INSTITUTO FEDERAL
Paraíba
Campus João Pessoa

Aula
1

Banco de Dados II

Atualizada em 18/09/2019

SQL Avançado

PostgreSQL

Professor:
Dr. Alex Sandro da Cunha Rêgo

alex@ifpb.edu.br

Objetivos

- Revisando a sintaxe SQL
  - SELECT, UPDATE, INSERT e DELETE
- Manipulando expressões
  - Funções matemáticas, caractere, data e conversão
- Condições de pesquisa
  - Faixas, listas e casamento de padrões
- Funções de agregação e agrupamento de dados
- Junções de tabelas
  - Inner Join, left outer join, right outer join
- Subconsultas e tabelas temporárias

Structured Query Language

- **Aspectos gerais da SQL**
  - Linguagem não procedural que requer do usuário qual dado é necessário sem especificar como obtê-lo
  - Poupa tempo de programação mas exige treino para dominá-lo
  - Linguagem interativa de consulta: consulta aos dados de um SGBD sem necessidades de programas
  - Suporte cliente-servidor: clientes se comunicam com o servidor de banco de dados através de comandos SQL
  - Controle de acesso: protege os dados contra aceso e manipulações não autorizadas

Structured Query Language

- **Aspectos Gerais da SQL**
  - Integridade dos dados: define as regras de integridades contra duplicidade, inconsistências e falhas no sistema.
  - Independência de fabricante: está incorporada em quase todos os SGBDs em seu padrão ANSI, com extensões proprietárias de cada fabricante.
  - Facilidade no entendimento: oferece um rápido entendimento, com comandos escritos em um inglês estruturado de alto nível
  - Definição de Dados (DDL): possibilida a definição da estrutura e organização dos dados, definição de índices, criação de tabelas, etc.

SELECT – Consultando Dados

- **Sintaxe Básica**

```
SELECT lista_de_colunas
FROM lista_de_tabelas
WHERE condições
```

- **Legenda**

  SELECT: especifica as colunas e expressões a serem exibidas no resultado da consulta

  FROM: especifica as tabelas que contém os dados a serem pesquisados

  WHERE: especifica as condições usadas para filtrar registros.

SELECT – Consultando Dados

- **Cláusulas adicionais**

  INTO: especifica uma nova tabela que conterá o resultado da consulta.

  ORDER BY: classifica o resultado da consulta com base em uma ou mais colunas.

  GROUP BY: agrupa as linhas das consultas com base nos valores de uma ou mais colunas

  HAVING: especifica as condições usadas para filtrar agrupamentos de dados no resultado da consulta. **Só deve ser usado com GROUP BY**

Consultas Simples

- **Exemplo1:** exibir todos os dados de todos os clientes (máscara '*')

```
SELECT * FROM cliente;
```

- **Exemplo 2:** exibir código, nome e telefone de todos os clientes (seleção de colunas)

```
SELECT idCliente,nome,fone FROM cliente;
```

- **Exemplo 3:** selecionar código, nome e telefone de todos os clientes, para uma tabela denominada **resultado**

```
SELECT idCliente,nome,fone INTO resultado
FROM cliente;
```

Consultas Simples

- **TOP n Registros**

```
SELECT <colunas>
FROM <tabelas>
WHERE <condição>
[ORDER BY <colunas>]
LIMIT n
```

- **Exemplo 4:** exibir código, nome e telefone dos 20 primeiros clientes cadastrados na empresa

```
SELECT idCliente,nome,fone
FROM cliente
LIMIT 20;
```

Consultas simples

- **Exemplo 5:** exibir código, nome e uma coluna **situção** contendo o valor 'classificado' para todos os clientes cadastrados

Valor pode ser envolvido por ' '

```
SELECT idCliente, nome, 'classificado'
situação FROM cliente;
```

- **Exemplo 6:** exibir o código, descrição e preço de custo dos produtos, renomeando a coluna **custo** para **preço de custo**

```
SELECT idProduto , descricao, custo AS
'Preço de Custo'
FROM produto;
```

Novo nome deve estar envolvido por "" se tiver espaço em branco

Consultas simples

- **Geração de Coluna Calculada**

```
SELECT <coluna1,coluna2,expressão>
FROM lista_de_tabelas
WHERE condições
```

- **<u>Exemplo 7</u>:** exibir código, quantidade em estoque, preço da venda e valor total (quantidade x preço da venda) para cada produto

```
SELECT idProduto, quantEst Quantidade,
venda, quantEst*venda AS Valor_Total
FROM produto;
```

Consultas Simples

- **<u>Exemplo 8</u>:** exibir o código (renomeando para matrícula), nome, salário e o salário com 30% de aumento para todos os funcionários.

```
SELECT idFuncionario Matricula, nome,
salario, salario*1.3 AS 'Novo Salario'
FROM funcionario;
```

Observe que após a palavra-chave **AS**, a nova coluna Pode vir envolvida por “” ou não. Entretanto, se o Nome da nova coluna tiver espaço em branco, passa A ser obrigatório o uso dos “”

Operadores Aritméticos

- Além de **operadores aritméticos**, podemos invocar **funções matemáticas** embutidas (consulte a referência do SGBD)

Operador	Descrição	Exemplo	Resultado
+	Adição	`2 + 4`	6
-	Subtração	`2 - 4`	-2
*	Multiplicação	`2 * 4`	8
/	Divisão inteira	`5 / 2`	3
%	Resto da divisão	`7 % 2`	1
^	Exponenciação	`2.0 ^ 3`	8
|/	Raiz quadrada	`|/ 81`	9
||/	Raiz cúbica	`||/27.0`	3
@	Valor absoluto	`@ -5.0`	5.0
!	Fatorial	`5!`	120

Funções Matemáticas

- Funções mais utilizadas
  - https://www.postgresql.org/docs/9.5/functions-math.html

Função	Descrição	Exemplo	Resultado
sin(v), cos(v), tan(v)	Seno, cosseno e tangente	`sin(90)`	`0.89399`
random()	Valor randômico na faixa 0.0 <= x <= 1.0	`random()`	`0.93839`
round(v,d)	Arredondamento	`round(37.4561, 2)`	`8`
div(x,y)	Inteiro quociente de x/y	`div(9/4)`	`2`
abs(x)	Valor absoluto	`abs(-5)`	`5`
ceil(x), floor(x)	Inteiro mais próximo maior/menor arg.	`ceil(3.678)` `floor(3.678)`	`4` `3`

Funções matemáticas

- **Exemplo 9:** exiba o nome do produto, preço de venda e o valor arredondado do preço de cada produto para 1 casa decimal

```
SELECT nome, venda 'Preço',
ROUND(preco,1) 'Preço com 1 casa decimal'
FROM produto;
```

- **Exemplo 10:** exiba o resultado do truncamento dos números 2.15 e 2.99

```
SELECT floor(2.99) primeiro, floor(2.15)
Segundo;
```

Funções para Strings

- **Operadores e Funções**
  - https://www.postgresql.org/docs/9.5/functions-string.html

Operador	Descrição	Exemplo	Resultado
||	Concatenção (str/str ou str/non-str)	'Post'||'gre'	Postgre
Função	**Descrição**	**Exemplo**	**Resultado**
`lower(s)`	Conversão de um string para minúsculo	`lower('ABC')`	abc
`upper(s)`	Conversão de um string para maiúsculo	upper('abc')	ABC
`left(s,n)`	Retornar os n primeiros caracteres	`left('abc',2)`	ab
`right(s,n)`	Retornar os n últimos caracteres	`right('abc',2)`	bc

Funções para Strings

- **Operadores e Funções**

Função	Descrição	Exemplo	Resultado
`char_length(s)`	Número de caracteres de uma string	char_length('IFPB')	4
`replace(s,match, str_replace)`	Substituição de ocorrências de um substring	replace('IFPB','PB', 'X')	'IFX'
`position(s in t)`	Obtém a posição de **s** em **t**	`position('a' in 'ca')`	2
`substring(s [from int] [for int]`	Extrai um substring a partir do intervalo especificado	`substring('IFPB' from 2 for 2)`	'FP'
`overlay(str placing rs from start_pos [for number_char])`	Substituir, a partir de uma string principal, um substring por outro	`overlay('IFPB' placing 'XX' from 3 for 2)`	'IFXX'

Funções para Strings

- **Exemplo 11:** exiba os 10 primeiros caracteres dos título dos livros em minúsculo, e seu respectivo ISBN na tabela **livros**

```
SELECT lower(left(titulo,10)), isbn
FROM livros;
```

- **Exemplo 12:** exiba os títulos dos livros que contém a palavra "SQL"

```
SELECT titulo FROM livro
WHERE position('SQL' in titulo) > 0;
```

  - **Nota**: position() é case-sensitive

Funções de Strings

- **Exemplo 13:** exiba o nome de cada aluno acrescido do status 'aprovado'

```
SELECT nome || '(aprovado)' Status
FROM aluno;
```

- **Exemplo 14:** exiba os 10 primeiros caracteres dos alunos matriculados

```
SELECT substring(nome from 1 for 10)
FROM aluno;
```

- **Exemplo 15:** exiba o nome dos alunos em caracteres minúsculos

```
SELECT lower(nome)
FROM aluno;
```

