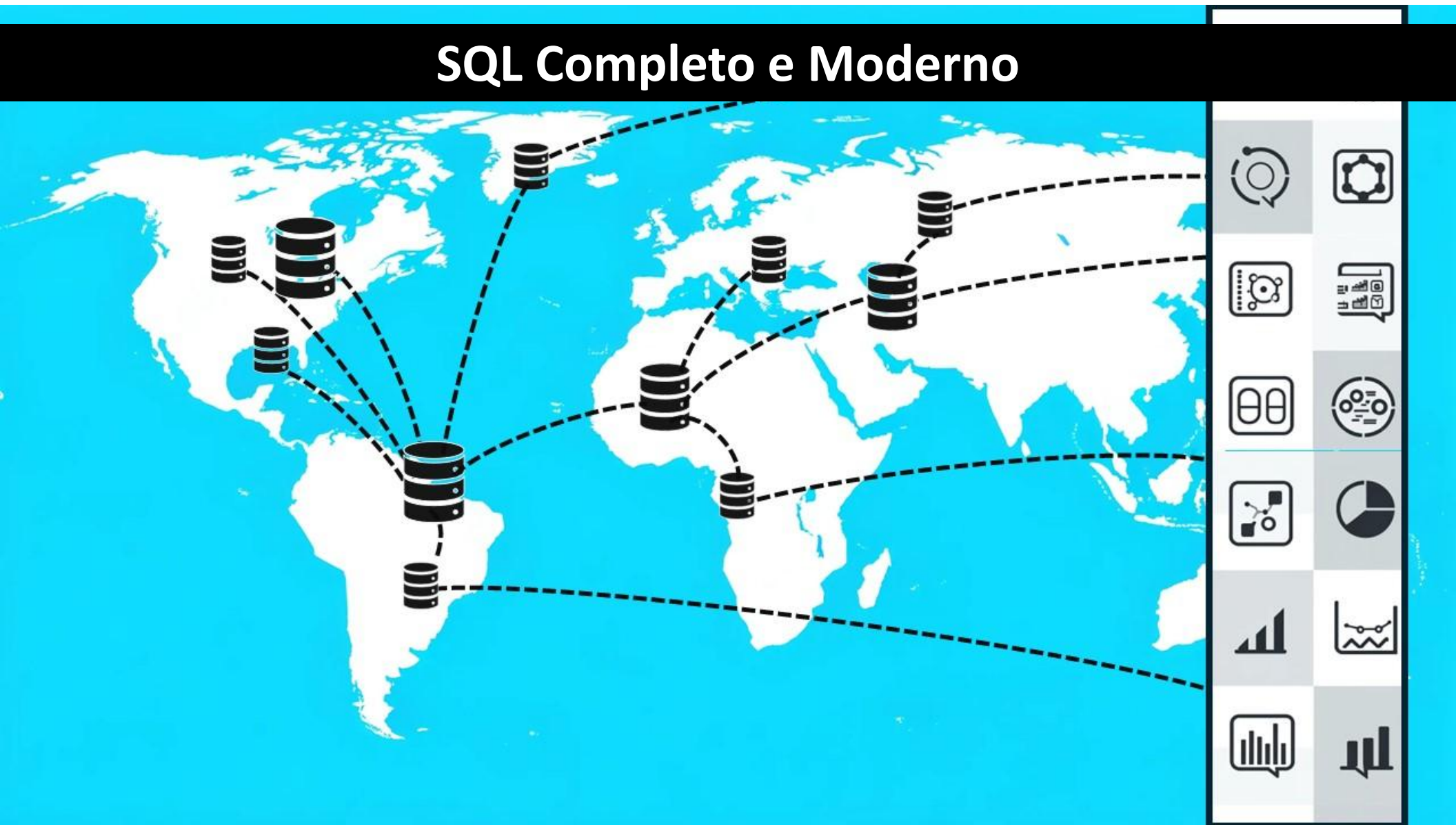
SQL Completo e Moderno



Tutor: Fernando Amaral

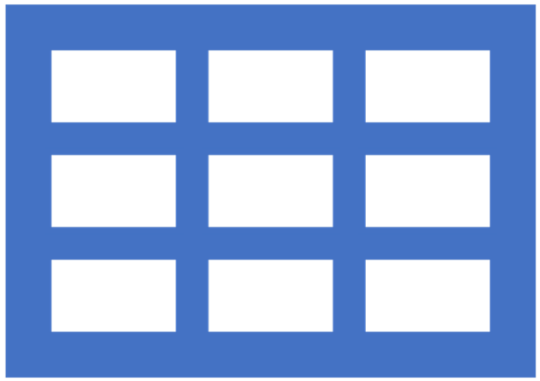
Estrutura do Curso

Orientações Gerais

- Vamos preparar um ambiente local
  - Postgres
  - Banco de Dados Populado
- O objetivo do curso é ser Agnóstico a Marca de Banco de Dados
- Material do curso para download
  - Scripts
  - Apostila em PDF (para referência)
- Dezenas de Atividades Diversas: Importante!

ID da Venda	Data da Venda	Produto	Quantidade	Preço Unitário	Total da Venda	Vendedor
1	2024-05-01	Camiseta	2	R$ 50	R$ 100	Ana
2	2024-05-02	Boné	1	R$ 30	R$ 30	Carlos
3	2024-05-02	Tênis	1	R$ 120	R$ 120	João
4	2024-05-03	Camiseta	1	R$ 50	R$ 50	Ana
5	2024-05-03	Tênis	2	R$ 120	R$ 240	Maria

Solução: Normalizar



- Organizar dados de forma a reduzir redundâncias e dependências
- Divide-se tabelas em tabelas menores
- A relação entre elas é mantida através de chaves primárias e estrangeiras

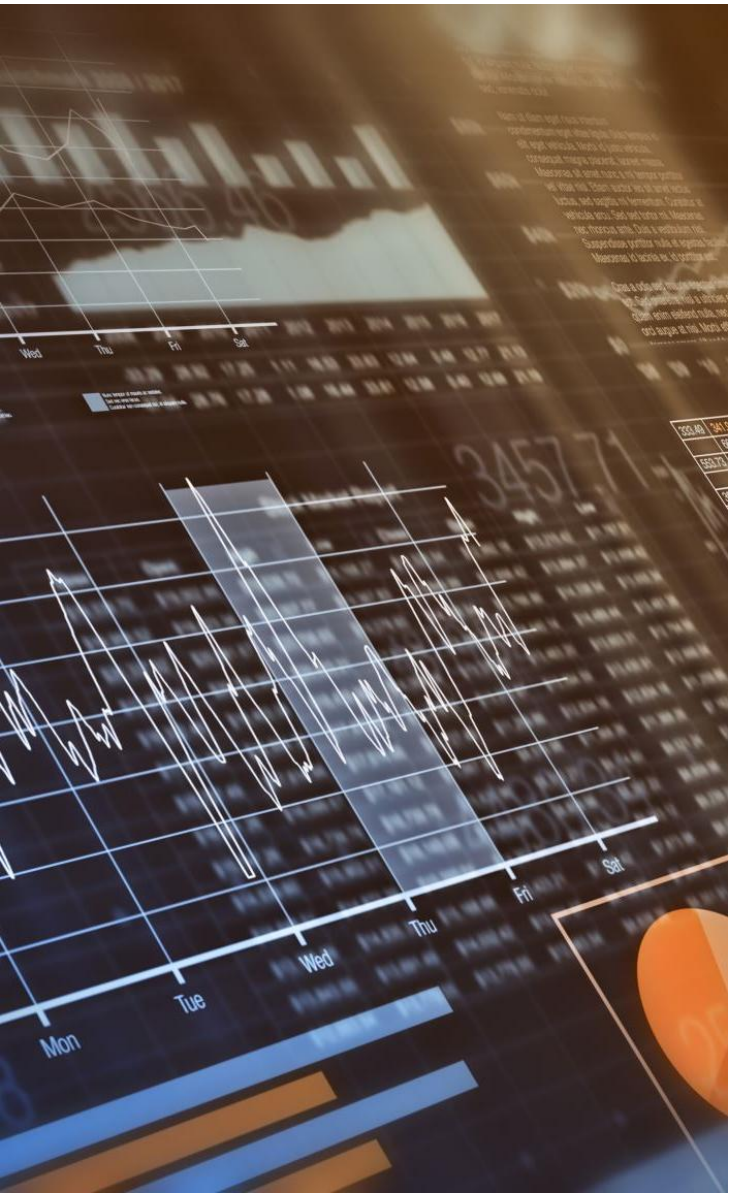
ID da Venda	Data da Venda	ID do Produto	ID do Vendedor	Quantidade	Total da Venda
1	2024-05-01	1	1	2	R$ 100
2	2024-05-02	2	2	1	R$ 30
3	2024-05-02	3	3	1	R$ 120
4	2024-05-03	1	1	1	R$ 50
5	2024-05-03	3	4	2	R$ 240

ID do Produto	Nome do Produto	Preço Unitário
1	Camiseta	R$ 50
2	Boné	R$ 30
3	Tênis	R$ 120

ID do Vendedor	Nome do Vendedor
1	Ana
2	Carlos
3	João
4	Maria

Consequências da Normalização:

- Consultas Complexas (joins)
- Desempenho
- Integridade Referencial
- Anomalias na Transação



Desnormalização

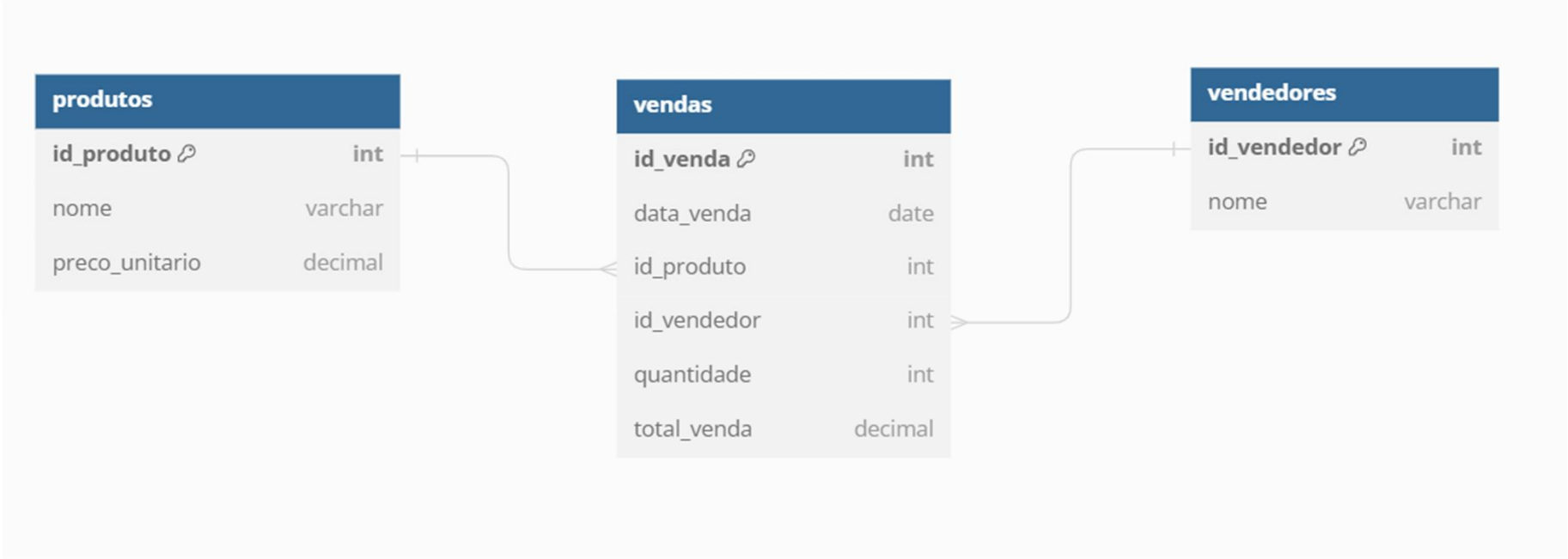
- Processo de Transformar novamente em uma única tabela, com todos os atributos
- Ótima Performance
- Usando em Relatórios, Sistemas, Dashboards etc.
- Views normalmente são Tabelas desnormalizadas

Banco de Dados

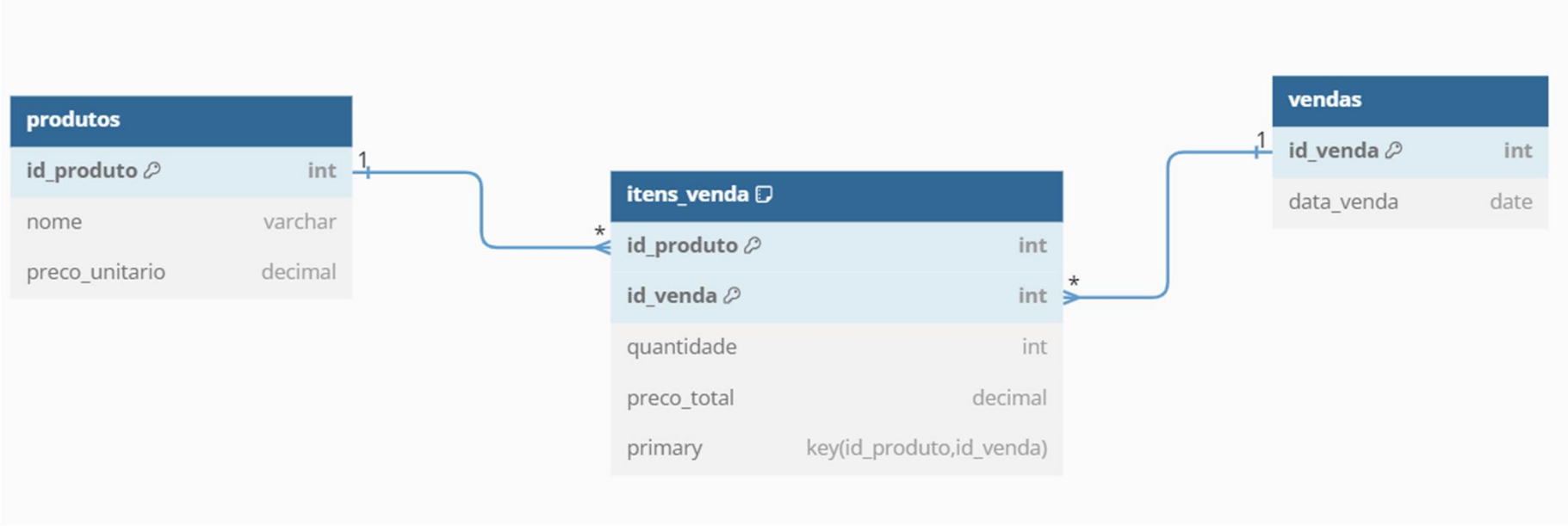
- Tabela/Entidade: Informação tabular
- Coluna/atributo/Campo
- Registro/Linha

