

**A TRANSMISSÃO DA  
AULA COMEÇA EM**

**INSTANTES**



# Sistemas de Informação

# **Bando de Dados 1**

Prof. Dr. Ronaldo Castro de Oliveira

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

FACOM - 2022



# Comando SELECT



# Consultas

---

- ▶ O processo de obtenção ou o comando para a obtenção de dados de um banco de dados é chamado **consulta**
- ▶ Em SQL, o comando SELECT é usado para especificar consultas
- ▶ As consultas simples em SQL correspondem às operações de SELEÇÃO, PROJEÇÃO e JUNÇÃO da álgebra relacional



# Sintaxe SELECT

---

```
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 [, ...] ] ]
```



# 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 [, ...] ) ]
```



# Comando SELECT

## Exemplos

---

*/\* Seleciona todos os campos na tabela ALUNO \*/*

```
SELECT *  
FROM ALUNO;
```

*/\* Selecionar todos os nomes e nros. de matrícula da tabela ALUNO \*/*

```
SELECT NMat, Nome  
FROM ALUNO;
```

*/\* Selecionar todos os nomes e nros. de matrícula da tabela ALUNO,  
renomeando a coluna nome para “aluno” \*/*

```
SELECT NMat, Nome AS Aluno  
FROM ALUNO;
```

**Exemplo banco – Company (Empresa):**

