



LABORATÓRIO DE BANCO DE DADOS

# SQL - Consultas



PostgreSQL

- SQL Básico
- Consulta com operadores
- Consultas com agrupamento e agregações
- Subconsultas
- Consultas com condicionais
- Operações a partir de conjuntos
- Outras funções



# Consultas em tabelas

## SELECT

- Executa consultas em tabelas
- Realiza a operação de projeção da álgebra relacional

```
SELECT * | <lista de atributos e funções>  
FROM <lista de tabelas>;
```

- Cláusula **SELECT**:
  - Lista os atributos e/ou funções a serem exibidos nos resultados das consultas
- Cláusula **FROM**:
  - Especifica as relações a serem examinadas na avaliação da consulta





# Consultas em tabelas com cláusulas condicionais

## SELECT ... WHERE ...

- A cláusula WHERE possibilita consultas em tabelas considerando cláusulas condicionais (filtros).

```
SELECT * | <lista de atributos e funções>  
FROM <lista de tabelas>  
[WHERE <condições>];
```

- Realiza a operação de **seleção da álgebra relacional**
- Os operadores que podem ser utilizados são: =, <, <=, >, >= e <>
- Exemplos:

-- Retorna todas as colunas da tabelas que possuem funcionários com nome James.

```
SELECT * FROM funcionario WHERE Pnome=James';
```



# Consultas em uma única tabela SELECT

- Exemplos:

-- Retorna todas as colunas de todos da relação funcionário

**SELECT \* FROM** *funcionario*;

**SELECT \* FROM** empresa.funcionario;

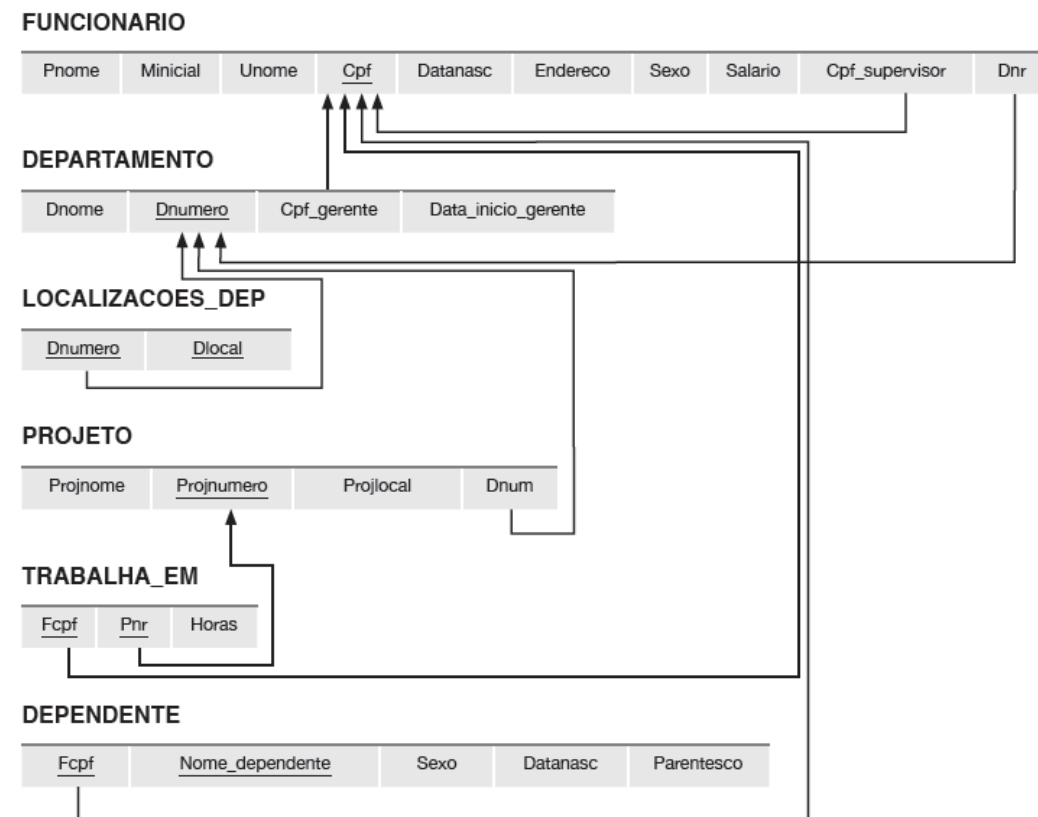
**SELECT \* FROM** public.funcionario;

-- Retorna as colunas Pnome e Unome da relação funcionário

**SELECT** Pnome, Unome **FROM** funcionario;

-- Retorna o produto de funcionário e departamento

**SELECT \* FROM** *funcionario*, *departamento*;



# Consultas em múltiplas tabelas SELECT

- Consulta com múltiplas tabelas com cláusula condicional

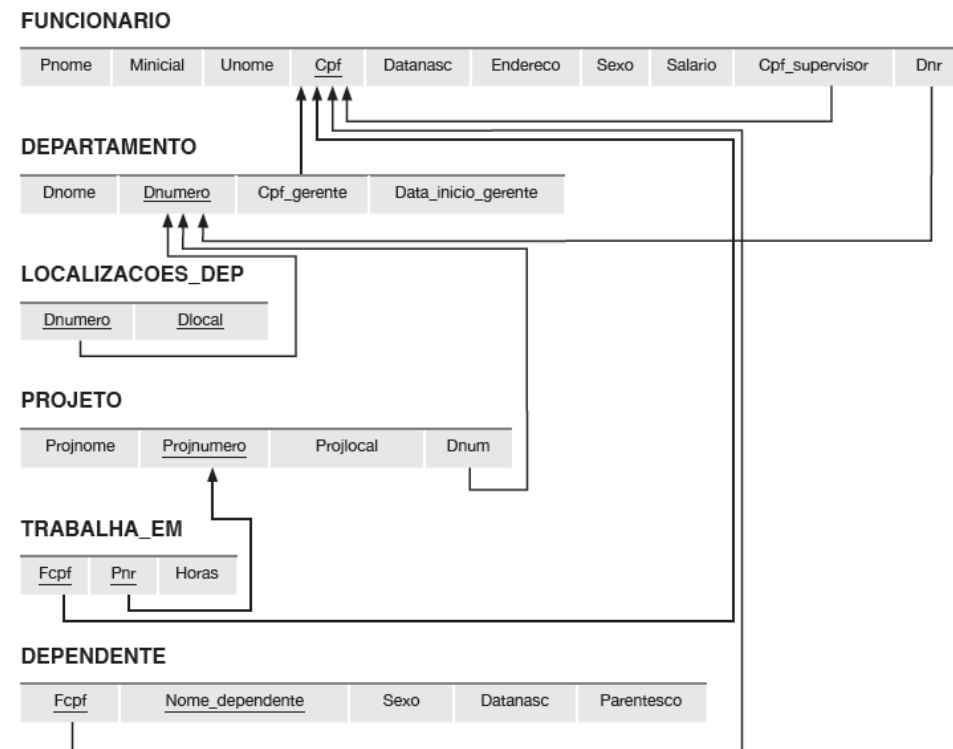
-- Retorna o nome do funcionário e o nome do departamento que aquele funcionário trabalha

```
SELECT funcionario.pnome, departamento.dnome  
FROM funcionario, departamento  
WHERE funcionario.dnr = departamento.dnumero;
```

- Consulta com múltiplas tabelas com cláusula produto cartesiano (todas as combinações de tuplas possíveis)

-- Retorna o nome do funcionário e o nome do departamento resultante do produto cartesiano de funcionário e departamento

```
SELECT funcionario.pnome, departamento.dnome  
FROM funcionario, departamento;
```



# Apelido, renomeação, variáveis de tupla AS

- Renomear
  - **Atributos**
    - Deve aparecer na cláusula **SELECT**
    - Útil para a visualização do resultado da query
  - **Relações**
    - Deve aparecer na cláusula **FROM**
    - Útil quando a mesma relação é utilizada mais do que uma vez na mesma consulta
  - **Sintaxe**  
nome\_antigo **AS** nome\_novo
- **AS** é um comando opcional



# Apelido, renomeação, variáveis de tupla AS

- Atributo

-- Retorna os dados da coluna Pnome, porém, a coluna nome está com o "apelido" nome\_funcionario

```
SELECT Pnome AS  
nome_funcionario FROM  
funcionario;
```

- Relação

-- Retorna o nome do funcionário, nome de seu departamento e seus dependentes

```
SELECT Pnome AS nome_funcionario,  
       Dnome AS nome_departamento ,  
       Nome_dependente AS dependente  
FROM funcionario AS func ,  
     departamento AS depto ,  
     dependete AS depe  
WHERE func.Cpf = depto.Dnumero  
AND depe.Fcpf = func.cpf;
```



# Apelido, renomeação, variáveis de tupla AS

- Mecanismos de nomeação podem ser utilizados para especificar **variáveis de tupla na cláusula WHERE**.