Funções de Data/Hora

- PostgreSQL possui um rico conjunto de funções para converter/manusear tipos de dados em formato data/hora (**datetime**)
  - Operadores (+,-,*)
  - Funções de extração
- Tipos de Dados:
  - **timestamp**: data/hora (2019-02-12 14:17:05) – 8bytes
  - **date**: data (2019-02-12) – 4bytes
  - **time**: hora do dia (00:00:00 a 24:00:00)
- Referência completa:
  - https://www.postgresql.org/docs/9.5/functions-datetime.html

Funções de Data/Hora

- Operadores

Operador	Exemplo	Resultado
+	date ‘2012-01-01’ + integer ‘9’ date '2001-09-28' + time '03:00'	‘2019-01-10’ date '2001-09-28 03:00:00’ timestamp
-	date '2019-02-10' - date '2019-02-01'	9 integer
+	time '01:00' + interval '3 hours'	'04:00:00' time
-	time '05:00' - time '03:00'	'02:00:00’ interval
*	5 * interval ‘2 day’ 60 * interval '1 second'	10 days interval '00:01:00’ interval
https://www.tutorialspoint.com/postgresql/postgresql_date_time.htm		

```
Tempo: 0,467 ms
dvdrental=# select date '2019-01-10' - date '2019-01-01';
 ?column?
----------
        9
(1 registro)
```

Funções de Data/Hora

- Funções

Função	Descrição
Current date/time and date/time creators	
CURRENT_DATE Retorno: **Date**	Devolve a data atual
CURRENT_TIME Retorno: **Time**	Devolve a hora atual
CURRENT_TIMESTAMP Retorno: **Timestamp**	Devolve o instante de tempo atual (ou **now()**)
age(data1[,data2]) Retorno: **interval**	Devolve o intervalo de tempo decorrido entre duas datas. Se data2 não for informada, assume **current_date**
make_date(y,m,d) Retorno: **Date** make_time(h,m,s) Retorno: **Time**	Cria data/hora a partir dos campos year,month,day, hour,minute,second

Funções de Data/Hora

- Funções

Função	Descrição
Partes de uma data/hora	
date_part('field',source) Retorno: **integer**	date fields: day, month, year, week, decade,dow time fields: hour, minute, second, miliseconds **source**: o campo/dado tipo **timestamp**, **interval, date** ou **time**
extract(field from source) Retorno: **integer**	Obtém um subcampo de um dado do tipo **timestamp**, **interval, date** ou **time**

  - O tipo **interval**: tipo de dados que permite **armazenar ou manipular** um período de tempo em anos, mês, dias, horas, minutos, etc.

```
postgres=# SELECT now(), now() - interval '2 hours 30 minutes' Passado;
              now              |            passado
-------------------------------+-------------------------------
 2019-09-05 09:43:16.084682-03 | 2019-09-05 07:13:16.084682-03
(1 registro)
```

Funções de Data/Hora

- **Exemplo 16:** exiba a quantidade de meses decorrido entre duas datas

```
SELECT EXTRACT( age(date '2019-01-01',
CURRENT_DATE)); -- current_date=2019-09-04
```

AGE(): 8 month and 13 days - Resultado: 8

- **Exemplo 17:** exiba o nome do cliente e a data de vencimento 'dd/mm' da tabela pagamentos

```
SELECT nome, date_part('day',data_venc)
|| '\' || date_part('month',data_venc) AS
resultado /* || operador de concatenação */
FROM pagamento;
```

Funções de Data/Hora

- **Exemplo 18:** exiba o dia da semana de uma data específica

```
SELECT extract(dow from date'2019-01-14');
/* Sunday(0) to Saturday(6)
   dow: day of week
   doy: day of year */
```

- **Exemplo 19:** exiba o id do empréstimo, a data de emprestimo e uma nova data de vencimento equivalente a 5 dias à frente.

```
SELECT numEmprestimo, dataEmprestimo,
dataEmprestimo + 5 AS dataFutura
FROM pagamento;
/* SELECT date '2019-02-15' + 5   Operador  + */
```

Funções de Data/Hora

- **Exemplo 20**: exiba o nome e mês de aniversário de todos os funcionários

```
SELECT nome, date_part('month',dataNasc)
AS 'Mes de Aniversario'
FROM funcionario;
```

- **Exemplo 21**: exiba a data de aniversário de todos os funcionários e a data atual nomeada para **Hoje**

```
SELECT dataNasc, current_date 'Hoje'
FROM funcionario;
```

Funções de Conversão de Tipo

- **Convenção: funções to_<type>()**
  - O primeiro argumento é o valor a ser formatado, enquanto que o segundo é um *template* que define o formato de saída

Função	Descrição
to_char(type,'padrão')	Converte um argumento de um tipo específico em texto type: timestamp, int, double, numeric, interval
to_number(text,padrão)	Converte um string em um valor numérico
to_date(text,padrão)	Converte um string para um valor date
`>>> CAST e notação ::`	
cast(expressão **as** type)	Converte um tipo de dado em outro type: integer, date, timestamp, text, real,...
`cast::type`	type: ..., numeric, money, boolean,...
https://www.postgresql.org/docs/9.5/functions-formatting.html http://www.postgresqltutorial.com/postgresql-cast/	

Funções de Conversão de Tipo

- **Exemplo 22:** exiba a data de nascimento 20/10/2000 no formato DD Mon YY

```
SELECT to_char(date '2000-10-20', 'DD Mon YY');
-- Retorna 20 Oct 00
```

- **Exemplo 23:** exiba a data 20/10/2000 como um dado do tipo **date**

```
SELECT to_date('2019-09-10', 'YYYY-MM-DD');
SELECT CAST('2019-09-10' AS DATE);
SELECT '10-SEP-2019':: AS DATE;
--ou então '2019-09-10'
```

Funções de Conversão de Tipo

- **Exemplo 24:** exiba o valor '145.15' como um dado do tipo **numérico**

```
SELECT to_number('145.15','999,999.99') ;
SELECT CAST('145.15' AS REAL);
SELECT '145.15'::REAL;
```

- **Exemplo 25:** exiba a data de hoje no formato adotado pelo Brasil

```
SELECT to_char(current_date, 'DD-MM-YYYY');
```

Distinção de Dados

- **Sintaxe**:

```
SELECT DISTINCT <lista_de_conlunas>
FROM <tabela>
```

- **Exemplo 26:** Exibir o código de todos os clientes que já fizeram pedido na empresa

```
SELECT DISTINCT idCliente
FROM pedido;
```

- **Exemplo 27:** Exibir os distintos códigos do cliente e vendedor de todos os pedidos

```
SELECT DISTINCT idCliente, idVendedor
FROM pedido;
```

Ordenação de Resultados

- **Sintaxe**:

```
SELECT <lista_de_conlunas> FROM <tabela>
ORDER BY <coluna(s)> [ASC/DESC]
```

O **ORDER BY** aceita a utilização de colunas nomeadas em sua cláusula

- **<u>Exemplo 28</u>:** exibir código, nome e telefone de todos os clientes, ordenado pelo nome do cliente de forma ascendente

```
SELECT idCliente, nome, fone
FROM cliente
ORDER BY nome; -- ASC é o default
```

Ordenação de Resultados

- **Exemplo 29:** exibir código, nome, preço de custo e de venda e a diferença venda/custo para todos os produtos, ordenado pela diferença de preço em ordem descendente

```
SELECT idProduto, nome, custo, venda,
custo-venda 'diferenca'
FROM produto ORDER BY diferenca DESC;
```

- **Exemplo 30:** exibir nome, cidade e CEP dos clientes, ordenado pela cidade em ordem ascend. e pelo cep em ordem descendente

```
SELECT nome, cidade, cep
FROM cliente ORDER BY cidade, cep DESC
```

Cláusula WHERE

- **Sintaxe**:

```
SELECT <lista_de_conlunas> FROM <tabela>
WHERE <coluna OPERADOR expressão>
```

- **Exemplo 31:** exibir todos os dados dos funcionários que nasceram a partir de 1950

```
SELECT *, to_char(cast(datanasc as date),
'DD-MM-YYYY') AS data_brasil
FROM funcionario
WHERE date_part('year',dataNasc) >= 1950;
/* to_char aceita tanto campos do tipo date quanto
timestamp. No exemplo, o CAST pode ser omitido */
```

Operadores do WHERE

Operador	Significado
=	**Igual.** Campo e expressão devem possuir o mesmo valor
<>, !=	**Diferente.** Campo e expressão devem possuir valores diferentes
LIKE	Campos com caracteres que correspondem aos da expressão. Só atua em campos do tipo caractere e distigue maiúsculas de minúsculas. É permitido o uso de máscaras % (percentual)- substitui uma sequência de caracteres _ (sublinhado) – substitui um único caractere
>,>=, <,<=	**Menor, menor igual, maior, maior igual.** Estabeleçe uma condição de relação comparativa entre um campo e uma expressão
BETWEEN	O valor do campo deve estar compreendido no intervalo definido
IN	O Valor do campo deve ser igual a um dos valores da lista
IS NULL	Seleciona as linhas com o valor NULL para o campo especificado

Cláusula WHERE

- **Exemplo 32:** Exibir o nome e estoque atual de todos os produtos que estão com o estoque atual abaixo do estoque mínimo, ordenado pelo descrição de forma ascendente

```
SELECT descricao, quantest FROM produto
WHERE estoqueAtual < estoqueMin
ORDER BY descricao;
```

- **Exemplo 33:** Exibir o nome de todos os produtos que possuam a ocorrência da palavra "diet"

```
SELECT descricao FROM produto
WHERE position('diet' in descricao) > 0;
```