```
SELECT *  
FROM employee;  
SELECT fname as “Nome”, salary as “Salário”  
FROM employee;
```

# Comando SELECT

## Exemplos

---

**DISTINCT** – remove as tuplas duplicadas da resposta

*/\* Selecciona todas as cidades \*/*

**SELECT** cidade  
**FROM** filiais;

Resultado correto, mas com muitas repetições



| Cidade         |
|----------------|
| São Paulo      |
| São Paulo      |
| São Paulo      |
| Rio de Janeiro |
| Rio de Janeiro |
| Angra dos Reis |
| Belo Horizonte |
| Belo Horizonte |
| Uberlândia     |
| Uberlândia     |

*/\* Seleccionar todas as cidades \*/*

**SELECT DISTINCT** cidade  
**FROM** filiais;



| Cidade         |
|----------------|
| São Paulo      |
| Belo Horizonte |
| Uberlândia     |
| Rio de Janeiro |
| Angra dos Reis |

Exemplo banco – Company (Empresa):

**SELECT DISTINCT** location **AS** “Localização”  
**FROM** projeto;



# Comando SELECT

## CLÁUSULA WHERE

---

**WHERE** - Especifica quais registros das tabelas listadas na cláusula FROM são afetados por uma instrução SELECT, UPDATE ou DELETE. Se você não especificar uma cláusula WHERE, a consulta retornará todas as linhas da tabela.

### EXEMPLO:

*/\* Selecciona nome/idade dos alunos cuja idade seja maior que 22 \*/*

**SELECT** Nome, Idade

**FROM** ALUNO

**WHERE** Idade > 22;

Exemplo banco – Company (Empresa):

**SELECT** fname **as** “Nome”, salary **as** “Salário”

**FROM** employee

**WHERE** salary > 40000;

# CLÁUSULA WHERE

---

## Sintaxe

WHERE *condition*

Onde **condition** é qualquer expressão que retorne um tipo booleano.

Qualquer linha que não satisfaça a condição **condition** será eliminada do resultado

## Exemplos :

*/\* Selecciona os números de matrícula dos alunos matriculados a partir de 30/07/2002 \*/*

**SELECT** NMat

**FROM** MATRICULA

**WHERE** Data > '30/07/2002';

*/\* Selecciona os números de matrícula dos alunos matriculados a partir de 30/07/2002 \*/*

**SELECT** M.NMat,A.Nome

**FROM** MATRICULA M,ALUNO A

## Exemplo banco – Company (Empresa):

**SELECT** fname **AS** “Nome”, lname **AS** “Sobrenome”

**FROM** employee

**WHERE** salary > 40000;

# CLÁUSULA WHERE OPERADOR BETWEEN...AND

---

**BETWEEN...AND:** Determina se o valor de uma expressão se situa dentro de um intervalo especificado de valores. Se o valor de *expr* estiver entre *<valor1>* e *<valor2>* (inclusive), o operador BETWEEN...AND retornará True ; caso contrário, retornará False.

## EXEMPLO:

*/\* Listar todos alunos com idade entre 20 e 22 anos\*/*

**SELECT** Nome, Idade

**FROM** Aluno

**WHERE** Idade **BETWEEN** 20 **AND** 22;

Exemplo banco – Company (Empresa):

**SELECT** fname **AS** “Nome”, salary **AS** “Salario”

**FROM** employee

**WHERE** salary **BETWEEN** 30000 **AND** 40000;

# CLÁUSULA WHERE

## OPERADOR IN

---

**IN:** Determina se o valor de uma expressão é igual a algum dos vários valores em uma lista especificada. Se *expr for encontrado na lista de valores*, o operador IN retornará True; caso contrário, retornará False.

### EXEMPLO:

*/\* Listar todos os alunos provenientes de São Paulo, São Carlos ou Rio de Janeiro\*/*

**SELECT \***

**FROM ALUNO**

**WHERE UPPER(Cidade) IN ('SAO PAULO', 'SAO CARLOS', 'RIO DE JANEIRO');** -- **UPPER transforma os caracteres do atributo Cidade em maiúsculo**

**Exemplo banco – Company (Empresa):**

**SELECT** pname **AS** “Nome Projeto”

**FROM** project

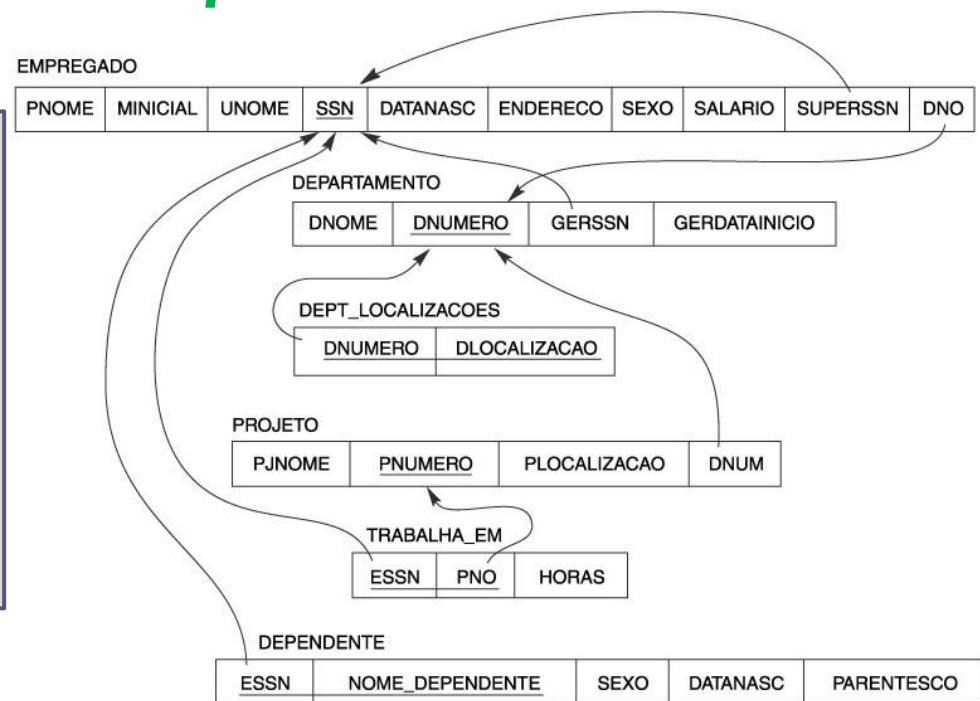
**WHERE** plocation **IN** (‘ Bellaire’, ‘Houston’);

# CLÁUSULA WHERE OPERADOR IN

**IN + subconsulta:** O operador IN pode trabalhar com subconsulta

*/\* Listar os nomes funcionários que trabalham em projetos \*/*