-- Recupera o nome e o endereço de todos os funcionários que trabalham para o departamento 'Administracao'

**SELECT** F.Pnome, F.Unome, F.Endereco

**FROM** funcionario F, departamento D

**WHERE** D.Dnome='Administracao' **AND** D.Dnumero=F.Dnr;





# SELECT - Atributos ambíguos

Em SQL o mesmo nome pode ser usado para dois ou mais atributos desde que estejam em **relações diferentes.**

- A ambiguidade dos nomes de atributos também surge no caso de consultas que se referem à mesma relação duas vezes
- Exemplo:
  - Para cada funcionário, recupere o primeiro e o último nome do funcionário e o primeiro e o último nome de seus supervisor imediato.

```
SELECT F.Pnome, F.Unome, S.Pnome, S.Unome  
FROM funcionario AS F, funcionario AS S  
WHERE F.cpf_supervisor = S.cpf;
```



# Comparações envolvendo valores NULL

## SELECT ... WHERE ...

- SQL permite consultas que verificam se o valor de um atributo é NULL

**SELECT \* FROM <lista de tabelas> WHERE <atributo> IS [NOT] NULL;**

**SELECT \* FROM funcionario WHERE endereco IS NULL;**

**SELECT \* FROM funcionario WHERE endereco IS NOT NULL;**

- Valor NULL pode ser usado para representar um valor:
  - **Desconhecido** (existe, mas não é conhecido)
    - Exemplo: a data de nascimento de uma pessoa não é conhecida
  - **Não disponível** (existe, mas é propositalmente omitido)
    - Exemplo: o usuário não informa o telefone comercial
  - **Não aplicável** (o atributo é indefinido para essa tupla)
    - Exemplo: o atributo "cônjuge" de uma pessoa que não é casada





# Comparações envolvendo valores NULL

## SELECT ... WHERE ...

Exemplo:

**SELECT \* FROM funcionario WHERE** ~~sexo = NULL;~~

**SELECT \* FROM funcionario WHERE** sexo **IS** NULL;

**SELECT \* FROM funcionario WHERE** sexo **IS NOT** NULL;





# Ordem de classificação na exibição de tuplas

## SELECT ... ORDER BY

- Ordena as tuplas que aparecem no resultado de uma consulta
  - **ASC (padrão):** ordem ascendente
  - **DESC:** ordem descendente
- Ordenação pode ser especificada em vários atributos
  - A ordenação referente ao primeiro atributo é prioritária. Se houver valores repetidos, então
  - é utilizada a ordenação referente ao segundo atributo, e assim por diante.

**SELECT \* | <lista de atributos e funções>**

**FROM <lista de tabelas>**

**[WHERE <condições>]**

**[ORDER BY <lista de atributos> | <posição>];**



# Ordem de classificação na exibição de tuplas

## SELECT ... ORDER BY

- Exemplo:

-- Seleciona os atributos que representam o nome, sexo, data de nascimento e parentesco da tabela dependente ordenando o sexo de forma descendente e o nome de forma ascendente

```
SELECT nome_dependente, sexo, datanasc, parentesco FROM dependente  
ORDER BY sexo DESC, nome_dependente ASC;
```

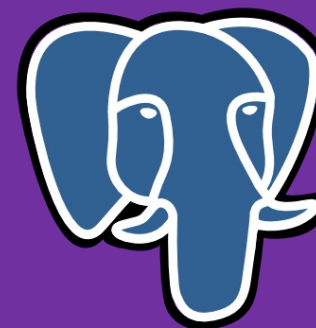
```
SELECT nome_dependente, sexo, datanasc, parentesco FROM dependente  
ORDER BY 2 DESC, 1 ASC;
```





LABORATÓRIO DE BANCO DE DADOS

# SQL - Consultas



PostgreSQL

- SQL Básico
- Consulta com operadores
- Consultas com agrupamento e agregações
- Subconsultas
- Consultas com condicionais
- Operações a partir de conjuntos
- Outras funções

# Consultas com operadores

## SELECT ... WHERE

- Operadores

- **AND**: conjunção de condições
- **OR**: disjunção de condições
- **NOT**: negação de condições
- **=, <>, >, <, >=, <=**
- **BETWEEN ... AND**: entre dois valores
- **LIKE, NOT LIKE**: comparação de cadeias de caracteres
  - % (porcentagem): substitui qualquer string
  - \_ (underscore): substitui qualquer caractere





# Consultas com operadores AND | OR | NOT

## Operador

- Conjunção de condições: **AND**

-- Retorna todos os funcionários cujo nome é Roberto e trabalha no departamento 5

```
SELECT * FROM funcionario WHERE pnome='Ahmad' AND dnr=4;
```

```
SELECT * FROM funcionario WHERE (pnome,dnr)=('Ahmad',4);
```





# Consultas com operadores AND | OR | NOT

## Operador

- Disjunção de condições: **OR**

-- Retorna todos os funcionários cujo nome é Roberto ou trabalha no departamento 5

**SELECT \* FROM funcionario WHERE Pnome=Ahmad' OR Dnr=5;**

- Negação de condições: **{AND | OR} NOT**

-- Retorna todos os funcionarios cujo nome é Roberto e não trabalha no departamento 5

**SELECT \* FROM funcionario WHERE Pnome=Ahmad' AND NOT Dnr=5;**





# Consultas com operadores WHERE ... BETWEEN ... AND

- Comparação entre dois valores
  - **BETWEEN** <valor1> **AND** <valo2>
  - **NOT BETWEEN** <valor1> **AND** <valo2>

```
SELECT * FROM funcionario  
WHERE salario BETWEEN 10000 AND 25000  
AND dnr=4;
```

- Semelhante a comparação: **salario** **>=** 10000 **AND** **salario** **<=**25000





# Comparação de cadeias de caracteres LIKE

- Condições de comparação apenas sobre partes de uma cadeia de caracteres, usando o operador de comparação LIKE

-- Recuperar todos os funcionários cujo endereço esteja em São Paulo.

```
SELECT Pnome, Unome FROM funcionario  
WHERE Endereco LIKE '%SaoPaulo,SP%';
```

-- Recuperar todos os funcionários nasceram durante a década de 1950 (xx/xx/xx5x).

```
SELECT Pnome, Unome FROM funcionario  
WHERE Datanasc LIKE '______5_';
```



# Comparação de cadeias de caracteres LIKE

- Caso os símbolos de % (porcentagem) ou \_ (underscore) estejam presentes na cadeia de caractere, é necessário a utilização de um caractere de escape.

Alguns SGBDs oferecem opções de LIKE que não diferenciam letras maiúsculas de minúsculas

Para o Postgres utiliza-se ILIKE (insensitive).





# Operações com Strings



PostgreSQL

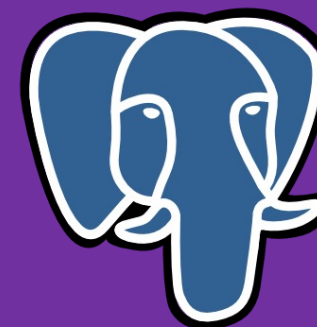
- É possível concatenar string: `||`  
`SELECT pnome || ' ' || unome AS NomeCompleto FROM funcionario;`
- Saber o tamanho da string: **CHAR\_LENGTH**  
`SELECT CHAR_LENGTH(pnome) AS TamanhoNome FROM funcionario;`
- Transformar em maiúsculo: **UPPER**  
`SELECT UPPER(pnome) AS Nome FROM funcionario;`
- Transformar em minúsculo: **LOWER**  
`SELECT LOWER(pnome) AS Nome FROM funcionario;`
- Transforma a cadeia de caracter para a primeira letra da palavra em maiúsculo e as demais em minúsculo (camelcase)  
`SELECT INITCAP(pnome || ' ' || unome) AS NomeCompleto FROM funcionario;`
- Remover o espaço em branco no nas extremidades da string: **TRIM**
  - `SELECT TRIM(endereco) AS NomeCompleto FROM funcionario;`





LABORATÓRIO DE BANCO DE DADOS

# SQL - Consultas



PostgreSQL

- SQL Básico
- Consulta com operadores
- Consultas com agrupamento e agregações
- Subconsultas
- Consultas com condicionais
- Operações a partir de conjuntos
- Outras funções

# Funções de agregação e agrupamento em SQL

- Funções utilizadas para **particionar** a relação em subconjunto de tuplas
  - Baseado no(s) **atributo(s) de agrupamento**
  - Aplicar a função a cada grupo desse tipo independentemente
- Atributos de **agrupamento** no **GROUP BY** também devem aparecer no **SELECT**
- Recupera os valores para as funções somente para aqueles grupos que satisfazem à condição imposta na cláusula **HAVING**
- **Valores NULL são descartados** quando as funções de agregação são aplicadas a determinada coluna (atributo)



