



**A TRANSMISSÃO DA  
AULA COMEÇA EM**

**INSTANTES**



# Gestão da Informação

# **Bando de Dados**

Prof. Dr. Ronaldo Castro de Oliveira

[ronaldo.co@ufu.br](mailto:ronaldo.co@ufu.br)

FACOM - 2022



# Comando AGRUPAMENTOS



# Agrupamentos

---

SELECT [ ALL | DISTINCT [ ON ( expressão [, ...] ) ] ]  
\* | expressão [ AS nome\_de\_saída ] [, ...]  
[ FROM item\_do\_from [, ...] ]  
[ WHERE condição ]  
[ GROUP BY expressão [, ...] ]  
[ HAVING condição [, ...] ]  
[ { UNION | INTERSECT | EXCEPT } [ ALL ] seleção ]  
[ ORDER BY expressão [ ASC | DESC | USING operador ] [, ...] ]  
[ LIMIT { contador | ALL } ]  
[ OFFSET início ]  
[ FOR UPDATE [ OF nome\_da\_tabela [, ...] ] ]



# Agrupamentos

- ▶ Podemos dividir o conjunto de tuplas de uma relação em grupos de acordo com algum critério, baseado nos valores dos atributos
- ▶ Por exemplo, na tabela *dependentes* as tuplas pode ser agrupadas de acordo com o parentesco do dependente
- ▶ Exemplo: pai, filho, irmã, irmão, tio, avó

Felipe	108	M	25/02/1997	Filho
Felipe	115	M	15/12/2004	Filho
Felipe	116	M	29/03/2001	Filho
Felipe	117	M	30/04/2000	Filho
Renato JR	1000	M	24/05/1968	Filho
Joaquim	1006	M	30/06/2005	Filho
Joao	101	M	11/10/1992	Filho
Paula	200	F	25/06/1975	Irmã
Maria	1201	F	20/06/1988	Irmã
Alberto	1202	M	15/02/1958	Irmão
João	1201	M	28/09/1992	Irmão
Tania	208	F	10/06/1950	Mãe
Maria_Inês	1203	F	15/11/1940	Mãe
Dolores Martins	704	F	20/05/1956	Mãe
Maria Junger	403	F	05/10/1954	Mãe
Jose Moreno	204	M	15/08/1970	Marido
Joao	207	M	05/05/1950	Pai

# Agrupamentos

- ▶ Por exemplo, na tabela *dependentes* as tuplas pode ser agrupadas de acordo com o parentesco do dependente
- ▶ Exemplo: pai, filho, irmã, irmão, tio, avó

Felipe	108	M	25/02/1997	Filho
Felipe	115	M	15/12/2004	Filho
Felipe	116	M	29/03/2001	Filho
Felipe	117	M	30/04/2000	Filho
Renato JR	1000	M	24/05/1968	Filho
Joaquim	1006	M	30/06/2005	Filho
Joao	101	M	11/10/1992	Filho
Paula	200	F	25/06/1975	Irmã
Maria	1201	F	20/06/1988	Irmã
Alberto	1202	M	15/02/1958	Irmão
João	1201	M	28/09/1992	Irmão
Tania	208	F	10/06/1950	Mãe
Maria_Inês	1203	F	15/11/1940	Mãe
Dolores Martins	704	F	20/05/1956	Mãe
Maria Junger	403	F	05/10/1954	Mãe
Jose Moreno	204	M	15/08/1970	Marido
Joao	207	M	05/05/1950	Pai

# Agrupamentos

*/\* parentesco de todos os dependentes\*/*

```
SELECT parentesco
FROM dependentes
GROUP BY parentesco;
```



Marido
Irmã
Filho
Pai
Irmão
Mãe

- ▶ Observe que na cláusula **SELECT** só podem constar os atributos presentes no **GROUP BY**
- ▶ Isso faz sentido pois, por exemplo, existem 2 irmãs cadastradas, qual delas apareceria no resultado?

*/\* parentesco de todos os dependentes\*/*

```
SELECT nome_dependente, parentesco
FROM dependentes
GROUP BY parentesco;
```



**ERRO:** coluna "dependentes.nome\_dependente" deve aparecer na cláusula **GROUP BY** ou ser utilizada em uma função de agregação

## Exemplo banco – Company (Empresa):

-- apresenta os parentescos de dependentes dos empregados

```
select d.relationship
      from dependent d
      group by d.relationship;
```

# Agrupamentos

- ▶ Mais de um atributo pode ser usado no agrupamento

*/\* agrupando as filiais por cidade/estado\*/*

**SELECT** cidade, uf

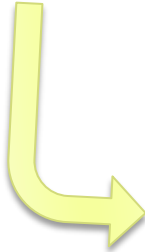
**FROM** filiais

**GROUP BY** cidade, uf

Tabela Filiais:

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG

Resultado da  
Consulta com  
GROUP BY



Rio de Janeiro	RJ
São Paulo	SP
Uberlândia	MG
Belo Horizonte	MG
Angra dos Reis	RJ



# Funções agregadas + Agrupamentos

- ▶ As funções agregadas (e.g., COUNT, MIN, MAX, AVG) podem ser usadas para cálculos com subgrupos de tuplas definidos pela cláusula GROUP BY

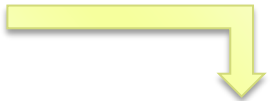
**/\* Qual o número de filiais por estado \*/**

```
SELECT uf, COUNT(uf) -- poderia ser COUNT(*)  
FROM filiais  
GROUP BY uf
```

Tabela Filiais:

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG

Resultado da  
Consulta com  
GROUP BY



	estado character(2)	count bigint
1	RJ	3
2	SP	3
3	MG	4

## Exemplo banco – Company (Empresa):

--apresenta a quantidade de projetos em cada localização

```
select p.plocation, count(p.plocation)  
  from project p  
 group by p.plocation;
```

# Agrupamentos


- Inserindo uma nova filial:  
Internet

*/\* agrupando as filiais por estado\*/*


```
SELECT UF  
FROM filiais  
GROUP BY UF
```

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG
Internet		

Resultado da  
Consulta com  
GROUP BY



	uf character(2)
1	
2	MG
3	RJ
4	SP



Campos Nulos  
também são  
agrupados

# Agrupamentos

## ► Inserindo uma nova filial: Internet

*/\* agrupando as filiais por estado e  
contando o número por estado\*/*


**SELECT UF, COUNT(\*)**

**FROM filiais**


**GROUP BY UF**

Resultado da  
Consulta com  
GROUP BY

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG
Internet		



	uf character(2)	count bigint
1		1
2	MG	4
3	RJ	3
4	SP	3



Campos Nulos  
também são  
agrupados e  
contados

# Agrupamentos


- Inserindo uma nova filial:  
Internet

*/\* agrupando as filiais por estado e  
contando o número por estado\*/*


```
SELECT UF, COUNT(UF)  
FROM filiais  
GROUP BY UF
```

Resultado da  
Consulta com  
GROUP BY

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG
Internet		



	uf character(2)	count bigint
1		0
2	MG	4
3	RJ	3
4	SP	3



Campos Nulos  
também são  
agrupados

# Agrupamentos


## ► Inserindo uma nova filial: Internet

*/\* agrupando as filiais por estado e  
contando o número por estado\*/*


```
SELECT UF, COUNT(Nome)
FROM filiais
GROUP BY UF
```