```
SELECT fname
FROM employee
WHERE ssn IN (
    SELECT essn
FROM works_on);
```

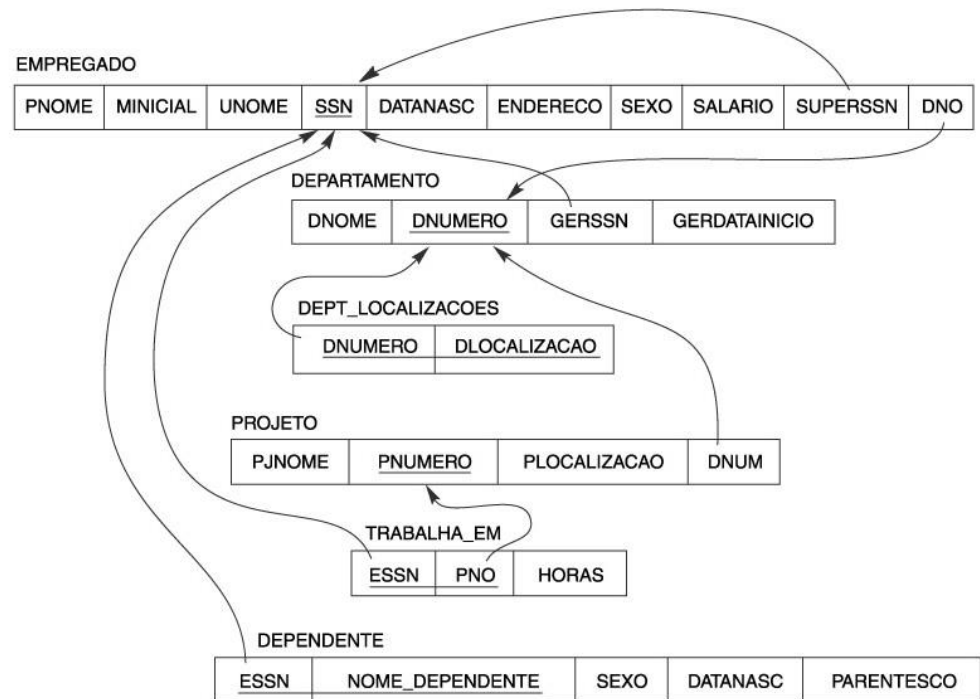


# CLÁUSULA WHERE OPERADOR IN

**NOT IN + subconsulta:** O operador IN pode trabalhar com subconsulta.

*/\* Listar os nomes funcionários que não trabalham em nenhum projeto \*/*

```
SELECT fname
FROM employee
WHERE ssn NOT IN (
  SELECT essn
  FROM works_on);
```



# Comando SELECT

## Consultas aninhadas

---

- ▶ Muitas vezes é preciso buscar uma informação no banco para, então, usá-la na condição de comparação de uma consulta
- ▶ Isso pode ser feito por meio de consultas aninhadas:
  - ▶ Forma-se um bloco SELECT-FROM-WHERE dentro da cláusula WHERE de outra consulta.
  - ▶ Essa outra consulta é chamada de consulta interna.

*/\* Listar os projetos dos funcionários de nome 'José'\*/*

```
SELECT pno
FROM works_on
WHERE essn IN (
    SELECT ssn
    FROM employee
    WHERE fnome = 'John')
```

Consulta aninhada  
(consulta interna)

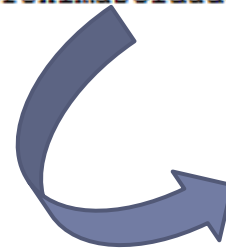
# Comando SELECT

## Consultas aninhadas – Operador IN

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 |
| Internet |                |    |

```
CREATE TABLE ProximasCidades(  
    cidade varchar(50));  
DELETE FROM ProximasCidades;  
INSERT INTO ProximasCidades  
    Select Distinct cidade  
    from filiais  
    where cidade is not null;  
INSERT INTO ProximasCidades VALUES ('Araguari'),('Ituiutaba');  
SELECT * FROM proximascidades
```



*/\* Qual o resultado? \*/*