# Funções de agregação e agrupamento em SQL

São usadas para resumir informações de várias tuplas em uma síntese de tupla única

- **Agrupamento**

- Cria subgrupos de tuplas antes do resumo

- Funções de agregação embutidas

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

- Essas funções podem ser usadas na cláusula **SELECT** ou em uma cláusula **HAVING**







# Funções de agregação e agrupamento em SQL

## GROUP BY ... HAVING ...

- Cláusula **GROUP BY**
  - Especifica os atributos de agrupamento
- Cláusula **HAVING**
  - Oferece uma condição sobre a informação de resumo

```
SELECT * | <lista de atributos e funções>  
FROM <lista de tabelas>  
[WHERE <condições>]  
[GROUP BY <atributos de agrupamento>]  
[HAVING <condições para agrupamento>];
```



# Principais diferenças entre WHERE E HAVING

- A diferença fundamental entre **WHERE** e **HAVING** é:
  - **WHERE** seleciona as linhas de entrada antes dos grupos e agregações serem computados (portanto, controla quais linhas serão agregadas)
  - **HAVING** seleciona linhas de grupo após os grupos e agregações serem computados
- A cláusula **WHERE** não pode conter funções de agregação
- A cláusula **HAVING** sempre contém funções de agregação (A rigor, é permitido escrever uma cláusula HAVING que não possui agregação, mas raramente é útil)



# Funções de agregação e agrupamento em SQL

- **Funções**

- Contagem: **COUNT( )**
  - Mínimo: **MIN( )**
  - Máximo: **MAX( )**
  - Soma: **SUM( )**
  - Média: **AVG( )**
- } São necessariamente números

- **Características**

- Recebe uma coleção de valores como entrada
- Retorna um único valor
- **Todas as funções de agregação, exceto COUNT(\*), ignoram valores NULL em sua coleção de saída**





# Funções de agregação e agrupamento em SQL

## COUNT

- A função **COUNT** retorna o total de linhas selecionadas
  - COUNT pode receber por parâmetro o nome da coluna ou um asterisco (\*)
    - Quando informado \* todas as linhas serão contabilizadas
  - Quando informado o nome de uma coluna, valores do tipo NULL são ignorados

```
SELECT COUNT(*) FROM funcionario;
```

```
SELECT COUNT(*) AS quantidade, sexo FROM funcionario GROUP BY sexo;
```

```
SELECT COUNT(sexo) AS quantidade, sexo FROM funcionario GROUP BY sexo;
```





# Funções de agregação e agrupamento em SQL

## MIN e MAX

- A função **MIN** retorna o valor **mínimo** de um conjunto de valores
- A função **MAX** retorna o valor **máximo** de um conjunto de valores

```
SELECT sexo, MIN(salario) AS min_salario, MAX(salario) AS  
max_salario FROM funcionario  
GROUP BY sexo;
```





# Funções de agregação e agrupamento em SQL

## SUM(soma)

Somente  
números

- A função **SUM** retorna a soma dos valores de uma coluna
  - Sintaxe: **SELECT SUM(<atributo>) FROM <nome\_da\_tabela>;**

```
SELECT sexo, SUM(salario) AS soma_salario  
FROM funcionario  
GROUP BY sexo;
```





# Funções de agregação e agrupamento em SQL

## AVG(média)

Somente  
números

- A função **AVG** retorna a média de valores de uma coluna
  - Sintaxe: **SELECT AVG(<atributo>) FROM <nome\_da\_tabela>;**

```
SELECT sexo, AVG(salario) AS media_salario  
FROM funcionario  
GROUP BY sexo;
```





# Funções de agregação e agrupamento em SQL

## HAVING

- Expressões na cláusula **HAVING** podem fazer referência tanto a **expressões agrupadas** quanto a **não agrupadas** (as quais necessariamente envolvem uma função de agregação)

- Exemplo:

```
SELECT x, SUM(y) AS soma_y  
FROM tabelaA  
GROUP BY x  
HAVING SUM(y) > 3;
```

Expressão não agrupada  
com função de agregação

```
SELECT x, SUM(y) AS soma_y  
FROM tabelaA  
GROUP BY x  
HAVING x > 10;
```

Expressão agrupada







# Funções de agregação e agrupamento em SQL

## HAVING

- Exemplo:

```
SELECT cpf, SUM(salario) AS somasalario  
FROM funcionario  
GROUP BY cpf  
HAVING SUM(salario) > 30000;
```

Expressão não agrupada  
com função de agregação

```
SELECT cpf, SUM(salario) AS somasalario  
FROM funcionario  
GROUP BY cpf  
HAVING cpf > 100000;
```

Expressão agrupada





# Funções de agregação e agrupamento SQL

## HAVING

- É possível utilizar a cláusula **HAVING** **sem GROUP BY**

```
postgres=# SELECT * FROM FUNCIONARIO;
```

pnome	minicial	unome	cpf	datanasc	endereco	sexo	salario	cpf_supervisor	dnr
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000.00	12345678	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000.00	888665555	4
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Housto, TX	M	40000.00	888665555	5
John	B	Smith	12345678	1965-01-09	731 Fondren, Houston, TX	M	30000.00	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000.00	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000.00	987654321	4
Robert	F	Scott	943775543	2042-06-21	2365 Newcastle Rd, Bellaire, TX	M	58000.00	888665555	1
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000.00	943775543	4
Vanessa	B	Borg	911887776	1965-01-17	3321 Castle, Spring, TX		10000.00	943775543	4
Asdrubal	B	Asd	91188711	1965-01-18	3321 Castle, Spring, TX	M	5000.00		4

(10 registros)

```
postgres=# SELECT AVG(salario) FROM FUNCIONARIO;
```

avg
31600.000000000000

(1 registro)

```
postgres=# SELECT AVG(salario) FROM FUNCIONARIO HAVING COUNT(*)>=10;
```

avg
31600.000000000000

(1 registro)

```
postgres=# SELECT AVG(salario) FROM FUNCIONARIO HAVING COUNT(*)>10;
```

avg
-----

(0 registro)





LABORATÓRIO DE BANCO DE DADOS

# SQL - Consultas



PostgreSQL

- SQL Básico
- Consulta com operadores
- Consultas com agrupamento e agregações
- Consultas com condicionais
- Subconsultas
- Operações a partir de conjuntos
- Outras funções

# Expressões de subconsultas

- Em algumas consultas é necessário recuperar algumas informações do BD e compará-las com algum outro tipo de informação (**tudo na mesma consulta**)
  - Tais consultas podem ser formuladas utilizando **sub consultas**
  - **É possível ter vários níveis de consulta**

```
SELECT ...  
FROM ...  
WHERE ... .... (SELECT ...  
                  FROM ...  
                  WHERE ...);
```

Operações para conectar consultas





# Expressões de subconsultas

## IN | NOT IN

**IN:** compara um valor **v** com um conjunto (ou multiconjunto) de valores **V** e avalia como **TRUE** se **v** for um dos elementos em **V**

```
SELECT <lista de atributos>  
WHERE <atributo>  
{IN | NOT IN}  
(SELECT <lista de atributos> FROM ...);
```

```
SELECT <lista de atributos>  
WHERE <atributo>  
{IN | NOT IN}  
(<lista de valores>);
```

Deve retornar exatamente uma coluna





# Expressões de subconsultas

## IN | NOT IN

-- Retorna os funcionários que trabalham nos departamentos 1 ou 5

```
SELECT * FROM funcionario WHERE dnr IN (1,5);
```

-- Retorna os funcionários que trabalham no departamento cujo gerente possui o cpf 987654321

```
SELECT * FROM funcionario WHERE dnr IN (SELECT dnumero from  
departamento where cpf_gerente='987654321');
```



# Funções de agregação e agrupamento SQL

- *Exemplo:*

- Retorne os funcionários que possuem dois ou mais dependentes

```
SELECT cpf, pnome, unome  
FROM funcionario f  
WHERE (SELECT COUNT(*) FROM dependente d WHERE d.fcpf = f.cpf) >= 2;
```

FUNCIONARIO

cpf	pnome	...
1163	Claudia	...
1164	Jorge	...
1165	Moacir	...
1167	Caio	...

DEPENDENTE

fcpf	pnome	...
1163	Amanda	...
1164	Fabio	...
1165	Alan	...
1163	Henrique	...
1164	Pedro	...
1163	Claudia	...



COUNT(\*) para cpf 1163 → 3

COUNT(\*) para cpf 1164 → 2

...



# Funções de agregação e agrupamento SQL

- Exemplo:

- Retorne os funcionários que possuem dois ou mais dependentes

```
SELECT cpf, pnome, unome
```

```
FROM funcionario f
```

```
WHERE f.cpf IN ( SELECT fcpf FROM dependente d GROUP BY d.fcpf HAVING COUNT(d.fcpf) >= 2 );
```

DEPENDENTE

fcpf	pnome	...
1163	Amanda	...
1164	Fabio	...
1165	Alan	...
1163	Henrique	...
1164	Pedro	...
1163	Claudia	...