A cláusula BETWEEN

- **Sintaxe**:

```
SELECT <lista_de_conlunas> FROM <tabela>
WHERE <coluna BETWEEN min AND max>
```

- **<u>Exemplo 34</u>:** Exibir os produtos (todas as colunas) cujo estoque atual esteja no interval de 10 a 30 unidades, ordenados pelo nome do produto em ordem ascendente

```
SELECT *
FROM produto
WHERE estAtual BETWEEN 10 AND 30
ORDER BY nome;
```

A cláusula BETWEEN

- **Exemplo 35:** Exibir todos os dados dos funcionários que nasceram na década de 60, ordenados pela data de nascimento em ordem descendente e pelo nome do funcionário em ordem ascendente.

```
SELECT *
FROM funcionario
WHERE datanasc BETWEEN '1960/01/01' AND
'1969/12/31'
ORDER BY datanasc DESC, nome;
```

O filtro para a data deve ser definido no formato configurado no banco (YYYY-MM-DD)

Operador IN

- **Sintaxe**:

```
SELECT <lista_de_conlunas>
FROM <tabela>
WHERE <coluna [NOT] IN (val1,val2,...,valn)
```

- **Exemplo 36:** Exibir todos os dados dos funcionários que residam em Manaíra ou Bessa, ordenados pelo nome em ordem ascendente.

```
SELECT * FROM funcionario
WHERE bairro IN ('Manaíra', 'Bessa')
ORDER BY nome;
```

A string na lista de filtragem é case-sensitive.

Operador IN

- **<u>Exemplo 37</u>:** Exibir todos os produtos que não sejam do tipo 2 ou 4, ordenados pelo tipo em ordem descendente e pelo nome em ordem ascendente.

```
SELECT * FROM produto
WHERE tipo NOT IN (2,4)
ORDER BY tipo DESC, nome;
```

Operador NULL

- **Sintaxe**:

```
SELECT <lista_de_conlunas>
FROM <tabela>
WHERE <coluna IS [NOT] NULL
```

Significado: ausente (não atribuído valor à coluna na inclusão)

- **Exemplo 38:** Exibir o nome e email de todos os funcionários que não possuem email, ordenados pelo nome em ordem ascendente.

```
SELECT * FROM funcionario
WHERE email IS NULL OR
char_length(trim(from email))=0
ORDER BY nome;
```

Operador NULL

- **<u>Exemplo 39</u>:** Exibir os dados de todos os funcionários cujo número do telefone foiinformado, ordenados nome do funcionário em ordem ascendente.

```
SELECT * FROM funcionario
WHERE fone IS NOT NULL
ORDER BY nome;
```

Operador NULL

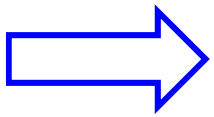
Curiosidade:

Você acha que as linhas com valores **NULL** serão retornadas de acordo com o SELECT abaixo?

```
SELECT * FROM funcionario
WHERE bairro <> 'Miramar';
```

****Não!****

```
 codigo |   nome   |    bairro
--------+----------+--------------
      1 | alex     | Jose Americo
      2 | danielle | Miramar
      3 | Arthur   | Geisel
      4 | Luana    |
      5 | Jessika  |
(5 registros)
```



```
 codigo |   nome   |    bairro
--------+----------+--------------
      1 | alex     | Jose Americo
      3 | Arthur   | Geisel
(2 registros)
```

```
 codigo |   nome   |    bairro
--------+----------+--------------
      1 | alex     | Jose Americo
      2 | danielle | Miramar
      3 | Arthur   | Geisel
(3 registros)
```

```
...bairro IS NOT NULL;
```

```
...bairro IS NULL;
```

```
 codigo |   nome   | bairro
--------+----------+--------
      4 | Luana    |
      5 | Jessika  |
(2 registros)
```

A cláusula LIKE

- **Sintaxe**:

```
SELECT <lista_de_colunas>
FROM <tabela>
WHERE <coluna> [NOT] LIKE <padrão-de-string>
```

  - **Casamento de padrão**: uma expressão que define o critério para seleção de strings
  - **%** (porcentagem) : sequência de caracteres
  - _ (sublinhado) : combina com um único caractere
- **Exemplo**
  - Se uma coluna do tipo string contém 'Sousa' ou 'Souza' em qualquer parte, você pode usar LIKE '%Sou_a%'

A cláusula LIKE

- **Exemplo 40:** Exibir código e nome de todos os funcionários que tenham o nome começando por "M", ordenados pelo nome em ordem ascendente.

```
SELECT idFuncionario,nome FROM funcionario
WHERE nome LIKE 'M%' -- case sensitive
ORDER BY nome;
```

- **Exemplo 41:** Exibir código e nome de todos os funcionários cujo nome termine com "Silva", ordenados pelo nome (ascendente).

```
SELECT idFuncionario,nome FROM funcionario
WHERE nome LIKE '%Silva' --case sensitive
ORDER BY nome;
```

A cláusula LIKE

- **Exemplo 42:** Exibir código e nome de todos os funcionários que tenham "Santos" em qualquer parte do nome.

```
SELECT idFuncionario,nome FROM funcionario
WHERE nome LIKE '%Santos%';--case sensitive
```

- **Exemplo 43:** Exibir código e nome de todos os funcionários que tenham o nome começando por "Mar" seguido de um caractere qualquer e logo depois a vogal "a"

```
SELECT idFuncionario,nome FROM funcionario
WHERE nome LIKE 'Mar_a%'; --case sensitive
```

Use **ILIKE** ao invés de **LIKE** para tornar *case-insensitive*.
Não é um padrão SQL, e sim uma extensão PostgreSQL

A cláusula LIKE

- **Exemplo 44:** Exibir nome e salário de todos os funcionários cujo valor de salário termina com 32 centavos

```
SELECT nome,salario FROM funcionario
WHERE salario::text LIKE '%32';
```

- **Exemplo 45:** Identificar nome e salário de todos os funcionários cujo salário apresenta o número 2 na segunda posição e termine com o número 3

```
SELECT codigo,nome FROM funcionario
WHERE nome LIKE '_2%3‘;
```

A cláusula SIMILAR TO

- **Sintaxe**

```
string [NOT] SIMILAR TO padrão ESCAPE char
```

  - Retorna **true** se um string combina com o padrão
  - Similar ao LIKE, porém, o padrão é interpretado como uma expressão regular
  - Aceita % e _, assim como o LIKE
  - **Metacaracteres**

| - Alternativa	**()** - agrupamento de itens em um único bloco lógico
{m} – repetição exatamente **m** vezes	* - repetição (0 ou mais)
+ - repetição (1 ou mais)	**?** – repetição (0 ou 1)

Para maiores informações, acesse:
https://www.postgresql.org/docs/9.0/functions-matching.html

A cláusula SIMILAR TO

- **Exemplo 46:** Exibir o nome e bairro dos funcionários que morem em bairros iniciados pela letra M ou C

```
SELECT nome,bairro FROM funcionario
WHERE bairro SIMILAR TO '(M|C)%';
```

- **Exemplo 47**: Exibir Todos os nomes começados por 'A' seguido da letra seja 'l' (com ou sem repetição)

```
SELECT nome FROM funcionario
WHERE nome SIMILAR TO 'Al+%';
```

Alex
Allan
Allliance

O entendimento de **expressões regulares** potencializa o uso do casamento de padrões como alternative ao **LIKE** ou **SIMILAR TO** (não está no escopo da disciplina)

A cláusula SIMILAR TO

- **<u>Exemplo 48</u>:** Exibir todas as mensagens de promoção de produtos com anúncio de 10% de desconto

```
SELECT mensagem FROM produto
WHERE mensagem SIMILAR TO '%10 !%%';
ESCAPE ! -- A '!' é o caractere de escape
```

A cláusula **ESCAPE** serve justamente pra evitar problemas de interpretação de consulta com o **LIKE** ou **SIMILAR TO**, especialmente quando símbolos especiais (*,+,{,},%) fazem parte da string mas devem ser vistos como um caractere normal

A cláusula SIMILAR TO

- **Exemplo 49:** Exibir o título e a descrição de todos os filmes, cuja descrição contenha, primeiramente, a palavra '**Emotional**' em qualquer parte do texto, seguido de '**Pioneer**' em qualquer posição depois da palavra 'Emotional'.

```
SELECT title,description FROM film
WHERE description
SIMILAR TO '%Emotional%Pioneer%';
-- Substituir o SIMILAR TO por LIKE tem o mesmo
-- efeito
-- database: dvdrental
```

Operadores Lógicos

- **Sintaxe**:

```
SELECT <lista_de_conlunas>
FROM <tabela>
WHERE <condição1> AND|OR <condição2>
AND|OR <condição3>
```

- **Exemplo 50:** Exibir código, nome, tipo e telefone dos clientes que sejam pessoa jurídica e que possuam telefone, ordenados pelo nome em ordem ascendente.

```
SELECT idCliente, nome, tipo, fone
FROM cliente WHERE tipo = 'J' AND
(email IS NOT NULL AND email <> '')
ORDER BY nome;
```

Operadores Lógicos

- **<u>Exemplo 51</u>:** Exibir código, nome, salário e idade de todos os funcionários homens com salário superior a R$ 1.500,00 e com idade entre 25 e 50 anos (inclusive).

```
SELECT idFuncionario, nome, salario,
date_part('year',age(dataNasc)) idade
FROM funcionario
WHERE sexo = 'M' AND salario > 1500 AND
date_part('year',age(dataNasc)) BETWEEN
25 AND 30;
```

Não utilize a coluna nomeada **idade** como expressão no **BETWEEN** (causa erro). Repetir a expressão.