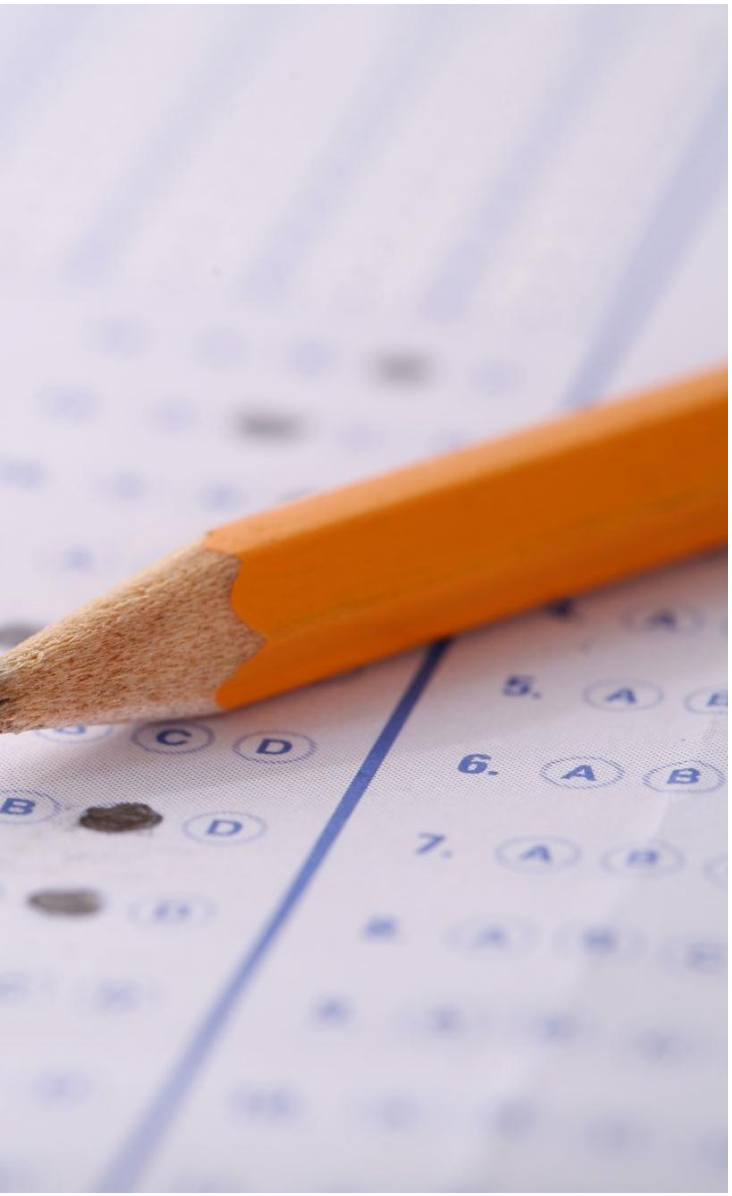
ID do Vendedor	Nome do Vendedor
1	Ana
2	Carlos
3	João
4	Maria

Chaves Primárias e Estrangeiras



Chaves Primárias Compostas e Relações N para N



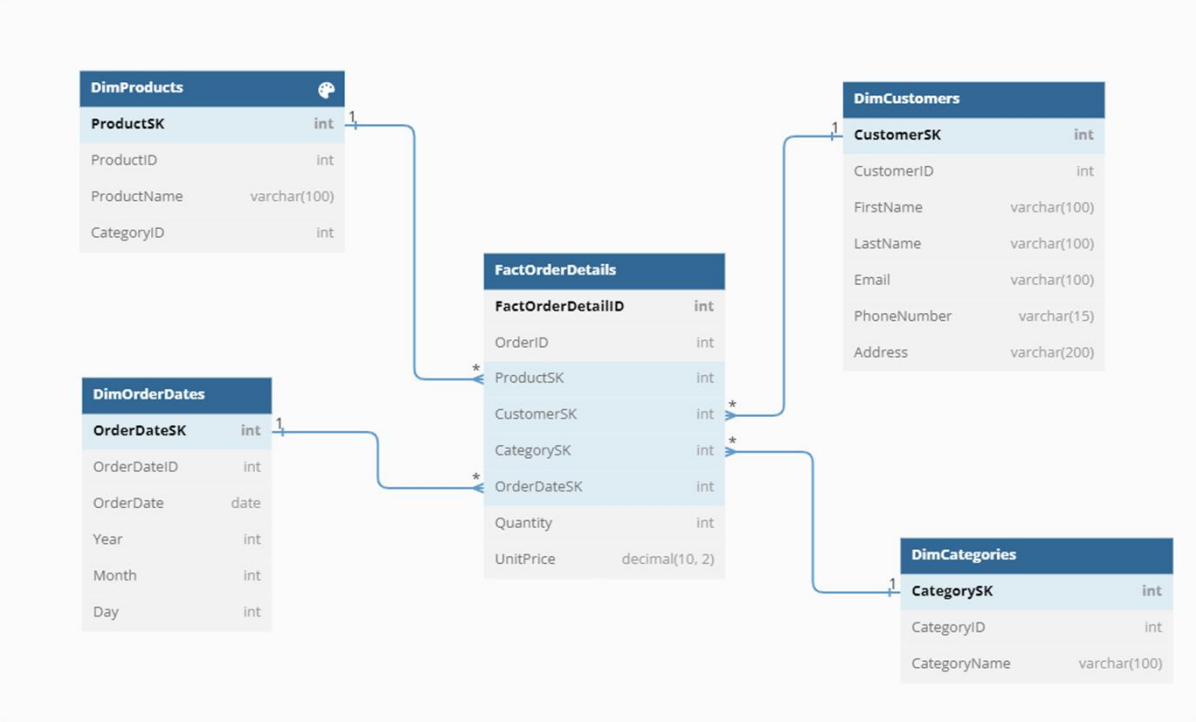


Outras Restrições de Integridade

- NOT NULL
- UNIQUE
- CHECK
- DEFAULT
- INDEX

Modelo Dimensional

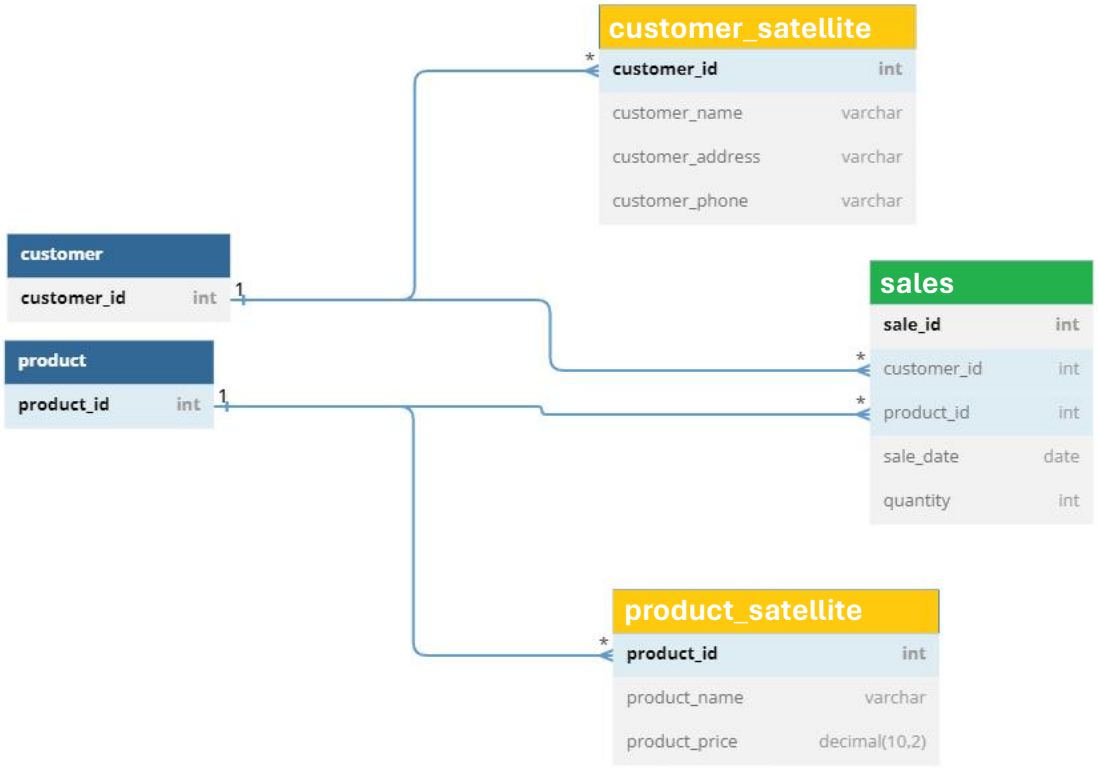
- O Fato é o dado central, é o tema do qual se quer analisar
- Dimensões: São os diversos pontos de vista sobre o qual se quer analisar o fato. Uma dimensão tempo é obrigatória
- Medidas: São valores que serão analisados



Flat Table

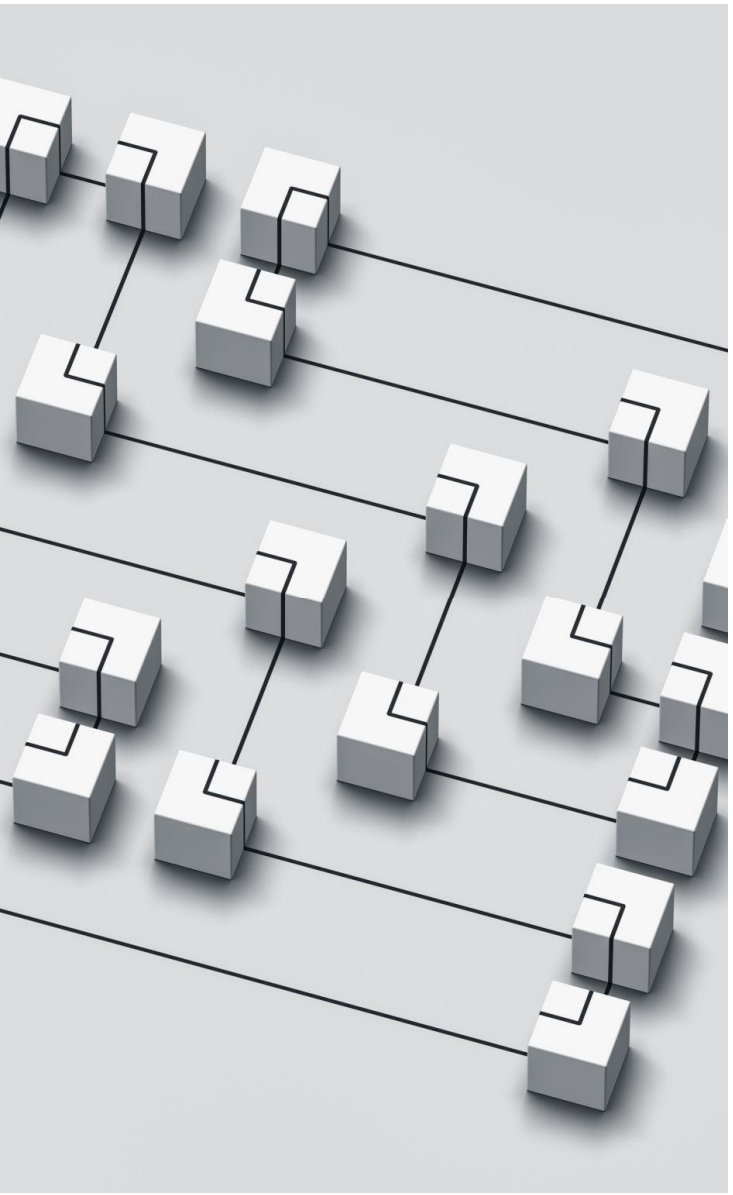
- Todas as informações em um única tabela
- Elimina necessidade de junções
- Redundância de dados
- Normalmente utilizada em DW:
  - Views
  - Views Materializadas

Data Vault



Hubs: Lista de chaves
Links: Estabelece a Relação entre Hubs
Satellite: Informação adicional sobre um hub ou link

	OLTP - Sistemas	OLAP – Data Warehouse
Objetivo	Operações	Análise
Otimizado para	Inserção, Atualização	Leitura
Forma	Transacional	Analítica
Modelo	Relacional	Dimensional
Integridade Referencial	Forte	Pouca ou Nenhuma
Armazenamento	Por Linha	Por Coluna
Usuários	Operacionais	Gerentes, coordenadores etc.
Tipos de Consultas	Simples	Complexas
Volume de Dados	Pequeno	Grande
Técnicas para Otimização	Índices	Clusters, Sort Keys, Partition Keys
Natureza dos Dados	Atuais e Detalhados	Históricos e menos detalhados
Exemplos:	SQL Server, Postgres, MySQL	Redshift, Snowflake, BigQuery