DEPENDENTE

fcpf
1163
1164

FUNCIONARIO

cpf	pnome	unome
1163	Claudia	...
1164	Jorge	...
1165	Moacir	...
1167	Caio	...





# Expressões de subconsultas

## EXISTS | NOT EXISTS

- Função **EXISTS**
  - Verificar se o resultado de uma consulta aninhada **correlacionada** é vazio ou não
- **EXISTS** e **NOT EXISTS**
  - Costumam ser usados em conjunto com uma consulta aninhada correlacionada
    - Recupera o nome dos funcionários que não possuem dependente
- **EXISTS(C)** retorna **TRUE** se **existe pelo menos uma tupla no resultado da consulta aninhada C**, e retorna **FALSE** em **caso contrário**.



# Expressões de subconsultas

- Atributo A **IN** (R: Resultado de subconsulta)
  - Verdadeiro se o valor do atributo A está em R
- Atributo A **NOT IN** (R: Resultado de subconsulta)
  - Verdadeiro se o valor do atributo A não está em R
- **EXISTS** (R: Resultado de subconsulta)
  - Verdadeiro se R é não vazio
- **NOT EXISTS** (R: Resultado de subconsulta)
  - Verdadeiro se R é vazio



# Expressões de subconsultas

## ANY | SOME

- Outros operadores de comparação podem ser usados para comparar um único valor  $v$ 
  - O operador = ANY (ou = SOME)
    - Retorna TRUE se o valor  $v$  for igual a *algum valor* no conjunto  $V$  e portanto é **equivalente a IN**

expressão operador ANY (subconsulta)  
expressão operador SOME (subconsulta)

$>$ ,  $<$ ,  $>=$ ,  $<=$ ,  $<>$

Deve retornar  
exatamente uma coluna





# Expressões de subconsultas

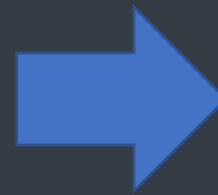
## ANY | SOME

Retorna os funcionários que possuem salário superior aos salários dos funcionários do departamento 5

```
SELECT pnome, salario  
FROM funcionario  
WHERE salario > SOME (SELECT salario FROM funcionario WHERE dnr=5);
```

PNOME	SALARIO
James	55000
Jennifer	43000
Franklin	40000
John	30000
Ramesh	38000
Joyce	25000
Ahmad	25000
Robert	58000
Alicia	25000
Vanessa	10000
Asdrubal	5000

SALARIOS DEP 5
40000
30000
38000
25000



PNOME	SALARIO
James	55000
Jennifer	43000
Franklin	40000
John	30000
Ramesh	38000
Robert	58000



# Expressões de subconsultas

## Comparações de valor com um conjunto - ALL

- ALL
- Compara **v operador ALL V** retorna **TRUE** se o valor de **v** for verdadeiro para todos os valores **no conjunto (ou multiconjunto) V**.
  - NOT IN equivale ao <> ALL

expressão **operador** ALL (subconsulta)

**>, <, >=, <=, <>**

Deve retornar  
exatamente uma coluna





# Expressões de subconsultas

## Comparações de valor com um conjunto - ALL

- Exemplo:

-- Retorna os nomes dos funcionários cujo salário é maior do que o salário de todos os funcionários no departamento 5

```
SELECT Unome, Pnome  
FROM FUNCIONARIO  
WHERE Salario > ALL (SELECT Salario FROM FUNCIONARIO WHERE dnr=5);
```





# Expressões de subconsultas

## ALL

Retorna os funcionários que possuem salário superior aos salários dos funcionários do departamento 5

```
SELECT pnome, salario  
FROM funcionario  
WHERE salario > ALL (SELECT salario FROM funcionario WHERE dnr=5);
```

PNOME	SALARIO
James	55000
Jennifer	43000
Franklin	40000
John	30000
Ramesh	38000
Joyce	25000
Ahmad	25000
Robert	58000
Alicia	25000
Vanessa	10000
Asdrubal	5000

SALARIOS DEP 5
40000
30000
38000
25000



PNOME	SALARIO
James	55000
Jennifer	43000
Robert	58000



# Expressões de subconsultas

- $A > \text{ANY}$  (R: Resultado de subconsulta)
  - Verdadeiro se o valor do atributo A é maior do que algum valor de R
- $A > \text{ALL}$  (R: Resultado de subconsulta)
  - Verdadeiro se o valor do atributo A é maior do que todo valor em R
- $A <> \text{ALL}$  (R: Resultado de subconsulta)
  - Verdadeiro se o valor de A não está em R
- $A = \text{ANY}$  (R: Resultado de subconsulta)
  - Verdadeiro se o valor de A está em R
- Outros operadores que podem ser combinados com ANY (ou SOME):  $>$ ,  $>=$ ,  $<$ ,  $<=$  e  $<>$







# Subconsultas na cláusula FROM

- A SQL permite que uma expressão de **subconsulta** seja utilizada na cláusula **FROM**

-- Retorne todos os funcionários que possuem salário maior ou igual ao salário médio de seu departamento

**SELECT** f1.pnome, salario, media\_salario, f1.dnr

**FROM** funcionario f1,

(**SELECT AVG(salario) media\_salario, dnr FROM funcionario GROUP BY dnr**) f2

**WHERE** f1.dnr = f2.dnr **AND** salario >= media\_salario;





# Subconsultas na cláusula FROM

- Subconsultas na cláusula FROM são aceitas pela maioria (mas não todas) as implementações SQL
- Algumas implementações exigem que a relação resultante da **subconsulta** recebam um **nome** mesmo que o nome nunca seja referenciado

-- Retorne o nome do funcionário que possui o salário maior que a média salarial de seu departamento

```
SELECT f1.pnome, salario, media_salario, f1.dnr
```

```
FROM funcionario AS f1,
```

```
(SELECT AVG(salario) media_salario, dnr FROM funcionario GROUP BY dnr) AS f2
```

```
WHERE f1.dnr = f2.dnr AND salario >= media_salario;
```





# Subconsultas na cláusula SELECT

- Em SQL é possível realizar subconsultas dentro da cláusula SELECT (subconsultas escalares).
- **Subconsultas escalares** podem ser definidas sem agregação (group by)

- Exemplo:

-- Retorna o nome, a quantidade de funcionários e o número do departamento

**SELECT**

    dnome, dnumero,

    (**SELECT** COUNT(\*) **FROM** funcionario **WHERE** dnumero=dnr) **AS** qtdfuncionarios

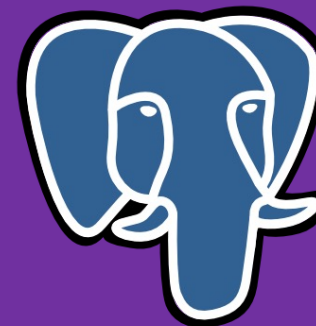
**FROM** departamento;





LABORATÓRIO DE BANCO DE DADOS

# SQL - Consultas



PostgreSQL

- SQL Básico
- Consulta com operadores
- Consultas com agrupamento e agregações
- Subconsultas
- Consultas com condicionais
- Operações a partir de conjuntos
- Outras funções



# Expressões condicionais

## CASE

- A expressão CASE do SQL é uma expressão condicional genérica, semelhante às declarações if/else de outras linguagens:

```
CASE WHEN condição THEN resultado  
[WHEN ...]  
[ELSE resultado]  
END;
```

- A condição é uma expressão que retorna um resultado *boolean*
  - Se o resultado for verdade, então o valor da expressão CASE é o resultado que segue a condição.
  - Se o resultado for falso, todas as cláusulas WHEN seguintes são analisadas da mesma maneira.
  - Se o resultado de nenhuma condição WHEN for verdade, então o valor da expressão CASE é o valor do resultado na cláusula ELSE.
  - Se a cláusula ELSE for omitida, e nenhuma condição for satisfeita, o resultado será nulo.





