

# Banco de Dados – IMD0401

## Aula 16 – SQL Data Query Language

João Carlos Xavier Júnior

[jcxavier@imd.ufrn.br](mailto:jcxavier@imd.ufrn.br)

# SQL - DQL

## ❑ Seleção:

- ❖ A forma básica do comando de seleção é:

```
SELECT <lista de atributos>  
FROM   <lista de tabelas>  
WHERE  <condição>
```

- ❖ Onde:

- **<lista de atributos>** é uma lista de nomes de atributos cujos valores serão ser recuperados pela consulta.
- **<lista de tabelas>** é uma lista de nomes de relações requeridas para processar a consulta.
- **<condição>** é uma expressão (lógica) que identifica as tuplas a serem recuperadas pela consulta.

# SQL - DQL

## ❑ Seleção:

### ❖ Exemplo:

```
select nome, cpf, genero from Cliente
```

### ❖ Resultado:

Output pane			
<b>Data Output</b> Explain Messages History			
	nome character varying(45)	cpf character varying(14)	genero character(1)
1	Roberto Carlos	111.111.111-01	M
2	Ana Maria	222.222.222-02	F
3	Francisco dos Santos	333.333.333-03	M
4	Angelina Jolie	444.444.444-04	F

# SQL - DQL

## ❑ Seleção:

❖ **Observação:** sempre use **filtros** em sua seleção.

```
select Nome, cpf, sexo from Cliente  
where cidade = 'Natal'  
and genero = 'M'
```

❖ Resultado:

Data Output Explain Messages History			
	nome character varying(45)	cpf character varying(14)	genero character(1)
1	Roberto Carlos	111.111.111-01	M


# SQL - DQL

## ❑ Seleção:

❖ **Observação:** nunca faça isso.

**select \* from Cliente**

❖ Resultado:



	idcliente integer	nome character varying(45)	cpf character varying(14)	genero character(1)	cidade character varying(15)
1	1	Roberto Carlos	111.111.111-01	M	Natal
2	2	Ana Maria	222.222.222-02	F	Parnamirim
3	3	Francisco dos Sa	333.333.333-03	M	
4	4	Angelina Jolie	444.444.444-04	F	

# SQL - DQL

## ❑ Seleção:

- ❖ A cláusula **SELECT** permite duplicação de resultados nas consultas.
- ❖ Para forçar a eliminação de duplicação, acrescenta-se a palavra chave **DISTINCT**.

### ❖ Exemplo:

```
select genero from Cliente  
select distinct genero from Cliente  
select all genero from Cliente
```

# SQL - DQL

## ❑ Seleção (distinct):

### ❖ Resultado:

```
select genero from Cliente
```

```
select distinct genero from Cliente
```

```
select all genero from Cliente
```

Data Output Explain	
	genero character(1)
1	M
2	F
3	M
4	F

Data Output Explain	
	genero character(1)
1	F
2	M

Data Output Explain	
	genero character(1)
1	M
2	F
3	M
4	F

# SQL - DQL

## ❑ Seleção:

- ❖ A cláusula SELECT pode conter funções que operam sobre uma coleção de valores de uma determinada coluna da tabela.
  
- ❖ O SQL fornece 5 funções agregadas:
  - COUNT: número de tuplas ou valores.
  - SUM: soma os valores de uma coluna.
  - AVG: calcula a média dos valores de uma coluna.
  - MAX: identifica o maior valor de uma coluna.
  - MIN: identifica o menor valor de uma coluna.



# SQL - DQL

## ❑ Seleção:

### ❖ Exemplos:

```
select count(genero) from Cliente
```

```
select count(distinct genero) from Cliente
```

Previous queries

```
select count(genero) from Cliente  
select count(distinct genero) from Cliente
```

Output pane

Data Output Explain Messages History

	count bigint
1	4

Previous queries

```
select count(genero) from Cliente  
select count(distinct genero) from Cliente
```

Output pane

Data Output Explain Messages History

	count bigint
1	2

# SQL - DQL

## ❑ Seleção:

❖ A cláusula SELECT permite que colunas possam ser **renomeadas** na consulta. A mesma ideia pode ser usada para tabelas.

❖ Exemplos:

```
select nome, cpf, genero, cidade as  
Localidade from Cliente
```

```
select      c.nome,      c.cpf,      c.genero,  
c.cidade as Localidade from Cliente c
```

# SQL - DQL

- ❑ Seleção: renomeando colunas e tabelas.

Previous queries

```
select nome, cpf, genero, cidade as Localidade from Cliente  
select c.nome, c.cpf, c.genero, c.cidade as Localidade from Cliente c
```

Output pane

Data Output Explain Messages History

	nome character varying(45)	cpf character varying(14)	genero character(1)	localidade character varying(15)
1	Roberto Carlos	111.111.111-01	M	Natal
2	Ana Maria	222.222.222-02	F	Parnamirim
3	Francisco dos Santos	333.333.333-03	M	
4	Angelina Jolie	444.444.444-04	F	

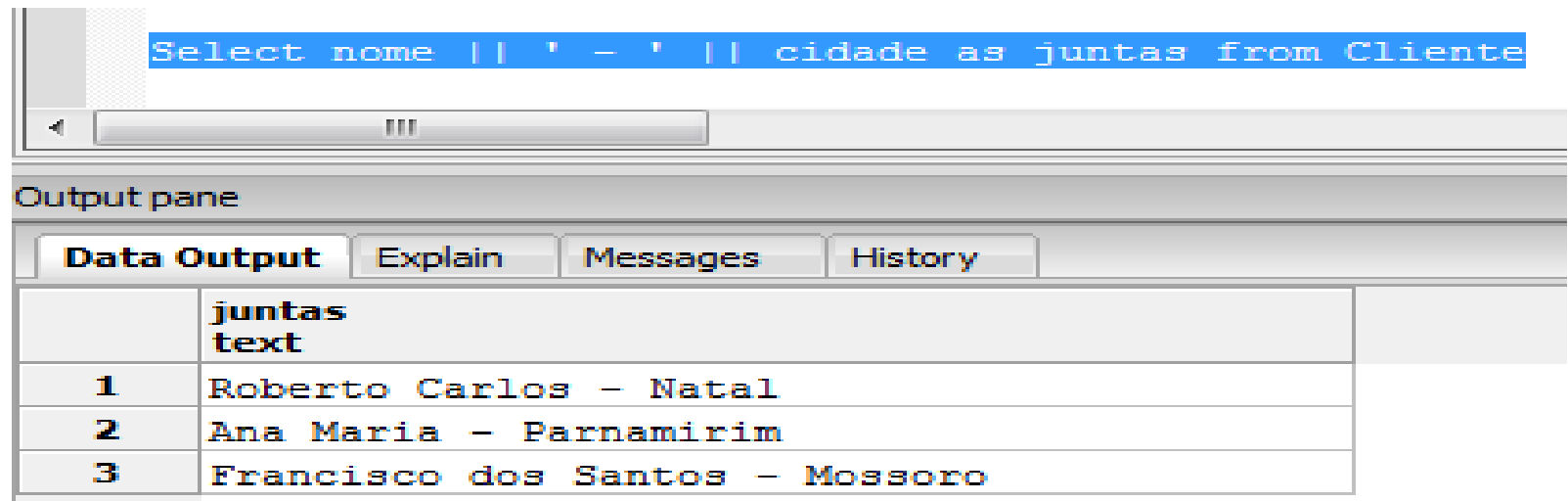
# SQL - DQL

## ❑ Seleção:

❖ SQL também permite função para concatenação ( || ).