Resultado da  
Consulta com  
GROUP BY

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG
Internet		



	uf character(2)	count bigint
1		1
2	MG	4
3	RJ	3
4	SP	3



Campos Nulos  
também são  
agrupados e  
contados


# Agrupamentos

- Inserindo uma nova filial:  
Internet


*/\* agrupando as filiais por estado e  
contando o número por estado\*/*

```
SELECT UF, COUNT(Cidade)  
FROM filiais  
GROUP BY UF
```

Resultado da  
Consulta com  
GROUP BY



	uf character(2)	count bigint
1		0
2	MG	4
3	RJ	3
4	SP	3



Campos Nulos  
também são  
agrupados

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG
Internet		


# Agrupamentos

- Inserindo uma nova filial:  
Internet

*/\* agrupando as filiais por estado e  
contando o número por estado\*/*

```
SELECT UF, COUNT(Distinct Cidade)  
FROM filiais  
GROUP BY UF
```

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG
Internet		



	uf character(2)	count bigint
1	MG	2
2	RJ	2
3	SP	1
4		0

Cada cidade só é  
contada uma vez

Campos Nulos  
também são  
agrupados

# Funções agregadas + Agrupamentos

- ▶ As funções agregadas (e.g., COUNT, MIN, MAX, AVG) podem ser usadas para cálculos com subgrupos de tuplas definidos pela cláusula GROUP BY

*/\* Qual o número de filiais por estado \*/*

**SELECT** estado, **COUNT**(estado) -- poderia ser **COUNT(\*)**

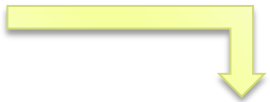
**FROM** filiais

**GROUP BY** estado

Tabela Filiais:

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG

Resultado da  
Consulta com  
GROUP BY



	estado character(2)	count bigint
1	RJ	3
2	SP	3
3	MG	4



# Comando SELECT

## CLÁUSULA HAVING

- ▶ **HAVING:** é semelhante à cláusula WHERE. HAVING elimina tuplas *agrupadas* que não satisfazem a uma determinada condição.
- ▶ Diferença com WHERE: WHERE filtra tuplas *individuais* antes da aplicação do GROUP BY, enquanto HAVING filtra grupo de tuplas criadas por GROUP BY. As condições de filtragem do HAVING devem ser feitas baseando-se nos atributos agrupados por GROUP BY

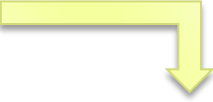
### EXEMPLO:

*/\* Listar os estados com  
mais de 3 filiais\*/*

```
SELECT uf
FROM filiais
GROUP BY uf
HAVING COUNT(*) >3
```

Tabela Filiais:

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG



	estado character(2)
1	MG

# Comando SELECT CLÁUSULA HAVING

## EXEMPLO:

*/\* Listar os estados com mais de 3 filiais  
renomeando o atributo de saída como nfiliais\*/*

**SELECT** UF, Count(\*) **AS** nfiliais


**FROM** filiais

**GROUP BY** UF

**HAVING** COUNT(\*) >3

Tabela Filiais:

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG



	uf character(2)	nfiliais bigint
1	MG	4

## Exemplo banco – Company (Empresa):

-- apresenta os parentescos de dependentes dos empregados diferente de 'SPOUSE'

```
select d.relationship as "Parentesco"  
      from dependent d  
      group by d.relationship  
      having d.relationship <> 'SPOUSE';
```

--apresenta a quantidade de projetos em cada localização com numero de projeto maior que 1

```
select p.plocation, count(p.plocation)  
      from project p  
      group by p.plocation  
      having count(p.plocation) >1;
```

# Comando SELECT CLÁUSULA HAVING

## EXEMPLO:

*/\* Listar todos os estados, exceto SP, com  
mais de 3 filiais renomeando o atributo de saída como nfiliais\*/*

**SELECT** UF, Count(\*) **AS** nfiliais

**FROM** filiais

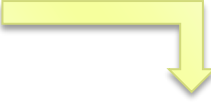
**WHERE** UF <> 'SP'

**GROUP BY** UF

**HAVING** COUNT(\*) >3

Tabela Filiais:

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG



	uf character(2)	nfiliais bigint
1	MG	4

# Comando SELECT CLÁUSULA HAVING


## EXEMPLO:

*/\* Listar todos os estados, exceto SP, com  
mais de 3 filiais renomeando o atributo de saída como nfiliais\*/*

**SELECT UF, Count(\*) AS nfiliais**  
**FROM filiais**  
**GROUP BY UF**  
**HAVING COUNT(\*) >3**  
**AND UF <> 'SP'**

Tabela Filiais:

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ
Barra	Rio de Janeiro	RJ
Angra	Angra dos Reis	RJ
Pampulha	Belo Horizonte	MG
Glória	Belo Horizonte	MG
Tibery	Uberlândia	MG
Centro	Uberlândia	MG



	uf character(2)	nfiliais bigint
1	MG	4

# Comando SELECT CLÁUSULA HAVING

## EXEMPLO: Erro comum

*/\* Listar os estados com  
mais de 3 filiais\*/*

**SELECT** UF, Count(\*) **AS** nfiliais  
**FROM** filiais  
**GROUP BY** UF  
**HAVING** nfiliais >3

Tabela Filiais:

Nome	Cidade	UF
Brás	São Paulo	SP
Limão	São Paulo	SP
Penha	São Paulo	SP
Tijuca	Rio de Janeiro	RJ

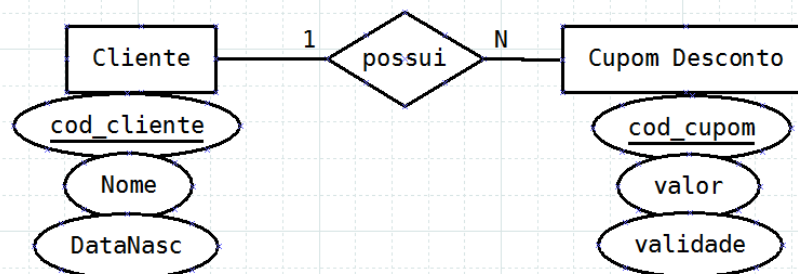
ERRO: coluna "nfiliais" não existe  
LINE 4: HAVING nfiliais >3

Centro	Uberlândia	MG
--------	------------	----

# Exemplo

```
SELECT * FROM cliente;
```

cod_cliente	nome	data_nasc
1	Maria	2000-01-05
2	Ana	1998-05-03
3	Carlos	1990-02-02



```
SELECT * FROM cupom_desconto;
```

num_cupom	valordesconto	validade	cod_cliente
1	0.3	2020-10-20	1
2	0.4	2020-10-20	1
3	0.15	2020-10-20	1
4	0.15	2020-10-20	2
5	0.15	2020-10-20	2

```
SELECT * FROM cliente NATURAL JOIN cupom_desconto;
```

cod_cliente	nome	data_nasc	num_cupom	valordesconto	validade
1	Maria	2000-01-05	1	0.3	2020-10-20
1	Maria	2000-01-05	2	0.4	2020-10-20
1	Maria	2000-01-05	3	0.15	2020-10-20
2	Ana	1998-05-03	4	0.15	2020-10-20
2	Ana	1998-05-03	5	0.15	2020-10-20

# Atributos no SELECT com GROUP BY

cod_cliente	nome	data_nasc	num_cupom	valordesconto	validade
1	Maria	2000-01-05	1	0.3	2020-10-20
1	Maria	2000-01-05	2	0.4	2020-10-20
1	Maria	2000-01-05	3	0.15	2020-10-20
2	Ana	1998-05-03	4	0.15	2020-10-20
2	Ana	1998-05-03	5	0.15	2020-10-20