# Funções de agregação e agrupamento SQL

## CASE

- A expressão **CASE** pode fazer parte da lista de agrupamento
  - Exemplo: contar a quantidade de usuários agrupando pela sua classificação

```
SELECT
    CASE
        WHEN salario <= 10000 THEN 'BAIXO'
        WHEN salario > 10000 AND salario <= 30000
        THEN 'MEDIO'
        WHEN salario > 30000 AND salario <= 50000
        THEN 'ALTO'
        ELSE 'SUPER ALTO'
    END AS classificacao_salario,
    COUNT(*) AS quantidade
FROM
    funcionario
GROUP BY
    classificacao_salario
ORDER BY 1;
```

Classificacao_salario	quantidade
ALTO	3
BAIXO	2
MEDIO	4
SUPER ALTO	2





# Consulta com CASE

- **O comando CASE**

- Permite mudar o valor de um dado
  - Exemplo: codificar o atributo sexo como 1 = masculino, 2 = feminino, 0 = indefinido (expressar os valores por extenso ao invés de usar código).

```
SELECT pnome,  
      (CASE  
        WHEN sexo='M' THEN 'Masculino'  
        WHEN sexo='F' THEN 'Feminino'  
        END) as sexo,  
      endereco,  
      salario  
FROM funcionario;
```

pnome	sexo	endereco	salario
James	Masculin		55000.0
	o	450 Stone, Houston, TX	0
Jennifer	Feminin		43000.0
	o	291 Berry, Bellaire, TX	0
Franklin	Masculin		40000.0
	o	638 Voss, Housto, TX	0
John	Masculin		30000.0
	o	731 Fondren, Houston, TX	0
Ramesh	Masculin		38000.0
	o	975 Fire, Oak Humble, TX	0
Joyce	Feminin		25000.0
	o	5631 Rice, Houston, TX	0
Ahmad	Masculin		25000.0
	o	980 Dallas, Houston, TX	0
Robert	Masculin	2365 Newcastle Rd, Bellaire, TX	58000.0
	o	TX	0
Alicia	Feminin		25000.0
	o	3321 Castle, Spring, TX	0
Vanessa		3321 Castle, Spring, TX	10000.0
	Masculin		0
Asdrubal	o	3321 Castle, Spring, TX	5000.00





# Utilizando CASE e IN

- É possível utilizar CASE com subconsultas IN
  - Retorne o nome, o cpf e 'sim' caso o funcionário também seja gerente e 'não' caso contrário

```
SELECT pnome, cpf,  
       CASE  
         WHEN cpf IN (  
             SELECT cpf_gerente  
             FROM departamento)  
         THEN 'sim'  
         ELSE 'não'  
       END funcionario_gerente  
FROM  
  funcionario;
```

```
postgres=# SELECT pnome, cpf,  
postgres=#       CASE  
postgres=#       WHEN cpf IN (SELECT cpf_gerente FROM departamento)  
postgres=#       THEN 'sim'  
postgres=#       ELSE 'não'  
postgres=#       END funcionario_gerente  
postgres=# FROM  
postgres=#       funcionario;  
  pnome  |      cpf      | funcionario_gerente  
-----+-----+-----  
James    | 888665555    | sim  
Jennifer | 987654321    | sim  
Franklin | 333445555    | sim  
John     | 12345678     | não  
Joyce    | 453453453    | não  
Ahmad    | 987987987    | não  
Robert   | 943775543    | não  
Alicia   | 999887777    | não  
Vanessa  | 911887776    | não  
Asdrubal | 91188711     | não  
(10 registros)
```







# Utilizando CASE e EXISTS

- É possível utilizar CASE com subconsultas IN
  - Retorne o nome, o cpf e 'sim' caso o funcionário também seja gerente e 'não' caso contrário

```
SELECT pnome, cpf,  
CASE  
    WHEN EXISTS (  
        SELECT * FROM departamento d  
        WHERE f.cpf=d.cpf_gerente)  
    THEN 'sim'  
    ELSE 'não'  
END funcionario_gerente  
FROM  
    funcionario f;
```

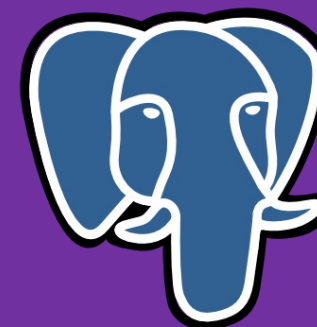
```
postgres=# SELECT pnome, cpf,  
postgres=#         CASE  
postgres=#         WHEN EXISTS (SELECT * FROM departamento d where f.cpf=d.cpf_gerente)  
postgres=#         THEN 'sim'  
postgres=#         ELSE 'não'  
postgres=#         END funcionario_gerente  
postgres=# FROM  
postgres=#         funcionario f;  
 pnome |      cpf      | funcionario_gerente  
-----+-----+-----  
James  | 888665555    | sim  
Jennifer | 987654321    | sim  
Franklin | 333445555    | sim  
John   | 12345678     | não  
Joyce  | 453453453    | não  
Ahmad  | 987987987    | não  
Robert | 943775543    | não  
Alicia | 999887777    | não  
Vanessa | 911887776    | não  
Asdrubal | 91188711     | não  
(10 registros)
```





LABORATÓRIO DE BANCO DE DADOS

# SQL - Consultas



PostgreSQL

- SQL Básico
- Consulta com operadores
- Consultas com agrupamento e agregações
- Subconsultas
- Consultas com condicionais
- Operações a partir de conjuntos
- Outras funções

# Operações sobre conjuntos em SQL

- SQL considera uma tabela como um **multiconjunto** (tuplas duplicadas podem aparecer em uma tabela ou resultado de uma consulta)
- SQL não elimina automaticamente tuplas duplicadas nos resultados das consultas
  - Operação de duplicatas é uma operação dispendiosa (para remover é necessário classificar as tuplas e depois eliminar as duplicatas)
- Uma tabela com uma chave é restrita a ser um conjunto uma vez o que o valor de chave precisa ser distinto em cada tupla.
- Para eliminar tuplas duplicadas pode-se utilizar a palavra chave **DISTINCT** na cláusula **SELECT**.





# Operações sobre conjuntos em SQL

## SELECT ... DISTINCT ...

- Executa consultas em tabelas removendo duplicatas de tuplas.

```
SELECT [DISTINCT] * | <lista de atributos e funções>  
FROM <lista de tabelas>;
```

- Exemplo:

-- Selecciona o nome de todos os funcionários da empresa sem duplicidade de primeiro nome

```
SELECT DISTINCT Pnome FROM funcionario;
```



# Operações sobre conjuntos em SQL

- Operação sobre conjuntos:

- **União** ( $R \cup S$ )

- **UNION**

- Une todas as linhas selecionadas por duas consultas, **eliminando as linhas duplicadas**
      - Gera uma relação que contém todas as tuplas pertencentes a R, a S, ou ambas R e S.

- **UNION ALL**

- Une todas as linhas selecionadas por duas consultas, **inclusive as linhas duplicadas**

- **Intersecção** ( $R \cap S$ )

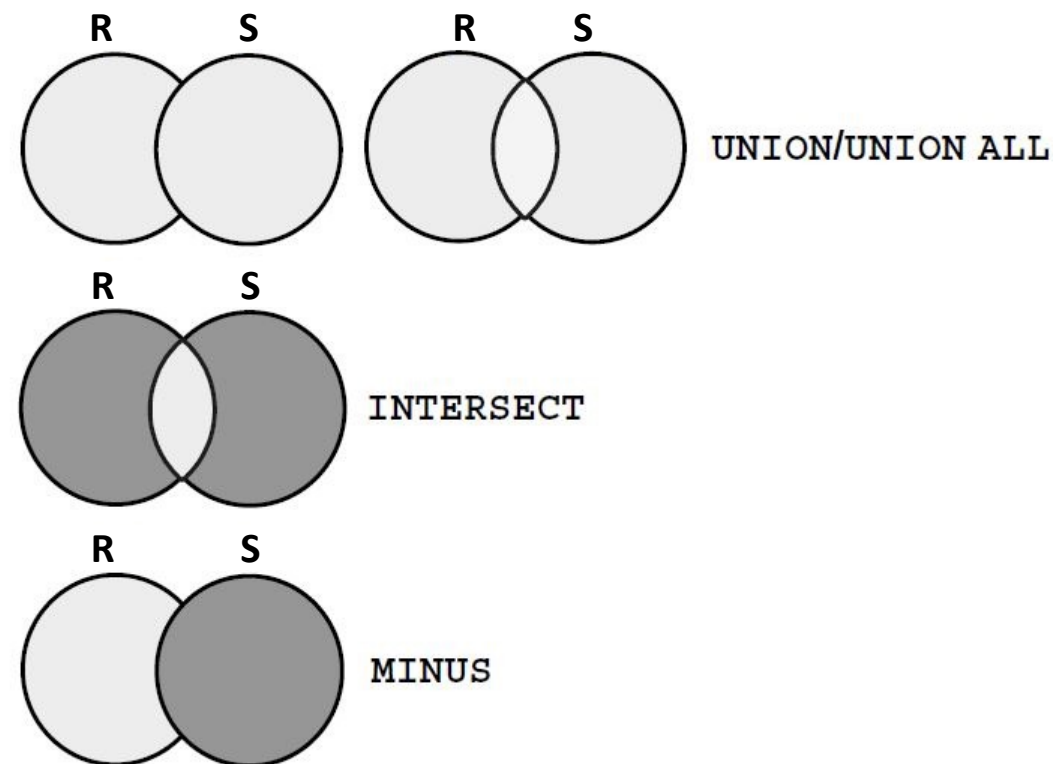
- **INTERSECT**

- Gera uma relação que contém todas as tuplas pertencentes tanto a R quanto a S

- **Diferença** ( $R - S$ )