❖ Exemplos:

```
select nome || ' - ' || cidade as  
juntas from Cliente
```



The screenshot shows a SQL query execution window. The query is: `Select nome || ' - ' || cidade as juntas from Cliente`. Below the query, there is an "Output pane" with tabs for "Data Output", "Explain", "Messages", and "History". The "Data Output" tab is selected, showing a table with the results of the query.

	juntas text
1	Roberto Carlos - Natal
2	Ana Maria - Parnamirim
3	Francisco dos Santos - Mossoro

# SQL - DQL

## □ From:

- ❖ A cláusula **FROM** corresponde ao **produto cartesiano** da álgebra relacional.
- ❖ A eliminação das tuplas incoerentes do produto cartesiano é feito através da cláusula **WHERE**.

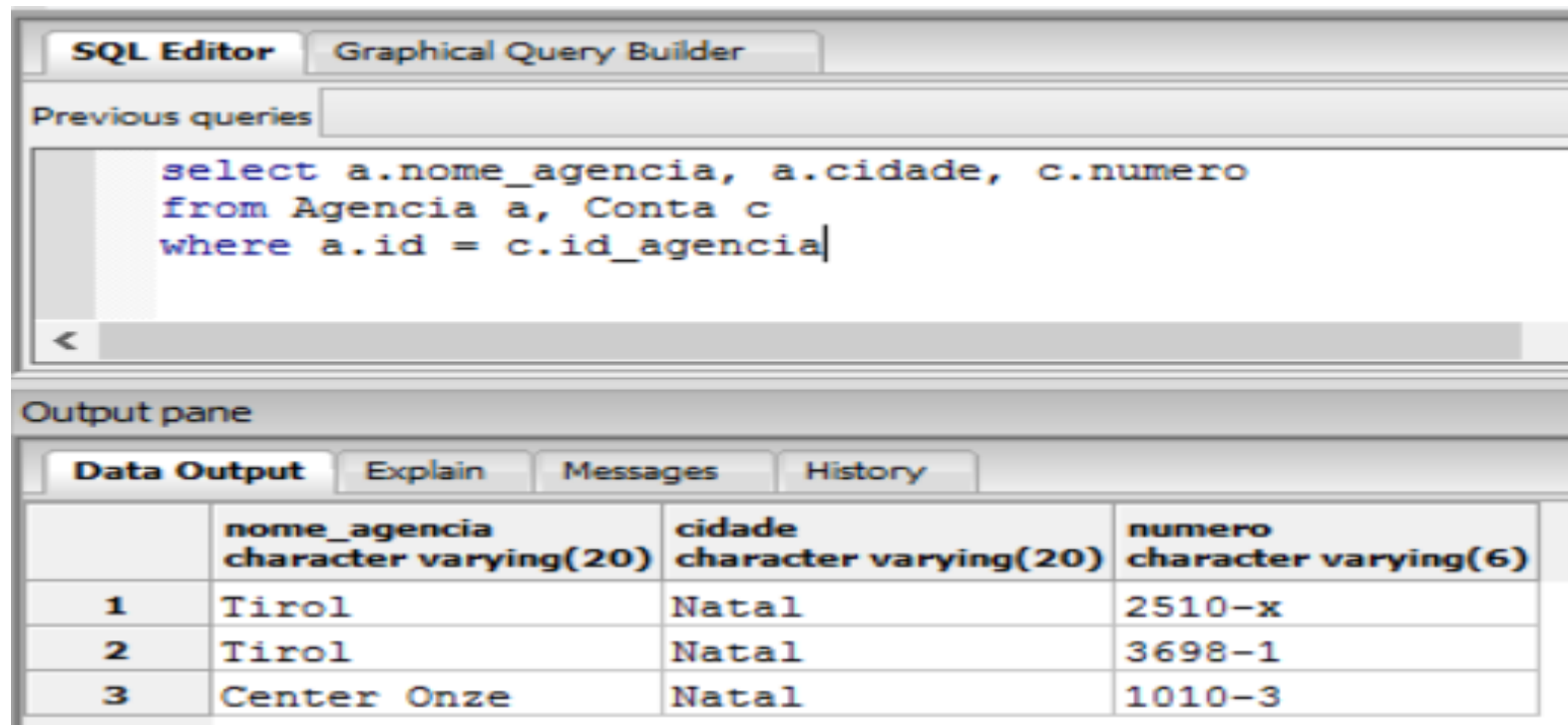
### ❖ Exemplo:

```
select a.nome_agencia, a.cidade,  
       c.numero  
from Agencia a, Conta c  
where a.id = c.id agencia
```

# SQL - DQL

□ From:

❖ Resultado:



The screenshot shows a database application interface. At the top, there are two tabs: "SQL Editor" and "Graphical Query Builder". Below the tabs is a text area for the SQL query. The query is: `select a.nome_agencia, a.cidade, c.numero from Agencia a, Conta c where a.id = c.id_agencia`. Below the query editor is an "Output pane" with four tabs: "Data Output", "Explain", "Messages", and "History". The "Data Output" tab is selected, showing a table with three columns: `nome_agencia` (character varying(20)), `cidade` (character varying(20)), and `numero` (character varying(6)). The table contains three rows of data.

	<code>nome_agencia</code> character varying(20)	<code>cidade</code> character varying(20)	<code>numero</code> character varying(6)
1	Tirol	Natal	2510-x
2	Tirol	Natal	3698-1
3	Center Onze	Natal	1010-3

# SQL - DQL

## □ Where:

- ❖ A cláusula **WHERE** usa os conectivos lógicos AND, OR e NOT. É permitido usar expressões aritméticas de comparação (=, <>, <, <=, >=, >).
- ❖ Operador BETWEEN permite que um atributo seja comparado dentro de uma faixa especificada.
- ❖ Exemplo:

```
Select idEmprestimo from Emprestimo  
where total between 90000 and 100000
```

# SQL - DQL

## □ Where:

❖ Operador **LIKE** permite a comparações em seqüências de caracteres.