## EXEMPLO:

*/\* Listar a quantidade de cupons por cliente\*/*

```
SELECT cliente.cod_cliente, COUNT(num_cupom) AS qte_cupom
FROM cliente INNER JOIN cupom_desconto
ON cliente.cod_cliente = cupom_desconto.cod_cliente
GROUP BY cliente.cod_cliente
```

cod_cliente	qte_cupom
1	3
2	2



# Atributos no SELECT com GROUP BY

cod_cliente	nome	data_nasc	num_cupom	valordesconto	validade
1	Maria	2000-01-05	1	0.3	2020-10-20
1	Maria	2000-01-05	2	0.4	2020-10-20
1	Maria	2000-01-05	3	0.15	2020-10-20
2	João	1999-05-02	4	0.15	2020-10-20
2	João	1999-05-02	5	0.15	2020-10-20

Somente atributos que estão no GROUP BY podem aparecer no SELECT

## EXEMPLO:

*/\* Listar a quantidade de cupons por cliente\*/*

```
SELECT cliente.cod_cliente, COUNT(num_cupom) AS qte_cupom
FROM cliente INNER JOIN cupom_desconto
ON cliente.cod_cliente = cupom_desconto.cod_cliente
GROUP BY cliente.cod_cliente
```

cod_cliente	qte_cupom
1	3
2	2



# Atributos no SELECT com GROUP BY

cod_cliente	nome	data_nasc	num_cupom	valordesconto	validade
1	Maria	2000-01-05	1	0.3	2020-10-20
1	Maria	2000-01-05	2	0.4	2020-10-20
1	Maria	2000-01-05	3	0.15	2020-10-20
2	João	1999-05-02	4	0.15	2020-10-20
2	João	1999-05-02	5	0.15	2020-10-20

Os outros atributos podem constar dentro de funções de agregação

## EXEMPLO:

*/\* Listar a quantidade de cupons por cliente\*/*

```
SELECT cliente.cod_cliente, COUNT(num_cupom) AS qte_cupom
FROM cliente INNER JOIN cupom_desconto
    ON cliente.cod_cliente = cupom_desconto.cod_cliente
GROUP BY cliente.cod_cliente
```

cod_cliente	qte_cupom
1	3
2	2

# Atributos no SELECT com GROUP BY

cod_cliente	nome	data_nasc	num_cupom	valordesconto	validade
1	Maria	2000-01-05	1	0.3	2020-10-20

Somente atributos que estão no **GROUP BY** podem aparecer no **SELECT**.  
No exemplo abaixo, mesmo que  
`cupom_desconto.cod_cliente = cliente.cod_cliente`  
um erro ocorrerá

## EXEMPLO:

*/\* Listar a quantidade de cupons por cliente \*/*

```
SELECT cupom_desconto.cod_cliente, COUNT(num_cupom) AS qte_cupom
FROM cliente INNER JOIN cupom_desconto
      ON cliente.cod_cliente = cupom_desconto.cod_cliente
GROUP BY cliente.cod_cliente
```

ERROR: column "cupom\_desconto.cod\_cliente" must appear in the GROUP BY clause or be used in an aggregate function

LINE 1: SELECT cupom\_desconto.cod\_cliente, COUNT(num\_cupom) AS qte\_c...

# Atributos no SELECT com GROUP BY

cod_cliente	nome	data_nasc	num_cupom	valordesconto	validade
1	Maria	2000-01-05	1	0.3	2020-10-20
1	Maria	2000-01-05	2	0.4	2020-10-20
					20-10-20
					20-10-20
					20-10-20

A partir do SQL 1999, é possível colocar outros atributos que não estão agrupados no SELECT desde que o atributo usado no agrupamento seja uma chave primária

## EXEMPLO:

*/\* Listar a quantidade de cupons por cliente \*/*

```
SELECT cliente.cod_cliente, cliente.nome, COUNT(num_cupom) AS qte_cupom
FROM cliente INNER JOIN cupom_desconto
ON cliente.cod_cliente = cupom_desconto.cod_cliente
GROUP BY cliente.cod_cliente
```

PRIMARY KEY no  
agrupamento

cod_cliente	<b>nome</b>	qte_cupom
1	<b>Maria</b>	3
2	<b>Ana</b>	2

# Consultas aninhadas

## EXISTS e NOT EXISTS

---

- ▶ **EXISTS (subconsulta)**
  - ▶ retorna TRUE se existir ao menos uma tupla no resultado da subconsulta
- ▶ **NOT EXISTS (subconsulta)**
  - ▶ retorna TRUE se subconsulta retornar um conjunto vazio (zero tuplas)
- ▶ **Exemplo:**

```
SELECT col1
FROM tab1
WHERE EXISTS (SELECT 1
              FROM tab2
              WHERE col2 = tab1.col2);
```

# Consultas aninhadas

## EXISTS e NOT EXISTS

---

- ▶ O otimizador do SGBD pode executar a consulta apenas até determinar que tem ao menos uma tupla como resultado
- ▶ Como o resultado depende apenas de se há tuplas no resultado, uma convenção comum é escrever as cláusulas exists na forma:
  - ▶ ... EXISTS(SELECT | FROM...WHERE ...)

# Exemplo

```
SELECT * FROM produto;
```

cod_produto	nome	valor
1	Queijo	15.00
2	Goiabada	8.00
3	Doce de leite	7.00

```
SELECT * FROM cliente;
```

cod_cliente	nome	data_nasc
1	Maria	2000-01-05
2	Ana	1998-05-03
3	Carlos	1990-02-02

```
SELECT * FROM compra;
```

cod_cliente	cod_produto	datahora
1	1	2016-10-20 00:00:00
1	2	2016-10-20 00:00:00
1	3	2016-10-20 00:00:00
2	1	2016-10-20 00:00:00
2	2	2016-10-21 00:00:00
2	2	2016-10-22 00:00:00

*-- Mostrar os clientes que fizeram compras*

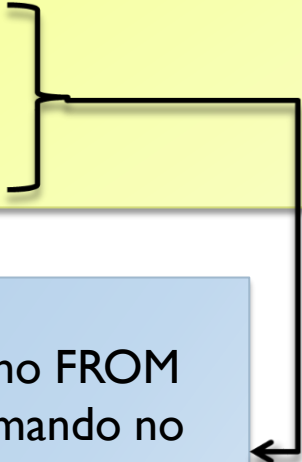
```
SELECT *  
FROM cliente  
WHERE EXISTS  
    ( SELECT 1  
      FROM compra  
      WHERE compra.cod_cliente =  
            cliente.cod_cliente)
```



# Exemplo

*-- Mostrar os clientes que fizeram compras*

```
SELECT *  
FROM cliente  
WHERE EXISTS  
    ( SELECT 1  
      FROM compra  
      WHERE compra.cod_cliente = cliente.cod_cliente)
```



Subconsulta com uma tabela no FROM  
(tabela compra) mas com comando no  
WHERE que usa uma tabela da consulta  
externa



# Exemplo

-- *Mostrar os clientes que fizeram compras*

```
SELECT *  
FROM cliente  
WHERE EXISTS  
    ( SELECT 1  
      FROM compra  
      WHERE compra.cod_cliente = cliente.cod_cliente)
```

Para cada tupla da consulta externa, a consulta interna é feita

cod_cliente	nome	data_nasc
1	Maria	2000-01-05
2	Ana	1998-05-03
3	Carlos	1990-02-02





