

# REVISÃO



# SQL – DDL

*(Data Language Definition -  
linguagem de definição de dados)*



# LINGUAGEM SQL

- SQL (*Structured Query Language*) é uma linguagem para criação e manipulação de bancos de dados relacionais.
- Tem como base a álgebra relacional e cálculo relacional.



# LINGUAGEM SQL

- **DDL** (*Data Definition Language*): linguagem de definição de dados
  - Define esquemas, tabelas, chaves primárias e estrangeiras, exclusão de esquemas, tabelas e colunas, alteração de tabelas.
  - Diz respeito à **estrutura** das tabelas e esquema do BD.
- **DML** (*Data Manipulation Language*): linguagem de manipulação de dados.
  - Consulta, inserção dos dados do BD, exclusão de dados, alteração de dados. Diz respeito aos **dados** das tabelas do BD – CRUD.
- **DCL** (*Data Control Language*): linguagem de controle de dados.
  - Define permissões.

# LINGUAGEM SQL

- SQL = DDL + DML + DCL
- Principais comandos:
  - DDL (Linguagem de definição de dados):
    - CREATE, DROP, ALTER
  - DML (Linguagem de manipulação de dados):
    - SELECT, INSERT, UPDATE, DELETE
- Conceitos:
  - Table = Relação (tabela)
  - Row = tupla (linha)
  - Column = atributo (coluna)

# DDL: PROPRIEDADES

- A DDL permite não só a especificação de um conjunto de relações (tabelas), como também informações acerca de cada uma das relações, incluindo:
  - O esquema de cada relação (estrutura);
  - O domínio dos valores associados a cada atributo (int, float, varchar, etc);
  - As regras de integridade (**chave, entidade, referencial**);
  - Informações acerca de cada uma das relações:
    - O conjunto de índices para manutenção de cada relação;
    - Informações sobre segurança e autoridade sobre cada relação;
    - A estrutura de armazenamento físico de cada relação no disco.

# DDL: CREATE DATABASE/ESQUEMA

- Antes de criar qualquer tabela é necessário criar um esquema (ou base de dados!!!)
- Em SQL, uma base de dados (ou esquema) é identificada através de um **nome**. Os elementos do esquema incluem tabelas, restrições, etc.

- **Sintaxe:**

```
CREATE DATABASE <nome>
```

- Exemplos:
  - CREATE SCHEMA Empresa;
  - CREATE SCHEMA Universidade;
  - CREATE DATABASE Hospital;

# DDL: TIPOS DE DOMÍNIOS

- Numéricos:
  - **INTEGER** : é um inteiro, abreviação da palavra, **integer** (em inglês).
  - **NUMERIC (p, d)**: é um numero de ponto fixo cuja precisão é definida pelo usuário. O número consiste de  $p$  dígitos (mais o sinal), sendo que  $d$  dos  $p$  dígitos estão à direita do ponto decimal.
    - Ex.: `numeric(6,2)`: 3000,00
  - **SERIAL**: números inteiros auto incrementáveis.
- Character/String:
  - **CHAR(n)**: é uma cadeia de caracteres de tamanho fixo, com o tamanho  $n$  definido pelo DBA. Abreviação de **character** (em inglês).
  - **VARCHAR(N)**: é uma cadeia de caracteres de tamanho variável, com o tamanho  $n$  máximo definido pelo DBA. Abreviação de **character varying** (em inglês).



# DDL: TIPOS DE DOMÍNIOS

- Booleano:
  - **BOOLEAN:** assume dois valores, falso e verdadeiro (true/ false).
- Data/Tempo:
  - **DATE:** tipo de dado que contém um ano (com 4 dígitos) mês e dia do mês sendo o formato “**aaaa/mm/dd**” o padrão do MySQL Workbench.
    - **Year (date)** – retorna o ano de uma data
    - **Month (date)** – retorna o mês de um data
    - **Day (date)** – retorna o dia de uma data
  - **TIME:** representa horário, com o seguinte formato: horas, minutos e segundos (00:00:00).
  - OBS: Alguns SGBDs oferecem o domínio **TIMESTAMP** que contém a data (ano, mês e dia) e horário (horas, minutos, segundos e milissegundos).

# DDL: TIPOS DE DOMÍNIOS

## OBSERVAÇÕES

- Uma **chave estrangeira** deve possuir o **mesmo tipo de domínio** da **chave primária** correspondente.
- O valor *nulo* é um membro de todos os domínios, isto é, **qualquer atributo pode assumir o valor NULL, exceto o atributo chave primária (restrição integridade de entidade)**;
- A SQL permite que a declaração de domínio de qualquer atributo inclua a especificação de **not null**, proibindo, assim, a inserção de um valor nulo para esse tipo de atributo (obrigatório na PK).

# DDL: CREATE TABLE, DROP TABLE

- **CREATE TABLE:**

- define a estrutura de uma tabela, suas restrições de integridade e cria a tabela vazia

- Sintaxe: `CREATE TABLE <nome_tabela> (...);`

- Ex.: CREATE TABLE empregado (atributo1 tipo, atributo 2, tipo ...);

- **DROP TABLE:**

- Remove todos os dados e a própria relação (estando vazia ou não).

- Sintaxe: `DROP TABLE <nome_tabela>;`

- Ex.: DROP TABLE empregado;

# DDL: CREATE TABLE – SINTAXE

## Exemplo

```
CREATE DATABASE EMPRESA;
```

```
CREATE TABLE DEPARTAMENTO (
```

```
    CodDep SERIAL NOT NULL,
```

```
    NomeDep VARCHAR(30),
```

```
    PRIMARY KEY (CodDep)
```

```
);
```

```
CREATE TABLE FUNCIONARIO (
```

```
    Matricula INTEGER NOT NULL,
```

```
    Nome VARCHAR(30) NOT NULL,
```

```
    Salario NUMERIC(8,2), Cargo VARCHAR(15) DEFAULT 'Analista',
```

```
    Estado CHAR(2),
```

```
    Idade INTEGER, CodDepto INT,
```

```
    PRIMARY KEY (Matricula),
```

```
    FOREIGN KEY (CodDepto) references DEPARTAMENTO (CodDep) ON  
    DELETE NO ACTION ON UPDATE NO ACTION
```

```
);
```

# DDL: CREATE TABLE - SINTAXE

- Criação de um campo que gere **códigos automáticos** não é padrão SQL, mas caso seja necessário, basta colocar o tipo **SERIAL** na criação do campo.
- Muito utilizado em relações que possuem **IDs**.