Operadores Lógicos

- **<u>Exemplo 52</u>:** Exibir todos os produtos com tipo igual a 1, 2 ou 5 e quantidade em estoque maior ou igual a 10, ou que tenham preço de venda superior a R$ 50,00 e tipo diferente de 1, 4 e 6

```
SELECT *
FROM produto
WHERE (tipo IN (1,2,5) AND estoqueAtual
>= 10) OR
(venda > 50.0 AND tipo NOT IN (1,4,6));
```

Funções Agregadas

- **Definição**
  - São funções que computam um único resultado para várias linhas de entrada (valor sumarizado)
  - O resultado depende do objetivo da função
- **Considerações**
  - Produzem um resultado único
  - Não atuam sobre valores nulos
  - São frequentemente utilizadas com as cláusulas **GROUP BY** e **HAVING** de uma instrução **SELECT**
  - Pode aparece na lista de colunas de um **SELECT**
  - Não pode ser utilizado na cláusula **WHERE**
- **Sintaxe**

```
funçãoAgregada( [DISTINCT] expressão)
```

Funções Agregadas: count()

- **count()**
  - Computa o número de linhas de uma determinada coluna

Função count()	Conta Duplicatas?	Conta valores NULL?
`count(*)`	**Sim**	**Sim**
`count(DISTINCT column)`	**Não**	**Não**
`Count(ALL column)`	**Sim**	**Não**

- **Exemplo 53:** Exibir quantidade de pedidos realizados no ano 1998.

```
SELECT count(*) quant FROM pedido
WHERE date_part('year',dataFatura) = 1998;
```

Funções Agregadas: count()

- **Exemplo 54:** Apresentar o número de cidades (dos clientes) que a empresa atende

```
SELECT count(DISTINCT cidade)
FROM cliente;
```

idCliente	nome	cidade
1	Alex	João Pessoa
2	Cândido	Campina Grande
3	Luiz	João Pessoa
4	Damires	Cabedelo
5	Leandro	Campina Grande
6	Paulo	NULL

```
count(*) = 6
count(DISTINCT cidade) = 3
count(ALL cidade) = 5
```

Funções Agregadas: sum()

- **sum()**
  - Obtem o somatório dos valores de uma coluna
  - **Sintaxe**

```
sum( DISTINCT | ALL expressão)
```

  - O modificador DISTINCT instrui sum() a realizar a soma eliminando as duplicatas
  - O modificador ALL instrui o sum() a considerar duplicatas, mas os valores NULL são descartados
    - ✓ A função sum() utiliza **ALL** como modificador default
- **Exemplo 55:** Exibir o total arrecadado com multas de atraso no mês de setembro

```
SELECT sum(valor) FROM multa -- biblioteca
WHERE date_part('month',dataDevolucao)=9;
```

Funções Agregadas: sum()

- **Exemplo 56:** Exibir a soma de todos os salários dos funcionários dos setores "Compras e Vendas" (COV) e "Marketing" (MKT), supondo um aumento de 32%

```
SELECT sum(salario*1.32) previsao
FROM funcionario
WHERE idSetor IN ('COV', 'MKT');
```

IMPORTANTE:
Não utilizar funções agregadas como condição de um WHERE

```
SELECT nome FROM funcionario
WHERE salario > avg(salario)
```

Funções Agregadas: avg()

- **avg(** DISTINCT | ALL expressão**)**
  - Determina a média aritmética de um conjunto de valores
- **<u>Exemplo 57</u>:** Exibir a média salarial dos empregados da empresa

```
SELECT avg(salario)
FROM funcionario;
```

- **<u>Exemplo 58</u>:** Exibir a média de idade das funcionárias do setor de atendimento

```
SELECT
avg(date_part('year',age(datanasc))
FROM funcionario
WHERE sexo = 'F' AND setor = 'ATD‘;
```

Funções Agregadas: avg()

- **Exemplo 59:** Exibir a média de salário dos funcionários com salário entre R$ 400,00 e R$ 800,00 que não sejam casados nem viúvos.

```
SELECT avg(salario)
FROM funcionario
WHERE salario BETWEEN 400 AND 800
AND estCivil NOT IN ('C', 'V');
```

Funções Agregadas: max(),min()

- **max()**
  - Determina o maior valor de um conjunto de valores
- **min()**
  - Determina o menor valor de um conjunto de valores
- **Exemplo 60:** Exibir o valor do maior salário pago a um funcionário

```
SELECT max(salario)
FROM funcionario;
```

- **Exemplo 61:** Exibir a menor temperatura observada no ano de 2019

```
SELECT min(temperatura) FROM medição
WHERE date_part('year',dataMed) = 2019;
```

Agrupamento de Dados

- **A cláusula GROUP BY**
  - Frequentemente usada com funções agregadas
  - Agrupa o resultado por uma ou mais colunas
  - Organiza as linhas de resultado em grupos de acordo com os valores das expressões apresentadas
- **Sintaxe**

```
SELECT <lista_de_conlunas> FROM <tabela>
WHERE <condição>
GROUP BY <coluna(s)>
HAVING <condição>
```

  - A cláusula HAVING (opcional) seleciona os grupos de acordo com a expressão (só pode ser usado com GROUP BY)

Agrupamento de Dados

- **Exemplo 62:** Exibir a quantidade de funcionários do sexo masculino e feminino

```
SELECT sexo, count(*) quant
FROM funcionario
GROUP BY sexo;
```

- **Exemplo 63:** Exibir a quantidade de funcionários e o total de salários para cada código de setor da empresa

```
SELECT idSetor, count(*) numFunc,
sum(salario) totSal FROM funcionario
GROUP BY idSetor;
```

Modifique a instrução para aparecer apenas o nome do setor e totalizadores.

Agrupamento de Dados

- **Exemplo 64:** Exibir a quantidade de pedidos realizado em cada dia do mês de outubro

```
SELECT dataFatura, count(*) quant
FROM pedido WHERE
date_part('month',dataFatura) = 10
GROUP BY dataFatura;
```

- **Exemplo 65:** Exibir a quantidade, o total de custo e a média dos preços de custo e de venda dos produtos de cada tipo de produto

```
SELECT idTipo, COUNT(*) Quant, SUM(custo)
'Custo Tot', AVG(custo) 'Custo Med',
AVG(venda) 'Venda Med' FROM produto
GROUP BY idTipo;
```

Agrupamento de Dados

- **<u>Exemplo 66</u>:** Exibir o código do produto, nome e quantidade de unidades vendidas, dos 5 produtos mais vendidos

```
SELECT i.idProduto, p.nome,
    (SUM(i.quant) total
FROM itens i, produtos p
WHERE i.idProduto = p.idProduto
GROUP BY i.idProd, p.nome
ORDER BY total DESC
LIMIT 5;
```

Agrupamento de Dados

- **Exemplo 67:** Exibir os pedidos com mais de 4 itens de produtos

```
SELECT idPedido, COUNT(idProduto) Quant
FROM itens
GROUP BY idPedido HAVING COUNT(idProduto)>4;
```

- **Exemplo 68:** Exibir a quantidade de pedidos realizado em cada ano, para os anos que efetuaram mais de 200 pedidos

```
SELECT date_part('year',dataFatura),
COUNT(*) quant FROM pedido
GROUP BY date_part('year',dataFatura)
HAVING COUNT(*) > 200;
```

Agrupamento de Dados

- **Exemplo 69:** Exibir o valor total de cada pedido (Valor total = quantidade * preço – desconto), onde a soma seja maior do que R$ 1.000,00

```
SELECT idPedido, SUM((i.preco * i.quant)-
(i.preco*i.quant*i.desconto/100)) 'Total'
FROM pedido p, itens i
WHERE p.idPedido = i.idPedido
GROUP BY idPedido
HAVING SUM((i.preco*i.quant)-
(i.preco*i.quant*i.desconto/100)) > 1000.00
ORDER BY idPedido;
```

Agrupamento de Dados

- **<u>Exemplo 70</u>:** Exibir para cada função, a quantidade de homens e mulheres e o total de salários por sexo, para as funções onde o total de salários seja maior do que R$ 1.000,00

```
SELECT idFuncao, sexo, COUNT(sexo) Quant,
SUM(salario) 'Salario Total'
FROM funcionario
GROUP BY idFuncao, sexo
HAVING SUM(salario) > 1000.00
ORDER BY idFuncao, sexo;
```

Modifique a consulta para que apareça o **nome da função**, ao invés do código

Agrupamento de Dados

- **<u>Exemplo 71</u>:** Exibir a quantidade de itens e o valor total dos produtos de cada tipo (total em reais disponíveis em estoque de acordo com o preço de venda), para os tipos onde o valor total de venda seja superior a R$ 1.000,00, ordenados pelo tipo

```
SELECT idTipo, COUNT(*) Quant,
SUM(venda*quantEst) 'Valor Total'
FROM produto
GROUP BY idTipo
HAVING SUM(venda*quantEst) > 1000.00
ORDER BY idTipo;
```

