Modelagem e implementação no Banco de Dados

Matheus José Ribeiro de Moura - 2023 0712 6158

Campus Rua Teresa
Vamos manter as informações! – Turma 9001 – 2024.3 FLEX

Objetivo da Prática

O objetivo da atividade era identificar os requisitos do sistema e transformá-los em um modelo adequado para ser possível aplicá-lo a uma situação real.

Explorar ferramentas de modelagem e gerenciadores de banco de dados para criação das tabelas, exercitar a sintaxe do SQL para criar, inserir e consultar todos os dados necessários foram propostos nessa prática.

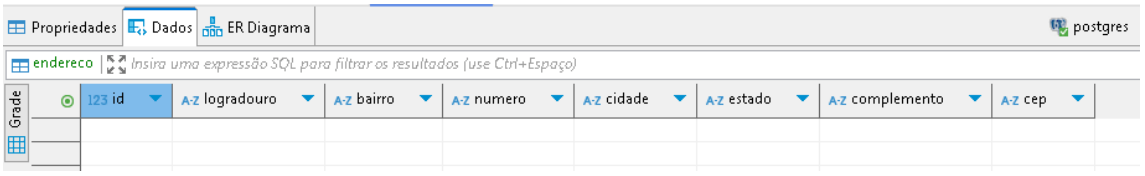
Obs.: Devido a liberdade dada no uso de ferramentas, os comandos deste documento foram realizados em PostgresSQL e as imagens tiradas do gerenciador DBeaver.

Repositório GIT: https://github.com/MatheusJRM/projeto-lojaDB.git

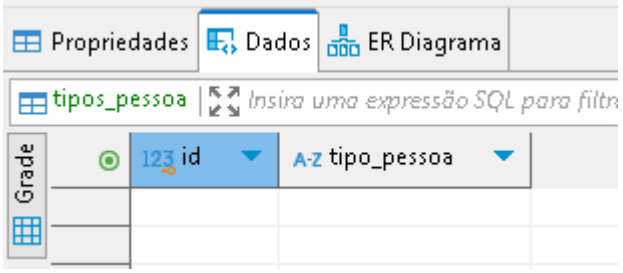
1º Procedimento | Criando o Banco de Dados

Códigos utilizados para criação das tabelas:

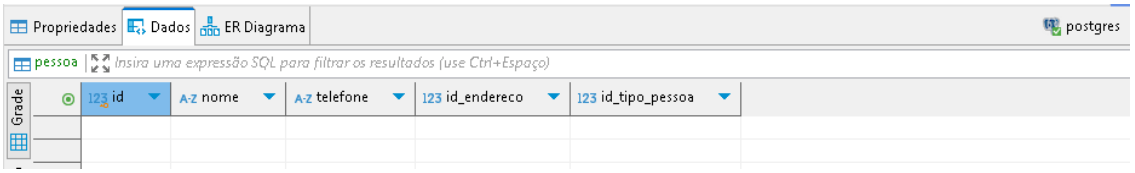
```
CREATE TABLE Endereco (
  id SERIAL PRIMARY key UNIQUE,
  logradouro VARCHAR(255) NOT NULL,
  bairro VARCHAR(255) NOT NULL,
  numero VARCHAR NOT NULL,
  cidade VARCHAR(255) NOT NULL,
  estado VARCHAR(255) NOT NULL,
  complemento VARCHAR(255),
  cep VARCHAR(8) NOT NULL
);
```



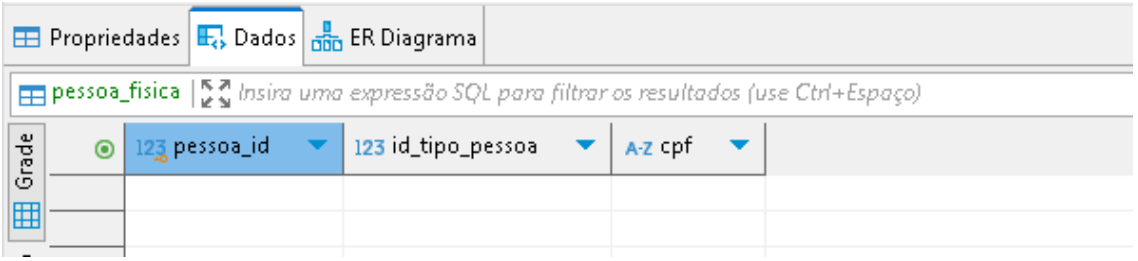

```
create table Tipos_Pessoa (id int primary key, Tipo_Pessoa VARCHAR(1));
```



