END

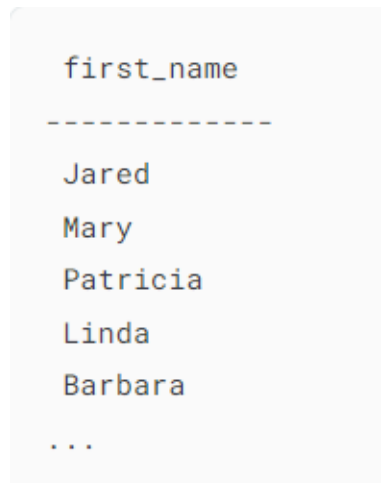
Este exemplo usa a instrução **SELECT** para encontrar os primeiros nomes de todos os clientes na tabela **customer**:



```
dvdrental/postgres@BancoTeste
```

Query Editor   Query History

```
1 SELECT first_name FROM customer;
```



```
first_name
-----
Jared
Mary
Patricia
Linda
Barbara
...
```



# PROGRAMAÇÃO BACK-END

A consulta a seguir usa a instrução **SELECT** para recuperar o **nome**, o **sobrenome** e o **email** dos clientes da tabela **customer**.

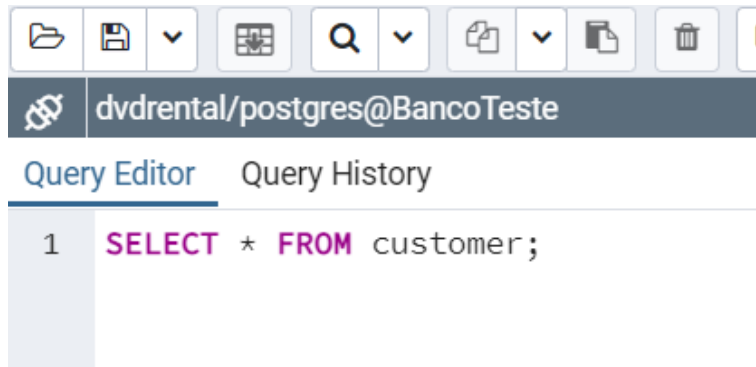
 dvdrental/postgres@BancoTeste  
Query Editor   Query History  

```
1  SELECT
2      first_name,
3      last_name,
4      email
5  FROM
6      customer;
```

first_name	last_name	email
Jared	Ely	jared.ely@sakilacustomer.org
Mary	Smith	mary.smith@sakilacustomer.org
Patricia	Johnson	patricia.johnson@sakilacustomer.org

# PROGRAMAÇÃO BACK-END

A consulta a seguir usa a instrução **SELECT \*** para recuperar dados de todas as colunas da tabela **customer**



Data Output		Explain	Messages	Notifications						
	customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)	address_id smallint	activebool boolean	create_date date		
1	524	1	Jared	Ely	jared.ely@sakilacustomer.org	530	true	2006-02-14		
2	1	1	Mary	Smith	mary.smith@sakilacustomer...	5	true	2006-02-14		
3	2	1	Patricia	Johnson	patricia.johnson@sakilacust...	6	true	2006-02-14		
4	3	1	Linda	Williams	linda.williams@sakilacusto...	7	true	2006-02-14		
5	4	2	Barbara	Jones	barbara.jones@sakilacusto...	8	true	2006-02-14		
6	5	1	Elizabeth	Brown	elizabeth.brown@sakilacust...	9	true	2006-02-14		
7	6	2	Jennifer	Davis	jennifer.davis@sakilacustom...	10	true	2006-02-14		
8	7	1	Maria	Miller	maria.miller@sakilacustome	11	true	2006-02-14		

# PROGRAMAÇÃO BACK-END

O exemplo a seguir usa a instrução **SELECT** para retornar os **nomes completos** e emails de todos os clientes da tabela **customer**

 **dvdrental/postgres@BancoTeste**

Query Editor

Query History


```
1  SELECT
2      first_name || ' ' || last_name,
3      email
4  FROM
5      customer;
```

	?column? text	email character varying (50)
1	Jared Ely	jared.ely@sakilacustomer.org
2	Mary Smith	mary.smith@sakilacustomer...
3	Patricia Johns...	patricia.johnson@sakilacust...
4	Linda Williams	linda.williams@sakilacusto...
5	Barbara Jones	barbara.jones@sakilacusto...
6	Elizabeth Bro...	elizabeth.brown@sakilacust...
7	Jennifer Davis	jennifer.davis@sakilacustom...
8	Maria Miller	maria.miller@sakilacustome...