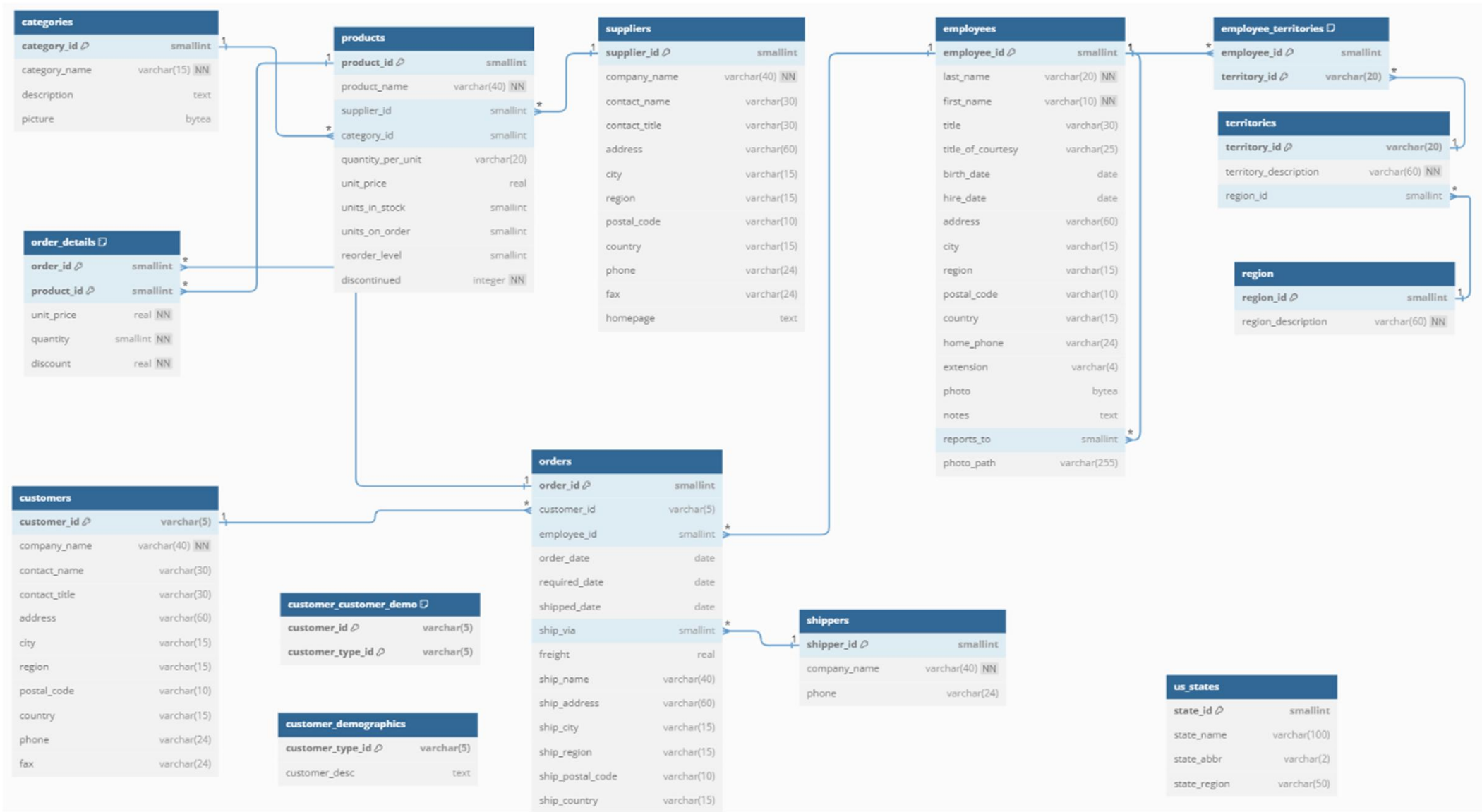
Modelos de Dados

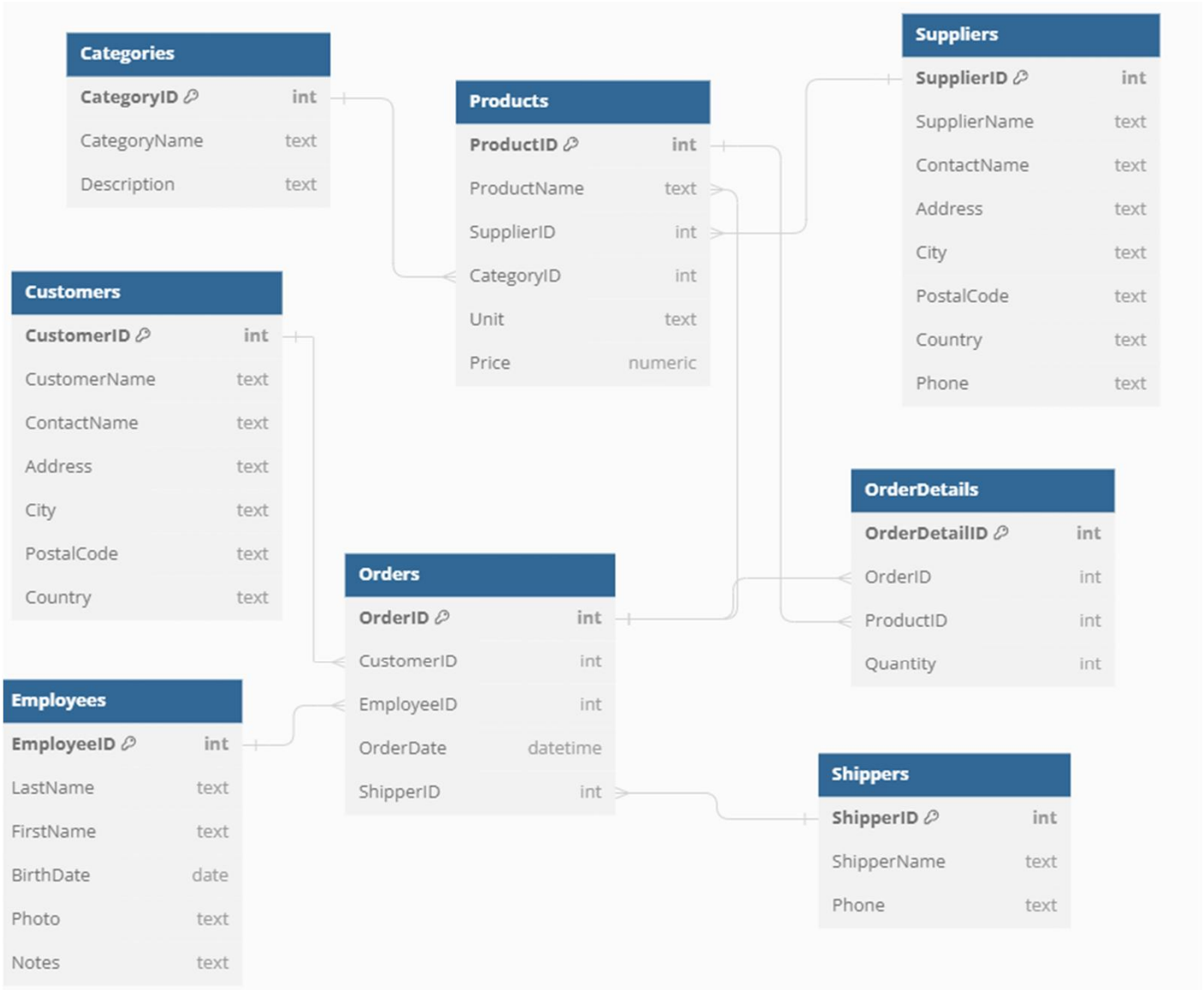
- Banco de Dados Northwind
  - Aulas:
    - Postgres instalado no seu computador
    - Scritps disponíveis para download
  - Atividades de Codificação
    - SQLite: Executado diretamente na plataforma
    - Simplificado
    - Pequenas diferenças no Modelo

- Próxima Aula: Modelo Northwind
- Depois: Modelo Northwind do SQLite

Northwind Traders

- Banco de dados de exemplo
- Importação e Exportação de Alimentos


categories
category_id smallint
category_name varchar(15) NN
description text
picture bytea
products
product_id smallint
product_name varchar(40) NN
supplier_id smallint
category_id smallint
quantity_per_unit varchar(20)
unit_price real
units_in_stock smallint
units_on_order smallint
reorder_level smallint
discontinued integer NN
suppliers
supplier_id smallint
company_name varchar(40) NN
contact_name varchar(30)
contact_title varchar(30)
address varchar(60)
city varchar(15)
region varchar(15)
postal_code varchar(10)
country varchar(15)
phone varchar(24)
fax varchar(24)
homepage text
employees
employee_id smallint
last_name varchar(20) NN
first_name varchar(10) NN
title varchar(30)
title_of_courtesy varchar(25)
birth_date date
hire_date date
address varchar(60)
city varchar(15)
region varchar(15)
postal_code varchar(10)
country varchar(15)
home_phone varchar(24)
extension varchar(4)
photo bytea
notes text
reports_to smallint
photo_path varchar(255)
employee_territories
employee_id smallint
territory_id varchar(20)
territories
territory_id varchar(20)
territory_description varchar(60) NN
region_id smallint
region
region_id smallint
region_description varchar(60) NN
order_details
order_id smallint
product_id smallint
unit_price real NN
quantity smallint NN
discount real NN
orders
order_id smallint
customer_id varchar(5)
employee_id smallint
order_date date
required_date date
shipped_date date
ship_via smallint
freight real
ship_name varchar(40)
ship_address varchar(60)
ship_city varchar(15)
ship_region varchar(15)
ship_postal_code varchar(10)
ship_country varchar(15)
customers
customer_id varchar(5)
company_name varchar(40) NN
contact_name varchar(30)
contact_title varchar(30)
address varchar(60)
city varchar(15)
region varchar(15)
postal_code varchar(10)
country varchar(15)
phone varchar(24)
fax varchar(24)
customer_customer_demo
customer_id varchar(5)
customer_type_id varchar(5)
customer_demographics
customer_type_id varchar(5)
customer_desc text
shippers
shipper_id smallint
company_name varchar(40) NN
phone varchar(24)
us_states
state_id smallint
state_name varchar(100)
state_abbr varchar(2)
state_region varchar(50)

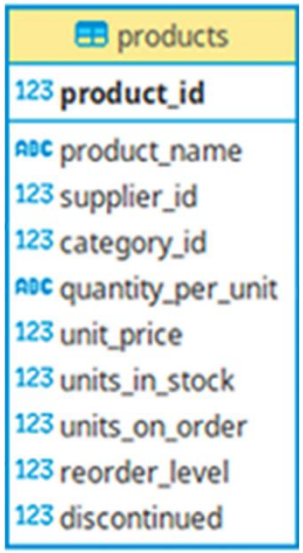

Categories
CategoryID int
CategoryName text
Description text
Products
ProductID int
ProductName text
SupplierID int
CategoryID int
Unit text
Price numeric
Suppliers
SupplierID int
SupplierName text
ContactName text
Address text
City text
PostalCode text
Country text
Phone text
Customers
CustomerID int
CustomerName text
ContactName text
Address text
City text
PostalCode text
Country text
OrderDetails
OrderDetailID int
OrderID int
ProductID int
Quantity int
Orders
OrderID int
CustomerID int
EmployeeID int
OrderDate datetime
ShipperID int
Employees
EmployeeID int
LastName text
FirstName text
BirthDate date
Photo text
Notes text
Shippers
ShipperID int
ShipperName text
Phone text

2.Fundamentos de SQL para Consultas

Consulta: Query

- Objetivo é recuperar um conjunto de dados tabulares, dados certos critérios
- Os dados pode prover de uma ou mais tabelas

Básico




```
SELECT
Coluna1,Coluna2 ... [*]
FROM
Tabela;
```

Básico

```
--consulta 1
SELECT
*
FROM
Products;

/* consulta 2
comentário de n linhas */
SELECT
product_name, unit_price
FROM
Products;
```

Where

Normalmente precisamos de um subconjunto dos dados

Precisamos informar ao SGBD como obter este subconjunto (condição)

Where

```sql
--Básico
SELECT
product_name, unit_price
FROM
products
WHERE
product_name = 'Geitost';
--Condições <,>,<>,<=,>=,=
SELECT
product_name, unit_price
FROM
products
WHERE
unit_price > 100;
```

Where

```sql
--Like %
SELECT
product_name, unit_price
FROM
products
WHERE
product_name like '%Bl%';

--Like _
SELECT
product_name, unit_price
FROM
products
WHERE
product_name like '%Blay_';
```

Where

```sql
-- ILike: Case Insensitive: b minúsculo
SELECT
product_name, unit_price
FROM
products
WHERE
product_name ilike '%blay_';

-- Not Like
SELECT
product_name, unit_price
FROM
products
WHERE
product_name NOT like '%Blay_';
```

Where

```sql
-- Between
SELECT
product_name, unit_price
FROM
products
WHERE
unit_price BETWEEN 50 AND 100;

-- IN
SELECT
product_name, quantity_per_unit
FROM
products
WHERE
quantity_per_unit
IN ('24 - 12 oz bottles','2 kg box');
```

Where

```sql
-- NOT IN
SELECT
product_name, quantity_per_unit
FROM
products
WHERE
quantity_per_unit
NOT IN ('24 - 12 oz bottles','2 kg box');
```

AND OR
NOT

Podemos ter condições que incluam mais de uma coluna

Usamos operadores lógicos para criar a cláusula

AND OR NOT

AND – todas as condições devem ser verdadeiras

OR – pelo menos uma das condições devem ser verdadeiras

NOT – seleciona registros que não atendem a condição

AND OR NOT

```sql
--AND
SELECT
product_name,unit_price,units_in_stock
FROM
products
WHERE
unit_price = 11
AND units_in_stock <=5;

--OR
SELECT
product_name,unit_price,units_in_stock
FROM
products
WHERE
unit_price = 11
OR units_in_stock <=5;
```

AND OR NOT

```sql
--NOT
SELECT
product_name,unit_price,units_in_stock
FROM
products
WHERE
NOT unit_price = 11;

--AND OR
-- AVALIA PRIMERO AND DEPOIS OR
SELECT
product_name,unit_price,units_in_stock
FROM
products
WHERE
unit_price = 10
AND units_in_stock <=5
OR discontinued = 1;
```

AND OR NOT

```sql
--AND OR
SELECT
product_name,unit_price,units_in_stock
FROM
products
WHERE
unit_price = 10
OR units_in_stock <=5
AND discontinued = 1;

--PARENTESES
SELECT
product_name,unit_price,units_in_stock
FROM
products
WHERE
unit_price = 10
OR (units_in_stock <=5
AND discontinued = 1);
```

Alias

```sql
-- alias para tabela
SELECT
prod.product_name ,
prod.unit_price ,
prod.units_in_stock
FROM
products  prod
WHERE
unit_price = 10
OR discontinued = 1

--alias para coluna
SELECT
prod.product_name as "Nome do Produto" ,
prod.unit_price Preço,
prod.units_in_stock Stock
FROM
products prod
WHERE
unit_price = 10
OR discontinued = 1
```

Funções de Agregação

Função	Descrição
AVG	Calcula a média dos valores em um conjunto.
SUM	Soma os valores em um conjunto.
COUNT	Conta o número de itens em um conjunto.
MIN	Encontra o menor valor em um conjunto.
MAX	Encontra o maior valor em um conjunto.