- Exemplo:

```
CREATE TABLE cidade (  
    id_cidade SERIAL NOT NULL,  
    nome_cidade VARCHAR(100) NOT NULL,  
    PRIMARY KEY (id_cidade)  
);
```

# DDL: ALTER TABLE

A SQL oferece instruções para definição do esquema da base de dados:

- **ALTER TABLE:**
- Usado para alterar o esquema da relação.
- permite modificar a definição de uma tabela
  - Operações de inserção, alteração e exclusão → atenção aos atributos e restrições de integridade
    - atributos chave não podem ser removidos;
- Sintaxe: `ALTER TABLE <nome da tabela> ;`

# DDL: ALTER TABLE

1. Sintaxe básica para inclusão de uma coluna:

**ALTER TABLE** nome\_tabela **ADD COLUMN** nome\_coluna tipo\_atributo;

Ex.: ALTER TABLE funcionario ADD COLUMN identidade varchar(10);

2. Sintaxe básica para exclusão de uma coluna:

**ALTER TABLE** nome\_tabela **DROP** nome\_coluna;

Ex.: ALTER TABLE empregado DROP estado;

3. Sintaxe básica para alteração do nome de uma coluna:

**ALTER TABLE** nome\_tabela **RENAME COLUMN** nome\_coluna\_atual TO  
novo\_nome\_coluna;

Ex.: ALTER TABLE empregado RENAME COLUMN Sexo TO Genero;

# DDL: ALTER TABLE

- **Observe-se que:**
  - A instrução **adiciona uma nova coluna com o valor vazio** para todas as linhas, isto é, **sem nenhum valor** armazenado.
  - O mesmo acontece quando há a criação de uma tabela...
    - A princípio ela não está “povoada” com dados, está vazia, ausente de valores (em outras palavras: não há linhas/ tuplas na tabela).
  - Os valores para as diversas linhas devem ser adicionadas através de instruções da DML (INSERT INTO...), que será assunto da próxima aula.



# DDL: Atributos, Primary Key, Foreign Key

- **Chave primária:**
  - PRIMARY KEY (PK)
- **Restrições de atributos (PostgreSQL):**
  - NOT NULL (não nulo) - NN
  - DEFAULT <value>
- **Chave estrangeira:**
  - FOREIGN KEY (FK)
  - **Cláusulas:** ON DELETE e ON UPDATE (CASCADE, SET NULL, SET DEFAULT, RESTRICT, NO ACTION)

# DDL: DROP DATABASE

- Para excluir uma base de dados (ou um esquema) existente, é necessário usar o comando DROP.

- **Sintaxe:** `DROP <nome_esquema>;`

- Exemplos:
  - DROP Empresa;
  - DROP Universidade;
  - DROP Hospital;

Cuidado ao remover uma base de dados. TODAS as tabelas, relacionamentos e dados contidos na base serão permanentemente removidos!!!

# SQL – DML

*(Data Manipulation Definition -  
linguagem de manipulação de  
dados)*

Parte 1



# DML - DEFINIÇÃO

- A Linguagem DML (*Data Manipulation Language*) é composta por 4 operações de manipulação de dados que estão representadas abaixo:
  - Inserir dados – INSERT
  - Excluir dados – DELETE
  - Atualizar dados – UPDATE
  - Recuperar dados (Consultar) – SELECT

# INSERT INTO (inserção)

- O comando **INSERT** insere dados em uma relação (tabela).
- **Sintaxe:**

```
INSERT INTO nome_tabela  
( atributo1,atributo2, ..., atributo n)  
VALUES ( valor1, valor2, ..., valor n)
```

# INSERT INTO (inserção)

- EXEMPLO: Inserir uma nova tupla (linha) na tabela Peca.

```
INSERT INTO Peca (Cod_Peca, Nome_Peca, Preço, Qte)
VALUES (380, 'Peca W', 77.00, 23) // caracteres entre aspas duplas
```

Peca			
<u>Cod_Peca</u>	Nome_Peca	Preço	Qte
56	Peca X	23.90	10
99	Peca Y	56.99	5
200	Peca Z	80.00	0
380	Peca W	77.00	23



# DELETE FROM (exclusão)

- O comando **delete** exclui dados (tuplas/linhas) em uma relação (tabela).
- **ATENÇÃO**  
**DROP (exclui estrutura) x DELETE (exclui dados)!!!**
- **Sintaxe:**

```
DELETE FROM nome_tabela  
WHERE condicao-de-exclusao
```

# DELETE FROM (exclusão)

- EXEMPLO: Excluir a peça com código 200 (**toda a linha**).

DELETE FROM Peca

WHERE Cod\_Peca = 200

Peca			
<u>Cod_Peca</u>	Nome_Peca	Preco	Qte
56	Peca X	23.90	10
99	Peca Y	56.99	5
200	Peca Z	80.00	0





# UPDATE/ SET (alteração)

- O comando **update** atualiza dados em uma relação (tabela).
- Quando há mudança de endereço, nome, valor, etc...
- **Sintaxe:**

```
UPDATE tabela
```

```
SET coluna1 = expressao1, ..., colunaN = expressaoN
```

```
WHERE condicao-de-alteracao
```

# UPDATE/ SET (alteração)

- EXEMPLO: Alterar o Preço da peça de código 200 para 90.00  
(antes era 80.00)

UPDATE Peca

SET Preço = 90.00

WHERE Cod\_Peca = 200

Peca			
<u>Cod_Peca</u>	Nome_Peca	Preço	Qte
56	Peca X	23.90	10
99	Peca Y	56.99	5
200	Peca Z	90.00	0



# SELECT (selecionar)

- O resultado de uma consulta SQL é uma relação com atributos e tuplas específicos.
- **Sintaxe:**

```
SELECT <lista de atributos>  
FROM <nome da(s) tabela(s)>  
WHERE <condicao_pesquisa1> AND <condicao_pesquisaN>
```

# SELECT (selecionar)

## CLÁUSULA WHERE

- A cláusula **WHERE** especifica as condições que devem ser satisfeitas.
- Usa conectores lógicos:
  - AND (e)
  - OR (ou)
  - NOT (não)

# SELECT (selecionar)

## CLÁUSULA WHERE

- Usa operadores de comparação:
  - > (maior)
  - < (menor)
  - = (igual)
  - <= (menor ou igual)
  - >= (maior ou igual)
- BETWEEN (entre um intervalo, **incluindo os extremos**)
  - facilita a especificação de condições numéricas que envolvam um intervalo, ao invés de usar os operadores =< e >=.

# SELECT (selecionar)

- **EXEMPLO 1:** Selecionar o código e o nome das peças com preço menor do que 70.00:

```
SELECT Cod_Peca, Nome_Peca  
FROM Peca  
WHERE Preco < 70.00
```

Peca			
<u>Cod_Peca</u>	Nome_Peca	Preco	Qte
56	Peca X	23.90	10
99	Peca Y	56.99	5
200	Peca Z	80.00	0

## RESULTADO:

Cod_Peca	Nome_Peca
56	Peca X
99	Peca Y

# SELECT (selecionar)

- **EXEMPLO 2:** Selecionar o nome e o preço das peças com preço maior que 50.00 e menor do que 70.00:

```
SELECT Nome_Peca, Preço
```

```
FROM Peca
```

```
WHERE Preço BETWEEN 50.00 AND 70.00
```

```
// WHERE Preço >= 50.00 AND Preço <= 70.00
```

Peca			
<u>Cod_Peca</u>	Nome_Peca	Preço	Qte
56	Peca X	23.90	10
99	Peca Y	56.99	5
200	Peca Z	80.00	0

## RESULTADO:

Nome_Peca	Preço
Peca Y	56.99

# SELECT (selecionar)

- O (\*) – ASTERISCO – seleciona **todos os atributos** da tabela.
- **EXEMPLO 3:** Selecionar todas informações das peças cuja quantidade em estoque seja maior ou igual a 10.