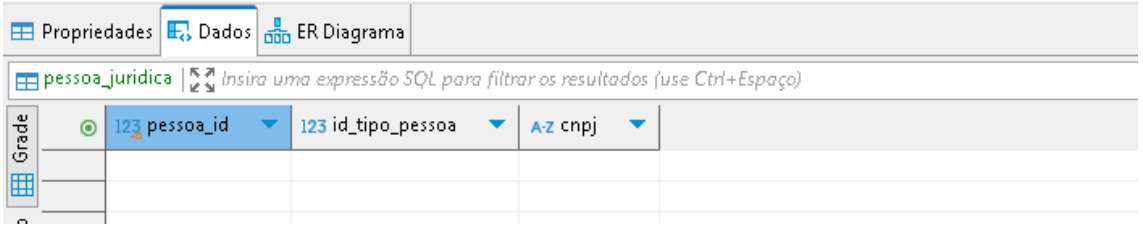

```
CREATE TABLE Pessoa (
  id SERIAL PRIMARY key
  UNIQUE, nome VARCHAR(255)
  NOT NULL, telefone
  VARCHAR(11) NOT NULL,
  id_endereco INTEGER,
  FOREIGN KEY (id_endereco) REFERENCES
  Endereco(id), id_tipo_pessoa int not null REFERENCES
  Tipos_pessoa(id), CONSTRAINT People_AltPK UNIQUE
  (id, id_tipo_pessoa)
);
```



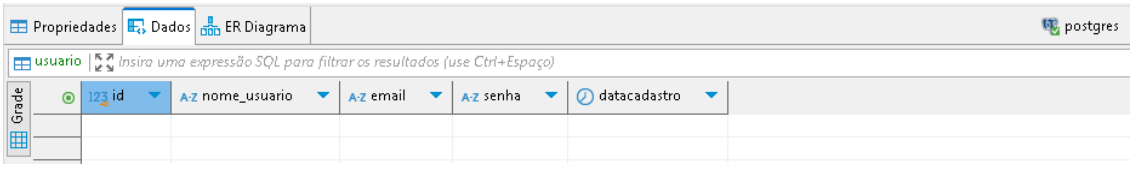

```
CREATE TABLE Pessoa_Fisica (
 pessoa_id INTEGER primary key unique references pessoa(id),
 id_tipo_pessoa int GENERATED ALWAYS AS (1) STORED, cpf
 VARCHAR(11) NOT null UNIQUE,
 FOREIGN KEY (pessoa_id, id_tipo_pessoa) REFERENCES Pessoa(id, id_tipo_pessoa)
);
```




```
CREATE TABLE Pessoa_Juridica (
        pessoa_id INTEGER primary key unique references pessoa(id),
        id_tipo_pessoa int GENERATED ALWAYS AS (2) STORED,
        FOREIGN KEY (pessoa_id, id_tipo_pessoa) REFERENCES Pessoa(id, id_tipo_pessoa),
  cnpj VARCHAR(14) NOT null UNIQUE
);
```

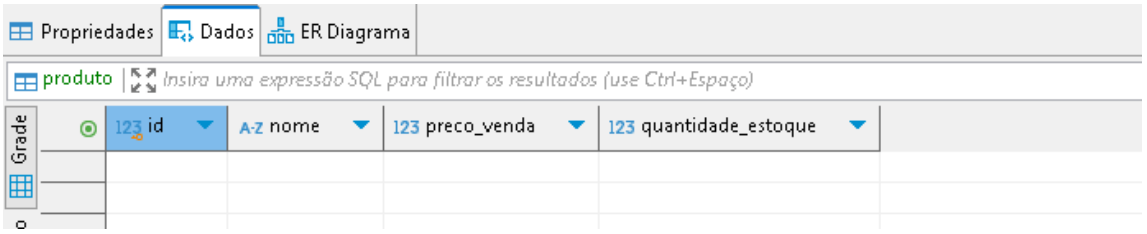



```
CREATE TABLE Usuario (
  id SERIAL PRIMARY key UNIQUE,
  nome_usuario VARCHAR(100) NOT NULL,
  email VARCHAR(255) NOT NULL
  UNIQUE, senha VARCHAR(16) NOT NULL,
  dataCadastro timestamp NOT NULL DEFAULT CURRENT_DATE
);
```