Funções de Agregação

```sql
-- avg
SELECT
AVG(prod.unit_price) "Preço Médio"
FROM
products as prod;

--sum
SELECT
SUM(prod.unit_price) "Soma dos Preços"
FROM
products as prod;
```

Funções de Agregação

```sql
--count
SELECT
COUNT(prod.unit_price) "Quantidade de Produtos"
FROM
products as prod;

--min
SELECT
MIN(prod.unit_price) "Menor Preço"
FROM
products as prod;
```

Funções de Agregação

```sql
--Max
SELECT
MAX(prod.unit_price) "Maior Preço"
FROM
products as prod;

SELECT
AVG(prod.unit_price) "Preço Médio"
FROM
products as prod;
```

Funções de Agregação

```sql
SELECT
AVG(prod.unit_price) "Preço Médio",
AVG(prod.units_in_stock) "Estoque Médio"
FROM
products as prod;
```

Order by, Limit, Offset

```sql
--order by
SELECT
product_id,product_name, unit_price
FROM
products
order by unit_price;

--decrescente
SELECT
product_id,product_name, unit_price
FROM
products
order by unit_price desc
```

Order by, Limit, Offset

```sql
--asc padrão
SELECT
product_id,product_name, unit_price
FROM
products
order by unit_price asc;

--limit
SELECT
product_id,product_name, unit_price
FROM
products
limit 5;
```

Order by, Limit, Offset

```sql
--order by, limit
SELECT
product_id,product_name, unit_price
FROM
products
order by unit_price desc
limit 5;

--offset  primeiro 5
SELECT
product_id,product_name, unit_price
FROM
products
order by unit_price desc
limit 5
```

Order by, Limit, Offset

```sql
--offset próximos 5
SELECT
product_id,product_name, unit_price
FROM
    products
ORDER BY unit_price DESC
LIMIT 5 OFFSET 5;

--próximos 5
SELECT
product_id,product_name, unit_price
FROM
    products
ORDER BY unit_price DESC
LIMIT 5 OFFSET 10;
```

Order by, Limit, Offset

```sql
--próximos 5
SELECT
product_id,product_name, unit_price
FROM
    products
ORDER BY unit_price DESC
LIMIT 5 OFFSET 15;

--20 produtos
SELECT
product_id,product_name, unit_price
FROM
products
order by unit_price desc
limit 20
```

Order by, Limit, Offset

```sql
--mais de uma coluna
select
product_id,units_in_stock, unit_price
from
products
order by units_in_stock asc, unit_price desc
```

Group By e Having

```sql
--group by
SELECT
country, count(country) total
FROM
customers
group by country;

--group by
SELECT
country,city, count(city) total
FROM
customers
group by country,city
order by total desc;
```

Group By e Having

```sql
--having
SELECT
country,city, count(city) total
FROM
customers
group by country,city
having count(city) > 2
order by total desc;
```

Distinct

```
SELECT
country, city
FROM
suppliers;

--distinct
SELECT
distinct country
FROM
suppliers;

--distinct
SELECT
distinct country, city
FROM
suppliers
order by country;
```

Valores Nulos

- Ausência de valor
- Diferente de:
  - '' Texto vazio
  - ' ' Espaço
  - Zero
- Precisamos Tratar de Forma Especial Nulos

Valores Nulos

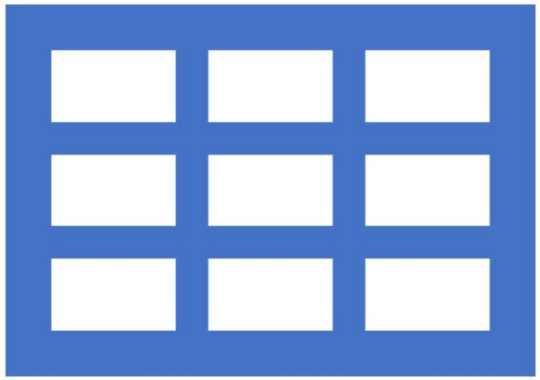
```sql
--total
SELECT
count(1)
FROM
suppliers;

--não nulos
SELECT
count(1)
FROM
suppliers
WHERE region IS NOT NULL;
```

Valores Nulos

```sql
--nulos
SELECT
count(1)
FROM
suppliers
WHERE region IS NULL;

--não é o mesmo que vazio
SELECT
count(1)
FROM
suppliers
WHERE region = '' or region = ' '
```

Valores Nulos

```sql
--regiões nulas
SELECT
    COALESCE(region, 'N/A') AS region
FROM
Suppliers;

--coalesce
SELECT
    region AS region
FROM
Suppliers;
```



UNION e UNION ALL

Union: retorna os registros de uma ou mais tabelas, elimina registros repetidos

Union all: retorna os registros de uma ou mais tabelas, inclusive registros repetidos

INTERSECT

- INTERSECT retorna as linhas que são resultados comuns a duas ou mais consultas.

EXCEPT

- EXCEPT retorna as linhas da primeira consulta que não existem nas consultas subsequentes.

UNION e UNION ALL

```sql
--union, sem duplicados
select distinct region from suppliers
union
select distinct region from customers

-- union all, incluindo duplicatas
select distinct region from suppliers
union all
select distinct region from customers
```

UNION e UNION ALL

```
--INTERSECT retorna as linhas que são resultados comuns
select distinct region from suppliers
INTERSECT
select distinct region from customers

--EXCEPT retorna as linhas da primeira consulta que não existem na segunda
select distinct region from suppliers
EXCEPT
select distinct region from customers;
--
select distinct region from customers
EXCEPT
select distinct region from suppliers;
```

Combinando Tudo

```sql
SELECT
    p.category_id AS "ID da Categoria",
    COUNT(p.product_id) AS "Quantidade de Produtos",
    AVG(p.unit_price) AS "Preço Médio",
    MAX(p.unit_price) AS "Preço Máximo"
FROM
    products p
WHERE
    p.discontinued = 0 AND
    p.product_name LIKE 'C%'
GROUP BY
    p.category_id
HAVING
    AVG(p.unit_price) > 50
ORDER BY
    "Quantidade de Produtos" DESC
LIMIT 5;
```

3.Dominando Joins

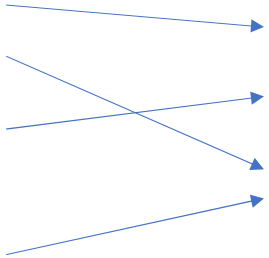
Junções

funcionarios		
emp_id	emp_nome	dept_id
1	Alice	1
2	Bob	3
3	Clara	2
4	David	
5	Eva	3

departamentos	
dept_id	dept_nome
1	RH
2	Marketing
3	TI
4	Financeiro

INNER JOIN

funcionarios		
emp_id	emp_nome	dept_id
1	Alice	1
2	Bob	3
3	Clara	2
4	David	
5	Eva	3



departamentos	
dept_id	dept_nome
1	RH
2	Marketing
3	TI
4	Financeiro

emp_id	emp_nome	dept_nome
1	Alice	RH
2	Bob	TI
3	Clara	Marketing
5	Eva	TI

LEFT JOIN

funcionarios		
emp_id	emp_nome	dept_id
1	Alice	1
2	Bob	3
3	Clara	2
4	David	
5	Eva	3

departamentos	
dept_id	dept_nome
1	RH
2	Marketing
3	TI
4	Financeiro

emp_id	emp_nome	dept_nome
1	Alice	RH
2	Bob	TI
3	Clara	Marketing
4	David	
5	Eva	TI

RIGHT JOIN

funcionarios		
emp_id	emp_nome	dept_id
1	Alice	1
2	Bob	3
3	Clara	2
4	David	
5	Eva	3

departamentos	
dept_id	dept_nome
1	RH
2	Marketing
3	TI
4	Financeiro

emp_id	emp_nome	dept_nome
1	Alice	RH
2	Bob	TI
3	Clara	Marketing
5	Eva	TI
		Financeiro

FULL OUTER JOIN

funcionarios		
emp_id	emp_nome	dept_id
1	Alice	1
2	Bob	3
3	Clara	2
4	David	
5	Eva	3

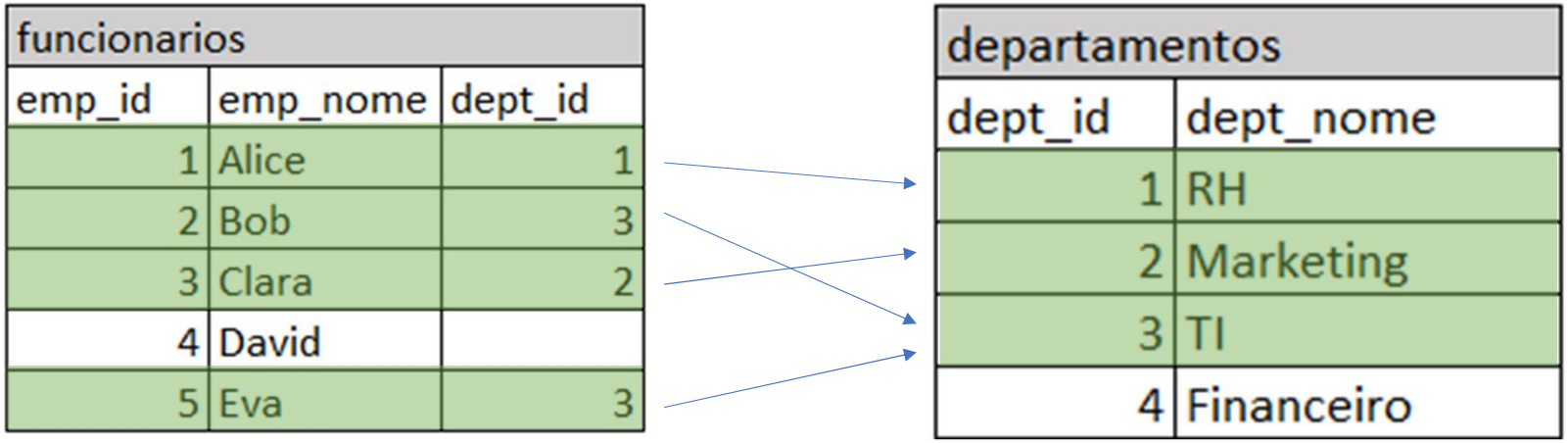
departamentos	
dept_id	dept_nome
1	RH
2	Marketing
3	TI
4	Financeiro

emp_id	emp_nome	dept_nome
1	Alice	RH
2	Bob	TI
3	Clara	Marketing
4	David	
5	Eva	TI
		Financeiro

CROSS JOIN

- Produto Cartesiano
- Lista todas as combinações possíveis
- Pode ter resultados imprevisíveis
- Pouco usual

INNER JOIN

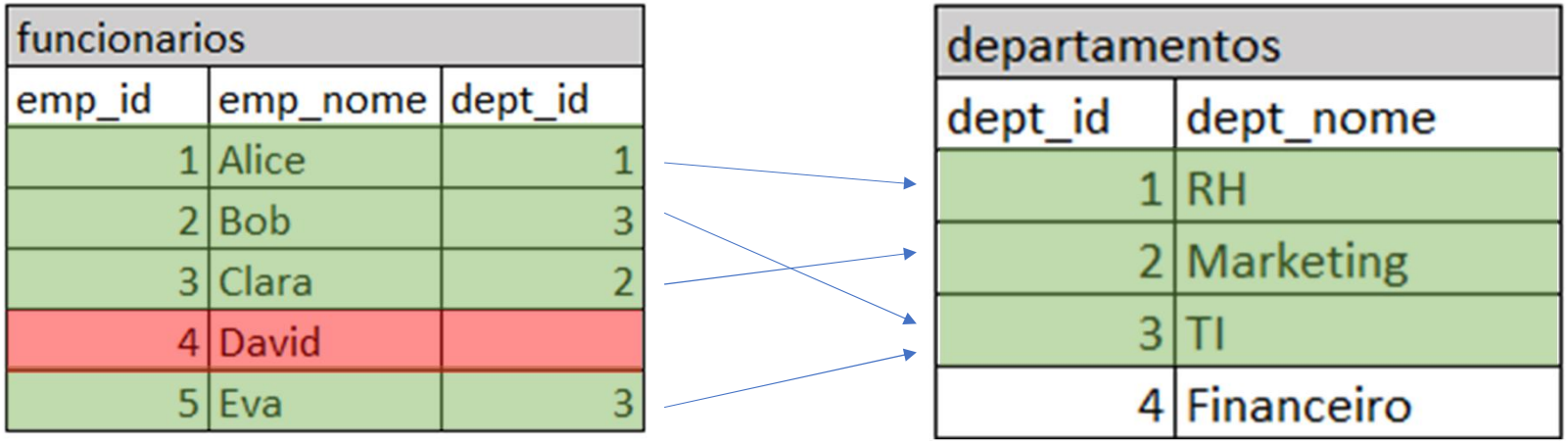


funcionarios		
emp_id	emp_nome	dept_id
1	Alice	1
2	Bob	3
3	Clara	2
4	David	
5	Eva	3

departamentos	
dept_id	dept_nome
1	RH
2	Marketing
3	TI
4	Financeiro

```sql
SELECT f.emp_id, f.emp_nome, d.dept_nome
FROM funcionarios f
INNER JOIN departamentos d ON f.dept_id = d.dept_id;
```

LEFT JOIN

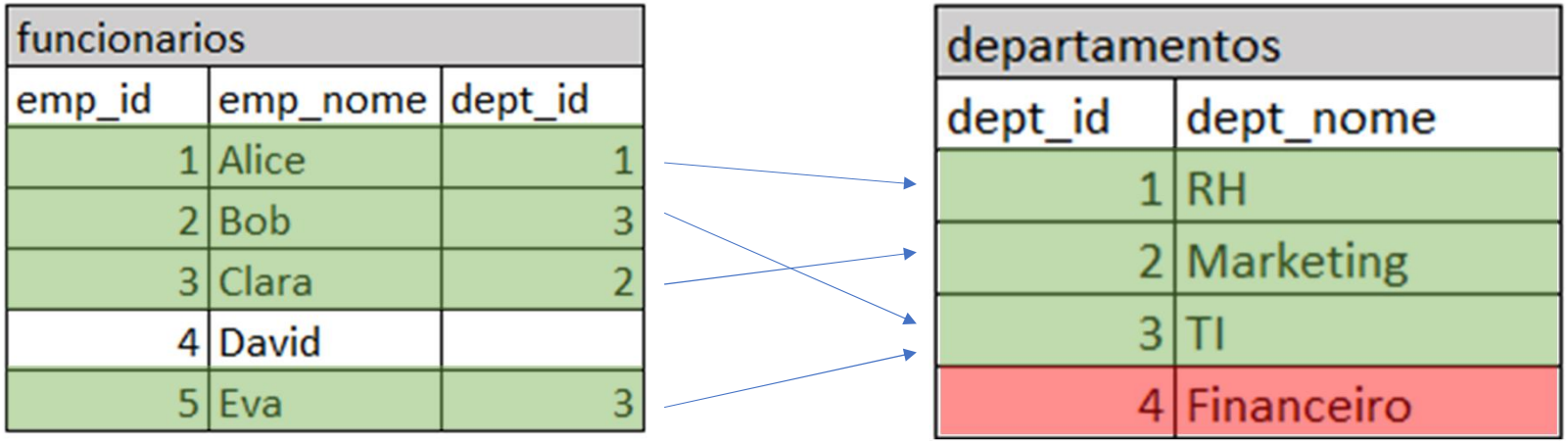


funcionarios		
emp_id	emp_nome	dept_id
1	Alice	1
2	Bob	3
3	Clara	2
4	David	
5	Eva	3

departamentos	
dept_id	dept_nome
1	RH
2	Marketing
3	TI
4	Financeiro

```sql
SELECT f.emp_id, f.emp_nome, d.dept_nome
FROM funcionarios f
LEFT JOIN departamentos d ON f.dept_id = d.dept_id;
```