# Exemplo

-- *Mostrar os clientes que fizeram compras*

```
SELECT *  
FROM cliente  
WHERE EXISTS  
    ( SELECT 1  
      FROM compra  
      WHERE compra.cod_cliente = cliente.cod_cliente)
```

cod_cliente	nome	data_nasc
1	Maria	2000-01-05
2	Ana	1998-05-03
3	Carlos	1990-02-02

Como temos 3 clientes, a subconsulta é executada 3 vezes

# Exemplo

-- *Mostrar os clientes que fizeram compras*

```
SELECT *  
FROM cliente  
WHERE EXISTS  
    ( SELECT 1  
      FROM compra  
      WHERE compra.cod_cliente = cliente.cod_cliente)
```

cod_cliente	nome	data_nasc
1	Maria	2000-01-05
2	Ana	1998-05-03
3	Carlos	1990-02-02

Para cada execução da subconsulta  
o EXISTS verifica se a subconsulta  
retornou algum resultado

# Exemplo

```
SELECT * FROM produto;
```

cod_produto	nome	valor
1	Queijo	15.00
2	Goiabada	8.00
3	Doce de leite	7.00

```
SELECT * FROM cliente;
```

cod_cliente	nome	data_nasc
1	Maria	2000-01-05
2	Ana	1998-05-03
3	Carlos	1990-02-02

```
SELECT * FROM compra;
```

cod_cliente	cod_produto	datahora
1	1	2016-10-20 00:00:00
1	2	2016-10-20 00:00:00
1	3	2016-10-20 00:00:00
2	1	2016-10-20 00:00:00
2	2	2016-10-21 00:00:00
2	2	2016-10-22 00:00:00

-- Mostrar os clientes que **NÃO** fizeram compras

```
SELECT *  
FROM cliente  
WHERE NOT EXISTS  
    ( SELECT 1  
      FROM compra  
      WHERE compra.cod_cliente =  
            cliente.cod_cliente)
```



# Exemplo

```
SELECT * FROM produto;
```

cod_produto	nome	valor
1	Queijo	15.00
2	Goiabada	8.00
3	Doce de leite	7.00

-- *Mostrar os clientes que compraram **todos** os produtos*

```
SELECT * FROM cliente;
```

cod_cliente	nome	data_nasc
1	Maria	2000-01-05
2	Ana	1998-05-03
3	Carlos	1990-02-02

```
SELECT * FROM compra;
```

cod_cliente	cod_produto	datahora
1	1	2016-10-20 00:00:00
1	2	2016-10-20 00:00:00
1	3	2016-10-20 00:00:00
2	1	2016-10-20 00:00:00
2	2	2016-10-21 00:00:00
2	2	2016-10-22 00:00:00



# Exemplo

```
SELECT * FROM produto;
```

cod_produto	nome	valor
1	Queijo	15.00
2	Goiabada	8.00
3	Doce de leite	7.00

```
SELECT * FROM cliente;
```

cod_cliente	nome	data_nasc
1	Maria	2000-01-05
2	Ana	1998-05-03
3	Carlos	1990-02-02

```
SELECT * FROM compra;
```

cod_cliente	cod_produto	data
1	1	2016-10-2
1	2	2016-10-2
1	3	2016-10-2
2	1	2016-10-2
2	2	2016-10-2
2	2	2016-10-2

-- *Mostrar os clientes que compraram **todos** os produtos*

```
SELECT *  
FROM cliente  
WHERE NOT EXISTS  
  
    (SELECT cod_produto  
     FROM produto  
  
     EXCEPT  
  
     SELECT cod_produto  
     FROM compra  
     WHERE compra.cod_cliente =  
           cliente.cod_cliente  
    )
```

-- *Mostrar os clientes que compraram **todos** os produtos*

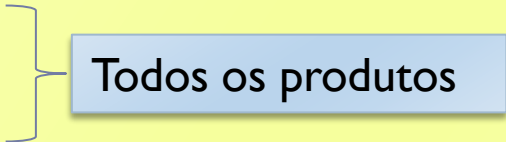
```
SELECT *  
FROM cliente  
WHERE NOT EXISTS  
  
  (SELECT cod_produto  
   FROM produto  
  
   EXCEPT  
  
   SELECT cod_produto  
   FROM compra  
   WHERE compra.cod_cliente = cliente.cod_cliente  
  )
```

Subconsulta  
correlacionada: ele é  
executada para cada  
cliente existente na  
base



-- *Mostrar os clientes que compraram **todos** os produtos*

```
SELECT *  
FROM cliente  
WHERE NOT EXISTS  
  
  (SELECT cod_produto  
   FROM produto  
  
   EXCEPT  
  
   SELECT cod_produto  
   FROM compra  
   WHERE compra.cod_cliente = cliente.cod_cliente  
  )
```



Todos os produtos



-- *Mostrar os clientes que compraram **todos** os produtos*

```
SELECT *  
FROM cliente  
WHERE NOT EXISTS  
  
  (SELECT cod_produto  
   FROM produto  
  
   EXCEPT  
  
   SELECT cod_produto  
   FROM compra  
   WHERE compra.cod_cliente = cliente.cod_cliente  
  )
```

Produtos para o  
cliente “atual”





-- *Mostrar os clientes que compraram **todos** os produtos*

```
SELECT *  
FROM cliente  
WHERE NOT EXISTS  
  
  (SELECT cod_produto  
   FROM produto  
  
   EXCEPT  
   )  
  SELECT cod_produto  
  FROM compra  
  WHERE compra.cod_cliente = cliente.cod_cliente  
  )
```

Ao subtrair o conjunto de todos os produtos dos produtos que o cliente comprou, teremos uma resposta vazia caso ele tenha comprado todos



-- *Mostrar os clientes que compraram **todos** os produtos*

```
SELECT *  
FROM cliente  
WHERE NOT EXISTS }  
  
    (SELECT cod_produto  
     FROM produto  
  
     EXCEPT  
  
     SELECT cod_produto  
     FROM compra  
     WHERE compra.cod_cliente = cliente.cod_cliente  
    )
```

**NOT EXISTS** retornará **TRUE** caso a subconsulta seja vazia, ou seja, caso o cliente tenha comprado todos os produtos



# Comando SELECT

## Consultas aninhadas correlacionadas

---

- ▶ Consultas aninhadas correlacionadas ocorrem quando o resultado da subconsulta (consulta interna) muda de acordo com a tupla que está sendo avaliada na consulta externa.
- ▶ Cuidado: isso pode ser muito lento, pois a subconsulta é reavaliada para cada linha da consulta externa. Se houver forma de evitar isso, sua consulta pode ser mais rápida



# Comando SELECT

## Itens do FROM

---

- ▶ É possível realizar consultas sobre os resultados obtidos em outras consultas. Isso pode ser feito adicionando a consulta na cláusula FROM
  - ▶ Relembrando a sintaxe

onde item\_do\_from pode ser um entre:

```
[ ONLY ] nome_da_tabela [ * ] [ [ AS ] alias [ ( alias_de_coluna [, ...] ) ] ]  
( seleção ) [ AS ] alias [ ( alias_de_coluna [, ...] ) ]  
nome_da_função ( [ argumento [, ...] ] ) [ AS ] alias [ ( alias_de_coluna [, ...]  
| definição_de_coluna [, ...] ) ]  
nome_da_função ( [ argumento [, ...] ] ) AS ( definição_de_coluna [, ...] )  
item_do_from [ NATURAL ] tipo_de_junção item_do_from [ ON condição_de_junção  
| USING ( coluna_de_junção [, ...] ) ]
```