SELECT \*

FROM Peca

WHERE Qte >= 10

Peca

<u>Cod_Peca</u>	Nome_Peca	Preco	Qte
56	Peca X	23.90	10
99	Peca Y	56.99	5
200	Peca Z	80.00	0

**RESULTADO:**

Cod_Peca	Nome_Peca	Preco	Qte
56	Peca X	23.90	10



# SELECT (selecionar)

- **EXEMPLO 4:** Selecionar o código, o nome, o preço e a quantidade de peças no estoque cujo código é 200:

SELECT \*

FROM Peca

WHERE Cod\_Peca = 200

## Peca


<u>Cod_Peca</u>	Nome_Peca	Preco	Qte
56	Peca X	23.90	10
99	Peca Y	56.99	5
200	Peca Z	80.00	0

## RESULTADO:

Cod_Peca	Nome_Peca	Preco	Qte
200	Peca Z	80.00	0

# SELECT (selecionar) – ORDER BY

## CLÁUSULA ORDER BY (ORDENAÇÃO E APRESENTAÇÃO DE TUPLAS)

- Aplicado apenas à operação **SELECT**, depois da cláusula **WHERE**
- A cláusula **ORDER BY** faz com que as tuplas do resultado de uma consulta apareçam em uma determinada ordem.
- Para especificar a forma de ordenação, devemos indicar:
  - **Desc**: ordem descendente (decrecente)
  - **Asc**: ordem ascendente (crescente)  **default**
- **Sintaxe:**

```
ORDER BY nome_atributo ASC  
ORDER BY nome_atributo DESC
```

# SELECT (selecionar)

## CLÁUSULA ORDER BY

- **EXEMPLO:** Selecionar o nome e a quantidade de todas as peças que há no estoque, por ordem decrescente do nome:

```
SELECT Nome_Peca, Qte  
FROM Peca  
ORDER BY Nome_Peca DESC
```

### Peca

<u>Cod_Peca</u>	Nome_Peca	Preco	Qte
56	Peca X	23.90	10
99	Peca Y	56.99	5
200	Peca Z	80.00	0

### RESPOSTA:

Nome_Peca	Qte
Peca Z	0
Peca Y	5
Peca X	10

# FUNÇÕES DE AGREGAÇÃO

- As funções de agregação servem para recuperar dados agregados, ou seja, dados que foram trabalhados sobre os dados armazenados.
- As principais são:
  - **SUM** (soma dos valores da coluna — tipo de dado numérico),
  - **MAX** (valor máximo),
  - **MIN** (valor mínimo),
  - **AVG** (average - média — deve ser numérico),
  - **COUNT** (contador de tuplas - as linhas - da relação).
- **ATENÇÃO: SUM  $\neq$  COUNT**

# FUNÇÕES DE AGREGAÇÃO

- EXEMPLO 1: Encontrar a soma dos preços de todas as peças, o maior preço, o menor preço e a média dos preços.

```
SELECT SUM(Preco), MAX(Preco), MIN(Preco), AVG(Preco)
FROM Peca;
```

## Peca

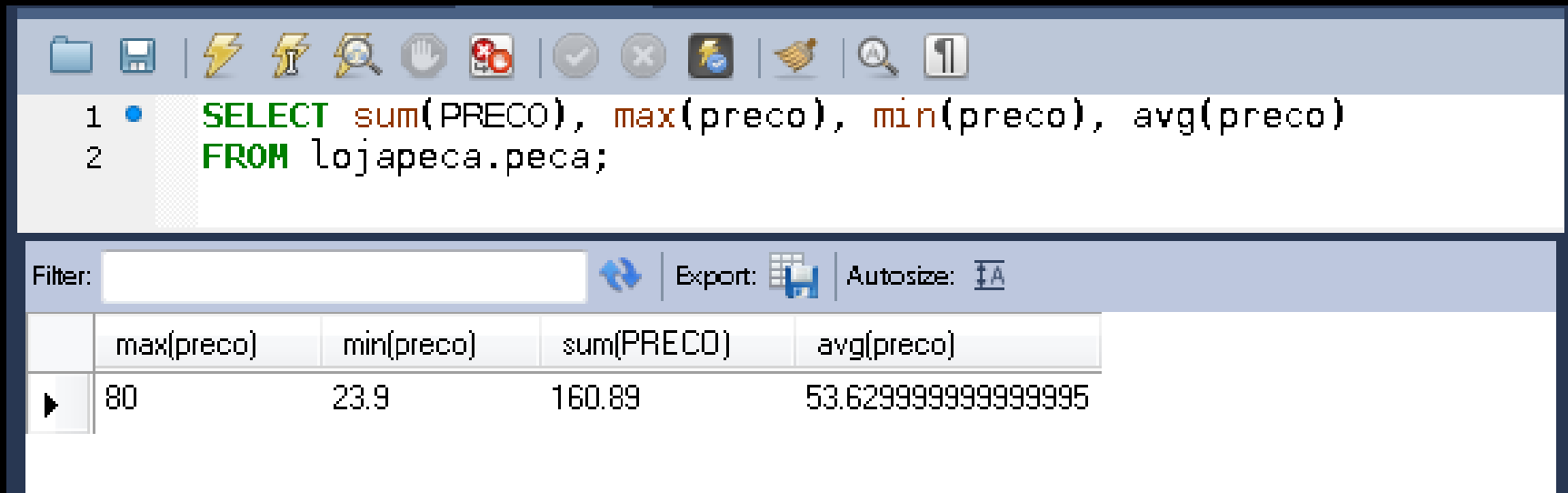
<u>Cod_Peca</u>	Nome_Peca	Preco	Qte
56	Peca X	23.90	10
99	Peca Y	56.99	5
200	Peca Z	80.00	0

## RESULTADOS:

SUM(Preco)	MAX (Preco)	MIN (Preco)	AVG (Preco)
160.89	80.00	23.90	53.62999999995

# FUNÇÕES DE AGREGAÇÃO

- Exemplo anterior executado no PostgreSQL  
(Schema: lojapeca; Tabela: peca)



The screenshot shows a PostgreSQL query editor window. The query is as follows:

```
1 SELECT sum(PRECO), max(preco), min(preco), avg(preco)
2 FROM lojapeca.pecas;
```

Below the query editor, the results are displayed in a table. The table has five columns: `max(preco)`, `min(preco)`, `sum(PRECO)`, and `avg(preco)`. The first row of data shows the values 80, 23.9, 160.89, and 53.629999999999995.