RIGHT JOIN

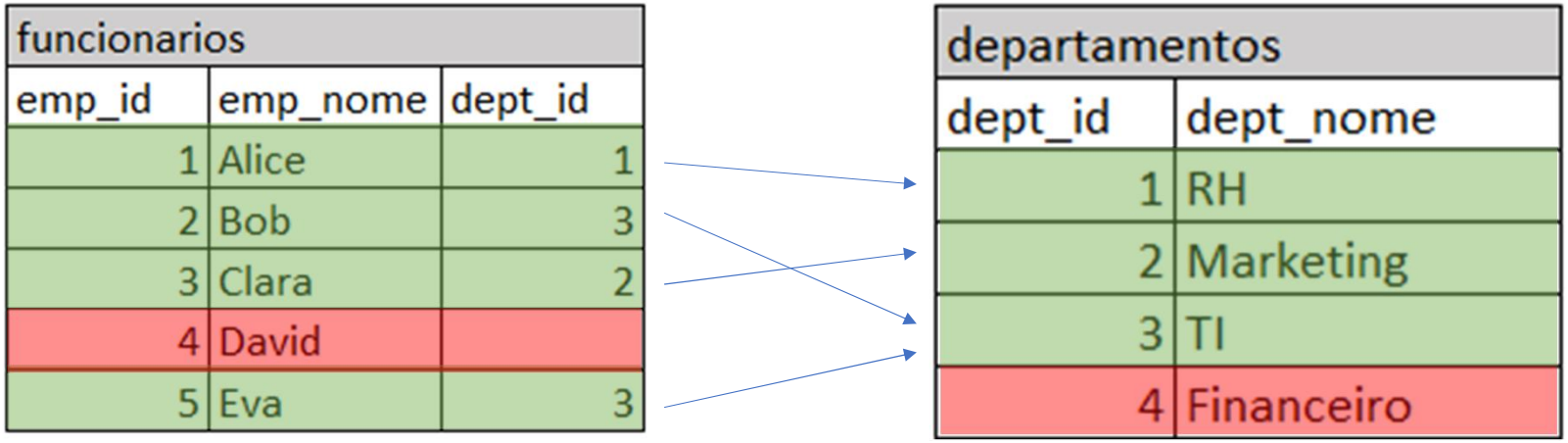


funcionarios		
emp_id	emp_nome	dept_id
1	Alice	1
2	Bob	3
3	Clara	2
4	David	
5	Eva	3

departamentos	
dept_id	dept_nome
1	RH
2	Marketing
3	TI
4	Financeiro

```sql
SELECT f.emp_id, f.emp_nome, d.dept_nome
FROM funcionarios f
RIGHT JOIN departamentos d ON f.dept_id = d.dept_id;
```

FULL OUTER JOIN



funcionarios		
emp_id	emp_nome	dept_id
1	Alice	1
2	Bob	3
3	Clara	2
4	David	
5	Eva	3

departamentos	
dept_id	dept_nome
1	RH
2	Marketing
3	TI
4	Financeiro

```sql
SELECT f.emp_id, f.emp_nome, d.dept_nome
FROM funcionarios f
FULL OUTER JOIN departamentos d ON f.dept_id = d.dept_id;
```

CROSS JOIN

```sql
SELECT f.emp_id, f.emp_nome, d.dept_id, d.dept_nome
FROM funcionarios f
CROSS JOIN departamentos d;
```

Joins

```sql
--inner
SELECT f.emp_id, f.emp_nome, d.dept_nome
FROM funcionarios f
INNER JOIN departamentos d ON f.dept_id = d.dept_id;

--left
SELECT f.emp_id, f.emp_nome, d.dept_nome
FROM funcionarios f
LEFT JOIN departamentos d ON f.dept_id = d.dept_id;
```

Joins

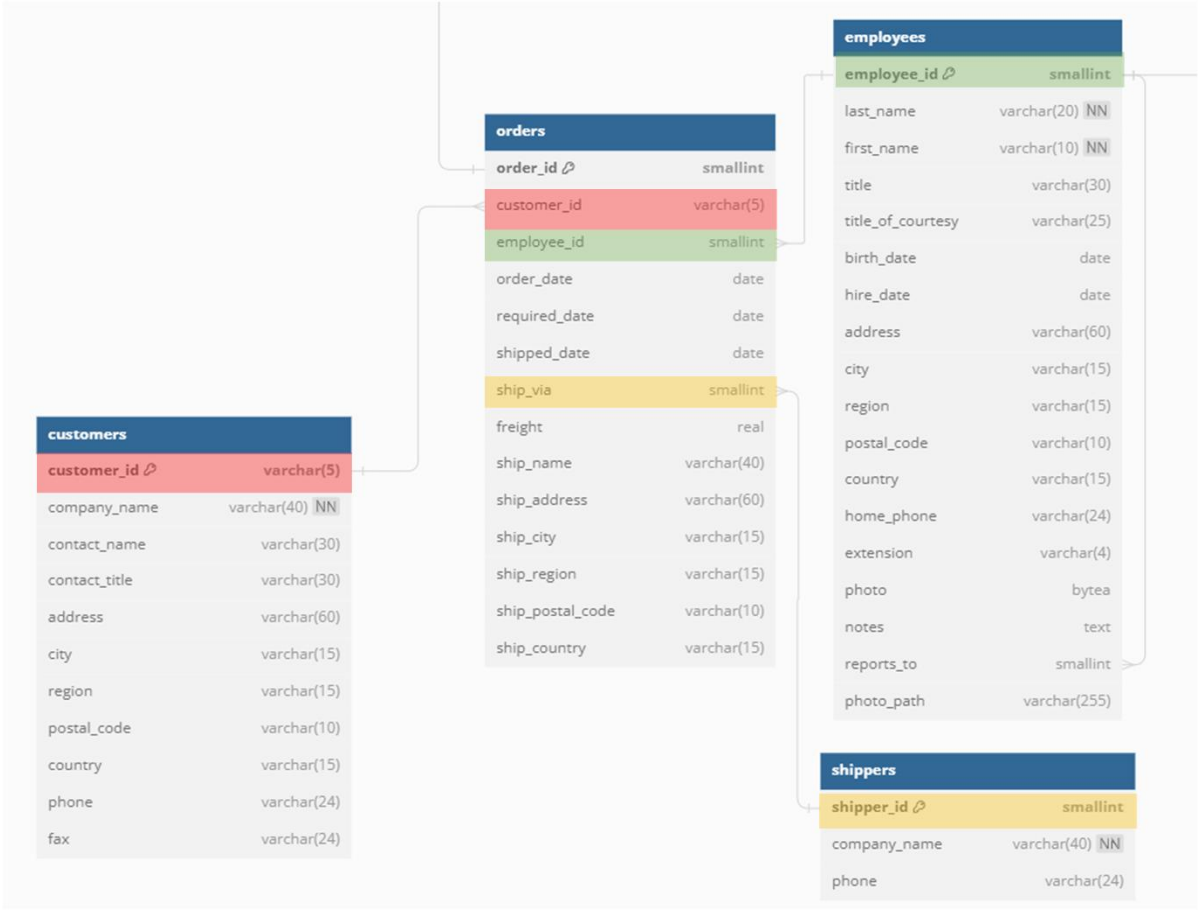
```sql
--right
SELECT f.emp_id, f.emp_nome, d.dept_nome
FROM funcionarios f
RIGHT JOIN departamentos d ON f.dept_id = d.dept_id;

--full outer
SELECT f.emp_id, f.emp_nome, d.dept_nome
FROM funcionarios f
FULL OUTER JOIN departamentos d ON f.dept_id = d.dept_id;
```

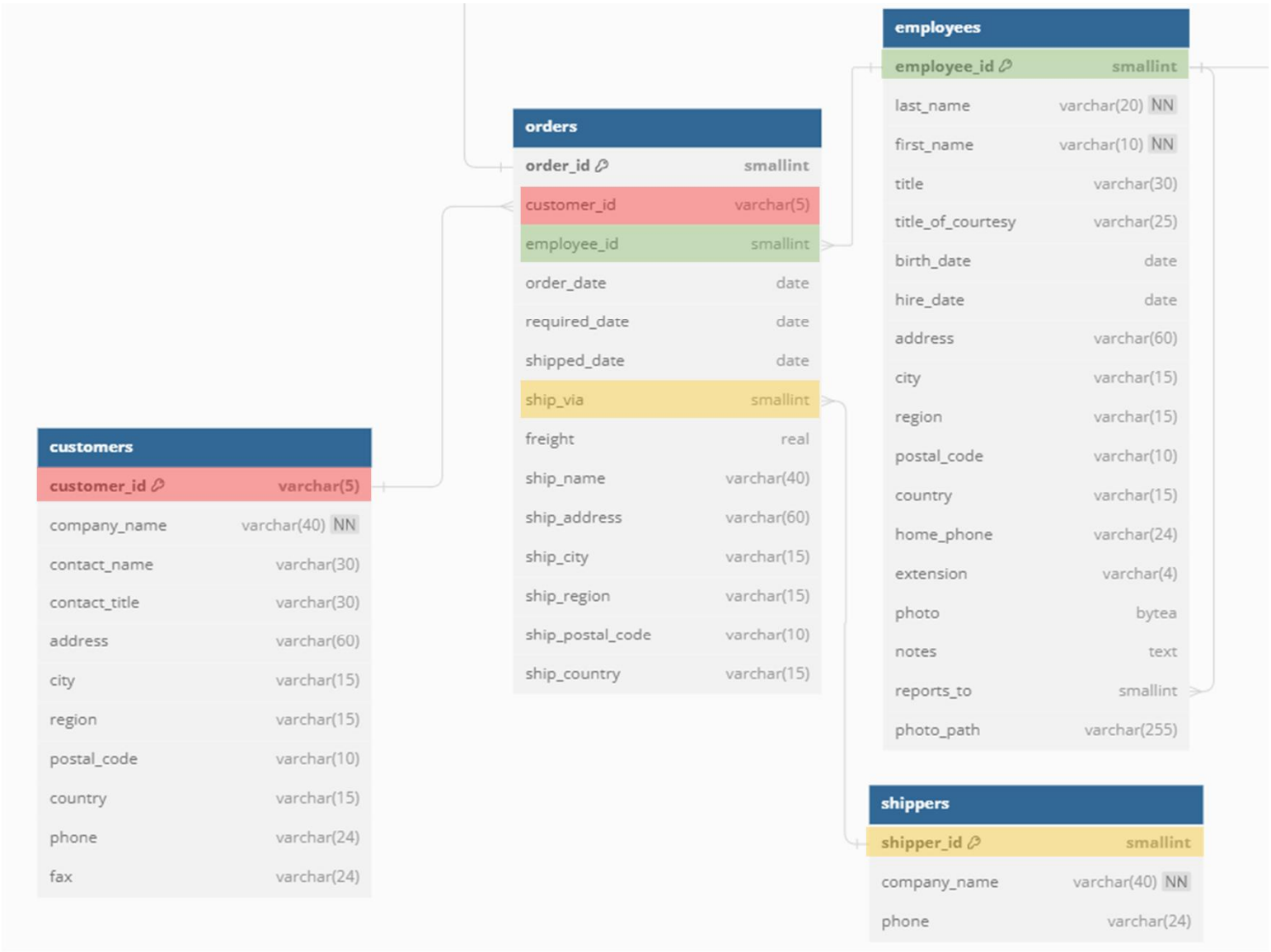
Joins

```sql
--cross
SELECT f.emp_id, f.emp_nome, d.dept_id, d.dept_nome
FROM funcionarios f
CROSS JOIN departamentos d;
```

Joins

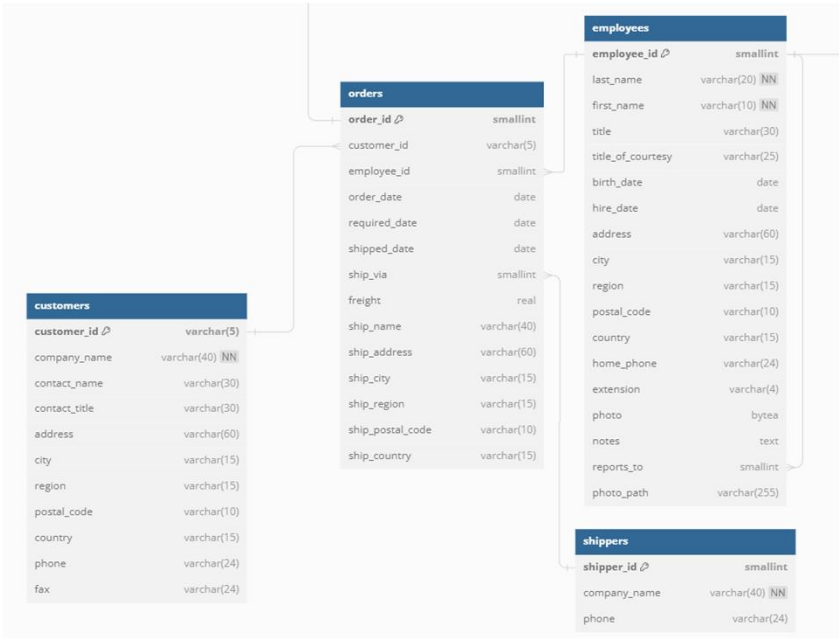
Trazer Pedidos, com nome do funcionário, nome do cliente e nome da transportadora




customers
customer_id varchar(5)
company_name varchar(40) NN
contact_name varchar(30)
contact_title varchar(30)
address varchar(60)
city varchar(15)
region varchar(15)
postal_code varchar(10)
country varchar(15)
phone varchar(24)
fax varchar(24)
orders
order_id smallint
customer_id varchar(5)
employee_id smallint
order_date date
required_date date
shipped_date date
ship_via smallint
freight real
ship_name varchar(40)
ship_address varchar(60)
ship_city varchar(15)
ship_region varchar(15)
ship_postal_code varchar(10)
ship_country varchar(15)
employees
employee_id smallint
last_name varchar(20) NN
first_name varchar(10) NN
title varchar(30)
title_of_courtesy varchar(25)
birth_date date
hire_date date
address varchar(60)
city varchar(15)
region varchar(15)
postal_code varchar(10)
country varchar(15)
home_phone varchar(24)
extension varchar(4)
photo bytea
notes text
reports_to smallint
photo_path varchar(255)
shippers
shipper_id smallint
company_name varchar(40) NN
phone varchar(24)

Joins

```
SELECT
    o.order_id,
    o.order_date,
    o.shipped_date,
    c.company_name AS customer_name,
    e.first_name || ' ' || e.last_name AS employee_name,
    s.company_name AS shipper_name
FROM
    orders o
LEFT JOIN customers c ON o.customer_id = c.customer_id
LEFT JOIN employees e ON o.employee_id = e.employee_id
LEFT JOIN shippers s ON o.ship_via = s.shipper_id;
```