# Comando SELECT

## Itens do FROM

---

### ► Exemplo

*/\* Listar os estados com 3 filiais (sem usar having)\*/*

**SELECT \***

**FROM (**

**SELECT** estado, count(\*) as nfiliais

**FROM** filiais

**GROUP BY** estado) **AS** t

**WHERE** t.nfiliais = 3



{[INNER] | {LEFT | RIGHT | FULL}[OUTER]} JOIN

# Sintaxe SELECT

---

```
SELECT [ ALL | DISTINCT [ ON ( expressão [, ...] ) ] ]  
* | expressão [ AS nome_de_saída ] [, ...]  
[ FROM item_do_from [, ...] ] – aula de hoje  
[ WHERE condição ]  
[ GROUP BY expressão [, ...] ]  
[ HAVING condição [, ...] ]  
[ { UNION | INTERSECT | EXCEPT } [ ALL ] seleção ]  
[ ORDER BY expressão [ ASC | DESC | USING operador ] [, ...] ]  
[ LIMIT { contador | ALL } ]  
[ OFFSET início ]  
[ FOR UPDATE [ OF nome_da_tabela [, ...] ] ]
```



# Itens FROM

---

onde item\_do\_from pode ser um entre:

[ ONLY ] nome\_da\_tabela [ \* ] [ [ AS ] alias [ ( alias\_de\_coluna [, ...] ) ] ]

( seleção ) [ AS ] alias [ ( alias\_de\_coluna [, ...] ) ]

nome\_da\_função ( [ argumento [, ...] ] ) [ AS ] alias [ ( alias\_de\_coluna [, ...]  
| definição\_de\_coluna [, ...] ) ]

nome\_da\_função ( [ argumento [, ...] ] ) AS ( definição\_de\_coluna [, ...] )

item\_do\_from [ NATURAL ] tipo\_de\_junção item\_do\_from [ ON condição\_de\_junção  
| USING ( coluna\_de\_junção [, ...] ) ]

Onde tipo\_de\_junção pode ser:

[ INNER ] JOIN

LEFT [ OUTER ] JOIN

RIGHT [ OUTER ] JOIN

FULL [ OUTER ] JOIN

CROSS JOIN

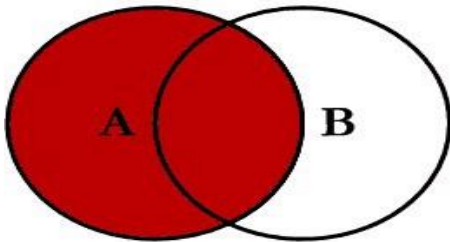
---



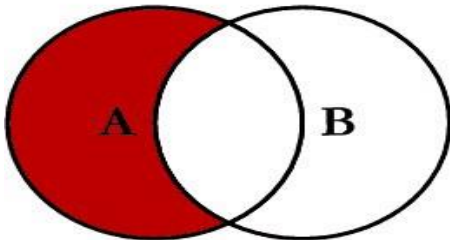


# Tipos de Junções

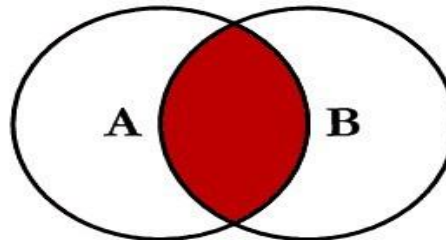
## SQL JOINS



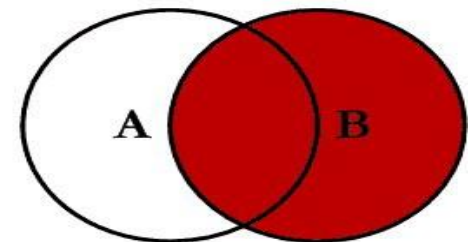
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



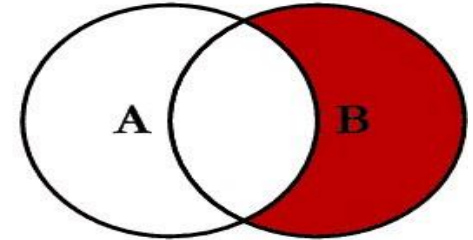
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



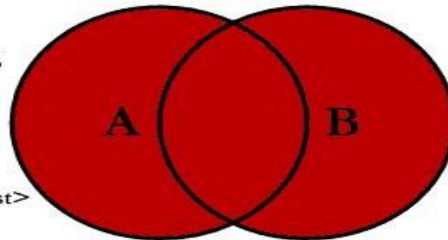
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



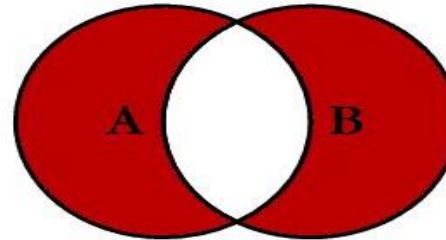
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```



*/\* Criando as tabelas\*/*

```
CREATE TABLE orientador (  
  id INT PRIMARY KEY,  
  nome VARCHAR(255)  
);
```

```
CREATE TABLE aluno (  
  matricula INT PRIMARY KEY,  
  nome VARCHAR(255),  
  orientador_id INT REFERENCES orientador(id)  
);
```

---





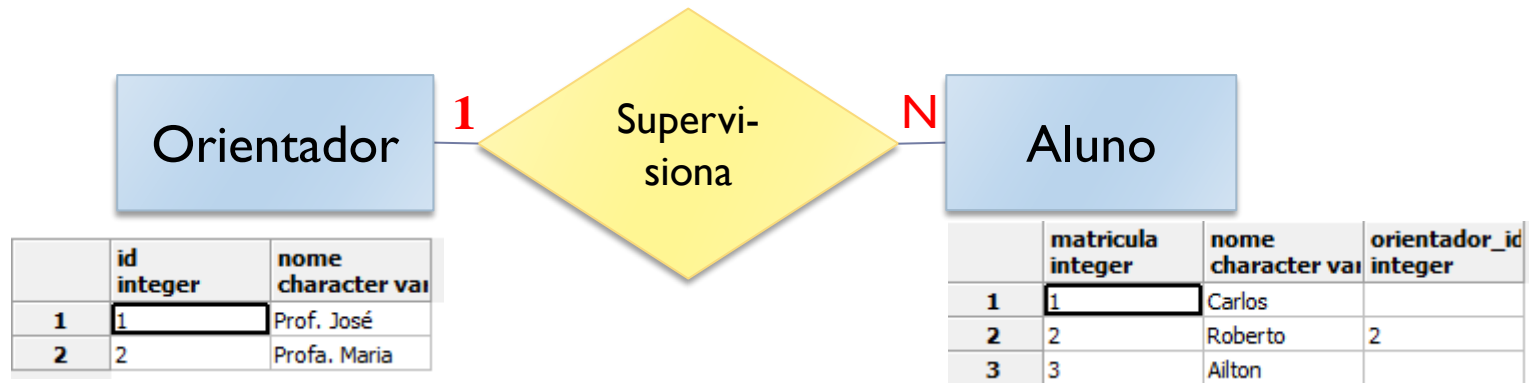
```
CREATE TABLE orientador (  
  id INT PRIMARY KEY,  
  nome VARCHAR(255)  
);
```

```
CREATE TABLE aluno (  
  matricula INT PRIMARY KEY,  
  nome VARCHAR(255),  
  orientador_id INT REFERENCES orientador(id)  
);
```