```
SELECT *  
FROM filiais  
WHERE cidade IN (  
    SELECT cidade  
    FROM ProximasCidades)
```

|   | <b>cidade</b><br>character varying(50) |
|---|--|
| 1 | Rio de Janeiro                         |
| 2 | Belo Horizonte                         |
| 3 | Uberlândia                             |
| 4 | Angra dos Reis                         |
| 5 | São Paulo                              |
| 6 | Araquari                               |
| 7 | Ituiutaba                              |



# Comando SELECT

## Consultas aninhadas – Operador IN

---

*/\* Qual o resultado?*

*São as cidades que já temos filiais mas que ainda estão na lista de próximas cidade*

*E a filial da Internet? \*/*

```
SELECT *  
FROM filiais  
WHERE cidade IN (  
  SELECT cidade  
  FROM ProximasCidades)
```

|    | nome<br>character varying(30) | cidade<br>character varying(50) | uf<br>character(2) |
|----|-------------------------------|---------------------------------|--------------------|
| 1  | Penha                         | São Paulo                       | SP                 |
| 2  | Limão                         | São Paulo                       | SP                 |
| 3  | Brás                          | São Paulo                       | SP                 |
| 4  | Glória                        | Belo Horizonte                  | MG                 |
| 5  | Pampulha                      | Belo Horizonte                  | MG                 |
| 6  | Centro                        | Uberlândia                      | MG                 |
| 7  | Tibery                        | Uberlândia                      | MG                 |
| 8  | Barra                         | Rio de Janeiro                  | RJ                 |
| 9  | Tijuca                        | Rio de Janeiro                  | RJ                 |
| 10 | Angra                         | Angra dos Reis                  | RJ                 |



# Comando SELECT

## Consultas aninhadas – Operador IN

---

*/\* Qual o resultado?*

*São as cidades que já temos filiais\*/*

```
SELECT *  
FROM ProximasCidades  
WHERE cidade IN (  
    SELECT cidade  
    FROM filiais)
```

|          | <b>cidade</b><br>character varying(50) |
|----------|--|
| <b>1</b> | Rio de Janeiro                         |
| <b>2</b> | Belo Horizonte                         |
| <b>3</b> | Uberlândia                             |
| <b>4</b> | Angra dos Reis                         |
| <b>5</b> | São Paulo                              |




# Comando SELECT

## Consultas aninhadas – Operador IN

---


*/\* Qual o resultado? \*/*

```
SELECT *  
FROM ProximasCidades  
WHERE cidade NOT IN (  
    SELECT cidade  
    FROM filiais)
```



|  | cidade<br>character varying(50) |
|--|---------------------------------|
|  |                                 |