❖ Exemplo:

```
select cpf, nome from Cliente  
where nome like '%Mar%'
```

<b>Data Output</b> Explain Messages History		
	<b>cpf</b> character varying(14)	<b>nome</b> character varying(45)
<b>1</b>	222.222.222-02	Ana Maria

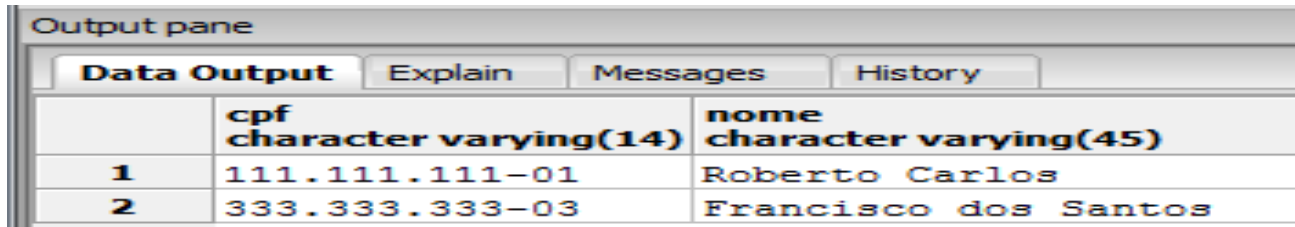


# SQL - DQL

□ Where:

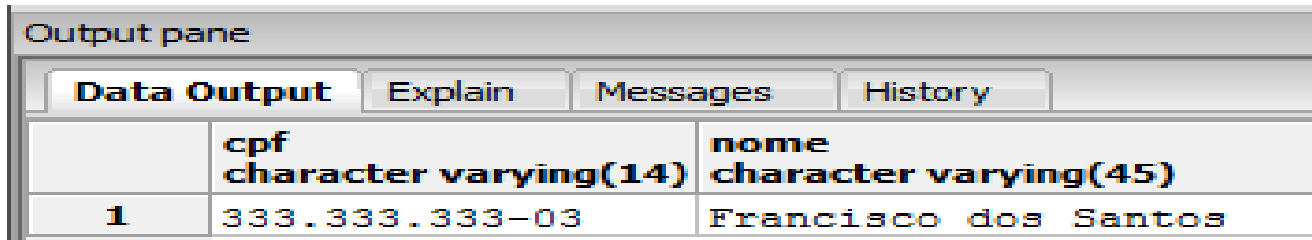
❖ Outros:

```
select cpf, nome from Cliente  
where nome like '%os%'
```



	cpf character varying(14)	nome character varying(45)
1	111.111.111-01	Roberto Carlos
2	333.333.333-03	Francisco dos Santos

```
select cpf, nome from Cliente  
where nome like 'Fr%'
```



	cpf character varying(14)	nome character varying(45)
1	333.333.333-03	Francisco dos Santos

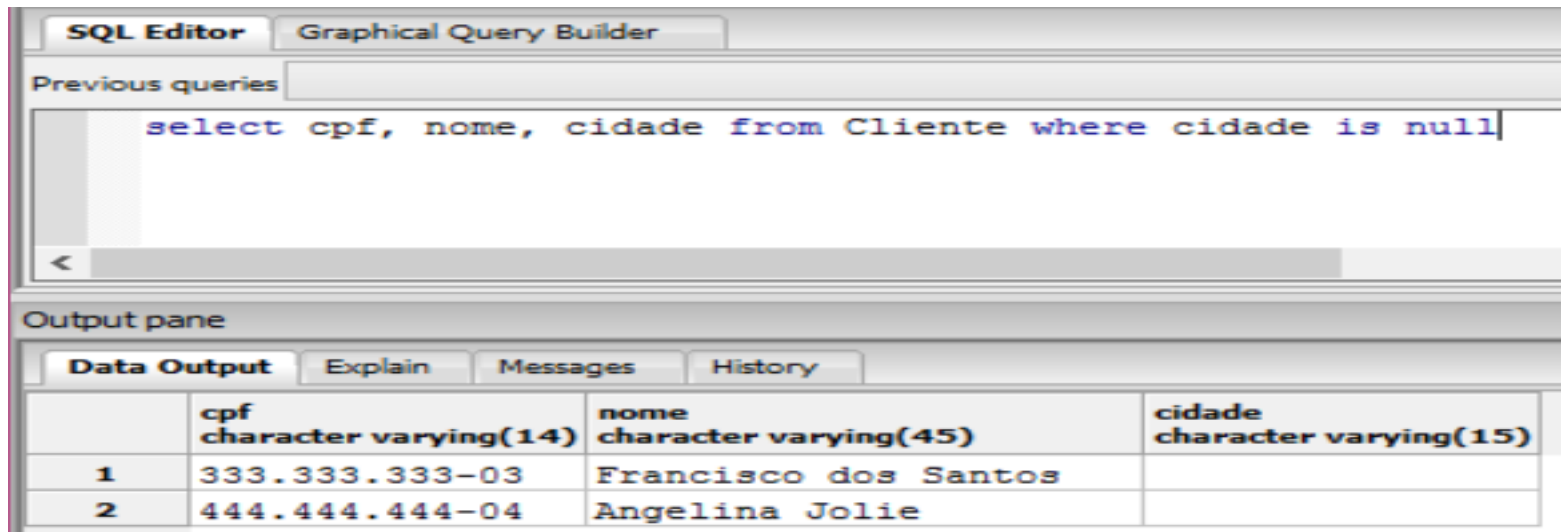
# SQL - DQL

## □ Where:

❖ Operador **IS NULL** verifica se o atributo é nulo.

❖ Exemplo:

```
select cpf, nome, cidade from Cliente  
where cidade is null
```



The screenshot shows a SQL Editor window with two tabs: "SQL Editor" and "Graphical Query Builder". The "SQL Editor" tab is active, displaying the query: `select cpf, nome, cidade from Cliente where cidade is null`. Below the query editor is an "Output pane" with four tabs: "Data Output", "Explain", "Messages", and "History". The "Data Output" tab is active, showing a table with the results of the query. The table has four columns: an index column, "cpf character varying(14)", "nome character varying(45)", and "cidade character varying(15)". There are two rows of data.

	cpf character varying(14)	nome character varying(45)	cidade character varying(15)
1	333.333.333-03	Francisco dos Santos	
2	444.444.444-04	Angelina Jolie	

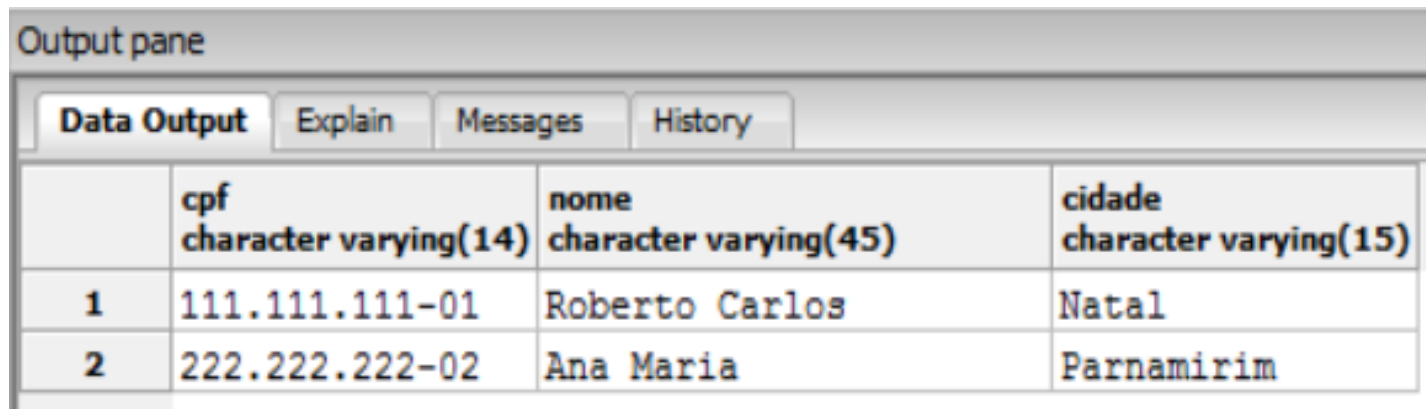
# SQL - DQL

## □ Where:

❖ Operador **IS NOT NULL** verifica se o atributo não é nulo.