*/\* Povoando as tabelas\*/*

```
INSERT INTO orientador VALUES (1,'Prof. José'), (2,'Profa. Maria');  
INSERT INTO aluno VALUES (1,'Carlos',NULL), (2,'Roberto',2),  
(3,'Ailton',NULL)
```





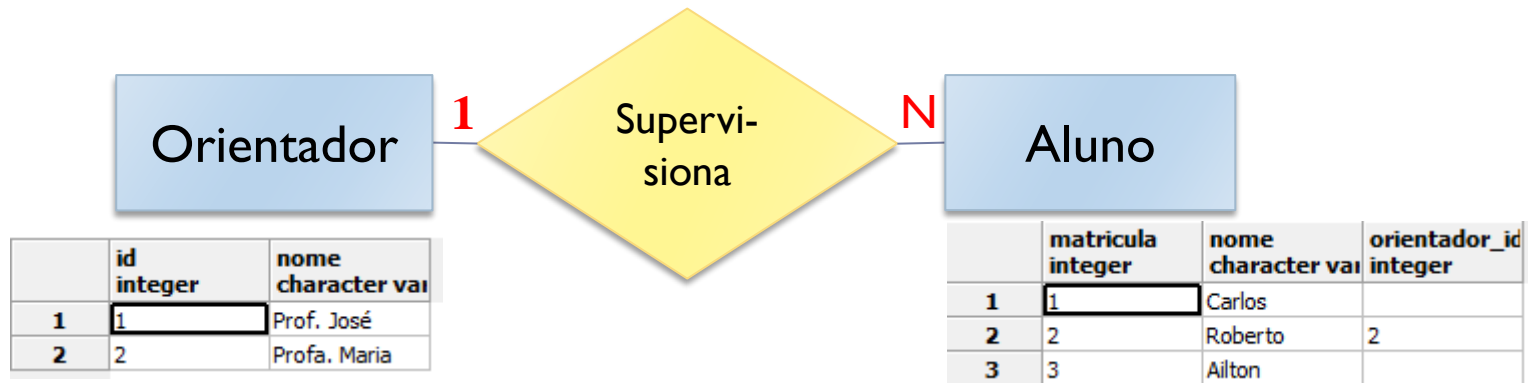
```

SELECT *
FROM orientador,aluno
WHERE aluno.orientador_id = orientador.id;

```

	id integer	nome character vari	matricula integer	nome character varying	orientador_id integer
1	2	Profa. Maria	2	Roberto	2





**SELECT \***

**FROM orientador INNER JOIN aluno**

**ON aluno.orientador\_id = orientador.id;**

	id integer	nome character varying	matricula integer	nome character varying	orientador_id integer
1	2	Profa. Maria	2	Roberto	2

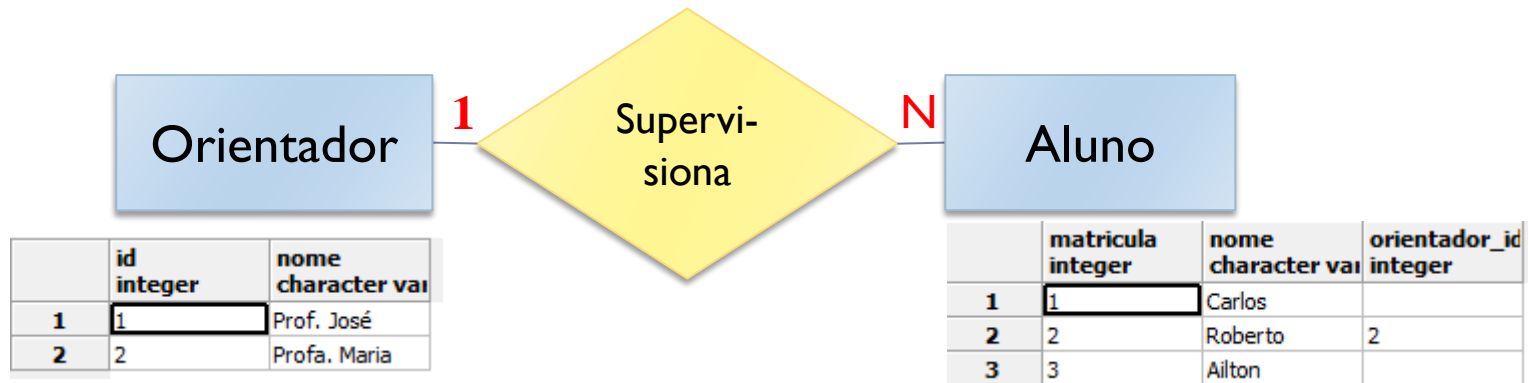
## Exemplo Banco de Dados - Company

-- Listar o nome e endereço dos empregados que trabalham no departamento 'Research'

**SELECT** fname, minit, lname, address

**FROM** (employee **INNER JOIN** department **ON** dno=dnumber)

**WHERE** dname='Research'



**SELECT \***

**FROM orientador LEFT OUTER JOIN aluno**

**ON aluno.orientador\_id = orientador.id;**

	id integer	nome character vai	matricula integer	nome character varying	orientador_id integer
1	1	Prof. José			
2	2	Profa. Maria	2	Roberto	2

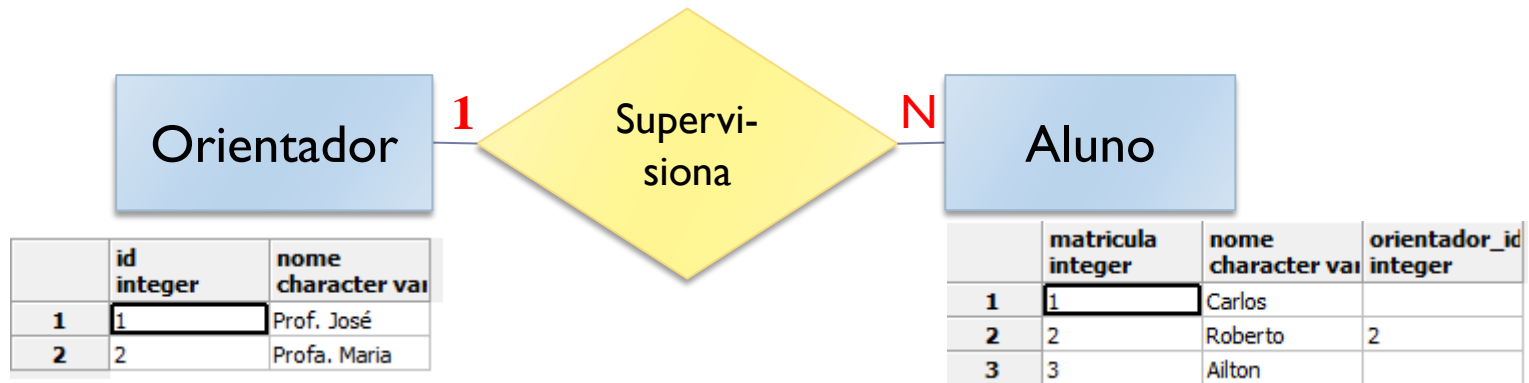
## Exemplo Banco de Dados - Company

-- Para cada empregado, liste o seu primeiro nome acompanhado do primeiro nome de seu supervisor, mesmo se o empregado não tiver supervisor

**SELECT** e.fname as employee\_name, s.fname as supervisor\_name

**FROM** (employee **AS** e **LEFT OUTER JOIN**

employee **AS** s **ON** e.superssn =s.ssn)



