

SQL – DML

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

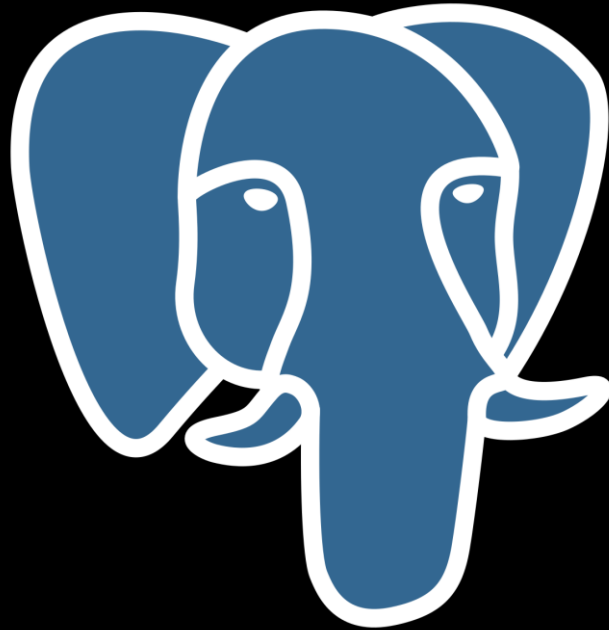
Parte 1



CREATE TABLE

- `CREATE TABLE nome_tabela (atributo1 tipo1 ... atributo tipoN);`
- `DEFAULT 'valor default'`
- `NOT NULL`
- `PRIMARY KEY`
- `FOREIGN KEY ... REFERENCES tabela (nomePK)`

DATA TYPES



<https://www.postgresql.org/docs/current/datatype.html>

ALTER TABLE tabela <expressão>

ADD COLUMN [IF NOT EXISTS]

This form adds a new column to the table, using the same syntax as `CREATE TABLE`. If `IF NOT EXISTS` is specified and a column already exists with this name, no error is thrown.

DROP COLUMN [IF EXISTS]

This form drops a column from a table. Indexes and table constraints involving the column will be automatically dropped as well. Multivariate statistics referencing the dropped column will also be removed if the removal of the column would cause the statistics to contain data for only a single column. You will need to say `CASCADE` if anything outside the table depends on the column, for example, foreign key references or views. If `IF EXISTS` is specified and the column does not exist, no error is thrown. In this case a notice is issued instead.

SET DATA TYPE

This form changes the type of a column of a table. Indexes and simple table constraints involving the column will be automatically converted to use the new column type by reparsing the originally supplied expression. The optional `COLLATE` clause specifies a collation for the new column; if omitted, the collation is the default for the new column type. The optional `USING` clause specifies how to compute the new column value from the old; if omitted, the default conversion is the same as an assignment cast from old data type to new. A `USING` clause must be provided if there is no implicit or assignment cast from old to new type.

When this form is used, the column's statistics are removed, so running `ANALYZE` on the table afterwards is recommended.

SET/DROP DEFAULT

These forms set or remove the default value for a column (where removal is equivalent to setting the default value to `NULL`). The new default value will only apply in subsequent `INSERT` or `UPDATE` commands; it does not cause rows already in the table to change.

SET/DROP NOT NULL

These forms change whether a column is marked to allow null values or to reject null values.

`SET NOT NULL` may only be applied to a column provided none of the records in the table contain a `NULL` value for the column. Ordinarily this is checked during the `ALTER TABLE` by scanning the entire table; however, if a valid `CHECK` constraint is found which proves no `NULL` can exist, then the table scan is skipped.

If this table is a partition, one cannot perform `DROP NOT NULL` on a column if it is marked `NOT NULL` in the parent table. To drop the `NOT NULL` constraint from all the partitions, perform `DROP NOT NULL` on the parent table. Even if there is no `NOT NULL` constraint on the parent, such a constraint can still be added to individual partitions, if desired; that is, the children can disallow nulls even if the parent allows them, but not the other way around.

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;

Restrições – ON DELETE, ON UPDATE

NO ACTION

Produce an error indicating that the deletion or update would create a foreign key constraint violation. If the constraint is deferred, this error will be produced at constraint check time if there still exist any referencing rows. This is the default action.