❖ Exemplo:

```
select cpf, nome, cidade from Cliente  
where cidade is not null
```



The screenshot shows a database application window with an "Output pane" at the top. Below the title bar are four tabs: "Data Output" (selected), "Explain", "Messages", and "History". The "Data Output" tab displays the results of a SQL query in a table format. The table has four columns: an index column, "cpf" (character varying(14)), "nome" (character varying(45)), and "cidade" (character varying(15)). There are two rows of data.

	cpf character varying(14)	nome character varying(45)	cidade character varying(15)
1	111.111.111-01	Roberto Carlos	Natal
2	222.222.222-02	Ana Maria	Parnamirim

# SQL - DQL

## □ Where:

❖ Operador **IN** permite que um atributo seja comparado com um conjunto.

❖ Exemplo:

```
Select nome, cpf from Cliente  
where uf in ('AC', 'RN')
```

	Data Output	Explain	Messages	History
	nome character varying(45)	cpf character varying(14)		
1	Roberto Carlos	111.111.111-01		
2	Ana Maria	222.222.222-02		
3	Marina Silva	555.555.555-05		

# Questões...



# SQL-DML e SQL – DQL

## ❑ Inclusão:

- ❖ Podemos inserir várias tuplas numa relação através de uma consulta (**SELECT**).
- ❖ Exemplo: Exemplo: inserir todas as tuplas de Alunos em ExAlunos.

```
Insert into ExAlunos  
Select Matricula, Nome, Endereco  
From Alunos  
where Matricula > 0
```

# SQL-DML e SQL – DQL

## Exemplo:

pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - BD\_AULA11 - alunos

File Edit View Help

Icons: Refresh, Undo, Redo, Print, Copy, Paste, Filter, Help, Limit: No limit

	matricula [PK] integer	nome character varying(40)	endereco character varying(40)
1	1	ANA	RUA A
2	2	ISABEL	RUA A
3	3	RAFAEL	RUA C
*			

pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - BD\_AULA11 - exalunos

File Edit View Help

Icons: Refresh, Undo, Redo, Print, Copy, Paste, Filter, Help, Limit: No limit

	matricula [PK] integer	nome character varying(40)	endereco character varying(40)
*			

# SQL-DML e SQL – DQL

## ❑ Resultado:

pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - BD\_AULA11 - exalunos

File Edit View Help

Icons: Refresh, Undo, Redo, Print, Save, Filter, Help, Limit: No limit

	matricula [PK] integer	nome character varying(40)	endereco character varying(40)
1	1	ANA	RUA A
2	2	ISABEL	RUA A
3	3	RAFAEL	RUA C
*			



# Questões...



# Subconsultas

# SQL - DQL

- ❑ Sub consulta ou **subquery** também chamada de **subselect** por alguns autores.
- ❑ Sub consulta (**In**):
  - ❖ Forma alternativa de especificar consultas envolvendo relacionamentos entre tabelas.
  - ❖ Filtragens prévias de dados na sub-consulta.
    - Apenas tuplas/atributos de interesse são combinados com dados da(s) tabela(s) da consulta externa.

# SQL - DQL

## □ Tabelas:

Edit Data - PostgreSQL 9.3 (localhost:5432) - DB\_teste - cliente

	idcliente [PK] serial	nome character varying(45)	cpf character varying(14)	genero character(1)	cidade character vary
1	1	Roberto Carlos	111.111.111-01	M	Natal
2	2	Ana Maria	222.222.222-02	F	Parnamirim
3	3	Francisco dos Santos	333.333.333-03	M	
4	4	Angelina Jolie	444.444.444-04	F	
5	5	Paulo Maluf	125.147.658-96	M	São Paulo
6	6	Celso Pitta	524.471.321-55	M	São Paulo
*					

Edit Data - PostgreSQL 9.3 (localhost:5432) - DB\_teste - devedor

	id [PK] integer	nome character varying(20)	cpf character varying(14)	numero_conta character varying(6)
1	1	Paulo Maluf	125.147.658-96	1010-3
2	2	Celso Pitta	524.471.321-55	3698-1
*				