	max(preco)	min(preco)	sum(PRECO)	avg(preco)
▶	80	23.9	160.89	53.629999999999995

# FUNÇÕES DE AGREGAÇÃO

- EXEMPLO 2: COUNT

- Recuperar a quantos tipos de peças são vendidos na loja.

```
SELECT COUNT(*) //ou COUNT(Cod_Peca)
```

```
FROM Peca;
```

Peca			
<u>Cod_Peca</u>	Nome_Peca	Preco	Qte
56	Peca X	23.90	10
99	Peca Y	56.99	5
200	Peca Z	80.00	0

## RESULTADO

(contagem de tuplas):

COUNT(*)
3

# SQL – DML

*(Data Manipulation Definition -  
linguagem de manipulação de  
dados)*





Parte 2













# Valores NULL

- Suponha que temos a tabela peca criada na última aula preenchida até o momento da seguinte forma

	<b>cod_peca</b> [PK] integer 	<b>nome_peca</b> character varying (30) 	<b>preco</b> numeric (6,2) 	<b>qte</b> integer 
1	1	Peca A	15.00	10
2	2	Peca B	8.00	20
3	3	Peca C	8.00	30
4	4	Peca B	8.00	10

# Valores NULL

- Suponha que temos a tabela peca criada na última aula preenchida até o momento da seguinte forma

Columns								+
		Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
		cod_pec	integer   v			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		nome_pec	character varying   v	30		<input checked="" type="checkbox"/>	<input type="checkbox"/>	
		preco	numeric   v	6	2	<input type="checkbox"/>	<input type="checkbox"/>	
		qte	integer   v			<input type="checkbox"/>	<input type="checkbox"/>	0

# Inserindo Valores NULOS

- Quando realizamos um INSERT e não passamos o campo, o banco de dados vai automaticamente inserir NULL no valor da célula.

# Inserindo Valores NULOS

Query

Query History

```
1 INSERT INTO peca (cod_pecas, nome_pecas, qte)
2 VALUES (5, 'Peca E', 15);
3 SELECT * FROM peca;
```

Data Output

Messages

Notifications

≡+

📄

▼

📋

🗑️

🗄️

⬇️

📈

	<div><div>cod_pecas</div><div>[PK] integer</div></div>	<div><div>nome_pecas</div><div>character varying (30)</div></div>	<div><div>preco</div><div>numeric (6,2)</div></div>	<div><div>qte</div><div>integer</div></div>
1	1	Peca A	15.00	10
2	2	Peca B	8.00	20
3	3	Peca C	8.00	30
4	4	Peca B	8.00	10
5	5	Peca E	[null]	15

# CUIDADO com o DEFAULT

- Lembre-se que colunas que tem o valor DEFAULT definido, não serão preenchidas com NULL, mas sim com o valor DEFAULT

# CUIDADO com o DEFAULT

Query

Query History

1

INSERT INTO peca (cod\_pecas, nome\_pecas, preco)

2

VALUES (6, 'Peca F', 17.00);

3

SELECT \* FROM peca;

Data Output

Messages

Notifications

≡+

📄

▼

📋

🗑️

🗄️

⬇️

📈

	<div>cod_pecas</div> <div>[PK] integer </div>	<div>nome_pecas</div> <div>character varying (30) </div>	<div>preco</div> <div>numeric (6,2) </div>	<div>qte</div> <div>integer </div>
1	1	Peca A	15.00	10
2	2	Peca B	8.00	20
3	3	Peca C	8.00	30
4	4	Peca B	8.00	10
5	5	Peca E	[null]	15
6	6	Peca F	17.00	0

# Inserindo Valor NULL

- Existe uma outra forma de definir um valor como NULL. Essa forma é deixando explícito no comando que a coluna deve receber valor NULL

# Inserindo Valor NULL

Query

Query History

1

2

3

INSERT INTO

peca

(cod\_peca, nome\_peca, preco, qte)

VALUES (7, 'Peca G', 17.00, NULL);

SELECT \* FROM peca;

Data Output

Messages

Notifications

≡+

📄

▼

📋

🗑️

🗄️

⬇️

📈

	<div><div>cod_peca</div><div>[PK] integer</div></div>	<div><div>nome_peca</div><div>character varying (30)</div></div>	<div><div>preco</div><div>numeric (6,2)</div></div>	<div><div>qte</div><div>integer</div></div>
1	1	Peca A	15.00	10
2	2	Peca B	8.00	20
3	3	Peca C	8.00	30
4	4	Peca B	8.00	10
5	5	Peca E	[null]	15
6	6	Peca F	17.00	0
7	7	Peca G	17.00	[null]



# Inserindo valor NULL

- Repare, que mesmo o campo QTE possuindo um valor DEFAULT, foi definido de forma explícita no INSERT que essa coluna deveria possuir um valor NULL

# ATENÇÃO

- Se você tentar inserir o valor NULL em uma coluna definida como NOT NULL, uma exceção (ERRO) será lançada pelo banco de dados.

Query	Query History	Scratch Pad ×
1	<code>INSERT INTO peca (cod_pecas, nome_pecas, preco, qte)</code>	
2	<code>VALUES (7, NULL, 17.00, 12);</code>	