```
SELECT *  
FROM ProximasCidades  
WHERE cidade NOT IN (  
    SELECT cidade  
    FROM filiais  
    WHERE cidade IS NOT NULL)
```



|   | cidade<br>character varying(50) |
|---|---------------------------------|
| 1 | Araquari                        |
| 2 | Ituiutaba                       |

# Operadores booleanos

| a     | b     | a AND b | a OR b |
|-------|-------|---------|--------|
| TRUE  | TRUE  | TRUE    | TRUE   |
| TRUE  | FALSE | FALSE   | TRUE   |
| TRUE  | NULL  | NULL    | TRUE   |
| FALSE | FALSE | FALSE   | FALSE  |
| FALSE | NULL  | FALSE   | NULL   |
| NULL  | NULL  | NULL    | NULL   |

| a     | NOT a |
|-------|-------|
| TRUE  | FALSE |
| FALSE | TRUE  |
| NULL  | NULL  |

|          |   |                | Resultado |
|----------|---|----------------|-----------|
| Araguari | = | São Paulo      | False     |
| Araguari | = | São Paulo      | False     |
| Araguari | = | São Paulo      | False     |
| Araguari | = | Rio de Janeiro | False     |
| Araguari | = | Rio de Janeiro | False     |
| Araguari | = | Angra dos Reis | False     |
| Araguari | = | Belo Horizonte | False     |
| Araguari | = | Belo Horizonte | False     |
| Araguari | = | Uberlândia     | False     |
| Araguari | = | Uberlândia     | False     |
| Araguari | = |                | NULL      |

**FALSE OR NULL => NULL**

\*lembre que NULL é **desconhecido**. Ou seja, desconhecemos o resultado da operação: se o *desconhecido* fosse TRUE, temos que

**FALSE OR TRUE => TRUE;**

se o *desconhecido* fosse FALSE, temos que

**FALSE OR FALSE => FALSE**

Assim, desconhecemos o resultado da operação, ou seja, o resultado é NULL

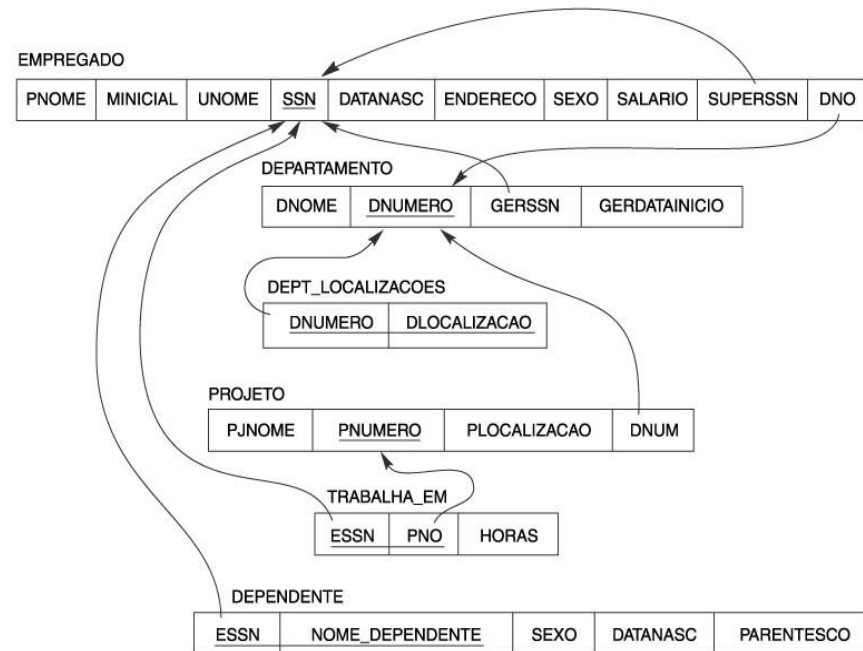
# Comando SELECT

## Consultas aninhadas

- ▶ Se um único atributo com uma única tupla é retornado da consulta externa, então podemos usar o operador =

*/\* Listar o funcionário com o maior salário\*/*

```
SELECT fname, lname, salary
FROM employee
WHERE salary = (
    SELECT max(salary)
    FROM employee)
```



# CLÁUSULA WHERE OPERADOR IS NULL

---

**IS NULL:** Determina se o valor de uma expressão é Nula (vide aula5)

## EXEMPLO:

*/\* Listar todas as disciplinas que NÃO possuem pré-requisito \*/*

**SELECT** Nome

**FROM** Discip

**WHERE** SiglaPreReq = **NULL**;

*-- forma errada de comparar nulos*

# CLÁUSULA WHERE OPERADOR IS NULL

---

**IS NULL:** Determina se o valor de uma expressão é Nula

## EXEMPLO:

*/\* Listar todas as disciplinas que NÃO possuem pré-requisito \*/*

**SELECT** Nome

**FROM** Discip

**WHERE** SiglaPreReq **IS NULL**;

*-- observe que o teste não é SiglaPreReq = NULL*



# CLÁUSULA WHERE

## OPERADOR LIKE

---

**LIKE:** Compara uma expressão de sequência com um padrão em uma expressão SQL. Para padrão, você pode utilizar caracteres curinga (por exemplo, Like 'MAK%', para 'MAKROMBOOKS') ou utilizar caracteres isolados (por exemplo, Like '\_OSE', para 'JOSE' e 'ROSE')

**EXEMPLO:** *Listar todos os alunos cujo nome termina em 'ina', ignorando as 3 primeiras letras \*/*

**SELECT** Nome

**FROM** Aluno

**WHERE UPPER(Nome) LIKE '\_\_\_INA';** -- São 3 '\_'. Usar '%'

**Exemplo banco – Company (Empresa):**

--Listar o nome de todo empregado cujo endereço está em Houston, Texas