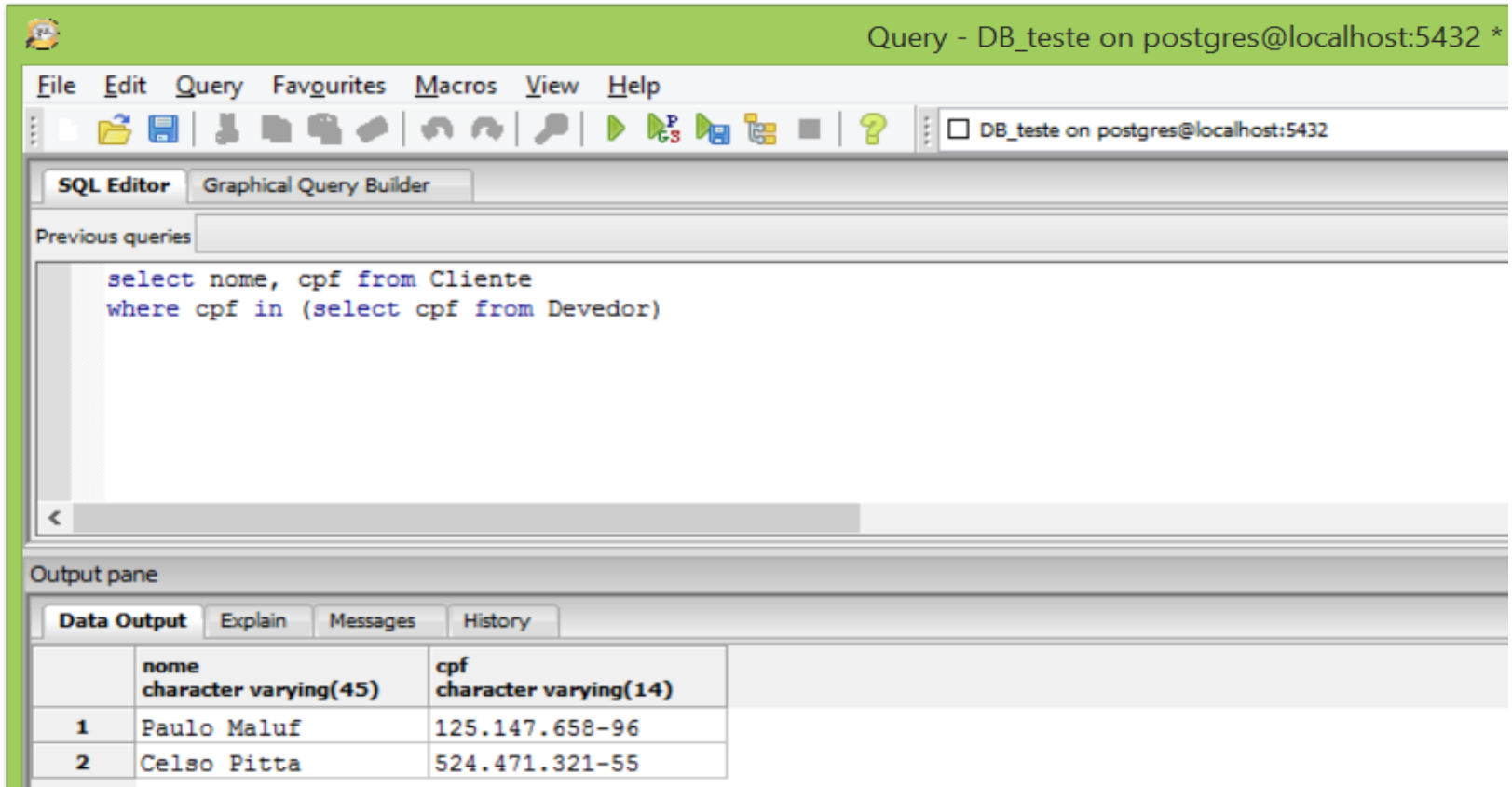
# SQL - DQL

□ Exemplo:

```
select nome, cpf from Cliente  
where cpf in (select cpf from Devedor)
```

# SQL - DQL

## □ Sub Consultas (In):



The screenshot shows a SQL client window titled "Query - DB\_teste on postgres@localhost:5432 \*". The window has a menu bar (File, Edit, Query, Favourites, Macros, View, Help) and a toolbar. Below the toolbar are tabs for "SQL Editor" and "Graphical Query Builder". The "SQL Editor" tab is active, showing the following SQL query:

```
select nome, cpf from Cliente
where cpf in (select cpf from Devedor)
```

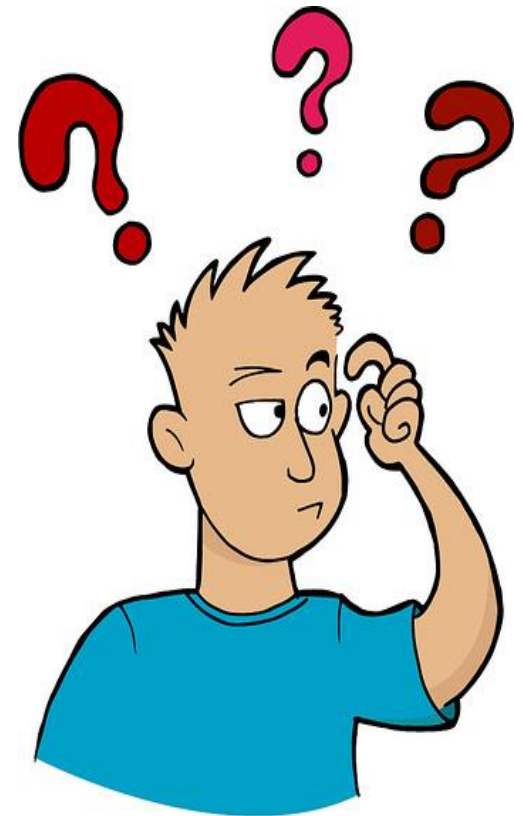
Below the query editor is a scroll bar. At the bottom of the window is the "Output pane" with tabs for "Data Output", "Explain", "Messages", and "History". The "Data Output" tab is active, displaying the results of the query in a table:

	nome character varying(45)	cpf character varying(14)
1	Paulo Maluf	125.147.658-96
2	Celso Pitta	524.471.321-55

# SQL - DQL

## ❑ Sub Consultas (**Exists**):

- ❖ A função **EXISTS** é usada para checar se o resultado de uma consulta aninhada é vazia ou não.
- ❖ Exemplo: encontre todos os clientes (nome e CPF) que tenham uma conta em alguma agência localizadas em **Natal**.



# SQL - DQL

## □ Tabelas:

Edit Data - PostgreSQL 9.3 (localhost:5432) - DB\_teste - agencia

	id [PK] integer	nome_agencia character varying(20)	cidade character varying(20)
1	1	Tirol	Natal
2	2	Center Onze	Natal
3	3	Currais Novos	Currais Novos
*			

Edit Data - PostgreSQL 9.3 (localhost:5432) - DB\_teste - conta

	id [PK] integer	numero character vary	id_agencia integer
1	1	2510-x	1
2	2	3698-1	1
3	3	1010-3	2
*			