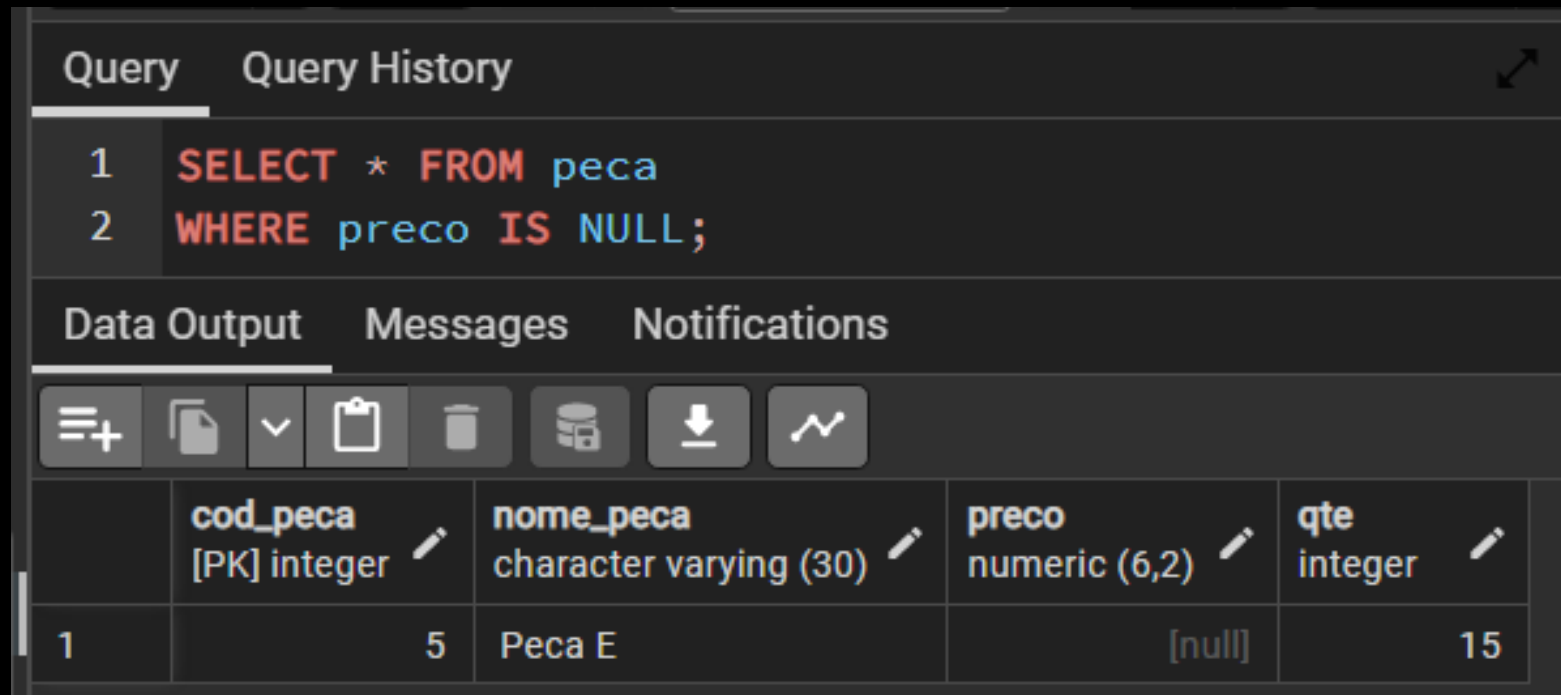
  

Data Output	Messages	Notifications
<p>ERROR: null value in column "nome_pecas" of relation "pecas" violates not-null constraint DETAIL: Failing row contains (7, null, 17.00, 12). SQL state: 23502</p>		

# Selecionando Valores Nulos

- Caso você queira retornar somente as linhas que possuam valores NULOS em uma determinada célula, a forma correta é utilizar o `IS NULL` e não `= NULL`

# FORMA CORRETA



The screenshot displays a database management interface. At the top, there are two tabs: 'Query' and 'Query History'. The 'Query' tab is active, showing a SQL query with two lines: '1 SELECT \* FROM peca' and '2 WHERE preco IS NULL;'. Below the query editor, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table of results. Above the table is a toolbar with icons for adding, saving, and other actions. The table has five columns: an index column, 'cod\_peca' (integer, PK), 'nome\_peca' (character varying (30)), 'preco' (numeric (6,2)), and 'qte' (integer). The first row of data shows the value '1' in the index column, '5' in the 'cod\_peca' column, 'Peca E' in the 'nome\_peca' column, '[null]' in the 'preco' column, and '15' in the 'qte' column.

```
1 SELECT * FROM peca
2 WHERE preco IS NULL;
```

	<b>cod_peca</b> [PK] integer	<b>nome_peca</b> character varying (30)	<b>preco</b> numeric (6,2)	<b>qte</b> integer
1	5	Peca E	[null]	15

# FORMA INCORRETA

Query

Query History

1

2

```
SELECT * FROM peca
WHERE preco = NULL;
```

Data Output

Messages

Notifications

≡+

📄

▼





📋

🗑️

🗄️

⬇️

📈

	<b>cod_pec</b> [PK] integer 	<b>nome_pec</b> character varying (30) 	<b>preco</b> numeric (6,2) 	<b>qte</b> integer 
--	--	---	---	---

# Selecionando Valores NÃO Nulos

- Caso você queira selecionar somente as linhas que NÃO possuam valor nulo em uma determinada coluna, é só utilizar o comando `IS NOT NULL`

# Selecionando Valores NÃO Nulos

Query

Query History

1

SELECT \* FROM peca

2

WHERE preco IS NOT NULL;

Data Output

Messages

Notifications

	<div>cod_pecaprecod_pec</div>
--	---

# Ordenando colunas com NULL

- Por default, caso você ordene um SELECT por uma coluna que possui células com valor NULL, essas células serão as últimas a serem retornadas



# Ordenando colunas com NULL

Query

Query History

1

SELECT \* FROM peca

2

ORDER BY preco;

Data Output

Messages

Notifications

	<div><div>cod_peca</div><div>[PK] integer</div></div>	<div><div>nome_peca</div><div>character varying (30)</div></div>	<div><div>preco</div><div>numeric (6,2)</div></div>	<div><div>qte</div><div>integer</div></div>
1	4	Peca B	8.00	10
2	2	Peca B	8.00	20
3	3	Peca C	8.00	30
4	1	Peca A	15.00	10
5	6	Peca F	17.00	0
6	7	Peca G	17.00	[null]
7	5	Peca E	[null]	15

# Ordenando colunas com NULL

- Caso você deseje que as células com valores NULL sejam as primeiras a serem retornadas no SELECT, utiliza o ORDER BY ... NULLS FIRST;

# Ordenando colunas com NULL

Query

Query History

1

SELECT \* FROM peca

2

ORDER BY preco NULLS FIRST;

Data Output

Messages

Notifications

	<div><div>cod_pec</div><div>[PK] integer</div><div></div></div>	<div><div>nome_pec</div><div>character varying (30)</div><div></div></div>	<div><div>preco</div><div>numeric (6,2)</div><div></div></div>	<div><div>qte</div><div>integer</div><div></div></div>
1	5	Peca E	[null]	15
2	2	Peca B	8.00	20
3	3	Peca C	8.00	30
4	4	Peca B	8.00	10
5	1	Peca A	15.00	10
6	6	Peca F	17.00	0
7	7	Peca G	17.00	[null]

# SELECT (selecionar)

## DISTINCT

- Tuplas duplicadas podem aparecer nas relações.
- No caso de desejarmos a eliminação de duplicidade, devemos inserir a palavra **DISTINCT** na cláusula **SELECT**.

### Observações:

- Funções agregadas normalmente consideram as tuplas duplicadas.
- Não é permitido o uso do **DISTINCT** com o **COUNT(\*)**.
- É válido usar o **DISTINCT** com **MAX** ou **MIN**, mesmo não alterando o resultado.