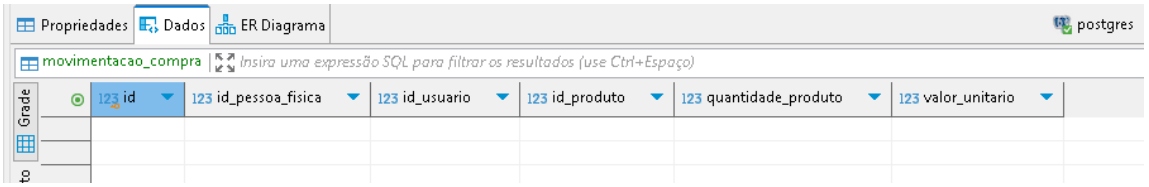



```
CREATE TABLE Produto (
  id SERIAL PRIMARY key UNIQUE,
  nome VARCHAR(255) NOT NULL,
  preco_venda NUMERIC NOT NULL,
```

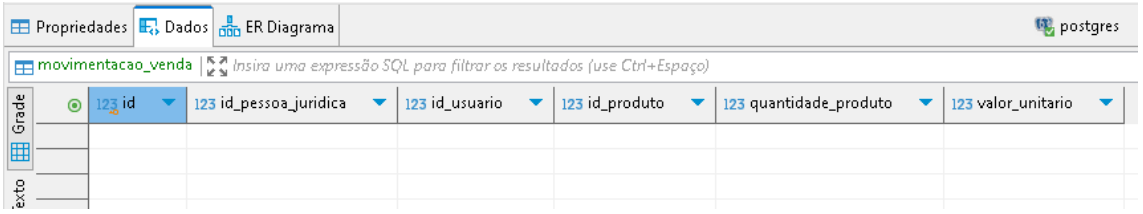
```
  quantidade_estoque INTEGER NOT NULL
);
```




```
CREATE TABLE Movimentacao_Compra (
  id SERIAL PRIMARY KEY,
  id_pessoa_fisica INTEGER NOT NULL,
  id_usuario INTEGER NOT NULL,
  id_produto INTEGER NOT NULL,
  quantidade_produto INTEGER NOT NULL,
  valor_unitario NUMERIC NOT NULL,
  FOREIGN KEY (id_pessoa_fisica) REFERENCES pessoa_fisica(pessoa_id),
  FOREIGN KEY (id_usuario) REFERENCES usuario(id),
  FOREIGN KEY (id_produto) REFERENCES produto(id)
);
```




```
CREATE TABLE Movimentacao_Venda (
  id SERIAL PRIMARY KEY,
  id_pessoa_juridica INTEGER NOT NULL,
  id_usuario INTEGER NOT NULL,
  id_produto INTEGER NOT NULL,
  quantidade_produto INTEGER NOT NULL,
  valor_unitario NUMERIC NOT NULL,
  FOREIGN KEY (id_pessoa_juridica) REFERENCES pessoa_juridica(pessoa_id),
  FOREIGN KEY (id_usuario) REFERENCES usuario(id),
  FOREIGN KEY (id_produto) REFERENCES produto(id)
);
```



Análises

a) Como são implementadas as diferentes cardinalidades, basicamente 1X1, 1XN ou NxN, em um banco de dados relacional?

R: **Cardinalidade 1x1 (um-para-um)**: Neste tipo de relacionamento, para cada registro em uma tabela, existe exatamente um registro correspondente em outra tabela, normalmente, a chave primária de uma das tabelas (Tabela A) é usada como chave estrangeira e primária na outra tabela (Tabela B).