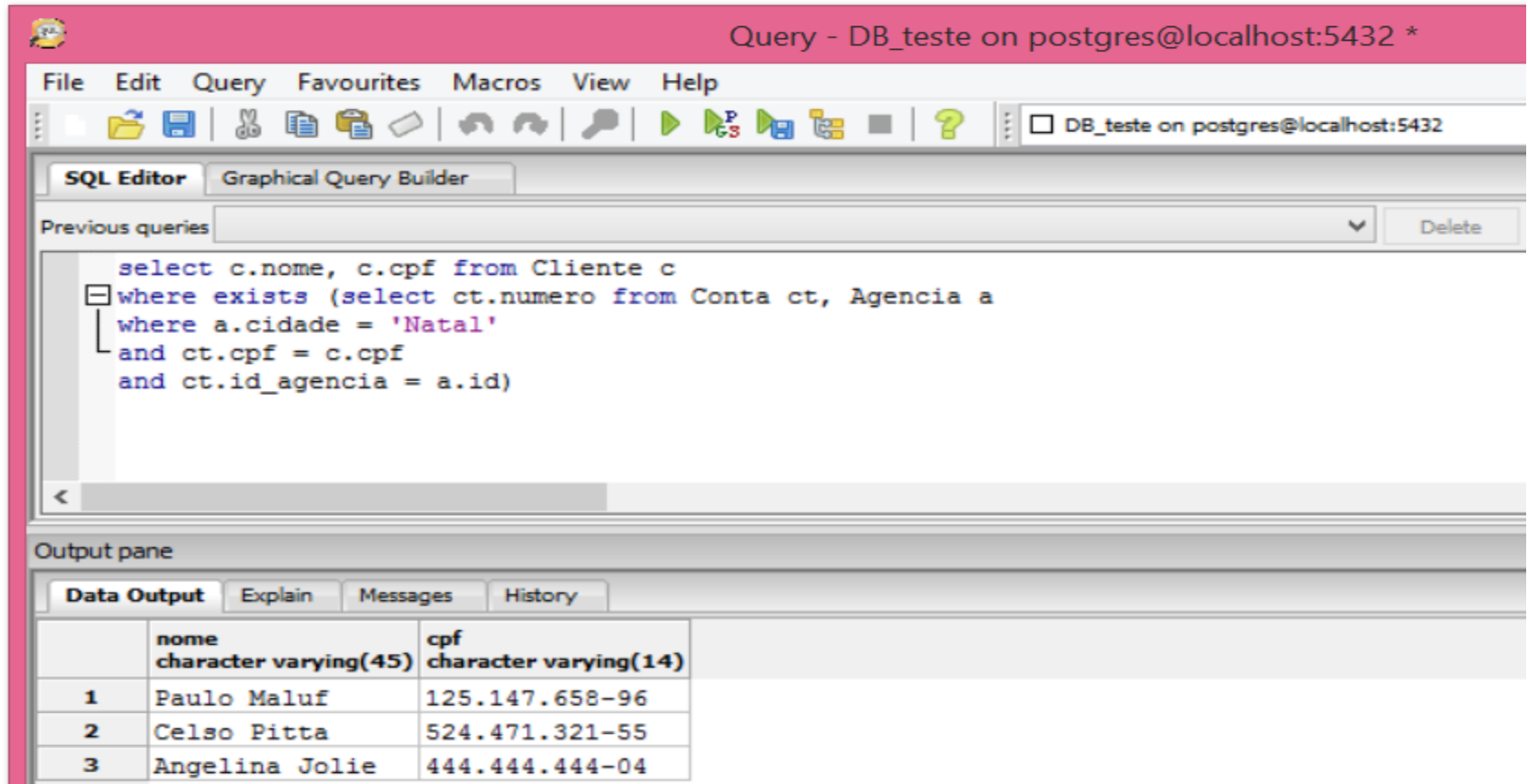
# SQL - DQL

❏ Exemplo:

```
select c.nome, c.cpf from Cliente c
where exists (select ct.id from Conta
ct, Agencia a
where a.cidade = 'Natal'
and ct.cpf = c.cpf
and ct.id_agencia = a.id)
```

# SQL - DQL

□ Resultado:



The screenshot shows a PostgreSQL SQL Editor window titled "Query - DB\_teste on postgres@localhost:5432 \*". The window has a menu bar (File, Edit, Query, Favourites, Macros, View, Help) and a toolbar with various icons. The "SQL Editor" tab is active, displaying the following SQL query:

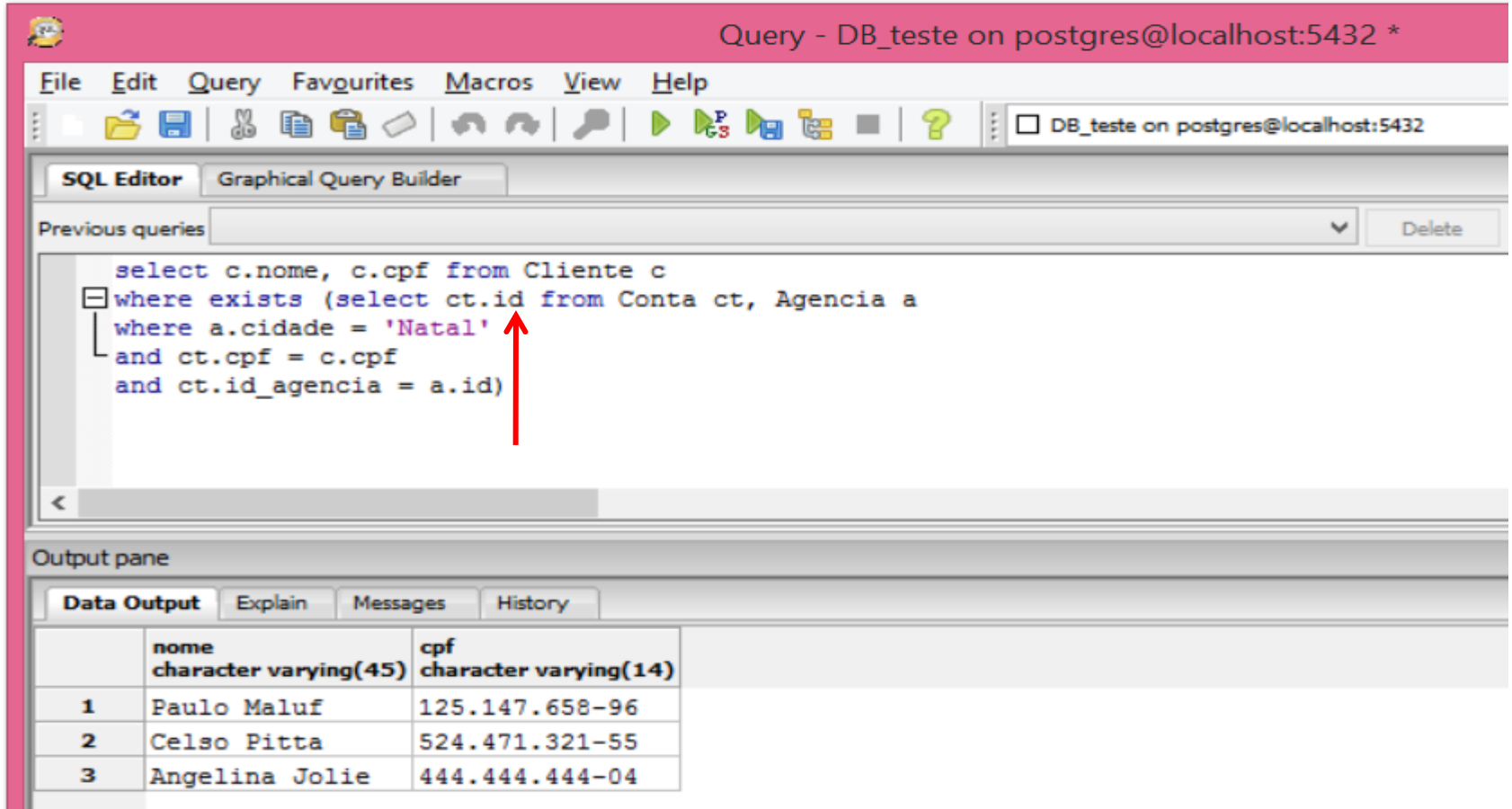
```
select c.nome, c.cpf from Cliente c
where exists (select ct.numero from Conta ct, Agencia a
where a.cidade = 'Natal'
and ct.cpf = c.cpf
and ct.id_agencia = a.id)
```

Below the query editor is the "Output pane" with tabs for "Data Output", "Explain", "Messages", and "History". The "Data Output" tab is selected, showing the results of the query in a table format:

	nome character varying(45)	cpf character varying(14)
1	Paulo Maluf	125.147.658-96
2	Celso Pitta	524.471.321-55
3	Angelina Jolie	444.444.444-04

# SQL - DQL

❏ Resultado:



The screenshot shows a PostgreSQL SQL Editor window titled "Query - DB\_teste on postgres@localhost:5432 \*". The window has a menu bar (File, Edit, Query, Favourites, Macros, View, Help) and a toolbar with various icons. The "SQL Editor" tab is active, displaying the following SQL query:

```
select c.nome, c.cpf from Cliente c
where exists (select ct.id from Conta ct, Agencia a
where a.cidade = 'Natal'
and ct.cpf = c.cpf
and ct.id_agencia = a.id)
```

A red arrow points to the string 'Natal' in the query. Below the editor is the "Output pane" with tabs for "Data Output", "Explain", "Messages", and "History". The "Data Output" tab is selected, showing a table with 3 rows and 2 columns: "nome" (character varying(45)) and "cpf" (character varying(14)).

	nome character varying(45)	cpf character varying(14)
1	Paulo Maluf	125.147.658-96
2	Celso Pitta	524.471.321-55
3	Angelina Jolie	444.444.444-04

# SQL - DQL

## ❑ Sub Consultas (**Atributo**):

- ❖ Quando há a necessidade de se incluir um atributo de outra entidade na seleção (consulta) sem influenciar no resultado final.