# Tabela Neste Momento

Query

Query History

1

SELECT \* FROM peca;

Data Output

Messages

Notifications

	<div>cod_peca</div> <div>[PK] integer</div>	<div>nome_peca</div> <div>character varying (30)</div>	<div>preco</div> <div>numeric (6,2)</div>	<div>qte</div> <div>integer</div>
1	1	Peca A	15.00	10
2	2	Peca B	8.00	20
3	4	Peca B	8.00	10
4	3	Peca A	8.00	30
5	6	Peca C	17.00	0
6	7	Peca C	17.00	[null]
7	5	Peca A	[null]	15

# SELECT (selecionar)

## DISTINCT

**Exemplo 1:** Obter o nome de todas peças (sem distinct).

```
SELECT nome_peca
```

```
FROM peca
```

**RESULTADO:**

Query

Query History

1

SELECT nome\_peca FROM peca;

Data Output

Messages

Notifications

<

# SELECT (selecionar)

## DISTINCT

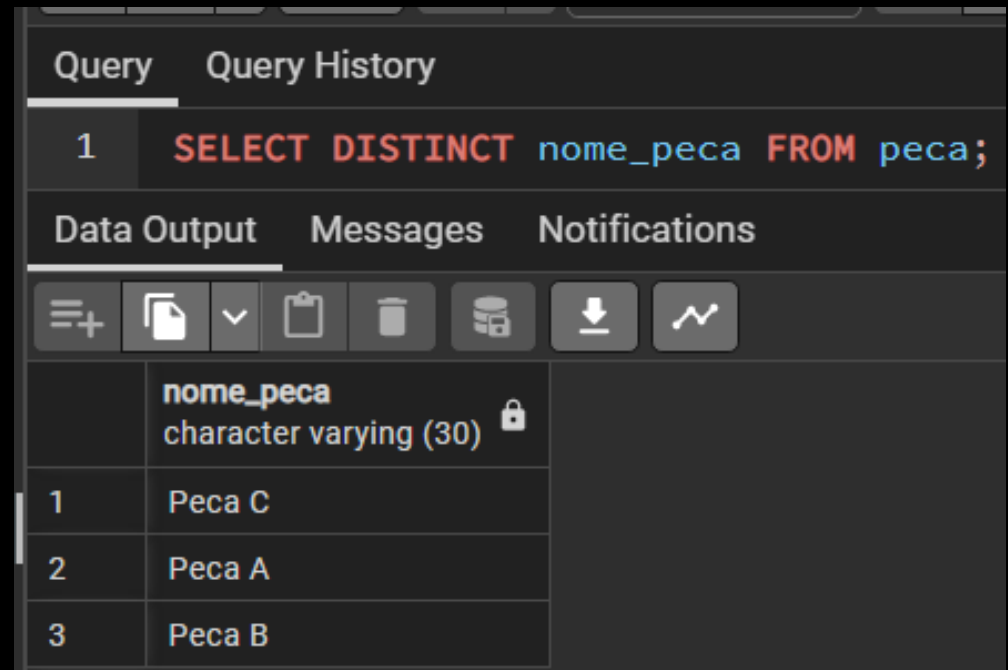
- **Exemplo2:** Obter o nome de todas as peças (usando a cláusula `distinct`).

## APARELHOS


**SELECT**

**DISTINCT** nome\_peca

**FROM** peca;



The screenshot shows a database query tool interface. At the top, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying the SQL query: `SELECT DISTINCT nome_peca FROM peca;`. Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with the results of the query. The table has two columns: 'nome\_peca' (character varying (30)) and a lock icon. The results are as follows:

	nome_peca character varying (30) 
1	Peca C
2	Peca A
3	Peca B

# SELECT (selecionar)

CLÁUSULA GROUP BY = agrupar por

- São funções que tomam uma coleção (conjunto ou subconjunto) de valores como entrada, retornando um valor simples.
- Só podem ser usadas em comandos SELECT.
- Normalmente utilizada em conjunto com as funções de agregação (COUNT, SUM, AVG, MIN, MAX).
- Agrupa as tuplas selecionadas com base em um de seus atributos (quando este atributo é igual em duas ou mais tuplas elas são agrupadas).
- Desta forma, a função de agregação será aplicada a cada grupo, e não a todas as tuplas.



# Tabela Neste Momento

```
1 SELECT * FROM peca ORDER BY cod_pecas;
```

Data Output Messages Notifications



	<b>cod_pecas</b> [PK] integer	<b>nome_pecas</b> character varying (30)	<b>preco</b> numeric (6,2)	<b>qte</b> integer	<b>veiculo</b> character varying (8)
1	1	Peca A	15.00	10	CARRO
2	2	Peca B	8.00	20	MOTO
3	3	Peca C	8.00	30	CAMINHAO
4	4	Peca D	8.00	10	CARRO
5	5	Peca E	[null]	15	CAMINHAO
6	6	Peca F	17.00	0	MOTO
7	7	Peca G	17.00	[null]	CARRO

# SELECT (selecionar)

CLÁUSULA GROUP BY = agrupar por

- Exemplo1: Obter o número de peças cadastradas.










```
SELECT COUNT (*)
```

```
FROM pecas
```

- RESULTADO

(contagem de tuplas):

ADABELLOS

Query	Query History	
1	<code>SELECT COUNT(*) FROM peca;</code>	
Data Output	Messages	Notifications
	       	
	count bigint	
1		7

# SELECT (selecionar)

CLÁUSULA GROUP BY = agrupar por

- Exemplo 2: Obter o n° de peças por veículo // contar por grupos

Query

Query History



1

SELECT veiculo, COUNT (1) FROM peca GROUP BY veiculo;

Data Output

Messages

Notifications

	veiculo character varying (8) 	count bigint 
1	MOTO	2
2	CAMINHAO	2
3	CARRO	3

# SELECT (selecionar)

CLÁUSULA GROUP BY = agrupar por

- Obter a soma da quantidade de peças por tipo de veículo.

Query

Query History

1

SELECT veiculo, SUM(qte) FROM peca GROUP BY veiculo;

Data Output

Messages

Notifications

	veiculo character varying (8)	sum bigint
1	MOTO	20
2	CAMINHAO	45
3	CARRO	20

# SELECT (selecionar)

## CLÁUSULA HAVING

- É utilizada para filtrar as linhas do grupo criado por GROUP BY.
- **Exemplo 2 (anterior) alterado:** obter a soma da quantidade de peças que sejam maiores que 20.

Query

Query History

1

SELECT veiculo, SUM(qte)

2

FROM peca

3

GROUP BY veiculo HAVING SUM(qte) > 20;

Data Output

Messages

Notifications

≡+

	veiculo character varying (8)	sum bigint
1	CAMINHAO	45

APARELHOS

# SELECT (selecionar)

## SELEÇÃO COM JUNÇÃO

- Quando os dados requeridos são de tabelas distintas, isto é, atributos de mais uma tabela são requeridos.
- Existe uma CONDIÇÃO DE JUNÇÃO, na qual os atributos **chave primária** e **chave estrangeira** das relações devem ser relacionados.

# SELECT (selecionar)

## SELEÇÃO COM JUNÇÃO

- **Ex.1:** Obter os nomes dos técnicos com experiência em secadora.

SELECT Nome

FROM TÉCNICOS, EXPERIÊNCIA

WHERE NumTec = NumTecnico AND Tipo = 'Secadora'

TÉCNICOS

NumTec	Nome	Cargo
297	Marco	Trainee
553	Hélio	Sênior
062	Tião	Sênior
718	Sílvio	Estagiário

EXPERIÊNCIA

NumTecnico	Tipo	AnosExp
553	Secadora	15
062	Lavadora	18
297	Torradeira	1
297	Secadora	1
718	Lavadora	5
062	Congelador	10
062	Secadora	12

# SELECT (selecionar)

## SELEÇÃO COM JUNÇÃO

- Ex.2: Obter nomes dos técnicos com experiência maior que 10 anos.