Observe que a primeira coluna da saída não tem nome, mas **?column?**.

# PROGRAMAÇÃO BACK-END

Para atribuir um nome a uma coluna temporariamente na consulta, você pode usar um alias de coluna: **AS column\_alias**

 dvdrental/postgres@BancoTeste

Query Editor   Query History


```
1  SELECT
2      first_name || ' ' || last_name as full_name,
3      email
4  FROM
5      customer;
```

Data Output   Explain   Messages   Notifications

	<div>full_name text</div>	<div>email character varying (50)</div>
1	Jared Ely	jared.ely@sakilacustomer.org
2	Mary Smith	mary.smith@sakilacustomer...
3	Patricia Johns...	patricia.johnson@sakilacust...
4	Linda Williams	linda.williams@sakilacusto...
5	Barbara Jones	barbara.jones@sakilacusto...
6	Elizabeth Brown	elizabeth.brown@sakilacust...



# PROGRAMAÇÃO BACK-END

## Outro exemplo de uso do alias

 dvdrental/postgres@BancoTeste

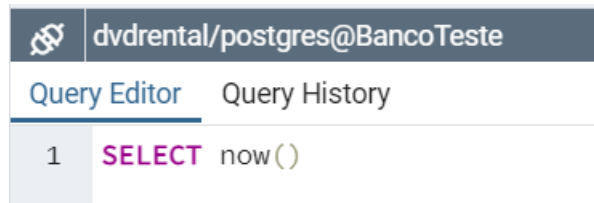
Query Editor   Query History

```
1  SELECT
2      first_name,
3      last_name AS surname
4  FROM customer;
```

	Data Output	Explain	Messages	Notifications
	<b>first_name</b> character varying (45)		<b>surname</b> character varying (45)	
1	Jared		Ely	
2	Mary		Smith	
3	Patricia		Johnson	
4	Linda		Williams	
5	Barbara		Jones	
6	Elizabeth		Brown	

# PROGRAMAÇÃO BACK-END

Neste exemplo, usamos a função **NOW()** na instrução **SELECT**. Ele retornará a **data** e **hora** atuais do **servidor PostgreSQL**.



Data Output		Explain	Messages	Notifications
	now timestamp with time zone			
1	2024-02-27 15:14:55.003221-05			

# PROGRAMAÇÃO BACK-END

Ao consultar dados de uma tabela, a instrução **SELECT** retorna linhas em uma **ordem não especificada**. Para classificar as linhas do conjunto de resultados, você usa a cláusula **ORDER BY** na instrução **SELECT**.

A cláusula **ORDER BY** permite classificar as linhas retornadas por uma cláusula **SELECT** em **ordem crescente** ou **decrescente** com base em uma expressão de classificação.

# PROGRAMAÇÃO BACK-END

O seguinte ilustra a sintaxe da **ORDER BY** cláusula:

```
SELECT
    select_list
FROM
    table_name
ORDER BY
    sort_expression1 [ASC | DESC],
    sort_expression2 [ASC | DESC],
    ...;
```



# PROGRAMAÇÃO BACK-END

A consulta a seguir usa a cláusula **ORDER BY** para classificar os clientes pelos nomes em ordem **crescente**:

	first_name character varying (45)	last_name character varying (45)
1	Aaron	Selby
2	Adam	Gooch
3	Adrian	Clary
4	Agnes	Bishop
5	Alan	Kahn
6	Albert	Crouse


Como a opção **ASC** é o padrão, pode-se **omiti-la**

dvdrental/postgres@BancoTeste	
Query Editor	Query History
1	<b>SELECT</b>
2	first_name,
3	last_name
4	<b>FROM</b>
5	customer
6	<b>ORDER BY</b>
7	first_name <b>ASC</b> ;

dvdrental/postgres@BancoTeste	
Query Editor	Query History
1	<b>SELECT</b>
2	first_name,
3	last_name
4	<b>FROM</b>
5	customer
6	<b>ORDER BY</b>
7	first_name;

# PROGRAMAÇÃO BACK-END

A **consulta** a seguir seleciona o nome e o sobrenome da tabela **customer** e classifica as linhas por valores na coluna do sobrenome em ordem **decrescente**:

 dvdrental/postgres@BancoTeste

Query Editor   Query History

```
1  SELECT
2    first_name,
3    last_name
4  FROM
5    customer
6  ORDER BY
7    last_name DESC;
```