Agrupamento de Dados

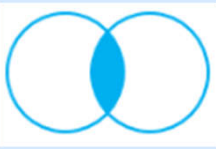
- **<u>Exemplo 72</u>:** Exibir a média de idade dos funcionários de cada sexo em cada setor da empresa. Exibir apenas os setores onde essa média de idade seja superior a 40 anos

```
SELECT idSetor, sexo,
AVG(date_part('year',age(dataNasc))) Idade
FROM funcionario
GROUP BY idSetor, sexo
HAVING
   AVG(date_part('year',age(dataNasc))) > 40;
```

Junção de Tabelas

- **INNER JOIN**:

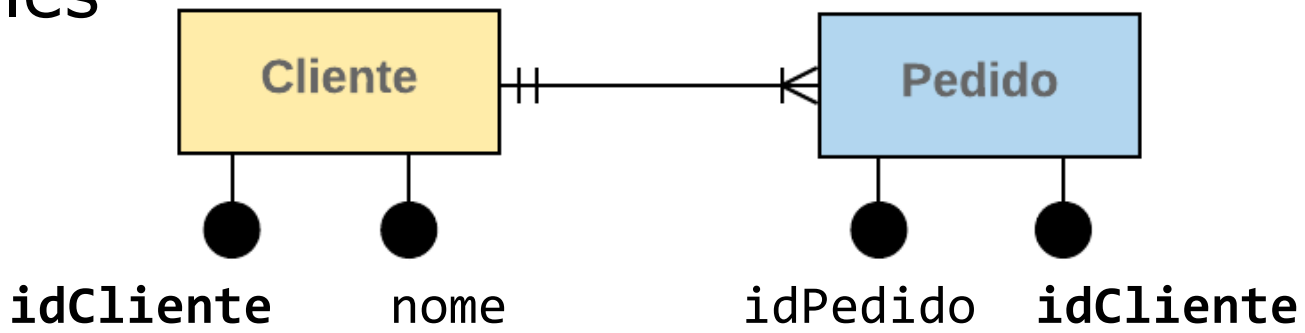
```
SELECT <lista_de_colunas>
FROM <tabela1> [alias1] [INNER] JOIN
     <tabela2> [alias2]
ON <condição de associação de chaves>
```



  - A junção entre duas tabelas é realizada usando uma coluna de conexão em comum existente em cada tabela.
  - O resultado da consulta contém somente linhas que correspondam ao critério estabelecido na condição de junção entre as mesmas (junção interior).
  - Tem o mesmo efeito que um SELECT/FROM/WHERE com ligação de chaves

Junção interna – INNER JOIN

- **Exemplo 73:** Exibir o código de cada pedido e o nome do cliente que realizou cada um deles



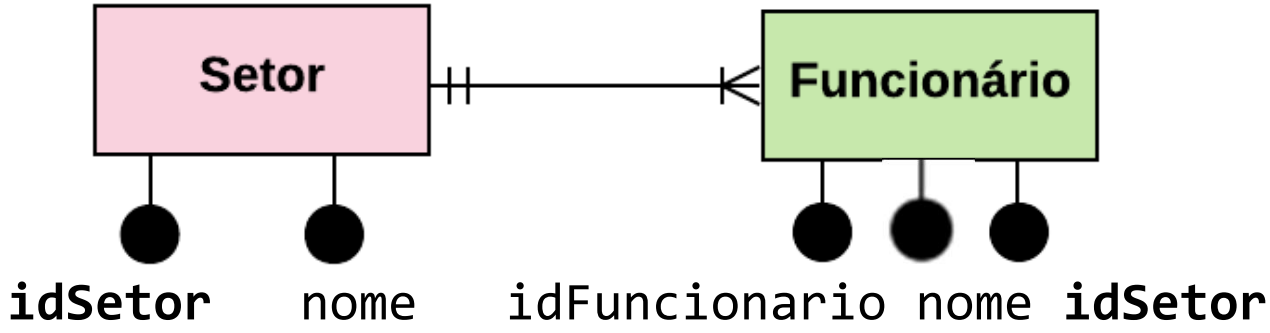
cliente.**idCliente** = pedido.**idCliente**

```
SELECT p.idPedido, c.nome FROM pedido p
INNER JOIN cliente c ON
p.idCliente=c.idCliente;

SELECT p.codPed, c.nome FROM pedido p,
cliente c WHERE p.idCliente = c.idCliente;
```

Junção interna – INNER JOIN

- **Exemplo 74:** Exibir, para cada funcionário, código e nome do funcionário, sigla e nome do setor onde trabalha, ordenados pela sigla do setor e nome do funcionário




```
SELECT f.idFuncionario, f.nome, s.idSetor,
s.nome FROM funcionario f INNER JOIN setor s
ON f.idSetor = s.idSetor
ORDER BY s.idSetor, f.nome;
```

Junção interna – INNER JOIN

- **Exemplo 75:** Exibir a quantidade, a média dos preços de venda e de custo dos produtos de cada tipo, mostrando o código e o nome do tipo dos produto.

```
SELECT t.idTipo, t.nome Tipo,
COUNT(p.idProduto) Quant, AVG(p.custo)
Media_Custo, AVG(p.venda) Media_venda
FROM produto p INNER JOIN tipo t ON
p.idTipo = t.idTipo
GROUP BY t.idTipo, t.nome;
```

Nota: a expressão count() também pode ser escrita com *. Neste exemplo, o GROUP BY deve ser com dois campos

Junção interna – INNER JOIN

- **Exemplo 76:** Exibir quantidade de pedidos constituídos por algum produto 'Bebida'.

```
SELECT COUNT( DISTINCT i.idPedido)
FROM produto p JOIN itens i
ON p.idProduto = i.idProduto
WHERE p.idTipo = 1; /*codigo do tipo*/
```

  - Um pedido pode ter mais de um item do mesmo tipo

- **Exemplo 77:** Exibir o valor total da folha de pagamento da empresa (salário + gratificação)

```
SELECT SUM(salario+gratific)
FROM funcionario f JOIN funcao u
ON f.idFuncao = u.idFuncao;
```

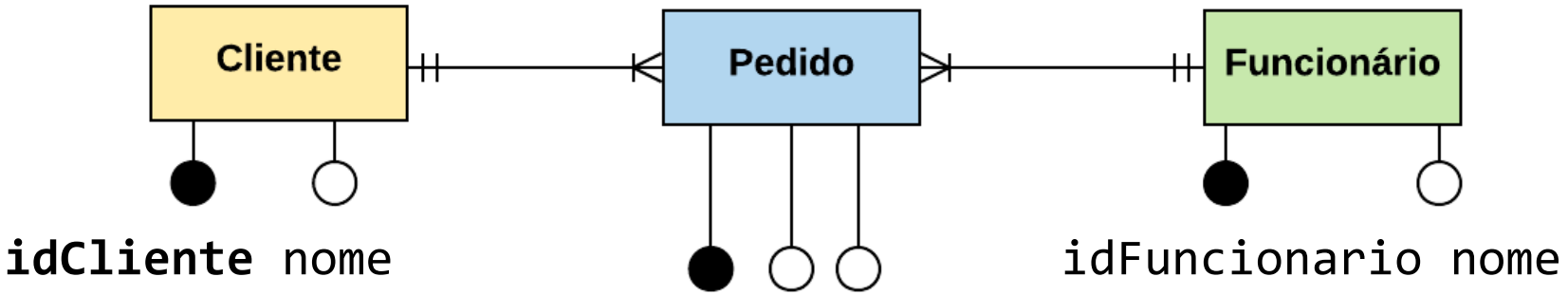
Junção interna – INNER JOIN

- **<u>Exemplo 78</u>:** Exibir o valor total (preço de venda X estoque atual) disponível em produtos, por tipo, exibindo o nome do tipo, ordenado pelo valor total em ordem descendente

```
SELECT t.nome, SUM(p.quantEst*p.venda)
Total
FROM tipo t INNER JOIN produto p ON
t.idTipo = p.idTipo
GROUP BY t.nome
ORDER BY Total DESC;
```

Junção interna – INNER JOIN

- **<u>Exemplo 79</u>:** Exibir a lista dos pedidos com o nome do cliente, do vendedor e a forma de envio de cada pedido




```
SELECT p.idPedido Pedido, c.nome Cliente, f.nome
Vendedor, p.via
FROM pedido p JOIN cliente c ON
p.idcliente=c.idCliente JOIN funcionario f ON
p.vendedor = f.idFuncionario ORDER BY
p.idPedido;
```

Junção interna – INNER JOIN

- **Exemplo 80:** Exibir o nome de cada funcionário, o nome da cidade onde cada um nasceu e o nome da cidade onde cada um reside, ordenados pelo nome do funcionário em ordem ascendente.

```
SELECT f.nome, c1.nome 'Cidade Natal',
c2.nome 'Cidade Residência'
FROM funcionario f JOIN cidade c1 ON
f.idNasceu = c1.idCidade JOIN cidade c2
ON f.idReside = c2.idCidade
ORDER BY f.nome;
```

Junção com Auto-relacionamento

- **INNER JOIN**:

```
SELECT <lista_de_colunas>
FROM <tabela1> [alias1] INNER JOIN
     <tabela2> [alias2]
ON <condição de associação de chaves>
```

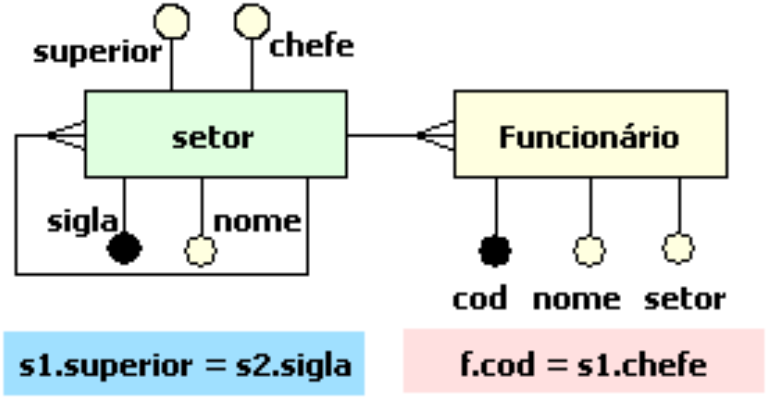
  - É uma junção de uma tabela **com ela mesma**, usando uma coluna de conexão.