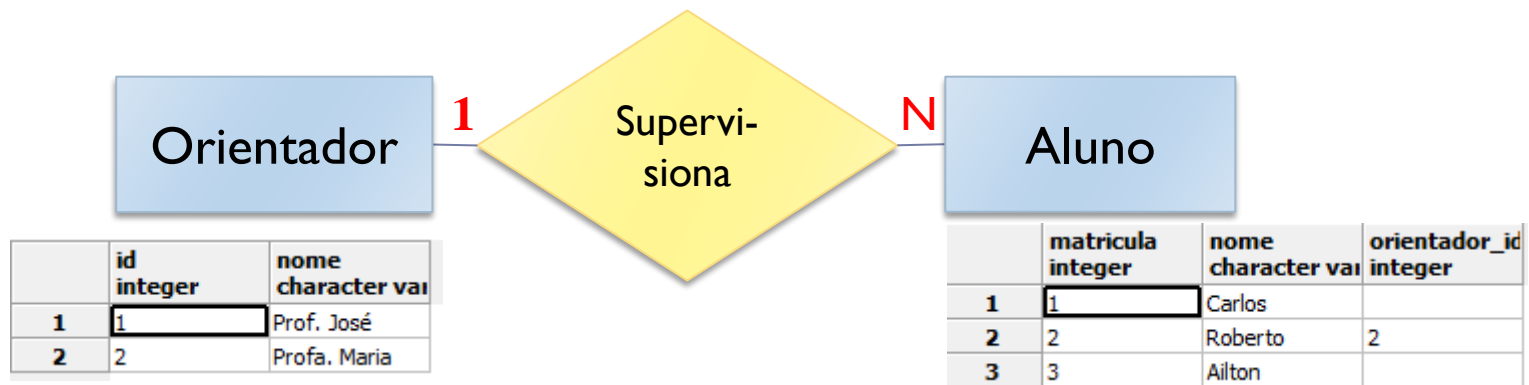
**SELECT \***  
**FROM** orientador **RIGHT OUTER JOIN** aluno  
**ON** aluno.orientador\_id = orientador.id;

	id integer	nome character variable	matricula integer	nome character varying	orientador_id integer
1			1	Carlos	
2	2	Profa. Maria	2	Roberto	2
3			3	Ailton	

### Exemplo Banco de Dados - Company

-- Para cada supervisor, liste o primeiro nome dos empregado que supervisiona, mesmo que não supervisione ninguém.

**SELECT** e.fname as employee\_name, s.fname as supervisor\_name  
**FROM** (employee **AS** e **RIGHT OUTER JOIN**  
employee **AS** s **ON** e.superssn =s.ssn)  
**ORDER BY** s.fname;



**SELECT \***

**FROM orientador FULL OUTER JOIN aluno**

**ON aluno.orientador\_id = orientador.id;**

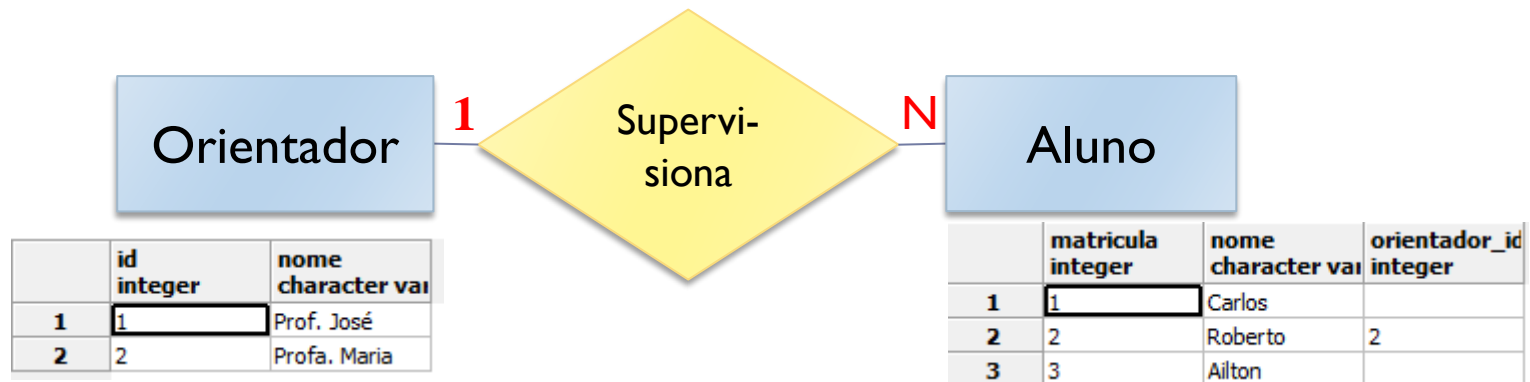
	id integer	nome character var	matricula integer	nome character var	orientador_id integer
1	1	Prof. José			
2	2	Profa. Maria	2	Roberto	2
3			1	Carlos	
4			3	Ailton	

## Exemplo Banco de Dados - Company

-- Liste o primeiro nome do supervisor e o primeiro nome de seus supervisionado, ordenado pelo primeiro. Mesmo se o empregado não for supervisor de ninguém, liste seu nome na primeira coluna e mesmo se o empregado não tiver supervisor, liste seu nome na segunda coluna.

```
SELECT s.fname as supervisor_name, e.fname as employee_name
FROM (employee AS e FULL OUTER join
      employee AS s ON e.superssn =s.ssn)
order by s.fname;
```





**SELECT \***

**FROM orientador CROSS JOIN aluno;**

	id integer	nome character vai	matricula integer	nome character vai	orientador_id integer
1	1	Prof. José	1	Carlos	
2	1	Prof. José	2	Roberto	2
3	1	Prof. José	3	Ailton	
4	2	Profª. Maria	1	Carlos	
5	2	Profª. Maria	2	Roberto	2
6	2	Profª. Maria	3	Ailton	

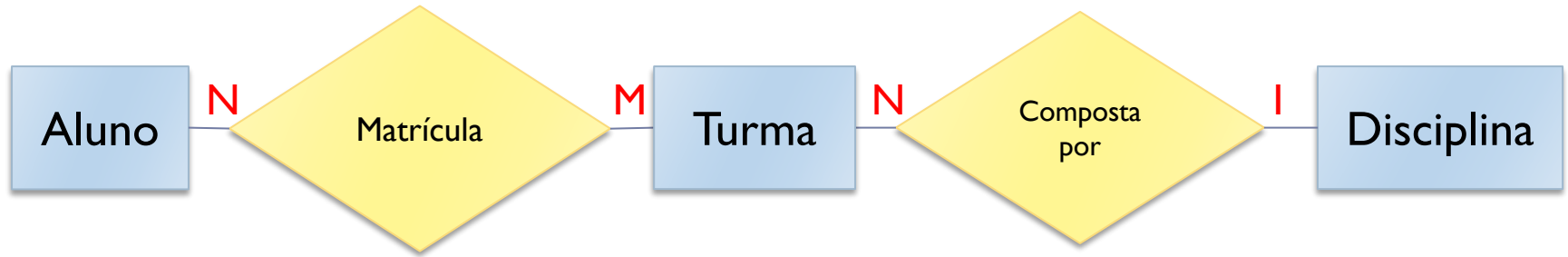
**OBS: O CROSS JOIN funciona igual ao produto cartesiano entre duas tabelas**