SELECT Nome

FROM TÉCNICOS, EXPERIÊNCIA

WHERE NumTec = NumTecnico AND AnosExp > 10

TÉCNICOS

NumTec	Nome	Cargo
297	Marco	Trainee
553	Hélio	Sênior
062	Tião	Sênior
718	Sílvio	Estagiário

EXPERIÊNCIA

NumTecnico	Tipo	AnosExp
553	Secadora	15
062	Lavadora	18
297	Torradeira	1
297	Secadora	1
718	Lavadora	5
062	Congelador	10
062	Secadora	12



# SELECT (selecionar)

## SELEÇÃO COM JUNÇÃO

**Ex.3:** Liste o nome dos técnicos e sua experiência em aparelhos da categoria 1.

### TIPOS

Tipo	Categoria	Taxa
Lavadora	1	20,00
Secadora	1	20,00
Torradeira	2	10,00
Congelador	1	8,00
Batedeira	2	25,00

### EXPERIÊNCIA

NumTecnico	Tipo	AnosExp
553	Secadora	15
062	Lavadora	18
297	Torradeira	1
297	Secadora	1
718	Lavadora	5
062	Congelador	10
062	Secadora	12

### TÉCNICOS

NumTec	Nome	Cargo
297	Marco	Trainee
553	Hélio	Sênior
062	Tião	Sênior
718	Sílvio	Estagiário

# SELECT (selecionar)

## SELEÇÃO COM JUNÇÃO

**Ex.3:** Liste o nome dos técnicos e sua experiência em aparelhos da categoria 1.

// colocar nomes das tabelas antes dos atributos

```
SELECT  TECNICOS.Nome, EXPERIÊNCIA.AnosExp,  
        TIPOS.Tipo  
FROM    TECNICOS, TIPOS, EXPERIENCIA  
WHERE   TIPOS.Tipo = EXPERIENCIA.Tipo AND  
        EXPERIENCIA.NumTecnico = TECNICOS.NumTec  
        AND TIPOS.Categoria = 1
```

# SELECT (selecionar)

## USO DE ALIASES

- Aliases → “apelidos” para as tabelas, usando a palavra reservada “AS”
- Permite associar um “nome de variável” para cada relação, a fim de simplificar os comandos SQL.
- Criando um SELECT com Aliases, temos:

```
SELECT  C.Nome, E.AnosExp, TP.Tipo
FROM    TECNICOS AS T, TIPOS AS TP, EXPERIENCIA AS E
WHERE   TP.Tipo = E.Tipo AND E.NumTecnico = T.NumTec
        AND TP.Categoria = 1
```

# CONSULTAS EM SQL - RESUMO

- Conjunto completo de cláusulas no comando SELECT-FROM
- Uma consulta em SQL pode consistir em até seis cláusulas:

```
SELECT [*] [DISTINCT] <lista de atributos> <funções agregação>
FROM <lista de tabelas>
[WHERE <condicao>]
[GROUP BY <lista de atributos para agrupamento>]
[HAVING <condicao para agrupamento, aceita funções agregação>]
[ORDER BY <atributo1, atributo2,..., atributo N> [ASC] [DESC]]
```

## OBSERVAÇÕES

- Apenas as cláusulas **SELECT** e **FROM** são obrigatórias.
- Quando existentes, as cláusulas devem aparecer na ordem especificada acima.
- O **ORDER BY** só pode ser utilizado após o ultimo **SELECT** (se a linguagem permitir).
- As cláusulas **GROUP BY** e **HAVING** só podem ser usados nos comandos **SELECT** individuais.

# JUNÇÕES

- É a possibilidade de se criar relacionamentos entre tabelas de forma a poder **recuperar dados de todas elas através de uma única consulta.**
- Através de uma operação chamada **JOIN** (junção) é possível consultar e manipular dados de mais de uma tabela usando apenas um comando **SELECT**.

# JUNÇÕES

- É importante utilizá-lo, porque tira da cláusula WHERE condições que são estritamente das junções (chave primária igual a chave estrangeira, por exemplo).
- Existem as variações de **junções internas e externas**.
- Internas: INNER JOIN, NATURAL JOIN

OBS.: A palavra INNER pode ser omitida.

# JUNÇÕES

## COMO FUNCIONA O COMANDO JOIN?

- Quando um comando **SELECT** especifica campos de duas tabelas **sem nenhuma restrição ou filtro**, o resultado será um número de linhas iguais à multiplicação do total de linhas da primeira tabela (N) pela segunda tabela (M) =  $N \times M$  tuplas.
- Isso ocorre devido ao fato de que, para cada linha da primeira tabela, todas as linhas da segunda são processadas.
- **Operações de junções tomam duas relações e têm como resultado uma outra relação.**

# JUNÇÕES

## FUNCIONAMENTO DO COMANDO JOIN

**CURSO**

<u>ID</u>	NomeC
1	Arquitetura
2	Nutrição

**JOIN**

**ALUNO**

<u>CIU</u>	Nome	IdCurso *
50200	Bia	2
50201	Rui	1
50202	Ana	2

### RESULTADO:

2 linhas x 3 linhas  
= 6 linhas

ID	NomeC	CIU	Nome	IdCurso
1	Arquitetura	50200	Bia	2
1	Arquitetura	50201	Rui	1
1	Arquitetura	50202	Ana	2
2	Nutrição	50200	Bia	2
2	Nutrição	50201	Rui	1
2	Nutrição	50202	Ana	2



# JUNÇÕES

## FUNCIONAMENTO DO COMANDO JOIN

### RESULTADO:

Porém, o resultado só fará sentido quando as chaves PK e FK forem equivalentes, ou seja,  $ID = IDCURSO$  (isso é feito usando o ON).

### RESULTADO CORRETO:

IDs devem ser iguais



ID	NomeC	CIU	Nome	IdCurso
1	Arquitetura	50200	Bia	2
1	Arquitetura	50201	Rui	1
1	Arquitetura	50202	Ana	2
2	Nutrição	50200	Bia	2
2	Nutrição	50201	Rui	1
2	Nutrição	50202	Ana	2

ID	NomeC	CIU	Nome	IdCurso
1	Arquitetura	50201	Rui	1
2	Nutrição	50200	Bia	2
2	Nutrição	50202	Ana	2

# JUNÇÕES

- **INNER JOIN (ou somente JOIN)**
- O inner join é uma junção interna.
- Junta os registros da tabela que tiver um **correspondente** na outra tabela, através das chaves primária e estrangeira.