The screenshot shows a SQL Editor window with a menu bar (File, Edit, Query, Favourites, Macros, View, Help) and a toolbar. The status bar indicates the connection is 'DB\_teste on postgres@localhost:5432'. The 'SQL Editor' tab is active, showing the following SQL query:

```
select d.nome, d.cpf, d.numero_conta,  
(select a.nome_agencia From Agencia a, Conta c  
where d.numero_conta = c.numero  
and c.id_agencia = a.id) from Devedor d
```

The 'Output pane' at the bottom shows the 'Data Output' tab with the following results:

	nome character varying(20)	cpf character varying(14)	numero_conta character varying(6)	nome_agencia character varying(20)
1	Paulo Maluf	125.147.658-96	1010-3	Center Onze
2	Celso Pitta	524.471.321-55	3698-1	Tirol

# SQL - DQL

## ❑ Cláusula de Organização:

- ❖ A cláusula **ORDER BY** possibilita a ordenação de tuplas.
- ❖ Podemos especificar DESC para ordem descendente ou ASC para ordem ascendente para cada atributo; ordem ascendente é o default.
- ❖ Exemplo: listar em ordem alfabética os nomes de todos os clientes.

```
Select nome, cpf, numero_conta  
From Devedor  
Order by nome
```

	nome character varying(20)	cpf character varying(14)	numero_conta character varying(6)
1	Celso Pitta	444.444.444-44	
2	Paulo Maluf	111.111.111-11	1020-1

# SQL - DQL

## ❑ Exemplos:

```
Select vendedor, cliente, valor  
From Venda  
Order by vendedor, valor
```

```
Select vendedor, cliente, valor  
From Venda  
Order by 3
```

```
Select vendedor, cliente, valor  
From Venda  
Order by 2 Desc
```

# SQL - DQL

## □ Cláusula para Agrupamento:

- ❖ A cláusula **GROUP BY** é usada para agrupar tuplas.
- ❖ Todas as expressões da cláusula **SELECT** têm que aparecer como colunas formadoras de grupos na cláusula **GROUP BY**.
- ❖ A cláusula **GROUP BY** também pode ser usada em consultas contendo a cláusula **WHERE**.
  - Nesse caso, a **GROUP BY** é codificada depois da cláusula **WHERE**.

# SQL - DQL

## ❏ Exemplo:

```
Select vendedor, cliente, nome  
From Vendas  
Where valor between 500 and 1000  
Group by cliente, vendedor, valor
```

## Resultado:

VENDEDOR	CLIENTE	VALOR
ANA	CLÁUDIA CASTRO	600,00
JOÃO	CLÁUDIA CASTRO	500,00
MARIA	FERDINANDO COLLOR	800,00
ANA	ZÉLIA CARDOSO	700,00

❖ As linhas que não atenderam a cláusula *where* não foram selecionadas.



# SQL - DQL

## ❑ Atenção:

- ❖ As funções agregadas **avg**, **sum**, **max**, **min** e **count**, **não podem** ser usadas em cláusula **GROUP BY**, pois geram um único valor e por isso não podem agrupar linhas.

```
Select vendedor, sum(valor) as total  
From Vendas  
Group by vendedor
```

VENDEDOR	TOTAL
ANA	1600,00
JOÃO	1000,00
MARIA	1000,00

- ❖ A função agregada está no *select*, e não na cláusula *group by*.

# SQL - DQL

## ❑ Cláusula para Agrupamento:

- ❖ A cláusula **HAVING** é usada para agrupar tuplas.
- ❖ A cláusula **HAVING** nos permite estreitar a área de atuação da cláusula **Group by** da mesma forma que a cláusula **Where** estreita a área de atuação de um **Select** , ou seja, através de uma condição de pesquisa.
- ❖ Ao contrário da clausula **Where**, a cláusula **Having** pode conter funções agregadas.

# SQL - DQL

## ❑ Exemplo:

```
Select vendedor, sum(valor) as total  
From Vendas  
Group by vendedor  
Having sum(valor) > 1000
```

### Resultado:

VENDEDOR	TOTAL
ANA	1600,00

# SQL - DQL

## ❑ Operações sobre Conjuntos:

- ❖ As operações sobre conjuntos união (**UNION**), interseção (**INTERSECT**), e exceção (**EXCEPT**) opera em relações e corresponde às operações da álgebra relacional  $\cup$ ,  $\cap$  e  $-$ .
- ❖ Cada uma das operações acima automaticamente elimina duplicados.
  - Para manter todos os duplicados tem-se que usar UNION ALL, INTERSECT ALL e EXCEPT ALL.

# SQL - DQL

□ Tabelas:

## Alunos

Output pane			
Data Output Explain Messages History			
	<b>matricula</b> <b>integer</b>	<b>nome</b> <b>character varying(40)</b>	<b>endereco</b> <b>character varying(40)</b>
<b>1</b>	1	ANA	RUA A
<b>2</b>	3	RAFAEL	RUA C
<b>3</b>	2	ISABEL	RUA A

## ExAlunos

Output pane			
Data Output Explain Messages History			
	<b>matricula</b> <b>integer</b>	<b>nome</b> <b>character varying(40)</b>	<b>endereco</b> <b>character varying(40)</b>
<b>1</b>	1	ABEL	RUA A
<b>2</b>	3	JOANA	RUA B
<b>3</b>	2	ISABEL	RUA A

# SQL - DQL

## ❑ Exemplo **UNION**:

```
Select nome, endereco From ExAlunos  
Union  
Select nome, endereco From Alunos  
Order by 1
```

Output pane		
Data Output Explain Messages History		
	nome character varying(40)	endereco character varying(40)
1	ABEL	RUA A
2	ANA	RUA A
3	ISABEL	RUA A
4	JOANA	RUA B
5	RAFAEL	RUA C

# SQL - DQL

## ❑ Exemplo UNION:

```
Select nome, endereco From ExAlunos  
Union all  
Select nome, endereco From Alunos  
Order by 1
```

Output pane		
Data Output Explain Messages History		
	<b>nome</b> character varying(40)	<b>endereco</b> character varying(40)
1	ABEL	RUA A
2	ANA	RUA A
3	ISABEL	RUA A
4	ISABEL	RUA A
5	JOANA	RUA B
6	RAFAEL	RUA C

# SQL - DQL

## ❑ Exemplo **INTERSECT**:

```
Select nome, endereco From ExAlunos  
Intersect  
Select nome, endereco From Alunos  
Order by 1
```

Output pane		
Data Output Explain Messages History		
	nome character varying(40)	endereco character varying(40)
1	ISABEL	RUA A

❖ Para esses dados não haverá diferença entre *Intersect* e *Intersect All*.



# SQL - DQL

## ❑ Exemplo **EXCEPT**:

```
Select nome, endereco From ExAlunos  
Except  
Select nome, endereco From Alunos  
Order by 1
```

Output pane		
Data Output Explain Messages History		
	nome character varying(40)	endereco character varying(40)
1	ABEL	RUA A
2	JOANA	RUA B

❖ Para esses dados não haverá diferença entre *Except* e *except All*.