Cardinalidade 1xN (um-para-muitos): Neste relacionamento, para cada registro em uma tabela, pode haver muitos registros correspondentes em outra tabela, a chave primária da tabela "um" (Tabela A) é referenciada como chave estrangeira na tabela "muitos" (Tabela B).

Cardinalidade NxN (muitos-para-muitos): Neste relacionamento, um registro em uma tabela pode estar relacionado a muitos registros em outra tabela e vice-versa, cria-se uma tabela associativa (ou tabela de junção) para representar o relacionamento entre as duas tabelas. Essa tabela contém chaves estrangeiras que apontam para as chaves primárias das duas tabelas relacionadas.

b) Que tipo de relacionamento deve ser utilizado para representar o uso de herança em bancos de dados relacionais?

R: Bancos de dados relacionais não possuem um suporte explícito para herança, como ocorre em linguagens orientadas a objetos, mas existem três abordagens principais para modelar herança:

- **Tabela Única:** Todos os atributos das classes pai e filhas são armazenados em uma única tabela. Pode haver colunas extras (não usadas por todas as subentidades) e uma coluna discriminadora para identificar qual subtipo cada linha representa.
- **Tabela por Subclasse**: Cada subclasse tem sua própria tabela, e a tabela da classe base armazena os atributos comuns. As tabelas das subclasses referenciam a tabela pai usando chaves estrangeiras.
- **Tabela por Concreta:** Cada subclasse tem sua própria tabela, e não há uma tabela para a classe pai. Todas as colunas da superclasse são repetidas nas tabelas das subclasses.

c) Como o SQL Server Management Studio permite a melhoria da produtividade nas tarefas relacionadas ao gerenciamento do banco de dados?

R: O **SQL Server Management Studio (SSMS)** oferece várias funcionalidades que podem melhorar significativamente a produtividade no gerenciamento de bancos de dados. Algumas dessas funcionalidades são: **Interface gráfica para administração, Ferramenta de consulta (Query Editor), Gerenciamento de backups e restaurações, Monitoramento e análise de desempenho, Gerenciamento de permissões, Designers gráficos de banco de dados, Automação de tarefas.**

2º Procedimento | Alimentando a Base

Códigos utilizados para preenchimento das tabelas:

```
insert into Usuario (nome_usuario, email, senha, datacadastro)
values ('op1', 'op1@gmail.com', 'op1', '2014-06-04 12:00');

insert into Usuario (nome_usuario, email, senha, datacadastro)
values ('op2', 'op2@gmail.com', 'op2', '2014-06-04 11:00');
```

Propriedades | Dados | ER Diagrama — postgres

usuario | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	id	nome_usuario	email	senha	datacadastro
1	1	op1	op1@gmail.com	op1	2014-06-04 12:00:00.000
2	2	op2	op2@gmail.com	op2	2014-06-04 11:00:00.000

```
insert into produto (nome, preco_venda, quantidade_estoque)
values ('Banana', 5.00, 100);
insert into produto (nome, preco_venda, quantidade_estoque)
values ('Laranja', 2.00, 500);
insert into produto (nome, preco_venda, quantidade_estoque)
values ('Manga', 4.00, 800);
```

Propriedades | Dados | ER Diagrama

produto | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	id	nome	preco_venda	quantidade_estoque
1	1	Banana	5	100
2	2	Laranja	2	500
3	3	Manga	4	800

```
insert into Pessoa (nome, telefone, id_endereco, id_tipo_pessoa)
values ('Matheus Jose', '24988375604', 1, 1);

insert into pessoa_fisica (pessoa_id, cpf)
values (1, '12752985754');
```

pessoa

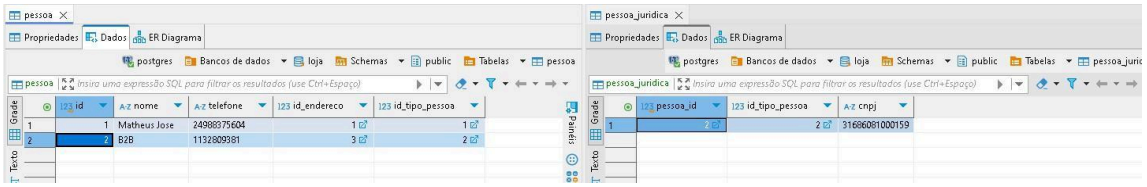
	id	nome	telefone	id_endereco	id_tipo_pessoa
1	1	Matheus Jose	24988375604	1	1
2	2	B2B	1132809381	3	2