## TÉCNICOS

NumTec	Nome	Cargo
--------	------	-------

## EXPERIÊNCIA

NumTecnico	Tipo	AnosExp
------------	------	---------

→ TÉCNICOS **INNER JOIN** EXPERIÊNCIA **ON**  
NumTec=NumTecnico

# JUNÇÕES

## TIPOS DE JUNÇÕES E CONDIÇÃO DE JUNÇÃO

Cada uma das variantes das operações de JUNÇÃO consiste em um *tipo de junção* e em uma *condição de junção*.

- **Condição de junção (ON)**
  - Definem quais tuplas das duas relações apresentam correspondência e quais atributos são apresentados no resultado de uma junção.
- **Tipo de junção (inner ou outer)**
  - Define como as tuplas em cada relação que não possuam nenhuma correspondência com as tuplas da outra relação devem ser tratadas.

# JUNÇÕES

- **USANDO INNER JOIN** com 2 tabelas.
- **Ex.3:** Liste o nome dos técnicos que possuem experiência em congelador.

SELECT Nome

FROM (TECNICOS INNER JOIN EXPERIENCIA ON  
NumTec=NumTecnico)

WHERE Tipo = 'congelador'

TÉCNICOS

NumTec	Nome	Cargo
297	Marco	Trainee
553	Hélio	Sênior
062	Tião	Sênior
718	Sílvio	Estagiário

EXPERIÊNCIA

NumTecnico	Tipo	AnosExp
553	Secadora	15
062	Lavadora	18
297	Torradeira	1
297	Secadora	1
718	Lavadora	5
062	Congelador	10
062	Secadora	12

# JUNÇÕES

- **USANDO INNER JOIN**  
com 3 tabelas.
- **Exemplo:** Liste o nome dos técnicos e sua experiência em aparelhos da categoria 1.

## TÉCNICOS

NumTec	Nome	Cargo
297	Marco	Trainee
553	Hélio	Sênior
062	Tião	Sênior
718	Sílvio	Estagiário

## TIPOS

Tipo	Categoria	Taxa
Lavadora	1	20,00
Secadora	1	20,00
Torradeira	2	10,00
Congelador	1	8,00
Batedeira	2	25,00

## EXPERIÊNCIA

NumTecnico	Tipo	AnosExp
553	Secadora	15
062	Lavadora	18
297	Torradeira	1
297	Secadora	1
718	Lavadora	5
062	Congelador	10
062	Secadora	12

# JUNÇÕES

USANDO INNER JOIN com 3 tabelas.

- **Exemplo:** Liste o nome dos técnicos e sua experiência em aparelhos da categoria 1.

```
SELECT  Nome, AnosExp
FROM ((TECNICOS INNER JOIN EXPERIENCIA ON
      NumTec = NumTecnico) INNER JOIN TIPOS ON Tipo =
      Tipo)
WHERE Categoria = 1
```

# JUNÇÕES

- **NATURAL JOIN**
- Com ele você não precisa identificar quais colunas serão comparadas, pois ele fará a comparação entre campos com mesmo nome.

→ TECNICOS **NATURAL JOIN** EXPERIENCIA

## TIPOS

Tipo	Categoria	Taxa
------	-----------	------

## EXPERIÊNCIA

NumTecnico	Tipo	AnosExp
------------	------	---------

→ Junção natural entre TECNICOS e EXPERIÊNCIA não dá (atributos com nomes diferentes)!!!

## TÉCNICOS

NumTec	Nome	Cargo
--------	------	-------

## EXPERIÊNCIA

NumTecnico	Tipo	AnosExp
------------	------	---------

# JUNÇÕES

- Existem as variações de junções internas e externas.
- Externas:
- O **OUTER JOIN** é uma junção externa.
- Temos:
  - **LEFT OUTER JOIN**
  - **RIGHT OUTER JOIN**

OBS.: A palavra **OUTER** pode ser omitida.



# JUNÇÕES

- **LEFT OUTER JOIN**

- A precedência é da tabela da Esquerda, isto é, todos os registros da primeira tabela serão mostrados independente se houver correspondente na outra tabela, após a equivalência das chaves através do “ON”.
- Pega todos os atributos da relação que está à esquerda, verifica se existe algum correspondendo à direita, caso afirmativo, retorna os atributos da direita e caso negativo, coloca o valor nulo (NULL) nos atributos.

# JUNÇÕES

- **LEFT OUTER JOIN**

→ LISTAR TODOS OS FUNCIONÁRIOS PERTENCENTES AS LOJAS.

```
SELECT F.NOMEFUN, L.NOME  
FROM FUNCIONARIOS AS F LEFT JOIN LOJAS AS L ON  
F.CODIGOL = L.CODIGO
```

FUNCIONÁRIOS

<u>CODIGOFUN</u>	CODIGOL*	NOMEFUN
1	1	JOÃO
2	1	JOAQUIM
3	2	JOSÉ
4	3	MÁRCIO
5	NULL	PEDRO

LOJAS

<u>CÓDIGO</u>	NOME
1	MATRIZ
2	FILIAL 1
3	FILIAL 2

# JUNÇÕES

- **LEFT OUTER JOIN**

Neste caso serão listados TODOS os funcionários mesmo os que não haja lojas cadastradas, POIS A PRIORIDADE É DA TABELA FUNCIONÁRIO.

RESULTADO:

NOMEFUN	NOME
JOÃO	MATRIZ
JOAQUIM	FILIAL 1
JOSÉ	FILIAL 1
MÁRCIO	FILIAL 2
PEDRO	NULL

# JUNÇÕES

- **RIGHT OUTER JOIN (join)**
- Prioridade da tabela à direita, isto é, todos os registros da segunda tabela serão mostrados independente se houver correspondente na outra tabela.
- Pega todos os atributos da relação que esta à direita, verifica se existe algum correspondente á esquerda, caso afirmativo, retorna os atributos da esquerda e caso negativo, coloca o valor nulo nos atributos.

# JUNÇÕES

- **RIGHT OUTER JOIN**

→ LISTAR TODOS OS FUNCIONÁRIOS PERTENCENTES AS LOJAS.

```
SELECT F.NOMEFUN, L.NOME  
FROM  FUNCIONARIOS AS F RIGHT JOIN LOJAS AS L ON  
      F.CODIGOL = L.CODIGO
```

FUNCIONÁRIOS

CODIGOFUN	CODIGOL	NOMEFUN
1	1	JOÃO
1	1	JOAQUIM
3	2	JOSÉ
4	3	MÁRCIO
5	NULL	PEDRO

LOJAS

CÓDIGO	NOME
1	MATRIZ
2	FILIAL 1
3	FILIAL 2

# JUNÇÕES

- **RIGHT OUTER JOIN**

Neste caso serão listados TODAS AS LOJAS com seus respectivos funcionários, POIS A PRIORIDADE É DA TABELA LOJAS.

## RESULTADO:

NOMEFUN	NOME
JOÃO	MATRIZ
JOAQUIM	FILIAL 1
JOSÉ	FILIAL 1
MÁRCIO	FILIAL 2

Neste caso o resultado do RIGHT JOIN seria igual ao resultado do INNER JOIN.