Regra

  - Para que a tabela possa se relacionar com ela mesma, devem ser informados dois aliases diferentes para a mesma tabela.

Junção com Auto-relacionamento

- **<u>Exemplo 81</u>:** Exibir a lista dos setores da empresa, o setor o qual está subordinado, nome do chefe do setorda setor.




```
SELECT s1.nome, s2.nome 'SetorSup', f1.nome
Chefe, f2.nome 'Chefe Super.' FROM setor s1
JOIN setor s2 ON s1.idSetor=s2.idSuperior
JOIN funcionario f1 ON f1.idFuncionario=
s1.chefe JOIN funcionario f2 ON
f2.idFuncionario = s2.chefe;
```

Junção Exterior

- **Definição**
  - Junção externa é uma seleção que não requer que os registros de uma tabela possuam registros equivalentes na outra
- **Tipos de Junções.**
  - LEFT OUTER JOIN junção à esquerda
  - RIGHT OUTER JOIN: junção à direita
  - FULL OUTER JOIN: junção completa
  - CROSS JOIN: junção cruzada

Junção Exterior

- **Esquema para exemplificação**

Nome_Agencia	*Num_empres*	*Total*
Tambaú	*E170*	*3000*
Torre	*E230*	*4000*
Centro	*E260*	*1700*

empréstimo

Nome_cliente	*Num_empres*
Jones	*E170*
Smith	*E230*
Carol	*E155*

devedor

Junção Exterior – LEFT JOIN

- **LEFT JOIN**

```
SELECT <lista_de_colunas>
FROM <tabela1> [alias1] LEFT [OUTER] JOIN
     <tabela2> [alias2]
ON <condição de associação de chaves>
```

  - Dadas duas tabelas A e B, o resultado de **left join** produz todas as linhas da tabela daesquerda (A), mesmo que a condição de junção não seja compatível com nenhuma linha da tabela à direita (B)
  - Retorna **todos os valores de um INNER JOIN**, mais todos os valores da tabela à esquerda que não correspondem com a tabela à direita, incluindo linhas com campo **NULL**

Junção Exterior – LEFT JOIN

- **<u>Exemplo 82 (INNER JOIN)</u>:** Exibir todos os campos da junção da tabela **empréstimo** com **devedor**

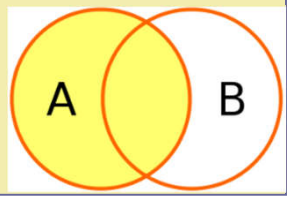
```
SELECT *
FROM emprestimo e INNER JOIN devedor d
ON d.num_empres = e.num_empres;
```

Agencia	*Num_empres*	*Total*	*cliente*	*Num_empres*
Tambaú	*E170*	*3000*	*Jones*	*E170*
Torre	*E230*	*4000*	*Smith*	*E230*

Junção Exterior – LEFT JOIN

- **<u>Exemplo 83</u>:** Exibir todos os campos da junção externa à esquerda da tabela **empréstimo** com **devedor**

```
SELECT *
FROM emprestimo e LEFT OUTER JOIN
     devedor d
ON d.num_empres = e.num_empres;
```



Agencia	*Num_empres*	*Total*	*cliente*	*Num_empres*
Tambaú	*E170*	*3000*	*Jones*	*E170*
Torre	*E230*	*4000*	*Smith*	*E230*
Centro	*E260*	*1700*	*NULL*	*NULL*

Junção Exterior – LEFT JOIN

- **<u>Exemplo 84</u>:** Exibir a lista das funções com os respectivos nomes dos funcionários que exercem cada uma delas, mesmo que exista função que não tenha nenhum funcionário, ordenados pelo nome da função e pelo nome do funcionário em ordem ascendente

```
SELECT fc.nome Funcao, f.nome Funcionario
FROM funcao fc LEFT JOIN funcionario f
ON fc.idFuncao=f.idFuncao
ORDER BY fc.nome, f.nome;
```

Junção Exterior – RIGHT JOIN

- **RIGHT JOIN**

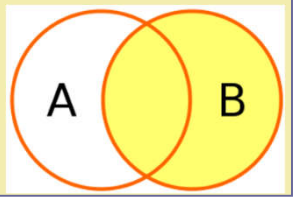
```
SELECT <lista_de_colunas>
FROM <tabela1> [alias1] RIGHT [OUTER] JOIN
     <tabela2> [alias2]
ON <condição de associação de chaves>
```

  - Dadas duas tabelas A e B, o resultado de **right join** produz todas as linhas da tabela da direita (B), mesmo que a condição de junção não seja compatível com nenhuma linha da tabela à direita (A)
  - Similar ao **LEFT JOIN**, porém, com a inversão do lado da tabela que terá todas as suas linhas incluídas no resultado

Junção Exterior – RIGHT JOIN

- **<u>Exemplo 85</u>:** Exibir todos os campos da junção externa da tabela **empréstimo** com **devedor**

```
SELECT *
FROM emprestimo e RIGHT JOIN
     devedor d
ON d.num_empres = e.num_empres;
```



Agencia	Num_empres	Total	cliente	Num_empres
Tambaú	E170	3000	Jones	E170
Torre	E230	4000	Smith	E230
NULL	E155	NULL	Carol	E155

Junção Exterior – RIGHT JOIN

- **Exemplo 86:** Exibir a lista dos tipos de produtos com os respectivos nomes dos produtos existentes para cada tipo, mesmo que exista tipo que não tenha produto, ordenados pelo nome do tipo e pelo nome do produto em ordem ascendente

```
SELECT t.nome Tipo, p.nome Produto
FROM produto p RIGHT JOIN tipo t
ON t.idTipo = p.idTipo
ORDER BY t.nome, p.nome;
```

Junção Exterior

- **FULL OUTER JOIN**

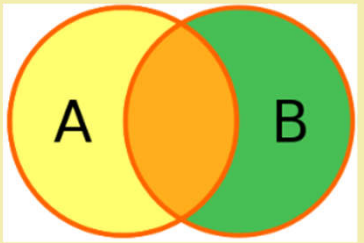
```
SELECT <lista_de_colunas>
FROM <tabela1> [alias1] FULL [OUTER] JOIN
     <tabela2> [alias2]
ON <condição de associação de chaves>
```

  - Dadas duas tabelas A e B, o resultado de **full join** é a combinação de todas as linhas da tabela da esquerda (A) e da direita (B), mesmo que a condição de junção não seja compatível entre as tabelas
  - Unificação do **LEFT JOIN** com **RIGHT JOIN**
  - Após a realização da junção interna, as tuplas da relação do lado esquerdo que não correspondem a nenhuma das tuplas do lado direito são preenchidas com NULL, e vice-versa

Junção Exterior – FULL JOIN

- **<u>Exemplo 87</u>:** Exibir todos os campos da junção externa da tabela **empréstimo** com **devedor**

```
SELECT * FROM emprestimo e
FULL OUTER JOIN devedor d
ON d.num_empres = e.num_empres;
```



Agencia	*Num_empres*	*Total*	*cliente*	*Num_empres*
Tambaú	*E170*	*3000*	*Jones*	*E170*
Torre	*E230*	*4000*	*Smith*	*E230*
Centro	*E260*	*1700*	*NULL*	*NULL*
NULL	*NULL*	*NULL*	*Carol*	*E155*

Junção Exterior – FULL JOIN

- **Exemplo 88:** Exibir a lista das funções com os respectivos nomes dos funcionários que exercem cada uma delas, mesmo que exista função sem funcionário e funcionário sem função

```
SELECT fc.nome Funcao, f.nome Funcionario
FROM funcao fc FULL JOIN funcionario f
ON fc.idFuncao = f.idFuncao
ORDER BY fu.nome, f.nome;
```

Junção Cruzada

- **CROSS JOIN**

```
SELECT <lista_de_colunas>
FROM <tabela1> [alias1] CROSS JOIN
     <tabela2> [alias2]
```

  - **Junção irrestrita**: gera um resultado formado por todas as combinações possíveis de uma linha da primeira tabela com uma linha da segunda
  - Não existe condição de junção
  - O resultado é chamado de produto cartesiano
  - Se a tabela **funcionario** tem 8 linhas e a tabela **setor** tem 4 linhas, teremos como resultado 8*4=32 linhas
  - O cross join raramente é usado (aplicar junção interior ou exterior é mais frequente)