	Data Output	Explain	Messages	Notifications
	<div>first_name character varying (45)</div>		<div>last_name character varying (45)</div>	
1	Cynthia		Young	
2	Marvin		Yee	
3	Luis		Yanez	
4	Brian		Wyman	
5	Brenda		Wright	
6	Tyler		Wren	

# PROGRAMAÇÃO BACK-END

A **Consulta** a seguir seleciona o nome e o sobrenome da tabela do cliente e classifica as linhas pelo **nome** em ordem **crescente** e pelo **sobrenome** em ordem **decrescente**:



dvdrental/postgres@BancoTeste

Query Editor

Query History

```
1 SELECT
2   first_name,
3   last_name
4 FROM
5   customer
6 ORDER BY
7   first_name ASC,
8   last_name DESC;
```

Data Output

Explain

Messages

Notifications

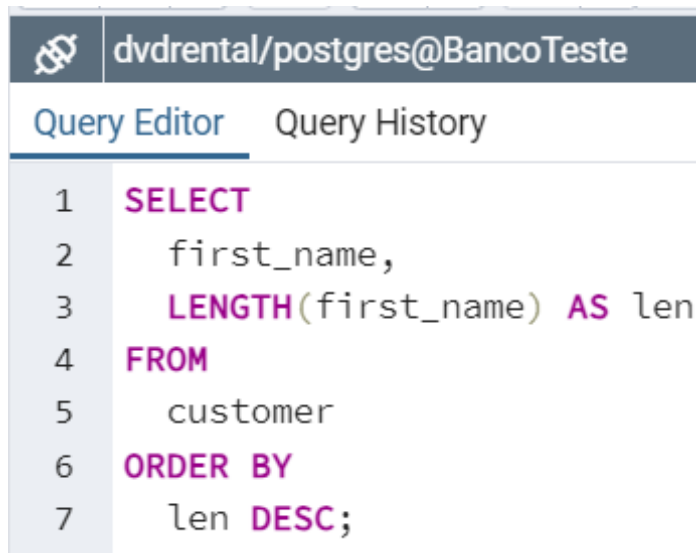
	<b>first_name</b> character varying (45)	<b>last_name</b> character varying (45)
325	Kay	Caldwell
326	Keith	Rico
327	Kelly	Torres
328	Kelly	Knott
329	Ken	Prewitt
330	Kenneth	Gooden
331	Kent	Arsenault



# PROGRAMAÇÃO BACK-END

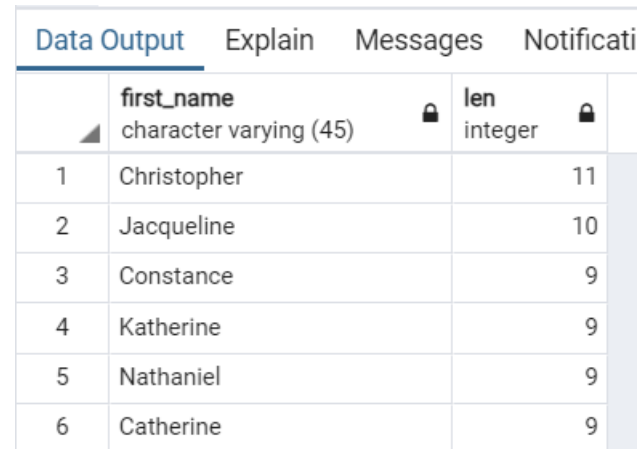
A função **LENGTH()** aceita uma **string** e retorna o **comprimento** dessa **string**.

A Consulta a seguir seleciona os primeiros nomes e seus comprimentos. Ele classifica as linhas pelo comprimento dos primeiros nomes



The screenshot shows a PostgreSQL Query Editor window with the title 'dvdrental/postgres@BancoTeste'. It has two tabs: 'Query Editor' and 'Query History'. The 'Query Editor' tab is active and contains the following SQL query:

```
1 SELECT
2   first_name,
3   LENGTH(first_name) AS len
4 FROM
5   customer
6 ORDER BY
7   len DESC;
```



The screenshot shows the 'Data Output' tab of the PostgreSQL Query Editor. It displays the results of the SQL query in a table format. The table has two columns: 'first\_name' (character varying (45)) and 'len' (integer). The results are ordered by 'len' in descending order.

	first_name character varying (45)	len integer
1	Christopher	11
2	Jacqueline	10
3	Constance	9
4	Katherine	9
5	Nathaniel	9
6	Catherine	9

# PROGRAMAÇÃO BACK-END

Introdução à cláusula **SELECT DISTINCT** do PostgreSQL

**SELECT DISTINCT** remove linhas duplicadas de um conjunto de resultados.

A cláusula **SELECT DISTINCT** retém uma linha para cada grupo de duplicatas.