```
SELECT e.fname, e.minit, e.lname
      FROM employee e
      WHERE e.address LIKE '%Houston%TX%';
```

--Listar o nome de todos os empregados nascidos na década de 50

```
SELECT e.fname, e.minit, e.lname
      FROM employee e
      WHERE CAST(e.bdate AS TEXT) LIKE '___6%';
```



# Comando SELECT

## CLÁUSULA ORDER BY

---

**ORDER BY:** Classifica os registros resultantes de uma consulta em um campo ou campos especificados, em ordem crescente ou decrescente. Os registros são classificados pelo primeiro campo listado após **ORDER BY**. Os registros que têm valores iguais naquele campo serão então classificados pelo valor no segundo campo listado e assim por diante.

**EXEMPLO:** *Listar alunos ordenado-os decendentemente por idade e por nome*

**SELECT** Nome, Idade

**FROM** Aluno

**ORDER BY** Idade **DESC**, Nome **DESC**; -- trocar **DESC** por **ASC**

**Exemplo banco – Company (Empresa):**

--Listar o nome de todo empregado cujo ordenado por salário e por nome

**SELECT** fname, lname, salary

**FROM** employee

**ORDER BY** salary **DESC**, fname **DESC**;

# Funções agregadas

- Funções agregadas calculam um único valor a partir de um conjunto de valores de entrada

| Function                           | Argument Type   | Return Type   | Description   |
|------------------------------------|---|---|---|
| <code>array_agg(expression)</code> | any   | array of the argument type  | input values concatenated into an array                                   |
| <code>avg(expression)</code>       | smallint, int, bigint, real, double precision, numeric, or interval | numeric for any integer-type argument, double precision for a floating-point argument, otherwise the same as the argument data type                             | the average (arithmetic mean) of all input values                         |
| <code>bit_and(expression)</code>   | smallint, int, bigint, or bit                                       | same as argument data type  | the bitwise AND of all non-null input values, or null if none             |
| <code>bit_or(expression)</code>    | smallint, int, bigint, or bit                                       | same as argument data type  | the bitwise OR of all non-null input values, or null if none              |
| <code>bool_and(expression)</code>  | bool  | bool  | true if all input values are true, otherwise false                        |
| <code>bool_or(expression)</code>   | bool  | bool  | true if at least one input value is true, otherwise false                 |
| <code>count(*)</code>              |   | bigint  | number of input rows  |
| <code>count(expression)</code>     | any   | bigint  | number of input rows for which the value of <i>expression</i> is not null |
| <code>every(expression)</code>     | bool  | bool  | equivalent to <code>bool_and</code>                                       |
| <code>max(expression)</code>       | any array, numeric, string, or date/time type                       | same as argument type   | maximum value of <i>expression</i> across all input values                |
| <code>min(expression)</code>       | any array, numeric, string, or date/time type                       | same as argument type   | minimum value of <i>expression</i> across all input values                |
| <code>sum(expression)</code>       | smallint, int, bigint, real, double precision, numeric, or interval | bigint for smallint or int arguments, numeric for bigint arguments, double precision for floating-point arguments, otherwise the same as the argument data type | sum of <i>expression</i> across all input values                          |
| <code>xmlagg(expression)</code>    | xml   | xml   | concatenation of XML values (see also <a href="#">Section 9.14.1.7</a> )  |

Mais funções: <http://www.postgresql.org/docs/8.4/static/functions-aggregate.html>

# Comando SELECT

## FUNÇÃO MIN( ) / MAX( )

---

**MIN( )** : Retorna o mínimo de um conjunto de valores contido em um campo na consulta.

**MAX( )**: Retorna o máximo de um conjunto de valores contido em um campo na consulta.

### EXEMPLO:

*/\* Verificar qual é a idade do aluno mais velho \*/*