- **EXCEPT**

- Gera uma relação que contém todas as tuplas pertencentes a R que não pertencem a S



# Operações sobre conjuntos em SQL

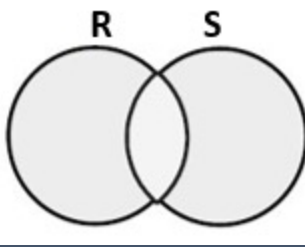
- As relações resultantes dessas operações de conjuntos são **conjuntos de tuplas**
  - Relações tuplas duplicadas são eliminadas do resultado
- Para realizar essas operações as tabelas precisam ser **compatíveis**
  - Relações com atributos com o **mesmo nome** e que apareçam na **mesma ordem** nas duas relações
- A SQL também permite essas operações em **multiconjuntos** (com tuplas repetidas)
  - Para isso basta utilizar a palavra chave **ALL**: **UNION ALL**, **EXCEPT ALL**, **INTERSECT ALL**





# Operações sobre conjuntos em SQL

## UNION



- **UNION**

-- Liste o nome, sexo e data de nascimento de todos os funcionários e dependentes que nasceram após 1945

```
SELECT Pnome as nome, sexo, datanasc
```

```
FROM funcionario
```

```
WHERE datanasc > '01/01/1945'
```

```
UNION
```

```
SELECT nome_dependente as nome, sexo, datanasc
```

```
FROM dependente
```

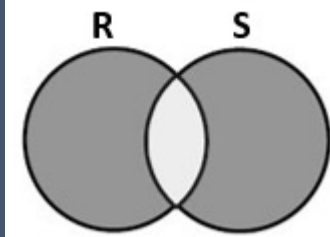
```
WHERE datanasc > '01/01/1945';
```





# Operações sobre conjuntos em SQL

## INTERSECT



- **INTERSECT**

-- Liste os nomes dos dependentes que possuem nome igual ao de algum dos funcionários

```
SELECT Nome_dependente  
FROM dependente  
INTERSECT  
SELECT Pnome  
FROM funcionario;
```

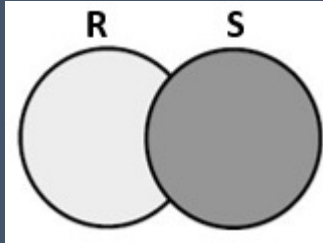






# Operações sobre conjuntos em SQL

## EXCEPT



- **EXCEPT**

-- Liste os nomes dos funcionários que não possuem dependentes

**SELECT** Pnome

**FROM** funcionario

**EXCEPT**

**SELECT** Pnome

**FROM** funcionario, dependente

**WHERE** funcionario.cpf = dependente.fcpf;





LABORATÓRIO DE BANCO DE DADOS

# SQL - Consultas



PostgreSQL

- SQL Básico
- Consulta com operadores
- Consultas com agrupamento e agregações
- Subconsultas
- Consultas com condicionais
- Operações a partir de conjuntos
- Window Functions



# *Window Functions*

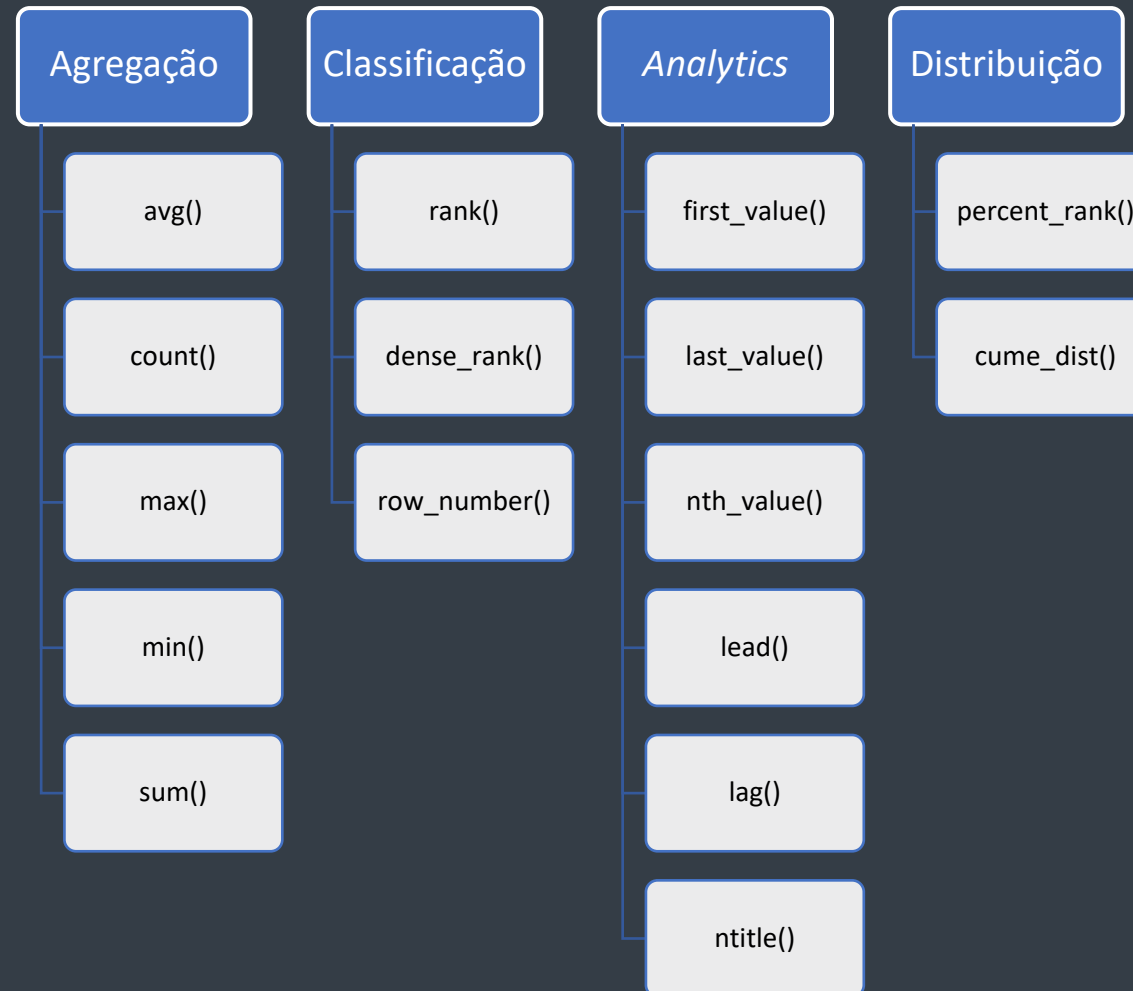
- Também são conhecidas como funções analíticas
- Calcula valores em um grupo de linhas, e retorna um único resultado para cada linha (diferente das funções de agregação)
- Um grupo de linhas correlacionadas é considerada uma “Janela” (OVER)
  - Permite trabalhar com totais, agrupamento, ordenação e cálculos complexos
- A partir dessas funções é possível calcular médias móveis, classificar itens, calcular somas cumulativas e realizar outras análises