A cláusula **SELECT DISTINCT** pode ser aplicada a **uma** ou **mais colunas** na lista de seleção da instrução **SELECT**.

# PROGRAMAÇÃO BACK-END


**O seguintes comandos ilustram a sintaxe da cláusula DISTINCT :**

```
SELECT  
    DISTINCT column1  
FROM  
    table_name;
```

```
SELECT  
    DISTINCT column1, column2  
FROM  
    table_name;
```

# PROGRAMAÇÃO BACK-END

Primeiro, crie a tabela **distinct\_demo** que possui três colunas: **id**, **bcolor**, **fcolor** usando a seguinte instrução **CREATE TABLE**:

 dvdrental/postgres@BancoTeste

Query Editor   Query History

```
1 CREATE TABLE distinct_demo (  
2   id SERIAL NOT NULL PRIMARY KEY,  
3   bcolor VARCHAR,  
4   fcolor VARCHAR  
5 );
```


Data Output   Explain   Messages   Notifications

CREATE TABLE

Query returned successfully in 67 msec.

# PROGRAMAÇÃO BACK-END

Segundo, insira algumas linhas na tabela **distinct\_demo** usando a instrução **INSERT**

 dvdrental/postgres@BancoTeste

Query Editor   Query History

```
1  INSERT INTO distinct_demo (bcolor, fcolor)
2  VALUES
3      ('red', 'red'),
4      ('red', 'red'),
5      ('red', NULL),
6      (NULL, 'red'),
7      ('red', 'green'),
8      ('red', 'blue'),
9      ('green', 'red'),
10     ('green', 'blue'),
11     ('green', 'green'),
12     ('blue', 'red'),
13     ('blue', 'green'),
14     ('blue', 'blue');
```

Data Output   Explain   Messages   Notifications


INSERT 0 12

Query returned successfully in 617 msec.



# PROGRAMAÇÃO BACK-END


Recupere os dados da tabela **distinct\_demo** usando a instrução **SELECT**.

```
 dvdrental/postgres@BancoTeste  
Query Editor  Query History  
1  SELECT  
2    id,  
3    bcolor,  
4    fcolor  
5  FROM  
6    distinct_demo;
```

Data Output				Explain	Messages	Notifications
	id [PK] integer	bcolor character varying	fcolor character varying			
1	1	red	red			
2	2	red	red			
3	3	red	[null]			
4	4	[null]	red			
5	5	red	green			
6	6	red	blue			
7	7	green	red			
8	8	green	blue			
9	9	green	green			
10	10	blue	red			
11	11	blue	green			
12	12	blue	blue			

# PROGRAMAÇÃO BACK-END

A consulta a seguir seleciona valores exclusivos na coluna bcolor da tabela e classifica o conjunto de resultados em ordem alfabética usando a cláusula ORDER BY

 dvdrental/postgres@BancoTeste

Query Editor   Query History

```
1 SELECT
2   DISTINCT bcolor
3 FROM
4   distinct_demo
5 ORDER BY
6   bcolor;
```

Data Output		Explain	Mes:
	<b>bcolor</b> character varying		
1	blue		
2	green		
3	red		
4	[null]		

# PROGRAMAÇÃO BACK-END

A instrução a seguir demonstra como usar a cláusula **DISTINCT** em **múltiplas colunas**

 dvdrental/postgres@BancoTeste

Query Editor Query History

```
1 SELECT
2   DISTINCT bcolor,
3   fcolor
4 FROM
5   distinct_demo
6 ORDER BY
7   bcolor,
8   fcolor;
```

	Data Output	Explain	Messages	Notifications
	<b>bcolor</b> character varying		<b>fcolor</b> character varying	
1	blue		blue	
2	blue		green	
3	blue		red	
4	green		blue	
5	green		green	
6	green		red	
7	red		blue	
8	red		green	
9	red		red	
10	red		[null]	