# OUTER JOINS

---

- ▶ Variante da operação de JOIN que baseia-se em valores NULL. O resultado de um OUTER JOIN é igual a de um INNER JOIN mas com a inclusão das tuplas que não satisfazem a condição de JOIN.
- ▶ Três variantes:
  - ▶ LEFT OUTER JOIN
    - ▶ As tuplas da tabela à esquerda que não obedecem a condição do JOIN aparecem na resposta
  - ▶ RIGHT OUTER JOIN
    - ▶ As tuplas da tabela à direita que não obedecem a condição do JOIN aparecem na resposta
  - ▶ FULL OUTER JOIN
    - ▶ As tuplas das duas tabelas que não obedecem a condição do JOIN aparecem na resposta

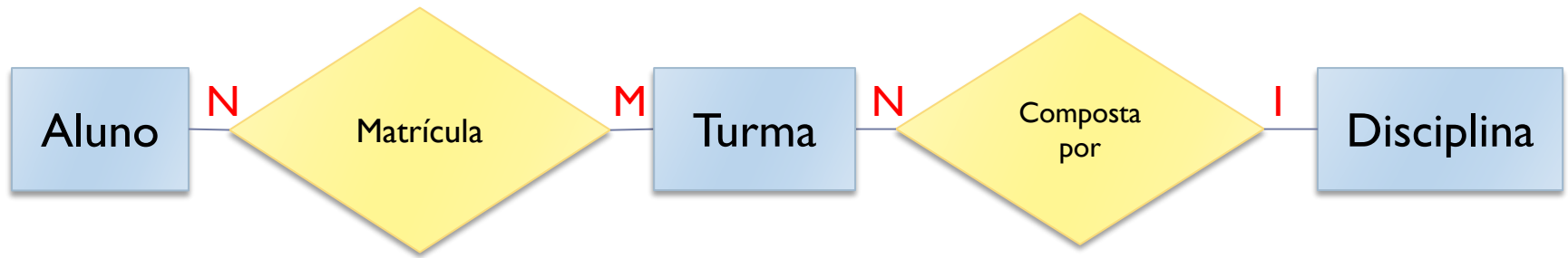




*/\* Liste as matriculas efetuadas em cada turma de cada disciplina \*/*

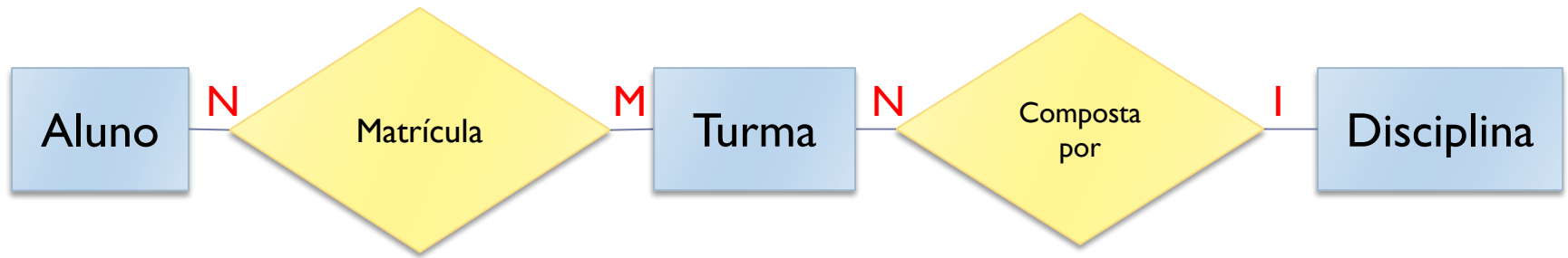
```
SELECT Aluno.NMat, Aluno.NOME, Matricula.CODIGOTURMA,  
Turma.SIGLA, Turma.NUMERO, Discip.NOME  
FROM Discip  
    INNER JOIN Turma  
    ON Discip.SIGLA = Turma.SIGLA  
    INNER JOIN Matricula  
    ON Turma.Codigo = Matricula.CODIGOTURMA  
    INNER JOIN Aluno  
    ON Aluno.NMat = Matricula.NMat;
```





*/\* Liste as matriculas efetuadas em cada turma de cada disciplina \*/*

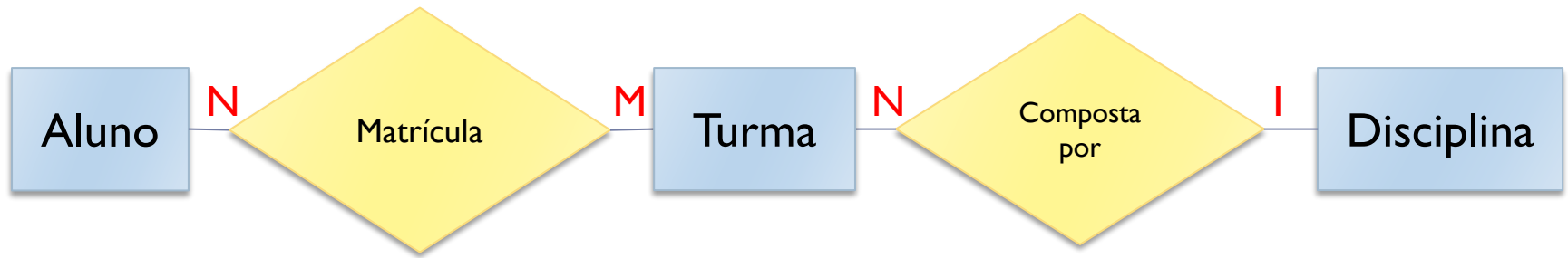
```
SELECT Aluno.NMat,Aluno.NOME,  
Matricula.CODIGOTURMA,  
Turma.SIGLA,Turma.NUMERO, Discip.NOME  
FROM Discip INNER JOIN  
    (Turma INNER JOIN  
        (Matricula INNER JOIN Aluno  
            ON Matricula.Nmat = Aluno.NMat )  
        ON Turma.Codigo = Matricula.CODIGOTURMA)  
    ON Discip.SIGLA = Turma.SIGLA;
```



*/\* Liste as matriculas efetuadas em cada turma de cada disciplina – alterando a ordem das junções\*/*

```
SELECT Aluno.NMat,Aluno.NOME,  
Matricula.CODIGOTURMA,  
Turma.SIGLA,Turma.NUMERO, Discip.NOME  
FROM Aluno INNER JOIN  
    (Matricula INNER JOIN  
        (Turma INNER JOIN Discip  
            ON Turma.SIGLA = Discip.SIGLA)  
        ON Matricula.CODIGOTURMA = Turma.Codigo)  
    ON Aluno.NMat = Matricula.NMat;
```





*/\* Liste as matriculas efetuadas em cada turma de cada disciplina – Comando equivalente usando WHERE\*/*

```
SELECT A.NMat,A.NOME, M.CODIGOTURMA,T.SIGLA,  
        T.NUMERO, D.NOME  
FROM Aluno A, Matricula M,Turma T, DiscipD  
WHERE T.SIGLA = D.SIGLA AND  
        M.CODIGOTURMA = T.Codigo AND  
        A.NMat = M.NMat;
```



# Junções Aninhadas

*/\* Para todo projeto localizado em 'Stafford', listar o número do projeto, o número do departamento que o controla e o último nome do gerente do departamento\*/*

```
SELECT pnumber, dnum, lname  
      FROM ((project JOIN department ON dnum=dnumber)  
            JOIN employee ON mgrssn=ssn)  
     WHERE plocation = 'Stafford';
```





**OBRIGADO A TODOS**

**DÚVIDAS**