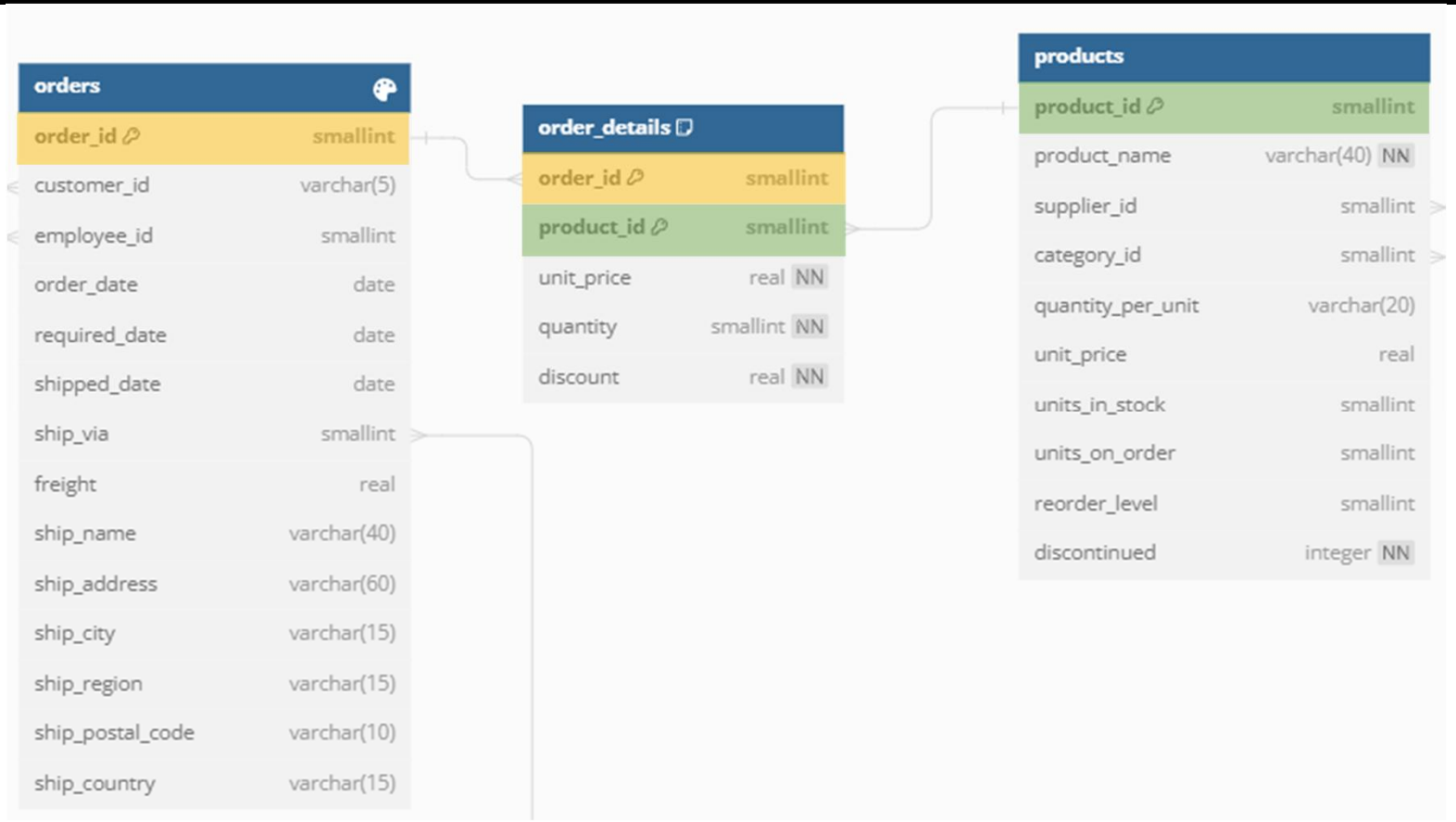


Joins

```sql
SELECT
    o.order_id AS "Número do Pedido",
    o.order_date AS "Data do Pedido",
    o.shipped_date AS "Data de Envio",
    c.company_name AS "Cliente",
    e.first_name || ' ' || e.last_name AS "Funcionário",
    s.company_name AS "Transportadora"
FROM
    orders o
INNER JOIN customers c ON o.customer_id = c.customer_id
LEFT JOIN employees e ON o.employee_id = e.employee_id
LEFT JOIN shippers s ON o.ship_via = s.shipper_id;
```

Joins N para N

Trazer Pedidos, com nome dos produtos

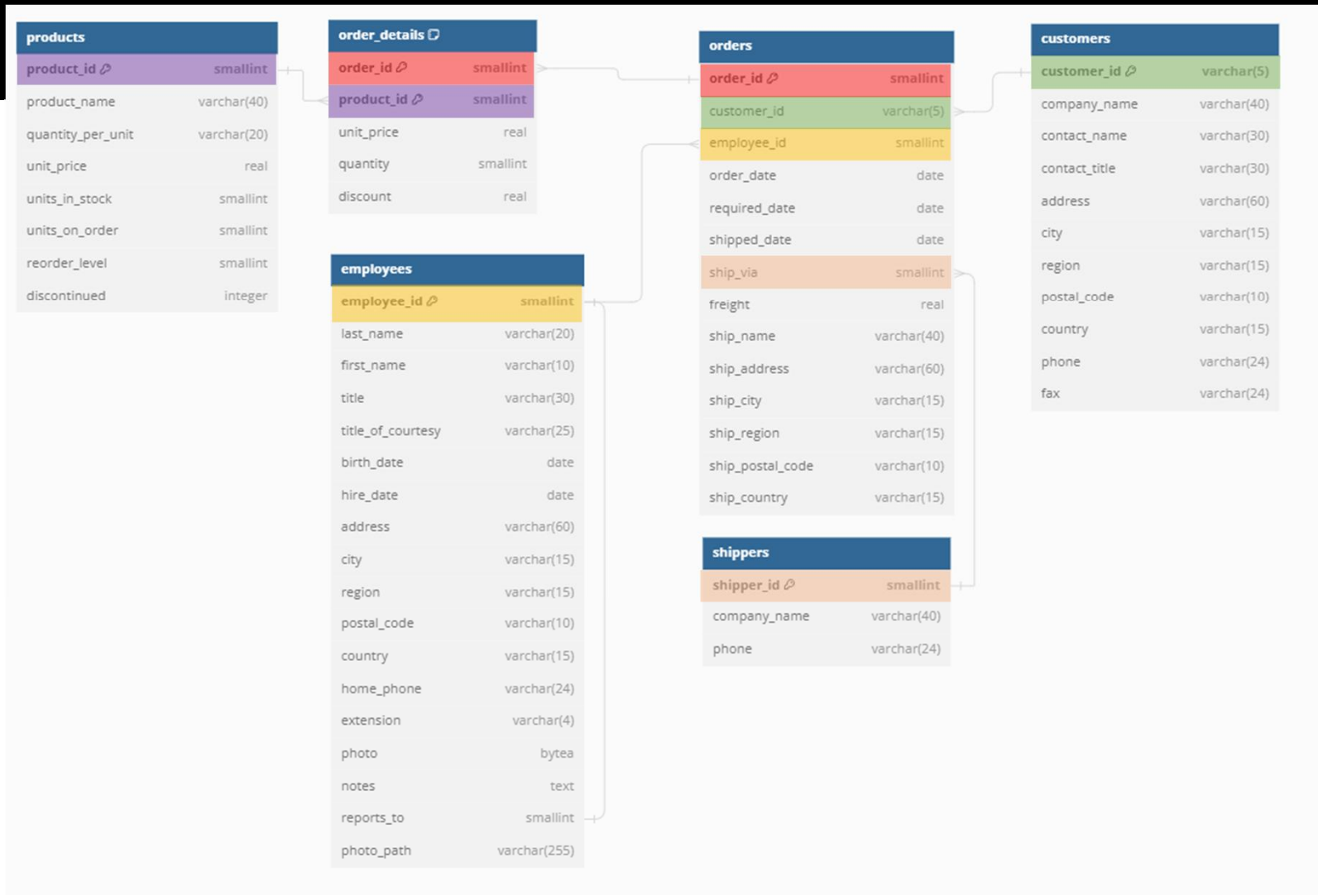


Joins N para N

```sql
SELECT
    o.order_id AS "Código do Pedido",
    o.order_date AS "Data do Pedido",
    p.product_name AS "Produto",
    d.unit_price AS "Preço Unitário",
    d.quantity AS "Quantidade",
    d.discount AS "Desconto"
FROM
    orders o
JOIN order_details d ON o.order_id = d.order_id
JOIN products p ON d.product_id = p.product_id
order by 1
```

Joins

Relatório detalhado de pedidos, incluindo cliente, funcionário, produtos com preços, quantidades e descontos, transportadora e data de envio



Joins

```sql
SELECT
    o.order_id AS "Código do Pedido",
    c.company_name AS "Cliente",
    e.first_name || ' ' || e.last_name AS "Funcionário",
    p.product_name AS "Produto",
    od.unit_price AS "Preço Unitário",
    od.quantity AS "Quantidade",
    od.discount AS "Desconto",
    s.company_name AS "Transportadora",
    o.shipped_date AS "Data de Envio"
FROM
    orders o
INNER JOIN customers c ON o.customer_id = c.customer_id
INNER JOIN employees e ON o.employee_id = e.employee_id
INNER JOIN order_details od ON o.order_id = od.order_id
INNER JOIN products p ON od.product_id = p.product_id
LEFT JOIN shippers s ON o.ship_via = s.shipper_id
ORDER BY 1;
```

Self Join


employees
employee_id smallint
last_name varchar(20) NN
first_name varchar(10) NN
title varchar(30)
title_of_courtesy varchar(25)
birth_date date
hire_date date
address varchar(60)
city varchar(15)
region varchar(15)
postal_code varchar(10)
country varchar(15)
home_phone varchar(24)
extension varchar(4)
photo bytea
notes text
reports_to smallint
photo_path varchar(255)
1
1
*

Self Join

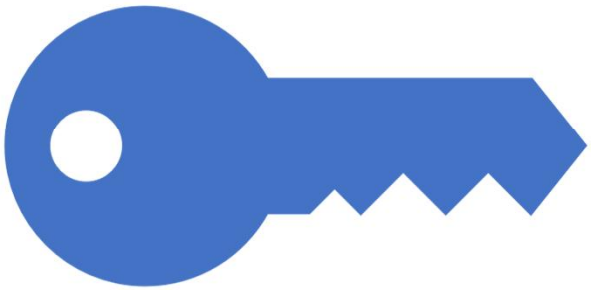
```sql
-- Self Join para Hierarquia
SELECT
    a.employee_id AS "Id do Funcionário",
    a.first_name || ' ' || a.last_name  AS "Nome do Funcionário",
    b.employee_id AS "Id do Supervisor",
    b.first_name || ' ' || b.last_name  AS "Nome do Supervisor"
FROM
    employees a
LEFT JOIN employees b ON a.reports_to = b.employee_id;
```

4.Insert, Update, Delete

Inserção de Dados

- Integridades devem ser respeitadas:
  - Chaves Primárias
  - Relações FK/PK
  - Colunas NOT NULL
  - Campos do tipo auto-incremento não devem ser incluídos (serial/Identity)*

- Na relação FK/PF:
  - Inclui-se primeiro a campo na tabela pai



Insert

```sql
--insert
INSERT INTO shippers (shipper_id, company_name, phone) VALUES
(7, 'Speedy Express', '555-1234');

--mais de um registro
INSERT INTO shippers (shipper_id, company_name, phone) VALUES
(8, 'My Package', '555-4556'),
(9, 'Travel With Me', '2344-4522');

--duas colunas
INSERT INTO shippers (shipper_id, company_name) VALUES
(10, 'Lets Go');
```

Insert

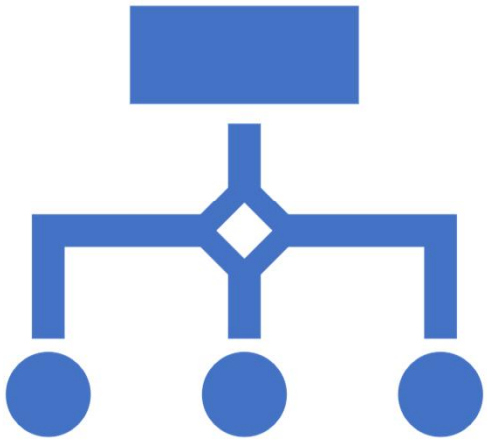
```sql
--sem colunas
INSERT INTO shippers VALUES
(11, 'Max Speed', '1212-5847');

select * from shippers;

--fk
INSERT INTO orders (
  order_id, customer_id, employee_id, order_date, required_date, shipped_date,
  ship_via, freight, ship_name, ship_address, ship_city, ship_region,
  ship_postal_code, ship_country
)
VALUES (
  11078, 'VINET', 5, '2023-09-05', '2023-10-03', '2023-09-08',
  2, 32.38, 'Vins et alcools Chevalier', '59 rue de lAbbaye', 'Reims', NULL,
  '51100', 'France'
);

select * from employees;
```

Insert

```sql
--violaçãode integrade referêncial
INSERT INTO orders (
  order_id, customer_id, employee_id, order_date, required_date, shipped_date,
  ship_via, freight, ship_name, ship_address, ship_city, ship_region,
  ship_postal_code, ship_country
) VALUES (
  11079, 'VINET', 10, '2023-09-05', '2023-10-03', '2023-09-08',
  2, 32.38, 'Vins et alcools Chevalier', '59 rue de lAbbaye', 'Reims', NULL,
  '51100', 'France'
);
```

Insert Into ... Select

- É possível usar Insert Into ... Select para inserir dados de uma tabela em outra
- Pode-se definir colunas tanto do insert quanto do Select
- O Select pode limitar registros usando-se Where...