# Window Functions

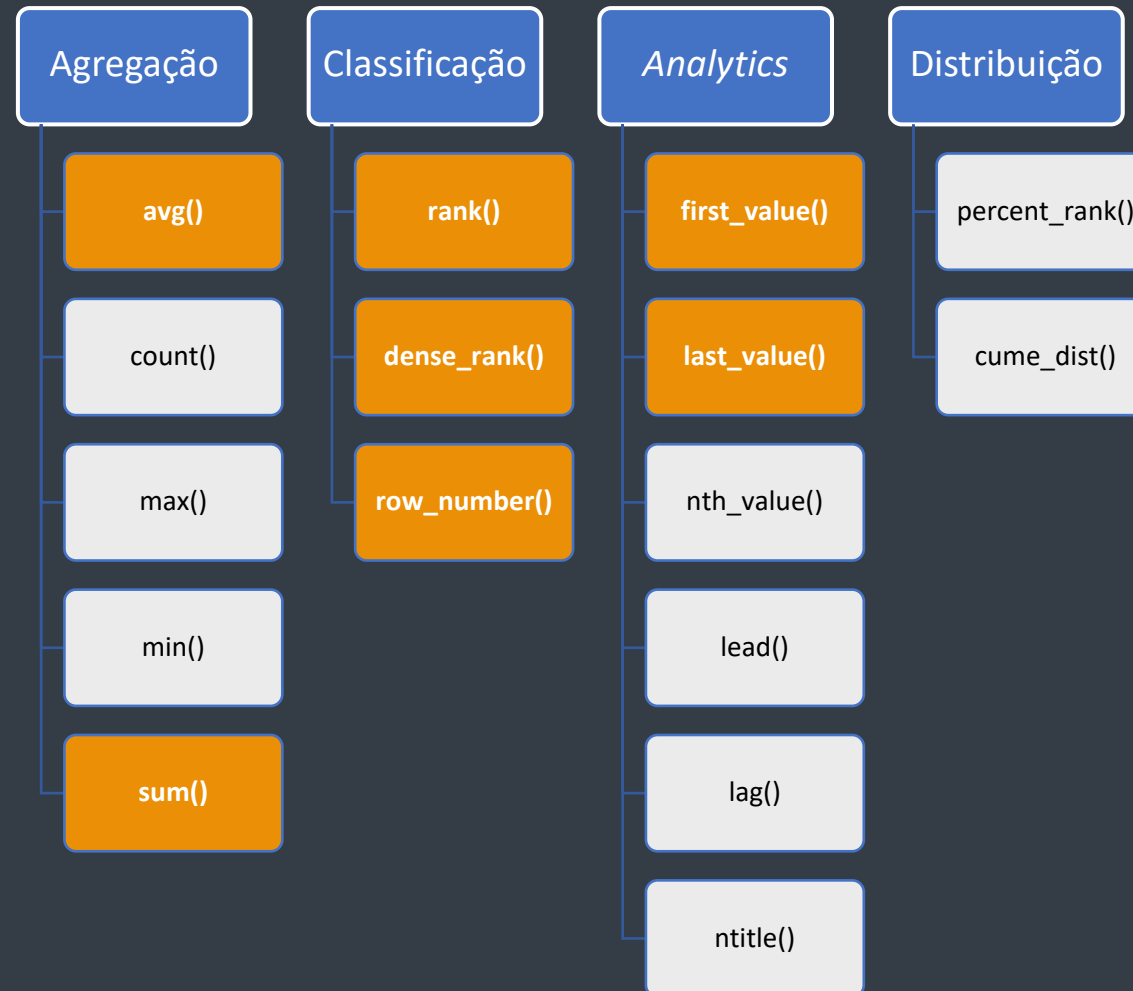
## Lista de Funções





# Window Functions

## Lista de Funções





# Window Functions

```
SELECT <column_1>, <column_2>, <window_function>() OVER (  
    [PARTITION BY <...>]  
    [ORDER BY <...>]  
    [<window_frame>]) <window_column_alias>  
FROM <table_name> ...;
```

- OVER determina exatamente como as linhas da consulta são divididas para processamento pela função de janela
- PARTITION BY (em OVER) divide as linhas em grupos, ou partições, que compartilham os mesmos valores da(s) expressão(ões) as quais a função analítica é aplicada
  - Cláusula opcional – se não for usada, todas as linhas na tabela de entrada farão parte de uma única partição
  - Várias expressões de partição são permitidas na cláusula PARTITION BY
- ORDER BY define a ordem em que as linhas são processadas pelas funções
  - Cláusula opcional na maioria das vezes, obrigatória quando há utilização de funções de navegação

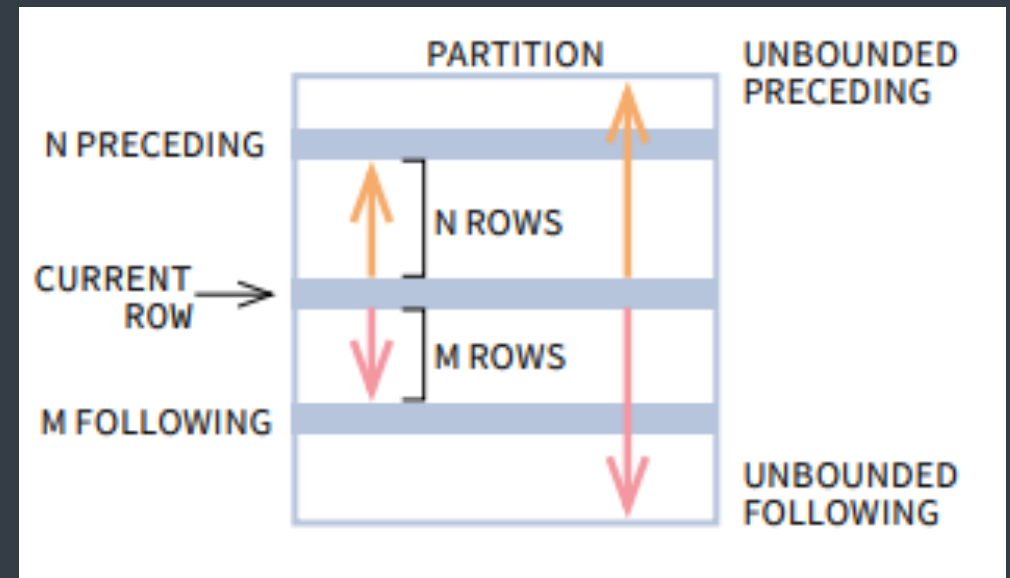




# Window Functions

```
SELECT <column_1>, <column_2>, <window_function>() OVER (  
  PARTITION BY <...>  
  ORDER BY <...>  
  <window_frame>) <window_column_alias>  
FROM <table_name> ...;
```

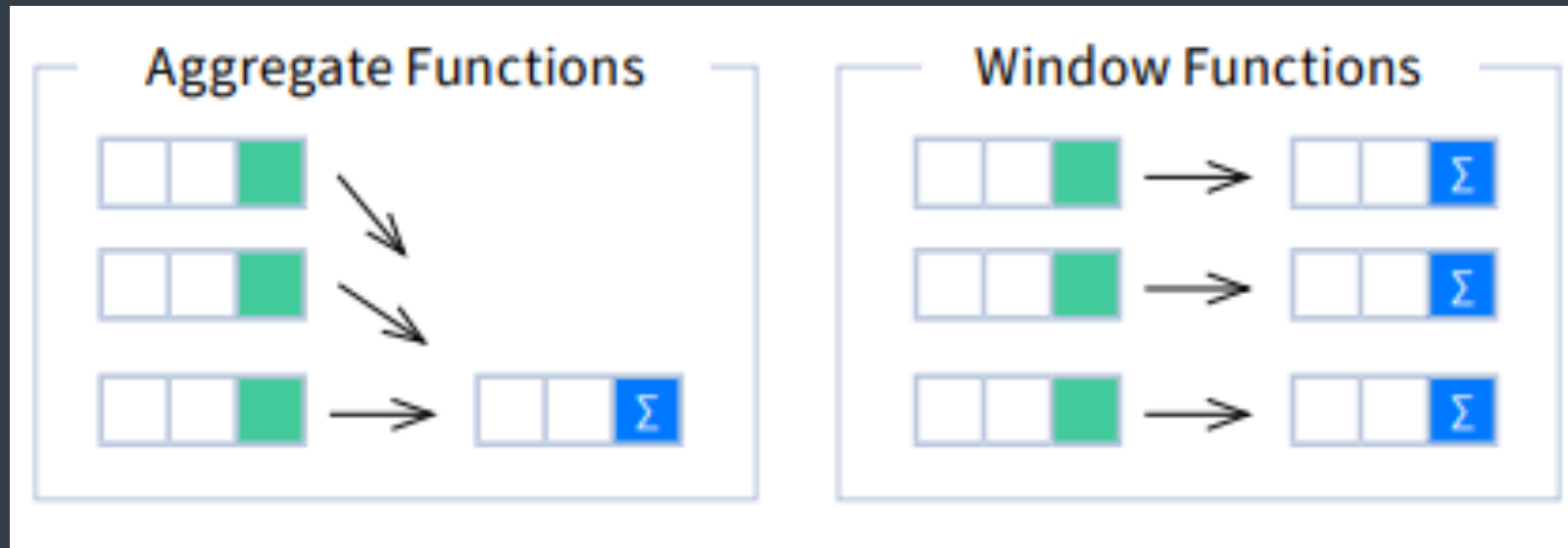
- window\_frame
  - Cláusula opcional
  - Ele identifica o conjunto de linhas usadas em cada cálculo.
  - Ex: ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING





# Aggregate Functions and Window Functions

- Pode ser considerada complementar às funções de agregação

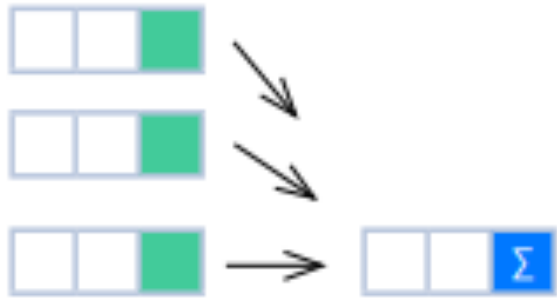




Nesse exemplo são calculados os subtotais de compras realizadas por categoria

item	compras	categoria
laranja	2	fruta
alho-poro	2	vegetal
maca	8	fruta
repolho	9	vegetal
alface	10	vegetal
couve	23	vegetal