Junção Cruzada

- **Visualização**

empregado

	nome character varying (30)	idSetor integer
1	Alex	10
2	Nathan	11
3	Luiz	[null]

setor

	idSetor integer	nome character varying (30)
1	10	TI
2	11	Marketing

```
SELECT *
FROM empregado CROSS JOIN setor;
```

	nome character varying (30)	idSetor integer	idSetor integer	nome character varying (30)
1	Alex	10	10	TI
2	Alex	10	11	Marketing
3	Nathan	11	10	TI
4	Nathan	11	11	Marketing
5	Luiz	[null]	10	TI
6	Luiz	[null]	11	Marketing

Subconsultas (subquery)

- **Definição**
  - Uma **subquery** é uma consulta SELECT **aninhada** (interna) a outra instrução SQL (principal)
  - Subconsultas também podem ser aplicadas a instruções INSERT
- **Propriedades**
  - Uma **subquery** deve ser delimitada entre parênteses e é avaliada apenas uma vez
  - O resultado de uma **subquery** é o conjunto de linhas (registros) a ser retornado à **query principal**
  - A **query** mais externa depende da **subquery** para ser avaliada
    - ✓ Em um comando SELECT, a resolução da **subquery** tem maior precedência. Uma vez resolvida, seu resultado é disponibilizado à **query** principal

Subconsultas (subquery)

- **Sintaxe**
  - A sintaxe é bastante dinâmica, pois podemos ter subconsultas como alvo para uma tabela na cláusula **FROM** ou valor para uma condição no **WHERE**
- **Retorno de uma subquery**
  1. Uma **subquery** de valor único retorna apenas um valor e pode ser usada no lugar de uma expressão utilizando operadores (=,!=,<=,>=,<,>)

     subquery de 1 coluna -> 1 valor

     ```
     ...WHERE col=(SELECT B …) /*Verdade se col=B*/
     ```

  2. Uma **subquery** de coluna única retorna múltiplas linhas e pode ser usada apenas em um WHERE utilizando cláusulas especiais

     1 coluna -> muitos valores

Subconsultas (subquery)

- **Exemplo 89:** exibir código, nome e preço de venda dos 10% primeiros produtos cadastrados

```
SELECT idProduto, nome, venda
FROM produto
LIMIT (SELECT COUNT(*)*0.1 FROM produto);
```

Subconsultas (subquery)

- **Exemplo 90:** Exibir o nome e o valor da gratificação das funções que têm a menor gratificação paga na empresa

```
SELECT nome, gratific FROM funcao
WHERE gratific = (SELECT MIN(gratific)
FROM funcao WHERE gratific <> 0 );
```

- **Exemplo 91:** Exibir o nome, salário e a porcentagem do total que o salário do funcionário representa.

```
SELECT nome Funcionario, salario,
100*(salario/(SELECT SUM(salario) FROM
funcionario) ) as Porcentagem
FROM funcionario;
```

Subconsultas (subquery)

- **Exemplo 92:** Exibir o código, o nome e a quantidade em estoque do produto que tem a maior quantidade em estoque da empresa

```
SELECT idProduto, nome, quantest
FROM produto
WHERE quantest = (SELECT MAX(quantest)
FROM produto );
```

Nota: função agregadas não podem ser usadas em cláusulas WHERE, logo, é incorreta a instrução abaixo:

```
SELECT idProduto, nome, quantest
FROM produto
WHERE quantest = MAX(quantest);
```

Subconsultas (subquery)

Uma **subquery** de seleção de uma coluna pode retornar uma lista de valores. A lista de valores pode ser usada em comparações com o uso do operador **IN**

- **Exemplo 93(a):** Exibir o código, nome do cliente e país de todos os clientes estrangeiros

```
SELECT c.idCliente, c.nome, p.nome 'Pais'
FROM cliente c JOIN cidade ci ON
c.idCidade = ci.idCidade JOIN pais p ON
ci.idPais = p.idPais
WHERE p.idPais IN (SELECT idPais FROM
pais WHERE idPais <> 'BRA' );
```

Subconsultas (subquery)

Neste mesmo exemplo, é possível apresentar uma consulta sem a necessidade de definir subconsulta

- **Exemplo 93(b):** Exibir o código, nome do cliente e país de todos os clientes estrangeiros

```
SELECT c.idCliente, c.nome, p.nome 'Pais'
FROM cliente c JOIN cidade ci ON
c.idCidade = ci.idCidade JOIN pais p ON
ci.idPais = p.idPais
WHERE p.idPais <> 'BRA';
```

Subconsultas (subquery)

- **Operadores relevantes**
  - > **ALL** : maior que todos
  - < **ALL**: menor que todos
  - <> **ALL**: diferente de todos (igual a **NOT IN**)
  - = **ANY**: igual a algum dos elementos da lista (o mesmo que **IN**)
  - > **ANY**: maior que algum dos elementos da lista
  - < **ANY**: menor que algum dos elementos da lista
  - <> **ANY**: diferente de algum dos elementos da lista (falso se igual a todos)
  - **Não é permitido "= ALL"**
  - **EXISTS**: teste de existência

Pode ser usado
>= ou **<=**

Subconsultas (subquery)

Cláusula **ANY** com operador relacional:

```
 … WHERE col > ANY (SELECT B FROM table)
/* Verdadeiro se existe col maior que algum B*/
```

- **Exemplo 94:** Exibir o nome, o tipo e o preço de venda dos produtos que não sejam dos tipos 3, 4 ou 5 e que tenham preço de venda maior que pelo menos o preço de um desses produtos.

```
SELECT nome, idTipo, venda FROM produto
WHERE idTipo NOT IN (3,4,5) AND
venda > ANY (SELECT venda FROM produto
WHERE idTipo IN (3,4,5) );
```

Subconsultas (subquery)

Cláusula **ALL** com operador relacional:

```
 … WHERE col > ALL (SELECT B FROM table)
/*Verdadeiro se existe col maior que todos de B*/
```

- **Exemplo 95:** Exibir o nome, o tipo e o preço de venda dos produtos que não sejam dos tipos 3, 4 ou 5 e que tenham preço de venda maior que o preço de todos os produtos dos tipos 3, 4 ou 5.

```
SELECT nome, idTipo, venda
FROM produto
WHERE idTipo NOT IN (3,4,5) AND
venda > ALL (SELECT venda FROM produto
WHERE idTipo IN (3,4,5) );
```

Subconsultas (subquery)

Teste de existência: A condição é **verdadeira** se a subquery retorna alguma linha. Caso nenhuma linha seja retornada, retorna **falso**

```
… WHERE EXISTS (SELECT B FROM table)
/* Verdadeiro se existe existe ao menos um valor
para B com numero de linhas <> 0*/
```

- **Exemplo 96:** Exibir o nome e o salário dos funcionários que têm o salário superior a R$ 2.000,00.

```
SELECT f1.nome, f1.salario
FROM funcionario f1
WHERE EXISTS( SELECT * FROM funcionario
f2 WHERE f2.salario > 2000 AND f1.codigo
= f2.codigo );
```

Subconsultas (subquery)

- **Exemplo 97:** Exibir o código dos clientes que fizeram pedido em agosto de 2017 e em junho de 2018.

```
SELECT DISTINCT a.idCliente
FROM pedido a
WHERE date_part('month',a.dataFatura)=08
AND date_part('year',a.dataFatura)=2017
AND
EXISTS
(SELECT * FROM pedido b WHERE
date_part('month',a.dataFatura)=06 and
date_part('year',a.dataFatura)=2018 AND
a.idCliente = b.idCliente);
```

Subconsultas (subquery)

- **Exemplo 98:** Exibir o nome de todos os clientes que tenham tanto conta quanto empréstimo no banco

```
SELECT nome_cliente FROM cliente d
WHERE EXISTS
(SELECT * FROM emprestico e
 WHERE d.codCliente = e.codCliente);
```

Subconsultas (subquery)

- **Subconsulta como alvo de um FROM**

```
SELECT <lista_de_colunas>
FROM (subconsulta) alias1
WHERE <condição>
```

  - As colunas e linhas do resultado de uma subconsulta são interpretadas como uma tabela

- **Exemplo 99:** Exibir os 5 primeiros números do pedido, código do cliente e quantidade de produtos pertencentes ao mesmo.

```
SELECT p.idPedido, p.idCliente, tb.quantid
FROM ( SELECT idPedido, COUNT(*) quantid FROM
itens GROUP BY idPedido) tb INNER JOIN pedido
p ON tb.idPedido = p.idPedido LIMIT 5;
```

SELECT - INTO

- **Consulta com criação de tabela**

```
SELECT <lista_de_colunas> INTO
<nova_tabela>
FROM <tabela(s)> [alias1]
WHERE <condição>
```

- **<u>Exemplo 100</u>:** Por meio de um SELECT, criar uma nova tabela que contenha o código do pedido, nome e telefone do cliente que fez cada um deles.

```
SELECT p.idPedido Pedido, c.nome Cliente,
c.fone INTO tabPedidos FROM pedido p
INNER JOIN cliente c
ON p.idCliente = c.idCliente;
```

União de Conjuntos

- **Cláusula UNION**
  - Utilizada para combinar o resultado de duas instruções SELECT
    - ✓ Baseada na operação matemática da **união de conjuntos**

```
SELECT <lista_de_colunas> FROM <tabela1>
UNION [ALL]
SELECT <lista_de_colunas> FROM <tabela2>
```

  - Os dois comandos podem até trazer dados de tabelas diferentes, desde que com o mesmo número de colunas e tipos de dados compatíveis para cada coluna correspondente
  - Na união de dois conjuntos, os **elementos repetidos são eliminados.**
    - ✓ Use **ALL** para manter as duplicatas