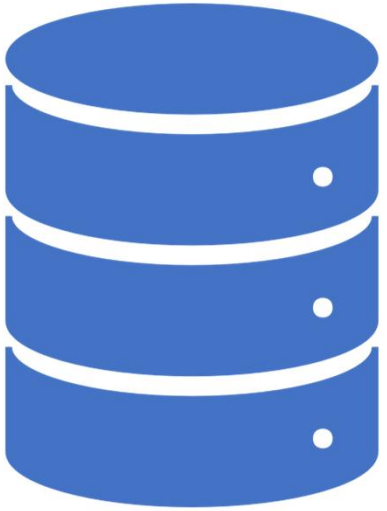


Insert Into Select

```sql
select supplier_id, company_name, phone from suppliers
where company_name = 'Escargots Nouveaux';

select * from shippers;

INSERT INTO shippers (shipper_id, company_name, phone)
SELECT supplier_id, company_name, phone
from suppliers
where company_name = 'Escargots Nouveaux'
```

Atualização

- Definimos colunas e valores que serão atualizados
- Normalmente há uma cláusula where



Update

```sql
--customers
select phone,postal_code from customers
where customer_id = 'ALFKI';

UPDATE customers
SET phone = '555-9999',
postal_code = '12209'
WHERE customer_id = 'ALFKI';
```

Update

```sql
--employees
select title,* from employees
where employee_id = 1;

UPDATE employees
SET title = 'Senior Sales Representative'
WHERE employee_id = 1;
```

Update

```sql
--update sem where
select unit_price from products;

UPDATE products
SET unit_price = unit_price * 1.10;
```

Update

```
--orders
Select freight from orders
WHERE order_date BETWEEN '1997-01-21' AND '1997-04-07'
  AND ship_country = 'USA';

UPDATE orders
SET freight = freight * 1.05
WHERE order_date BETWEEN '1997-01-21' AND '1997-04-07'
  AND ship_country = 'USA';
```

Update

```sql
--orders
UPDATE orders
SET ship_address = NULL
WHERE order_id = 11077;

select ship_address from orders
WHERE order_id = 11077;

UPDATE orders
SET ship_address = (SELECT address FROM customers
          WHERE customer_id = orders.customer_id)
WHERE order_id = 11077;
```

Delete

```sql
DELETE FROM shippers
WHERE shipper_id = 7;

DELETE FROM shippers
WHERE shipper_id between 8 AND 9;

DELETE FROM shippers
WHERE company_name IN ('Lets Go','Max Speed')

DELETE FROM orders
WHERE order_id = 11078
```

Grant, Revoke

Permissões

- Grant e Revoke concedem permissões sobre objetos do banco de dados

Tipos de Permissões

- SELECT
- INSERT
- UPDATE
- DELETE
- TRUNCATE
- REFERENCES: permite a criação de FKs
- TRIGGER: permite criar trigger na tabela
- CONNECT: permite conectar com banco de dados específico
- USAGE: permite o uso de schemas
- CREATE: permite a criação de novos objetos
- TEMPORARY/TEMP:

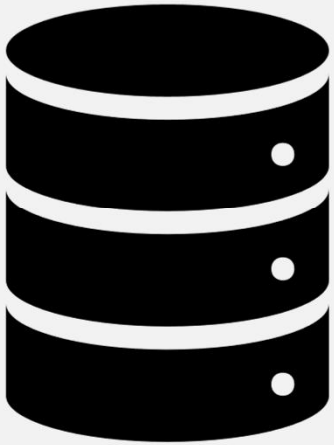
Permissões

```sql
CREATE USER etl WITH PASSWORD 'senha123';
GRANT SELECT ON TABLE employees TO etl;

SELECT grantee, privilege_type
FROM information_schema.table_privileges
WHERE table_name='employees' AND grantee='etl';

REVOKE SELECT ON TABLE employees FROM etl;
DROP USER etl;
```

6. Controle de Transações e Níveis de Isolamento

Banco de Dados Normalizados

- O fato dos banco de dados relacionais possuírem o mesma operação em diversas tabelas, devido a normalização, traz um efeito colateral:
  - Operações de negócio podem ser registradas de forma inconsistente



Exemplo

- Transação debita dinheiro de uma conta
- Transação que iria creditar dinheiro em outra conta falha
- Resultado: dinheiro desaparece!

- Solução:
  - Tratar o processo como uma única transação:
    - Ou todas são confirmadas, ou todas são desfeitas

ACID

- **Atomicidade: transações tratadas como únicas**
- **Consistência: a transação deve ir de um estado consitente para outro estado consistente**
- **Isolamento: quando as transanções devem ser visíveis. Pode prevenir fenômenos como leitura suja e fantasmas**
- **Durabilidade: uma vez que a transação for comfirmada, ele persistirá**

Comandos

- Begin: Inicia o controle de transações
- Commit: Confirma as transações
- Rollback: Desfaz as transações
- Savepoint: Cria um ponto de controle

Controle de Transações

```sql
-- Inicia a transação
BEGIN;

-- Insere um novo pedido
INSERT INTO orders (order_id,order_date, customer_id, employee_id)
VALUES (200,'2023-10-05', 'VINET', 5);

-- Insere um detalhe para o pedido
INSERT INTO order_details (order_id, product_id, quantity, unit_price,discount)
VALUES (200, 11, 10, 14.00,0);

-- Tenta selecionar as últimas inserções
SELECT * FROM orders WHERE order_id = 200;
SELECT * FROM order_details WHERE order_id = 200

-- rollback
ROLLBACK;
COMMIT;
```

Problemas Comuns em Transações

- Leituras Sujas
- Leituras Não Repetíveis
- Leituras Fantasmas

Introdução aos Níveis de Isolamento de Transação

- Uma transação é uma sequência de operações realizadas como uma única unidade lógica
- Níveis de Isolamento garantem a integridade dos dados durante a transação

Read Committed

- Nível de Isolamento mais comum

Repeatable Read

- Pode Repetir Leitura com os mesmos dados

Serializable

- Nível mais alto de isolamento

Bloqueios (Locks)

- Impede outras transações de Ler ou Escrever dados

Gerenciando Bloqueios

- Deadlocks
- Estratégias de Prevenção

DDL

DDL

```sql
--criar tabela
CREATE TABLE tabela_temporaria
(id SERIAL PRIMARY KEY, nome VARCHAR(100), idade INT );

--consultar estrutura
SELECT * FROM
    information_schema.columns
WHERE
    table_name = 'tabela_temporaria';

--inserir registros
INSERT INTO tabela_temporaria (nome, idade) VALUES
('Ana', 28),
('Bruno', 34),
('Clara', 25);

SELECT * FROM tabela_temporaria;
```

DDL

```sql
/* delete x truncate */
DELETE FROM tabela_temporaria;

--truncate, remove registros sem log
TRUNCATE TABLE tabela_temporaria RESTART IDENTITY;

-- Adicionar uma nova coluna
ALTER TABLE tabela_temporaria ADD COLUMN nova_coluna VARCHAR(255);

SELECT * FROM tabela_temporaria;

-- Remover uma coluna
ALTER TABLE tabela_temporaria DROP COLUMN nova_coluna;
```

DDL

```sql
/* adicionar coluna novamente */
ALTER TABLE tabela_temporaria ADD COLUMN nova_coluna VARCHAR(255);

-- Modificar o tipo de dados de uma coluna
ALTER TABLE tabela_temporaria ALTER COLUMN nova_coluna
    TYPE INT USING nova_coluna::INT;

-- Modificar o valor padrão de uma coluna
ALTER TABLE tabela_temporaria ALTER COLUMN nova_coluna SET DEFAULT 0;

INSERT INTO tabela_temporaria (nome, idade) VALUES
('Ana', 28),
('Bruno', 34),
('Clara', 25);

SELECT * FROM tabela_temporaria;
```

DDL

```sql
-- Adicionar uma constraint UNIQUE
ALTER TABLE tabela_temporaria ADD CONSTRAINT unica_nome UNIQUE(nome);

INSERT INTO tabela_temporaria (nome, idade) VALUES
('Ana', 28);

-- Adicionar uma constraint NOT NULL
ALTER TABLE tabela_temporaria ALTER COLUMN nome SET NOT NULL;

INSERT INTO tabela_temporaria (idade) VALUES
(28);
```

DDL

```sql
-- Adicionar uma constraint CHECK
ALTER TABLE tabela_temporaria ADD CONSTRAINT check_idade CHECK (idade >= 18);

INSERT INTO tabela_temporaria (nome, idade) VALUES
('Pedro', 8);

-- Remover uma constraint
ALTER TABLE tabela_temporaria DROP CONSTRAINT check_idade;

CREATE INDEX idx_nome ON tabela_temporaria (nome);

ALTER INDEX idx_nome RENAME TO idx_nome_tabela_temporaria;

DROP INDEX idx_nome_tabela_temporaria;

-- Drop Table
DROP TABLE IF EXISTS tabela_temporaria;
```

8.Views

Views

```sql
-- view padrão
CREATE VIEW view_padrao AS
SELECT
    o.order_id AS "Código do Pedido",
    c.company_name AS "Cliente",
    e.first_name || ' ' || e.last_name AS "Funcionário",
    p.product_name AS "Produto",
    od.unit_price AS "Preço Unitário",
    od.quantity AS "Quantidade",
    od.discount AS "Desconto",
    s.company_name AS "Transportadora",
    o.shipped_date AS "Data de Envio"
FROM
    orders o
INNER JOIN customers c ON o.customer_id = c.customer_id
INNER JOIN employees e ON o.employee_id = e.employee_id
INNER JOIN order_details od ON o.order_id = od.order_id
INNER JOIN products p ON od.product_id = p.product_id
LEFT JOIN shippers s ON o.ship_via = s.shipper_id
ORDER BY 1;

SELECT * FROM view_padrao;

SELECT "Código do Pedido","Cliente","Funcionário"
FROM view_padrao
WHERE "Código do Pedido" = 10248
```

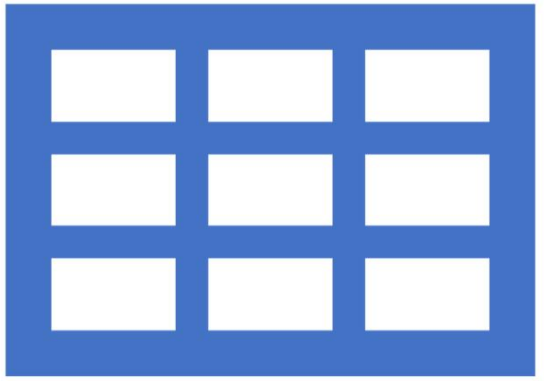
View Materializada

```sql
--view materializada
CREATE MATERIALIZED VIEW view_materializada AS
SELECT
   o.order_id AS "Código do Pedido",
    c.company_name AS "Cliente",
    e.first_name || ' ' || e.last_name AS "Funcionário",
    p.product_name AS "Produto",
    od.unit_price AS "Preço Unitário",
    od.quantity AS "Quantidade",
    od.discount AS "Desconto",
    s.company_name AS "Transportadora",
    o.shipped_date AS "Data de Envio"
FROM
    orders o
INNER JOIN customers c ON o.customer_id = c.customer_id
INNER JOIN employees e ON o.employee_id = e.employee_id
INNER JOIN order_details od ON o.order_id = od.order_id
INNER JOIN products p ON od.product_id = p.product_id
LEFT JOIN shippers s ON o.ship_via = s.shipper_id
ORDER BY 1;

SELECT * FROM view_materializada;

SELECT "Código do Pedido","Cliente","Funcionário"
FROM view_materializada
WHERE "Código do Pedido" = 10248
```

9.Windows Functions

Window Functions

- Janela: é um conjunto definido de linhas relacionados
- OVER: define como as linhas devem ser agrupadas em uma janela
- PARTITION BY: cria partições quando a divisão é baseada em colunas
- ORDER BY: ordem das linhas dentro da partição ou em toda a consulta (quando não há partição). O resultado é calculado sobre todos os dados



Janela com Partições

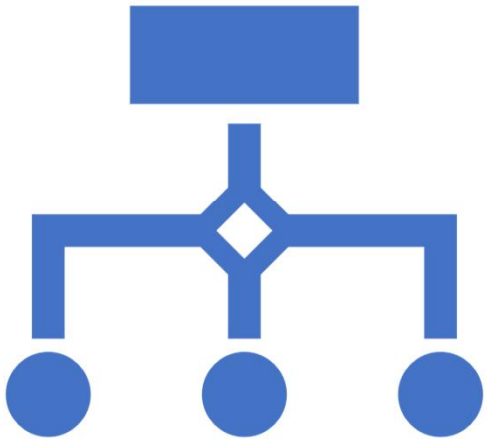
ALFKI	10643	1997-08-25	1
ALFKI	10692	1997-10-03	2
ALFKI	10702	1997-10-13	3
ALFKI	10835	1998-01-15	4
ALFKI	10952	1998-03-16	5
ALFKI	11011	1998-04-09	6
ANATR	10308	1996-09-18	1
ANATR	10625	1997-08-08	2
ANATR	10759	1997-11-28	3
ANATR	10926	1998-03-04	4
ANTON	10365	1996-11-27	1
ANTON	10507	1997-04-15	2
ANTON	10535	1997-05-13	3
ANTON	10573	1997-06-19	4
ANTON	10677	1997-09-22	5
ANTON	10682	1997-09-25	6
ANTON	10856	1998-01-28	7
AROUT	10355	1996-11-15	1
AROUT	10383	1996-12-16	2
AROUT	10453	1997-02-21	3
AROUT	10558	1997-06-04	4
AROUT	10707	1997-10-16	5

Partição Única

60	Camembert Pierrot	1577	1
59	Raclette Courdavault	1496	2
31	Gorgonzola Telino	1397	3
56	Gnocchi di nonna Alice	1263	4
16	Pavlova	1158	5
75	Rhönbräu Klosterbier	1155	6
24	Guaraná Fantástica	1125	7
40	Boston Crab Meat	1103	8
62	Tarte au sucre	1083	9
71	Flotemysost	1057	10
2	Chang	1057	10
21	Sir Rodney's Scones	1016	12
41	Jack's New England Cl...	981	13
76	Lakkalikööri	981	13

Principais funções de Janela

- ROW_NUMBER: atribui número sequencial a cada linha dentro da partição
- RANK: atribui uma classificação
- DENSE_RANK: atribui uma classificação mas não pula números repetidos
- NTILE: cria grupos baseado em algum valor e um n especificado
- LAG: acessa dados da linha anterior
- LEAD: acessa dados da linha seguinte



Window Functions

```sql
--ROW_NUMBER() atribui numero sequencial a cada linha dentro da partição
--sequenciar os pedidos do cliente por data
SELECT customer_id, order_id, order_date,
       ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date)
       AS "Sequência"
FROM orders;
```

Window Functions

```sql
--RANK() atribui uma classificação
--cria ranking de produtos mais vendidos (quantidade)
SELECT od.product_id, p.product_name, SUM(od.quantity) as "Quantidade Total",
       RANK() OVER (ORDER BY SUM(od.quantity) DESC) AS "Ranking"
FROM order_details od
INNER JOIN products p ON (p.product_id = od.product_id)
GROUP BY od.product_id,p.product_name;
```

Window Functions

```sql
--DENSE_RANK() atribui uma classificação dentro da partição
--cria ranking de produtos mais vendidos
--não pula números repetidos
SELECT od.product_id, p.product_name, SUM(od.quantity) as "Quantidade Total",
       DENSE_RANK() OVER (ORDER BY SUM(od.quantity) DESC) AS "Ranking"
FROM order_details od
INNER JOIN products p ON (p.product_id = od.product_id)
GROUP BY od.product_id,p.product_name;
```

Window Functions

```sql
--NTILE cria grupos baseado em algum valor e um n especificado
--dividir fretes em 4 quantis
SELECT order_id, freight,
       NTILE(4) OVER (ORDER BY freight DESC) AS "Quartis"
FROM orders;
```

Window Functions

```sql
--LAG acessa dados da linha anterior
--compara preço do produto com pedido anterior
SELECT order_id, product_id, unit_price,
       LAG(unit_price, 1) OVER (PARTITION BY product_id ORDER BY order_id)
        AS "Preço Anterior"
FROM order_details;
```