### Aggregate Functions



categoria	sum
fruta	10
vegetal	44

### Window Functions



item	compras	categoria	sum
laranja	2	fruta	10
maca	8	fruta	10
alho-poro	2	vegetal	44
repolho	9	vegetal	44
alface	10	vegetal	44
couve	23	vegetal	44

```
select categoria, sum(compras)
from producao
group by categoria;
```

```
select item, compras, categoria, sum(compras)
over(partition by categoria) as sum
from producao;
```



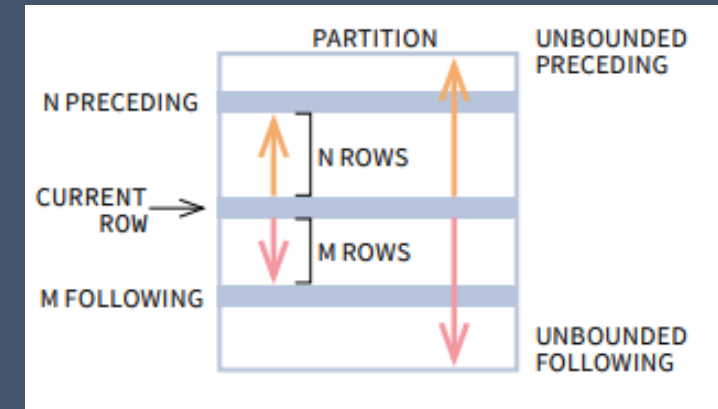
Nesse exemplo são calculadas as somas cumulativas para cada categoria

item	compras	categoria
laranja	2	fruta
alho-poro	2	vegetal
maca	8	fruta
repolho	9	vegetal
alface	10	vegetal
couve	23	vegetal



item	compras	categoria	sum
laranja	2	fruta	2
maca	8	fruta	10
alho-poro	2	vegetal	2
repolho	9	vegetal	11
alface	10	vegetal	21
couve	23	vegetal	44

```
select item, compras, categoria, sum(compras)
over(
  partition by categoria
  order by compras
  rows between unbounded preceding and current row) as sum
from producao;
```



# MÉDIA MÓVEL



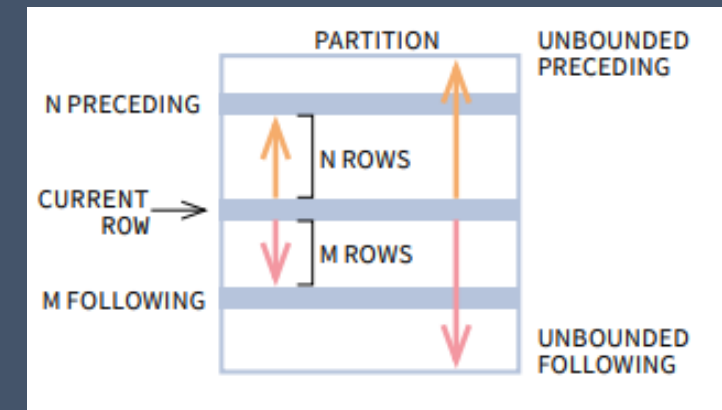
Cálculo da média móvel de compras, considerando o limite inferior uma linha anterior a linha atual e o limite superior uma linha após a linha atual.

item	compras	categoria
laranja	2	fruta
alho-poro	2	vegetal
maca	8	fruta
repolho	9	vegetal
alface	10	vegetal
couve	23	vegetal



item	categoria	compras	media_movel
laranja	fruta	2	2,00
alho-poro	vegetal	2	4,00
maca	fruta	8	6,30
repolho	vegetal	9	9,00
alface	vegetal	10	14,00
couve	vegetal	23	16,50

```
SELECT item, categoria, compras,  
AVG(compras) over (ORDER BY compras ROWS BETWEEN  
1 preceding AND 1 following) media_movel  
FROM producao;
```



## Classificação – rank / dense\_rank / row\_number



- As **funções de numeração** atribuem valores inteiros a cada linha com base na ordenação (ORDER BY)
  - RANK – linhas com o mesmo valor na coluna de ordenação, recebem o mesmo valor de classificação. A próxima linha recebe a classificação de acordo com o número de linhas
  - DENSE\_RANK – sempre incrementa 1 sem pular valor
  - ROW\_NUMBER – retorna a ordem em que as linhas aparecem

Calcula a classificação de cada item com base no número de compras

x	rank	dense_rank	row_number
1	1	1	1
2	2	2	2
2	2	2	3
5	4	3	4
8	5	4	5
10	6	5	6
10	6	5	7

```
SELECT x,  
       RANK() OVER (ORDER BY x ASC) AS rank,  
       DENSE_RANK() OVER (ORDER BY x ASC) AS dense_rank,  
       ROW_NUMBER() OVER (ORDER BY x ASC) AS row_num  
FROM numeros;
```



## Cálculo da classificação de cada item com base no número de compras

item	compras	categoria
laranja	2	fruta
alho-poro	2	vegetal
maca	8	fruta
repolho	9	vegetal
alface	10	vegetal
couve	23	vegetal



item	compras	categoria	rank
laranja	2	fruta	1
alho-poro	2	vegetal	1
maca	8	fruta	3
repolho	9	vegetal	4
alface	10	vegetal	5
couve	23	vegetal	6

```
select item, compras, categoria, rank()  
over(order by compras) as rank  
from producao;
```



## NAVEGAÇÃO – FIRST VALUE / LAST VALUE



- As funções de navegação retornam o valor da expressão especificada com base na janela e sua ordenação
  - FIRST\_VALUE – retorna o valor da primeira linha
  - LAST\_VALUE – retorna o valor da última linha

Retorna o item mais conhecido da categoria com base na categoria e informações de compra

item	compras	categoria
laranja	2	fruta
alho-poro	2	vegetal
maca	8	fruta
repolho	9	vegetal
alface	10	vegetal
couve	23	vegetal



item	compras	categoria	primeiro	ultimo
laranja	2	fruta	laranja	maca
maca	8	fruta	laranja	maca
alho-poro	2	vegetal	alho-poro	couve
repolho	9	vegetal	alho-poro	couve
alface	10	vegetal	alho-poro	couve
couve	23	vegetal	alho-poro	couve

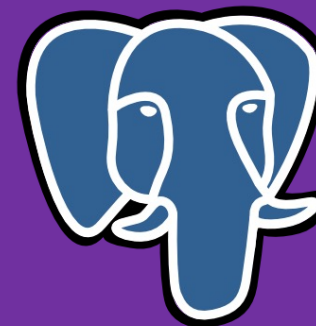
```
SELECT item, compras, categoria,  
FIRST_VALUE(item) OVER (PARTITION BY categoria  
ORDER BY compras ROWS BETWEEN unbounded preceding AND unbounded following) AS primeiro,  
LAST_VALUE(item) OVER (PARTITION BY categoria  
ORDER BY compras ROWS BETWEEN unbounded preceding AND unbounded following) AS ultimo  
FROM producao;
```





LABORATÓRIO DE BANCO DE DADOS

# SQL - Consultas



PostgreSQL

- SQL Básico
- Consulta com operadores
- Consultas com agrupamento e agregações
- Subconsultas
- Consultas com condicionais
- Operações a partir de conjuntos
- Outras funções



# Funções: limitar casas decimais

## TRUNC | ROUND



PostgreSQL

- **TRUNC**: não faz o arredondamento, somente limita as casas decimais
- **ROUND**: não faz o arredondamento, somente limita as casas decimais

**TRUNC**(<valor>, quantidade de casas decimais)

**ROUND**(<valor>, quantidade de casas decimais)

```
SELECT sexo, TRUNC(AVG(salario),2) AS media_salario FROM funcionario GROUP BY sexo;
```

```
SELECT sexo, ROUND(AVG(salario),2) AS media_salario FROM funcionario GROUP BY sexo;
```

```
SELECT TRUNC(1.345,2) ; -- resultado = 1.34
```

```
SELECT ROUND(1.345,2) ; -- resultado = 1.35
```







# Funções: limitando quantidade de tuplas

## LIMIT



PostgreSQL

- A cláusula LIMIT é utilizada para limitar o número de resultados de uma consulta SQL

-- Retorna as 5 primeiras tuplas da consulta

```
SELECT * FROM funcionario LIMIT 5;
```

-- Retorna as 5 primeiras tuplas da consulta com início na quinta tupla (OFFSET indica a posição de início da paginação)

-- Mostra o resultado da tupla 6 até a tupla 10

```
SELECT * FROM funcionario OFFSET 5 LIMIT 5;
```





# Concatenação de string

- Para tipo de dados *string*, o operador de concatenação || pode ser utilizado para juntar valores.
- **SELECT 'Nome: ' || Pnome FROM FUNCIONARIO;**





# Função COALESCE

COALESCE(valor [, ...])

- A função COALESCE retorna o primeiro de seus argumentos que não for nulo.
- Só retorna nulo quando todos os seus argumentos são nulos.
- Geralmente é útil para substituir o valor padrão quando este é o valor nulo, quando os dados são usados para exibição.