```
SELECT MAX(Idade)
FROM ALUNO;
```

*/\* Verificar o(s) nome(s) e a idade do(s) aluno(s) mais novo(s) \*/*

```
SELECT Nome, Idade
FROM ALUNO
WHERE Idade IN (SELECT MIN(Idade)
FROM ALUNO);
```

### Exemplo banco – Company (Empresa):

--Maior salário dos empregados

```
SELECT MAX(salary)
FROM employee;
```

### Exemplo banco – Company (Empresa):

--Empregado mais velho

```
select fname, bdate
from employee
where bdate in (select min(bdate)
from employee);
```

# Comando SELECT

## FUNÇÃO SUM( )

---

**SUM( ):** Retorna a soma de um conjunto de valores contido em um campo especificado em uma consulta. A função Sum ignora os registros que contenham campos Null.

### EXEMPLO:

*/\* Calcular a quantidade de créditos oferecidos \*/*

```
SELECT SUM(NNCred) AS TotalCreditos  
FROM DISCIP;
```

### Exemplo banco – Company (Empresa):

--Apresentar a somatória de todos os salários pagos

```
SELECT SUM(salary) AS “Total de Salários”  
FROM employee;
```

# Comando SELECT

## FUNÇÃO AVG( )

---

**AVG( ):** Calcula a média aritmética de um conjunto de valores contido em um campo especificado em uma consulta.

### EXEMPLO:

*/\* Calcular a idade média de alunos cadastrados \*/*

```
SELECT AVG(Idade) AS Media  
FROM ALUNO
```

### Exemplo banco – Company (Empresa):

--Apresentar a media de todos os salários pagos

```
SELECT AVG(salary) AS “Total de Salários”  
FROM employee;
```

# Comando SELECT

## FUNÇÃO COUNT( )

**COUNT( ):** Calcula o número de registros retornado por uma consulta. A função **COUNT** não conta registros que tenham campos *NULL*, exceto quando *expr* for o caractere curinga asterisco (\*).

### EXEMPLO:

*/\* Contar quantos alunos fazem iniciação científica (IC) –  
Observe que registros onde cpf\_orientador é NULL não são  
considerados\*/*

**SELECT COUNT**(cpf\_orientador)

**FROM** aluno;

*/\* Contar quantos orientadores de IC existem. Note que um orientador pode orientar mais de um  
aluno \*/*

**SELECT COUNT**( **DISTINCT** cpf\_orientador)

**FROM** aluno;

| NMAT | Nome   | cpf_orientador |
|------|--------|----------------|
| 1    | Carlos | 222.222.222-22 |
| 2    | José   |                |
| 3    | Maria  | 555.555.222-12 |
| 4    | André  | 222.222.222-22 |

### Exemplo banco – Company (Empresa):

-- Apresenta o numero de numero de registros de projetos

```
select count(*)
```

```
    from project p
```

-- Apresenta o numero de localizações dos projetos distintos

```
select count(distinct p.plocation)
```

```
    from project p
```

# COUNT()

---

- ▶ É possível usar a palavra chave **DISTINCT** com o count. Desta forma, os valores duplicados nas colunas não são contados.

*/\* Qual o número de estados  
atendidos pela empresa \*/*

**SELECT COUNT (DISTINCT uf )  
FROM filiais**

**Resposta: 3**

| 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 |

# Observações Funções Agregadas

---

- ▶ Uma função agregada não pode aparecer com um atributo da tabela

*/\* Verificar qual é a idade e o nome do aluno mais velho \*/*

**SELECT** nome, **MAX**(Idade)

**FROM** ALUNO;

**→ ERRO ←**

- ▶ Funções agregadas podem aparecer com atributos quando é utilizado o operador **GROUP BY**, explicado mais adiante





# 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

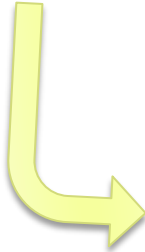
**FROM** filiais

**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

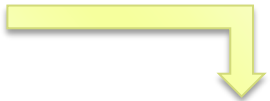
**/\* Qual o número de filiais por estado \*/**