Window Functions

```sql
--LEAD acessa dados da linha seguinte
--mostra o preço do próximo pedido de um produto
SELECT order_id, product_id, unit_price,
       LEAD(unit_price, 1) OVER (PARTITION BY product_id ORDER BY order_id)
       AS "Próximo Preço"
FROM order_details;
```

10.CTEs

CTE

```sql
--total do pedido, com cliente
WITH PedidoSoma AS (
    SELECT order_id, SUM(unit_price * quantity) AS total_price
    FROM order_details
    GROUP BY order_id
)
SELECT o.order_id, o.customer_id, p.total_price
FROM orders o
JOIN PedidoSoma p ON o.order_id = p.order_id
ORDER BY total_price DESC;
```

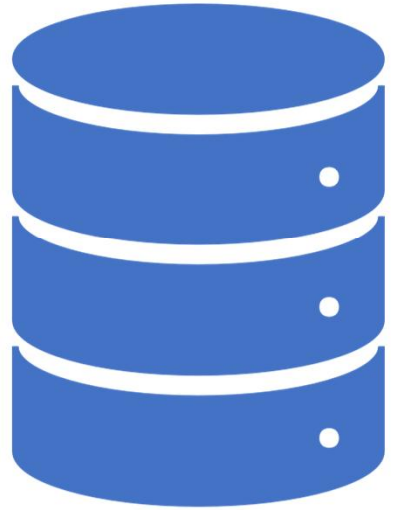
CTE

```sql
--identificar produtos mais vendidos comparados com a média
WITH TotalVendas AS (
    SELECT product_id, SUM(quantity) AS total_quantity
    FROM order_details
    GROUP BY product_id
), MaisVendidos AS (
    SELECT product_id, total_quantity
    FROM TotalVendas
    WHERE total_quantity > (
        SELECT AVG(total_quantity) FROM TotalVendas
    )
)
SELECT p.product_id, p.product_name, mv.total_quantity
FROM products p
JOIN MaisVendidos mv ON p.product_id = mv.product_id
ORDER BY mv.total_quantity DESC;
```

11.Stored Procedures e UDFs e Tiggers

Programação em Bancos de Dados

- Stored Procedure
- UDF
- Trigger



Stored Procedure

```
--stored procedure
CREATE OR REPLACE PROCEDURE add_product(
   product_id INT, pname VARCHAR, pcategory INT,
    psupplier INT, pprice NUMERIC, discontinued INT)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO products (product_id,product_name, category_id,
                          supplier_id, unit_price, discontinued)
    VALUES (product_id, pname, pcategory, psupplier, pprice, discontinued);
    COMMIT;
END;
$$;
```

Execução

```sql
--execução
CALL add_product(99,'Novo Produto', 5, 1, 19.99,0);

SELECT * FROM products;

DELETE FROM products where product_id = 99;

DROP PROCEDURE add_product;
```

Trigger

```sql
--criamos coluna para data de atualização
ALTER TABLE products ADD COLUMN last_updated TIMESTAMP;

SELECT last_updated FROM products limit 1;

--função de atualização
CREATE OR REPLACE FUNCTION update_product_last_updated()
RETURNS TRIGGER AS $$
BEGIN
    NEW.last_updated = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger

```
-trigger
CREATE TRIGGER product_update_before
BEFORE UPDATE ON products
FOR EACH ROW
EXECUTE FUNCTION update_product_last_updated();

--teste
INSERT INTO products (product_id,product_name, category_id,
                          supplier_id, unit_price, discontinued)
VALUES (99,'Novo Produto', 5, 1, 19.99,0);

UPDATE products SET unit_price = 12.00 WHERE product_name = 'Novo Produto';

SELECT product_name, last_updated FROM products WHERE product_name = 'Novo Produto';

DROP TRIGGER product_update_before ON products;
DROP FUNCTION update_product_last_updated;
ALTER TABLE products DROP COLUMN last_updated;
DELETE FROM products where product_id = 99;
```

UDF

```sql
CREATE OR REPLACE FUNCTION total_order_price(o_id INT)
RETURNS NUMERIC AS $$
DECLARE
    total_price NUMERIC;
BEGIN
    SELECT SUM(unit_price * quantity)
    INTO total_price
    FROM order_details
    WHERE order_id = o_id;
    RETURN total_price;
END;
$$ LANGUAGE plpgsql;

SELECT o.order_id, total_order_price(o.order_id)
FROM ORDERS o;
```

12.Funções Diversas

Funções de Data e Hora

```sql
-- Retorna a data e hora atuais
SELECT NOW();

-- Retorna a data atual
SELECT CURRENT_DATE;

-- Extrai o ano da data do pedido
SELECT order_date,
       EXTRACT(YEAR FROM order_date) AS "Ano",
       EXTRACT(MONTH FROM order_date) AS "Mês",
       EXTRACT(DAY FROM order_date) AS "Dia",
       EXTRACT(QUARTER FROM order_date) AS "Trimestre"
FROM orders;
```

Funções de Data e Hora

```sql
-- Calcula tempo do pedido
SELECT order_id, AGE(order_date, CURRENT_DATE) AS age
FROM orders;

-- Converte string para data
SELECT TO_DATE('20240101', 'YYYYMMDD') AS "Data";

-- Converte data para string
SELECT TO_CHAR(NOW(), 'YYYY-MM-DD') AS "Data";

-- Trabalhando com Zonas Horárias
SELECT order_date,order_date AT TIME ZONE 'UTC' AT TIME ZONE 'America/New_York'
FROM orders WHERE order_id = 10248;
```

Funções de String

```sql
-- Retorna o número de caracteres no nome do produto
SELECT product_name, LENGTH(product_name) AS "Tamanho"
FROM products;

-- Extrai uma substring do nome do produto
SELECT product_name, SUBSTRING(product_name FROM 1 FOR 5) AS "Abreviatura"
FROM products;

-- Retorna a posição da substring 'Ch' no nome do produto
SELECT product_name, POSITION('Ch' IN product_name) AS "Posição"
FROM products;
```

Funções de String

```sql
-- Remove espaços do início e do fim do nome do cliente
SELECT customer_id, TRIM(company_name) AS "Remove Espaços"
FROM customers;

-- Converte o nome do cliente para maiúsculas
SELECT customer_id, UPPER(company_name) AS "Maísculas",
LOWER(company_name) AS "Minúsculas"
FROM customers;
```

Funções de Matemáticas

```sql
-- Arredonda
SELECT  unit_price, quantity,unit_price * quantity AS "Total",
    ROUND((unit_price * quantity)::numeric, 2)
    AS "Total Arredondado"
FROM order_details;

-- Retorna o menor inteiro
SELECT product_name,unit_price, CEILING(unit_price) AS "Preço Inteiro"
FROM products;
```

Funções de Matemáticas

```sql
-- Retorna o valor absoluto da mudança
SELECT product_id,unit_price, ABS(unit_price - 20) AS "Mudança de Preço"
FROM products;

-- Gera um número aleatório
SELECT product_id, RANDOM() AS "Número Aleatório"
FROM products;
```

Funções Lógicas

```sql
-- Retorna o primeiro valor não-nulo da lista
SELECT order_id, COALESCE(ship_region, 'N/A') AS "Região"
FROM orders;

-- Retorna null se o preço unitário é igual a 11, senão retorna o preço
SELECT product_id,unit_price, NULLIF(unit_price,11) AS "Preço Unitário"
FROM products;

-- Retorna o maior
SELECT product_id,unit_price, GREATEST(unit_price, 20) AS "Preço Unitário"
FROM products;

-- Retorna o menor
SELECT product_id,unit_price, LEAST(unit_price, 20) AS "Preço Unitário"
FROM products;
```

Funções de Conversão e Formatação

```sql
-- Converte o preço unitário para texto
SELECT product_name, CAST(unit_price AS VARCHAR) AS "Preço Unitário"
FROM products;

-- Formata a data do pedido em formato específico
-- FM remove espaços e zeros
SELECT order_id, FORMAT('Pedido feito em %s ',
        TO_CHAR(order_date, 'FMDay,FMDDth FMMonth, YYYY')) AS "Data Formatada"
FROM orders;
```

Case

```sql
SELECT product_name, unit_price,
       CASE
           WHEN unit_price < 10 THEN '$'
           WHEN unit_price BETWEEN 10 AND 20 THEN '$$'
           ELSE '$$$'
       END AS "Categoria de Preços"
FROM products;
```

Subconsultas

```sql
--produtos cujo preço são maior que a média
SELECT product_name, unit_price
FROM products
WHERE unit_price > (SELECT AVG(unit_price) FROM products);
```

Subconsultas

```sql
--funcionários com pedidos na Alemanha e Brazil
SELECT DISTINCT e.first_name, e.last_name
FROM employees e
WHERE e.employee_id IN (
    SELECT o.employee_id
    FROM orders o
    JOIN customers c ON o.customer_id = c.customer_id
    WHERE c.country in ('Germany','Brazil') AND e.employee_id = o.employee_id
);
```

Any e All

```sql
--produtos com preço maior que qualquer produto do fornecedor 1
SELECT product_name, unit_price
FROM products
WHERE unit_price > ANY (SELECT unit_price FROM products WHERE supplier_id = 1);

--produtos cujo preço é maior que todos produtos  do fornecedor 1
SELECT product_name, unit_price
FROM products
WHERE unit_price > ALL (SELECT unit_price FROM products WHERE supplier_id = 1);
```

Exists

```sql
--funcionários com pedidos na Alemanha e Brazil
SELECT DISTINCT e.first_name, e.last_name
FROM Employees e
WHERE EXISTS (
    SELECT order_id
    FROM orders o
    JOIN customers c ON o.customer_id = c.customer_id
    WHERE c.country in ('Germany','Brazil') AND e.employee_id = o.employee_id
);
```

Extract

```sql
--pedidos por ano
SELECT EXTRACT(YEAR FROM order_date) AS "Ano",
    COUNT(1) AS "Total de Pedidos"
FROM orders
GROUP BY EXTRACT(YEAR FROM order_date);
```

ROLLUP, CUBE, GROUPING

```sql
SELECT c.country, EXTRACT(YEAR FROM o.order_date) AS "Ano",
    SUM(od.quantity * od.unit_price) AS "Total de Vendas"
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_details od ON o.order_id = od.order_id
GROUP BY ROLLUP (c.country, EXTRACT(YEAR FROM o.order_date))
order by 1, 3
```

ROLLUP, CUBE, GROUPING

```sql
SELECT c.country, EXTRACT(YEAR FROM o.order_date) AS "Ano",
    p.category_id, SUM(od.quantity * od.unit_price) AS "Total de Vendas"
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_details od ON o.order_id = od.order_id
JOIN products p ON od.product_id = p.product_id
GROUP BY CUBE (c.country, EXTRACT(YEAR FROM o.order_date), p.category_id)
order by 1,2,3
```

ROLLUP, CUBE, GROUPING

```sql
SELECT c.country, EXTRACT(YEAR FROM o.order_date) AS "Ano", p.category_id,
    SUM(od.quantity * od.unit_price) AS "Total de Vendas"
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_details od ON o.order_id = od.order_id
JOIN products p ON od.product_id = p.product_id
GROUP BY GROUPING SETS ((c.country, EXTRACT(YEAR FROM o.order_date)), (p.category_id), ())
order by 1,2,3,4
```

Filter

```sql
--Permite fazer cálculos específicos em funções de agregação
SELECT
    SUM(unit_price * quantity) AS "Total de Vendas",
    SUM(unit_price * quantity) FILTER (WHERE unit_price > 20)
    AS "Total Produtos Mais Caros",
    AVG(quantity) FILTER (WHERE quantity > 10)
  AS  "Média de Maiores Vendas"
FROM order_details;
```

Hashes

```sql
ALTER TABLE employees ADD COLUMN password_hash TEXT;

UPDATE employees
SET password_hash = md5('senha')
WHERE employee_id = 1;

--visualizando o hash
SELECT employee_id ,password_hash
FROM employees
WHERE employee_id = 1;
```

Hashes

```sql
--testando se o usuário informou a senha correta
SELECT employee_id
FROM employees
WHERE md5('senha') = password_hash
AND employee_id = 1;

ALTER TABLE employees DROP COLUMN password_hash;
```

Criptografia

```sql
CREATE EXTENSION IF NOT EXISTS pgcrypto;

ALTER TABLE employees ADD COLUMN salary bytea;

UPDATE employees
SET salary = pgp_sym_encrypt(7000::text, '374f61c8f2106cd239197a6514b')
WHERE employee_id = 1;

--verificando o salário
SELECT employee_id ,encode(salary, 'hex') ,
    pgp_sym_decrypt(salary, '374f61c8f2106cd239197a6514b')
FROM employees
WHERE employee_id = 1;

ALTER TABLE employees DROP COLUMN salary;
```

Tipos Compostos

```sql
--array
--coluna de array
ALTER TABLE products ADD COLUMN tags TEXT[];

--Inserindo
UPDATE products SET tags = ARRAY['orgânico', 'importado'] WHERE product_id = 5;

SELECT * FROM products
WHERE product_id = 5;

-- Adicionando um item
UPDATE products SET tags = array_append(tags, 'eco-friendly') WHERE product_id = 5;

-- Removendo um item
UPDATE products SET tags = array_remove(tags, 'importado') WHERE product_id = 5;
```

Tipos Compostos

```sql
-- Consultando um item no array
SELECT product_name,tags FROM products WHERE 'orgânico' = ANY (tags);

ALTER TABLE products DROP COLUMN tags;
```

Tipos Compostos

```sql
--tipo composto
-- Criando um tipo composto
CREATE TYPE product_info AS (
    manufacturer TEXT,
    warranty_period INT
);

-- Adicionando a coluna do tipo
ALTER TABLE products ADD COLUMN additional_info product_info;

-- Inserindo dados
UPDATE products SET additional_info = ROW('Acme Corp', 24) WHERE product_id = 5;

SELECT * FROM products
WHERE product_id = 5;
```

Json

```sql
--criando tabela
CREATE TABLE produtos (
    id serial PRIMARY KEY,
    nome VARCHAR(100),
    atributos JSON
);

--adicionando dados
INSERT INTO produtos (nome, atributos)
VALUES
('Camiseta', '{"cor": "azul", "tamanho": "M", "preco": 25.50}'),
('Calça', '{"cor": "preta", "tamanho": "G", "preco": 50.00}');

SELECT *
FROM produtos
WHERE atributos->>'cor' = 'azul';
```

Json

```sql
UPDATE produtos
SET atributos = json_build_object(
    'cor', (atributos->>'cor'),
    'tamanho', (atributos->>'tamanho'),
    'preco', '30.00'
)
WHERE id = 1;

SELECT *
FROM produtos
WHERE (atributos->>'preco')::numeric = 30;

DROP TABLE produtos;
```

Schemas

- **table**
- **schema.table**
- **database.schema.table**