# SQL - DQL

## □ Funções de Junção:

❖ A junção (**JOIN**) corresponde a operação da álgebra relacional ►◄.

❖ Existem quatro tipos de junção:

- INNER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN.

❖ As expressões para junção podem ser usadas na cláusula FROM ou em qualquer outro lugar.

# SQL - DQL

## ❑ INNER JOIN **versus** Produto Cartesiano:

- ❖ Exemplo: encontre os empréstimos com seus respectivos clientes.

```
select d.nome, a.nome_agencia, a.cidade  
from Devedor d, Agencia a, Conta c  
where d.numero_conta = c.numero  
and c.idagencia = a.idagencia
```

**ou**

```
select d.nome, a.nome_agencia, a.cidade  
from Devedor d  
[inner] join Conta c on c.numero = d.numero_conta  
[inner] join Agencia a on a.idagencia = c.idagencia
```

# SQL - DQL

## ❑ INNER JOIN versus Produto Cartesiano:

Data Output	Explain	Messages	Notifications																				
<table><thead><tr><th></th><th>nome character varying (40)</th><th>nome_agencia character varying (20)</th><th>cidade character varying (20)</th></tr></thead><tbody><tr><td>1</td><td>Celso Pitta</td><td>Tirol</td><td>Natal</td></tr><tr><td>2</td><td>Paulo Maluf</td><td>Center Onze</td><td>Natal</td></tr></tbody></table>		nome character varying (40)	nome_agencia character varying (20)	cidade character varying (20)	1	Celso Pitta	Tirol	Natal	2	Paulo Maluf	Center Onze	Natal			<table><tr><th>Data Output</th><th>Explain</th><th>Messages</th><th>Notifications</th></tr><tr><td colspan="4">Successfully run. Total query runtime: 63 msec. 2 rows affected.</td></tr></table>	Data Output	Explain	Messages	Notifications	Successfully run. Total query runtime: 63 msec. 2 rows affected.			
	nome character varying (40)	nome_agencia character varying (20)	cidade character varying (20)																				
1	Celso Pitta	Tirol	Natal																				
2	Paulo Maluf	Center Onze	Natal																				
Data Output	Explain	Messages	Notifications																				
Successfully run. Total query runtime: 63 msec. 2 rows affected.																							

Data Output	Explain	Messages	Notifications																				
<table><thead><tr><th></th><th>nome character varying (40)</th><th>nome_agencia character varying (20)</th><th>cidade character varying (20)</th></tr></thead><tbody><tr><td>1</td><td>Celso Pitta</td><td>Tirol</td><td>Natal</td></tr><tr><td>2</td><td>Paulo Maluf</td><td>Center Onze</td><td>Natal</td></tr></tbody></table>		nome character varying (40)	nome_agencia character varying (20)	cidade character varying (20)	1	Celso Pitta	Tirol	Natal	2	Paulo Maluf	Center Onze	Natal			<table><tr><th>Data Output</th><th>Explain</th><th>Messages</th><th>Notifications</th></tr><tr><td colspan="4">Successfully run. Total query runtime: 61 msec. 2 rows affected.</td></tr></table>	Data Output	Explain	Messages	Notifications	Successfully run. Total query runtime: 61 msec. 2 rows affected.			
	nome character varying (40)	nome_agencia character varying (20)	cidade character varying (20)																				
1	Celso Pitta	Tirol	Natal																				
2	Paulo Maluf	Center Onze	Natal																				
Data Output	Explain	Messages	Notifications																				
Successfully run. Total query runtime: 61 msec. 2 rows affected.																							

# SQL - DQL

## ❑ Exemplo:

```
Select e.nome, e.fone, t.salario  
From Empregado e  
Join Trabalha t on t.idEmpregado = e.idEmpregado
```

Output pane

	nome character varying(40)	fone character varying(9)	salario double precision
1	Roberto Carlos	3211-0987	600.15
2	Ronaldo Fenômeno	9898-0900	2600
3	Fernanda Montenegro	3451-1012	1500
4	Zeca Pagodinho	7777-7777	350

# SQL - DQL

## ❑ LEFT OUTER JOIN:

- ❖ Exemplo: encontre os empréstimos com seus respectivos clientes, se houver.

```
Select d.nome, e.data  
From Emprestimo e  
Left [outer] join devedor d on  
d.codigo_cliente = e.codigo_cliente
```

- ❖ Nota: **LEFT OUTER JOIN** exibe todos os empréstimos, definindo nulo para os empréstimos que não estão associados.

# SQL - DQL

## ❏ Exemplo:

```
Select e.nome, e.fone, t.salario  
From Empregado e  
Left join Trabalha t on  
t.idEmpregado = e.idEmpregado
```

Output pane

	nome character varying(40)	fone character varying(9)	salario double precis
1	Roberto Carlos	3211-0987	600.15
2	Ronaldo Fenômeno	9898-0900	2600
3	Fernanda Montenegro	3451-1012	1500
4	Zeca Pagodinho	7777-7777	350
5	Fernando Collor	3456-9876	

# SQL - DQL

## ❑ RIGHT OUTER JOIN:

- ❖ Exemplo: encontre os empréstimos, se houver, com seus respectivos clientes.

```
Select d.nome, e.data  
From Emprestimo e  
Right [outer] join devedor d on  
d.codigo_cliente = e.codigo_cliente
```

- ❖ Nota: **RIGHT OUTER JOIN** exibe todos os empréstimos, definindo nulo para os clientes que têm nenhum empréstimo realizado.



# SQL - DQL

## ❑ Exemplo:

```
Select e.nome, e.fone, t.salario  
From Empregado e  
Right join Trabalha t on  
t.idEmpregado = e.idEmpregado
```

Output pane

	nome character varying(40)	fone character varying(9)	salario double precision
1	Roberto Carlos	3211-0987	600.15
2	Ronaldo Fenômeno	9898-0900	2600
3	Fernanda Montenegro	3451-1012	1500
4	Zeca Pagodinho	7777-7777	350

❖ Por que o resultado ficou igual ao resultado anterior do ***INNER JOIN***?

# SQL - DQL

## ❑ FULL OUTER JOIN:

- ❖ Exemplo: encontre os empréstimos, se houver, com seus respectivos clientes, se houver.

```
Select d.nome, e.data  
From Emprestimo e  
Full [outer] join devedor d on  
d.codigo_cliente = e.codigo_cliente
```

- ❖ Nota: **FULL OUTER JOIN** exibe todos os empréstimos, definindo nulo para os clientes que têm nenhum empréstimo realizado e os empréstimos que estão associados a nenhum cliente.

# SQL - DQL

## ❑ Exemplo:

```
Select ta.salario, tu.data_demissao  
From Trabalhou tu  
Full join Trabalha ta on  
ta.idEmpregado = tu.idEmpregado
```

Output pane		
Data Output Explain Messages History		
	salario double precision	data_demissao date
1	600.15	
2	2600	
3	1500	
4	350	
5		2006-12-30

---

# That's all folks...