```
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<br>character(2) | count<br>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<br>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<br>character(2) | count<br>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<br>character(2) | count<br>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<br>character(2) | count<br>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<br>character(2) | count<br>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<br>character(2) | count<br>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(\*)**

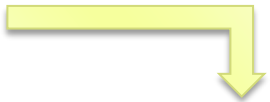
**FROM** filiais

**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<br>character(2) | count<br>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

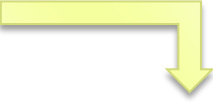
### EXEMPLO:

*/\* 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<br>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<br>character(2) | nfiliais<br>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

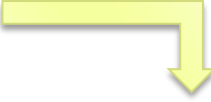
**WHERE** UF <> 'SP'

**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<br>character(2) | nfiliais<br>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<br>character(2) | nfiliais<br>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 |

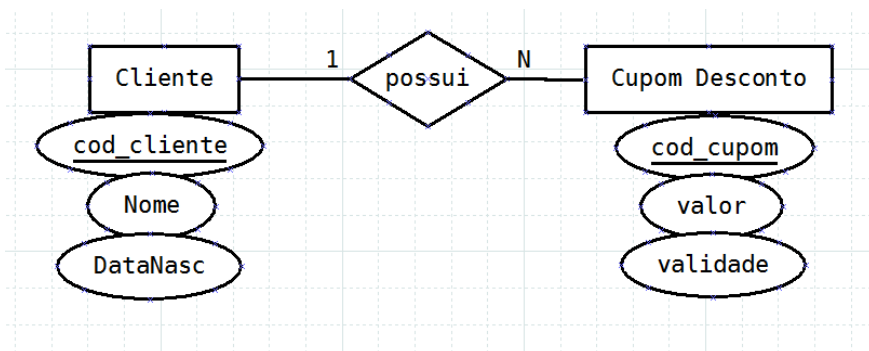
ERRO: coluna "nfiliais" não existe  
LINE 4: HAVING nfiliais >3

|        |            |    |
|--------|------------|----|
| 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*

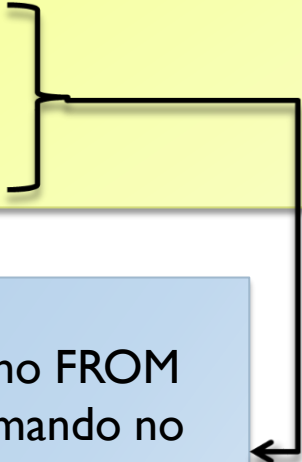
```
SELECT *  
FROM cliente  
WHERE EXISTS  
    ( SELECT 1  
      FROM compra  
      WHERE compra.cod_cliente =  
            cliente.cod_cliente)
```



# 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*

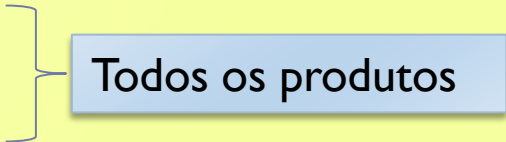
```
SELECT *  
FROM cliente  
WHERE NOT EXISTS  
  
  (SELECT cod_produto  
   FROM produto  
  
   EXCEPT  
  
   SELECT cod_produto  
   FROM compra  
   WHERE compra.cod_cliente = cliente.cod_cliente  
  )
```

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



# Referências

---

- ▶ Slides adaptados da aula da Profa. Josiane M. Bueno (*in memoriam*)
- ▶ Slides Prof. Humberto Razente
- ▶ Slides Profa. Sandra de Amo



**OBRIGADO A TODOS**

**DÚVIDAS**