União de Conjuntos

- **Cláusula UNION**
  - Pode ser utilizada para combinar mais de um SELECT

```
<query1> UNION <query2> UNION <query3>
```

  - **Não é obrigatório** que as colunas de união das tabelas envolvidas tenham o mesmo nome, porém, a coluna resultante leva o nome da coluna definida na **query1**

```
SELECT a FROM tab1 UNION
SELECT b FROM tab2
```

  - A cláusula **ORDER BY** pode ser aplicada para ordenar o resultado como **última** instrução após o **UNION**

Interseção de Conjuntos

- **Cláusula INTERSECT**
  - Retorna todas as linhas que estão ambas em **query1** e **query2**
    - ✓ Baseada na operação matemática da **interseção de conjuntos**

```
<query1> INTERSECT [ALL] <query2>
[INTERSECT <query3>]
```

  - A interseção pode ser realizada com mais de duas *queries*
  - Na **interseção** de dois conjuntos, os **elementos repetidos são eliminados** (como se utilizasse a cláusula DISTINCT)
    - ✓ Utilize **ALL** para manter as duplicatas

Diferença de Conjuntos

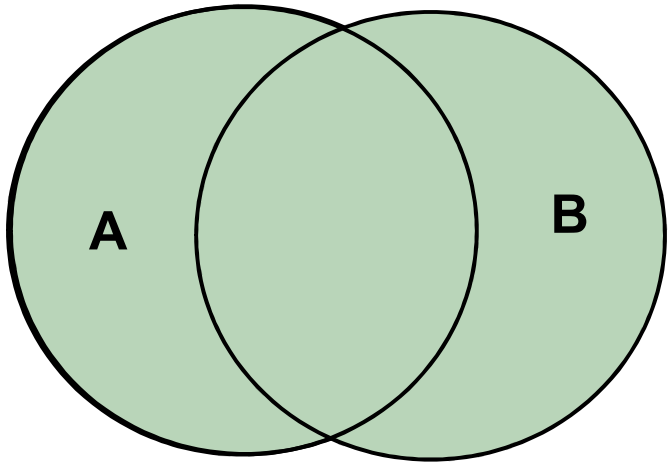
- **Cláusula EXCEPT**
  - Retorna todas as linhas que estão no resultado de **query1** mas que não se encontram em **query2**
    - ✓ Baseada na operação matemática da **diferença de conjuntos**

```
<query1> EXCEPT [ALL] <query2>
[EXCEPT <query3>]
```

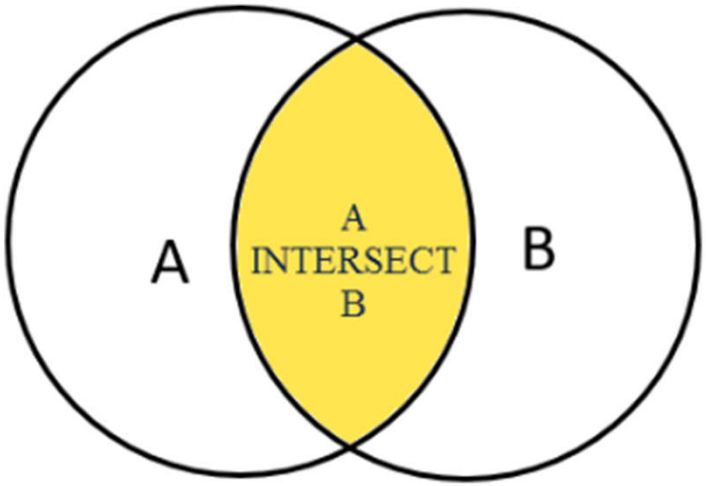
  - A diferença pode ser realizada com mais de duas *queries*
  - Na **diferença** de dois conjuntos, os **elementos repetidos são eliminados**
    - ✓ Utilize **ALL** para manter as duplicatas

UNION-INTERCEPT-EXCEPT

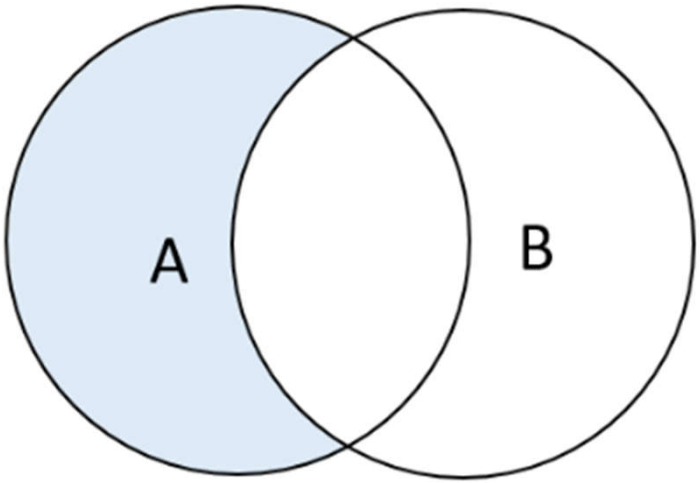
- Diagrama de Venn



União



Interseção



Diferença

Operações com Conjuntos

- Considere os campos e povoamento das tabelas **autor** e **editora** ilustrados a seguir:

```
projeto=# select * from autor;
   nome    |     cidade      | estado
-----------+-----------------+--------
 Joao      | Rio de Janeiro  | RJ
 Ana       | Campina Grande  | PB
 Carlos    | Joao Pessoa     | PB
 Sergio    | Sao Paulo       | SP
 Julia     | Aracaju         | SE
 Sandro    | Curitiba        | PR
 Cristiane | Manaus          | AM
 Marcos    | Recife          | PE
(8 registros)
```

```
projeto=# select * from editora;
     cidade      | estado
-----------------+--------
 Rio de Janeiro  | RJ
 Porto Alegre    | RS
 Sao Paulo       | SP
 Recife          | PE
 Salvador        | BA
 Boston          | MA
 Miami           | FL
(7 registros)
```

Operações com Conjuntos

- **Exemplo 101:** Exiba a lista de todas as cidades e estados onde existem autores ou editoras distribuídos pelo país, ordenado pelo estado.

```
SELECT cidade, estado FROM autor
UNION
SELECT cidade, estado FROM editora
ORDER BY estado;
/* ORDER BY apenas no último Select */
```

Nota: se existirem **cidades** de mesmo nome em estados diferentes, o UNION identifica como resultados distintos

Operações com Conjuntos

- **<u>Exemplo 102</u>:** Exiba a lista de todas as cidades dos autores onde não há editora.

```
SELECT cidade FROM autor
EXCEPT
SELECT cidade FROM editora;
```

```
    cidade
---------------
 Curitiba
 Campina Grande
 Joao Pessoa
 Manaus
 Aracaju
(5 registros)
```

INVERSÃO:
Editora > autor

```
    cidade
--------------
 Salvador
 Porto Alegre
 Boston
 Miami
(4 registros)
```

Operações com Conjuntos

- **<u>Exemplo 103</u>:** Exiba a lista de todas as cidades em que se encontram autores e editoras

```
SELECT cidade FROM autor
INTERSECT
SELECT cidade FROM editora;
```

```
     cidade
----------------
 Recife
 Sao Paulo
 Rio de Janeiro
(3 registros)
```

Tabelas Temporárias

- **Introdução**
  - Tabelas de curta duração, existente apenas durante a duração de uma sessão de banco de dados
  - O PostgreSQL descarta as tabelas temporárias automaticamente ao término de uma sessão ou transação
  - Uma tabela temporária pode ter o mesmo nome de uma tabela permanente, embora não seja recomendado. Porém, o acesso à tabela física só será possível quando a temporária for removida
- **Sintaxe**:

```
CREATE TEMPORARY TABLE (
...
);
```

Tabelas Temporárias

- **<u>Exemplo 104</u>:** Criar uma tabela temporária com o nome **aluno**, inserir dados e realizar consulta para listar todos os alunos.

```
CREATE TEMP TABLE aluno (
  idAluno integer PRIMARY KEY,
  name varchar NOT NULL
);
INSERT INTO aluno VALUES (1,'Alex’);
INSERT INTO aluno VALUES (2,‘Damires’);
INSERT INTO aluno VALUES (3,‘Petrônio’);
SELECT * FROM aluno;
```

Nota: utilize apenas quando necessário, uma vez que requer gravação em disco (afeta performance)

Exercícios

- Resolver a lista de Exercícios 1
  - SQL Básico e Avançado - Biblioteca
- Resolver a lista de Exercícios 2
  - SQL Básico e Avançado – Controle de Pedidos
- Resolver a lista de Exercícios 3
  - SQL Avançado – O esquema MyMail
- Resolver a lista de Exercícios 4
  - SQL Avançado (subconsultas)-Controle de Pedidos
- Resolver a lista de Exercícios 5
  - SQL Básico e Avançado - dvdrental

Referências Bibliográficas

- PostgreSQL 11.1 Documentation. Disponivel em https://www.postgresql.org/docs/11/index.html
- PostgreSQL Tutorial. Disponível em https://www.tutorialspoint.com/postgresql/index.htm
- A step-by-step guide to PostgreSQL Temporary Table. Disponível em http://www.postgresqltutorial.com/postgresql-temporary-table/
- Practical Exercises with subqueries. https://www.w3resource.com/sql-exercises/sql-subqueries-exercises.php https://www.tutorialspoint.com/sql_certificate/subqueries_to_solve_queries_questions.htm