RESTRICT

Produce an error indicating that the deletion or update would create a foreign key constraint violation. This is the same as `NO ACTION` except that the check is not deferrable.

CASCADE

Delete any rows referencing the deleted row, or update the values of the referencing column(s) to the new values of the referenced columns, respectively.

SET NULL [(*column_name* [, ...])]

Set all of the referencing columns, or a specified subset of the referencing columns, to null. A subset of columns can only be specified for `ON DELETE` actions.

SET DEFAULT [(*column_name* [, ...])]

Set all of the referencing columns, or a specified subset of the referencing columns, to their default values. A subset of columns can only be specified for `ON DELETE` actions. (There must be a row in the referenced table matching the default values, if they are not null, or the operation will fail.)

If the referenced column(s) are changed frequently, it might be wise to add an index to the referencing column(s) so that referential actions associated with the foreign key constraint can be performed more efficiently.

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:

<u>Cod_Peca</u>	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

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

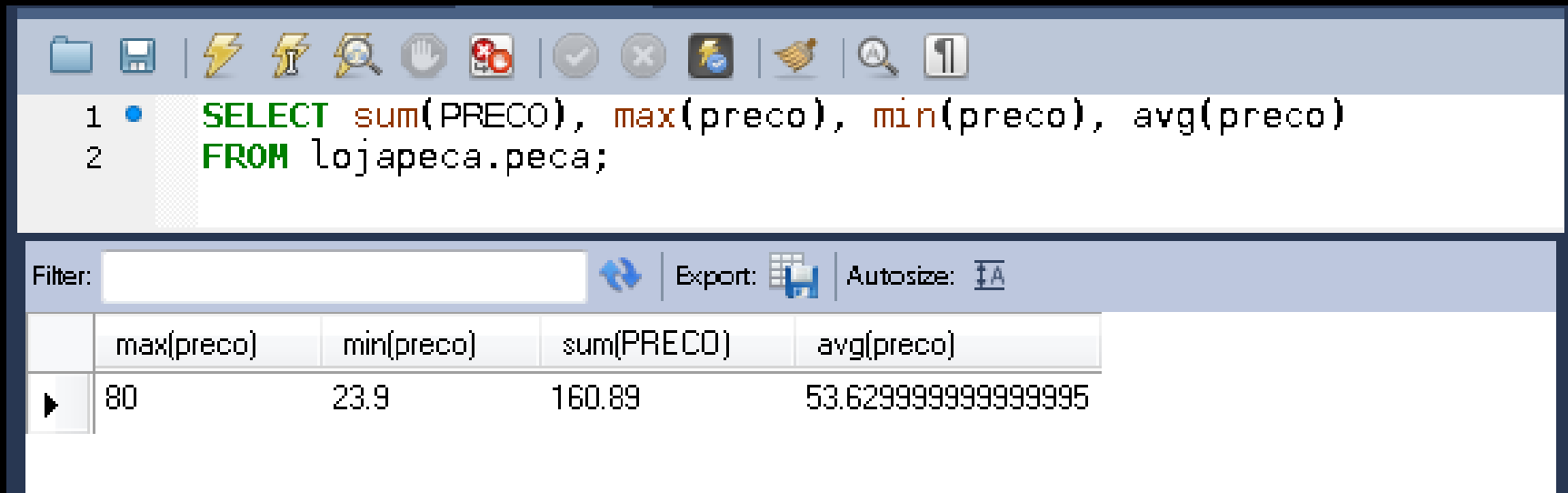
<u>Cod_Peca</u>	Nome_Peca	Preco	Qte
56	Peca X	23.90	10
99	Peca Y	56.99	5
200	Peca Z	80.00	0

RESULTADOS:

SUM(Preco)	MAX (Preco)	MIN (Preco)	AVG (Preco)
160.89	80.00	23.90	53.629999999995

FUNÇÕES DE AGREGAÇÃO

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



The screenshot shows a PostgreSQL query editor interface. The query is: `SELECT sum(PRECO), max(preco), min(preco), avg(preco) FROM lojapeca.pecas;`. The results are displayed in a table with columns: `max(preco)`, `min(preco)`, `sum(PRECO)`, and `avg(preco)`. The values are 80, 23.9, 160.89, and 53.629999999999995 respectively.

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

	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