pessoa_fisica

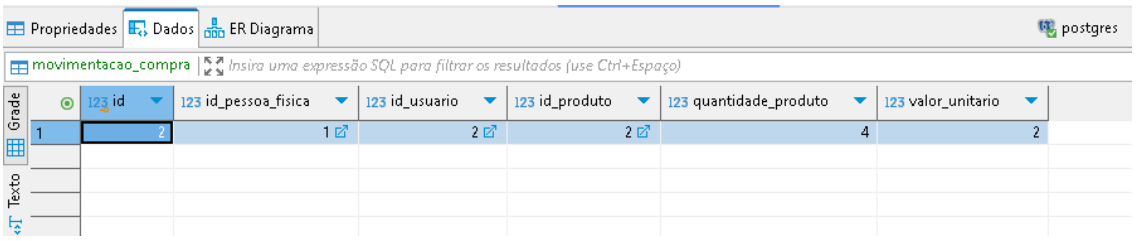
	pessoa_id	id_tipo_pessoa	cpf
1	1	1	12752985754

```
insert into Pessoa (nome, telefone, id_endereco, id_tipo_pessoa)
values ('B2B', '1132809381', 3, 2);

insert into pessoa_juridica (pessoa_id, cnpj)
values (2, '31686081000159');
```

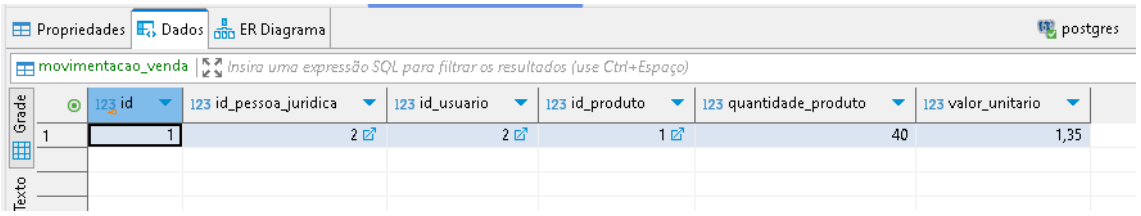


```
insert into movimentacao_compra (id_pessoa_fisica, id_usuario, id_produto, quantidade_produto,
valor_unitario)
values (1, 2, 2, 4, 2.00);
```



id	id_pessoa_fisica	id_usuario	id_produto	quantidade_produto	valor_unitario
2	1	2	2	4	2

```
insert into movimentacao_venda (id_pessoa_juridica, id_usuario, id_produto, quantidade_produto,
valor_unitario)
values (2, 2, 1, 40, 1.35);
```

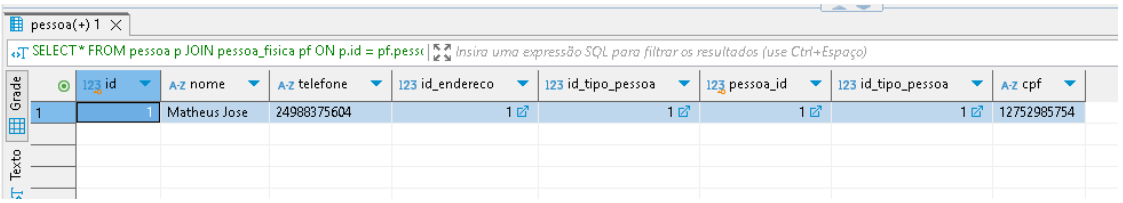


id	id_pessoa_juridica	id_usuario	id_produto	quantidade_produto	valor_unitario
1	2	2	1	40	1,35

Códigos utilizados para as consultas:

- *Dados completos de pessoas físicas*:

```
SELECT *
FROM pessoa p
JOIN pessoa_fisica pf ON p.id = pf.pessoa_id
WHERE pf.pessoa_id = p.id;
```



id	nome	telefone	id_endereco	id_tipo_pessoa	pessoa_id	id_tipo_pessoa	cpf
1	Matheus Jose	24988375604	1	1	1	1	12752985754

- *Dados completos de pessoas jurídicas:*

```
SELECT *
FROM pessoa p
JOIN pessoa_juridica pj ON p.id = pj.pessoa_id
WHERE pj.pessoa_id = p.id;
```

pessoa(+) 1

SELECT * FROM pessoa p JOIN pessoa_juridica pj ON p.id = pj.pe: | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	id	nome	telefone	id_endereco	id_tipo_pessoa	pessoa_id	id_tipo_pessoa	cnpj
1	2	B2B	1132809381	3	2	2	2	31686081000159

- *Movimentações de venda, com produto, fornecedor, quantidade, preço unitário e valor total:*

```
SELECT
m.id,
p.nome AS produto,
pj.cnpj AS vendedor,
m.quantidade_produto,
m.valor_unitario,
m.quantidade_produto * m.valor_unitario AS valor_total
FROM
movimentacao_venda m
INNER JOIN produto p ON m.id_produto = p.id
INNER JOIN pessoa_juridica pj ON m.id_pessoa_juridica = pj.pessoa_id;
```

movimentacao_venda(+) 1

SELECT m.id, p.nome AS produto, pj.cnpj AS vendedor, m.quantid | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	id	produto	vendedor	quantidade_produto	valor_unitario	valor_total
1	1	Banana	31686081000159	40	1,35	54

- *Movimentações de compra, com produto, comprador, quantidade, preço unitário e valor total:*

```
SELECT
m.id,
p.nome AS produto,
pf.cpf AS comprador,
m.quantidade_produto,
m.valor_unitario,
m.quantidade_produto * m.valor_unitario AS valor_total
FROM
movimentacao_compra m
INNER JOIN produto p ON m.id_produto = p.id
INNER JOIN pessoa_fisica pf ON m.id_pessoa_fisica = pf.pessoa_id;
```

movimentacao_compra(+) 1

SELECT m.id, p.nome AS produto, pf.cpf AS comprador, m.quanti | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	id	produto	comprador	quantidade_produto	valor_unitario	valor_total
1	1	Laranja	12752985754	4	2	8

- *Valor total das vendas agrupadas por produto:*

```
SELECT
p.nome AS produto,
SUM(m.quantidade_produto * m.valor_unitario) AS valor_total
FROM
movimentacao_venda m
INNER JOIN produto p ON m.id_produto = p.id
GROUP BY
p.nome;
```

produto 1

SELECT p.nome AS produto, SUM(m.quantidade_produto * m.val | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	produto	valor_total
1	Banana	54

- *Valor total das compras agrupadas por produto:*

```
SELECT
p.nome AS produto,
SUM(m.quantidade_produto * m.valor_unitario) AS valor_total
FROM
movimentacao_compra m
INNER JOIN produto p ON m.id_produto = p.id
GROUP BY
p.nome;
```

produto 1

SELECT p.nome AS produto, SUM(m.quantidade_produto * m.val | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	produto	valor_total
1	Laranja	8

- *Operadores que não efetuaram movimentações de venda:*

```
SELECT
u.nome_usuario
FROM
usuario u
LEFT JOIN movimentacao_venda m ON u.id = m.id_usuario
WHERE
m.id IS NULL;
```

usuario 1

SELECT u.nome_usuario FROM usuario u LEFT JOIN movimentaca | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	nome_usuario
1	op1

- *Valor total de compra, agrupado por operador:*

```
SELECT
u.nome_usuario,
SUM(m.quantidade_produto * m.valor_unitario) AS valor_total
FROM
movimentacao_compra m
INNER JOIN usuario u ON m.id_usuario = u.id
GROUP BY
u.nome_usuario;
```

usuario 1

SELECT u.nome_usuario, SUM(m.quantidade_produto * m.valor_u | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	nome_usuario	valor_total
1	op2	8

- *Valor total de venda, agrupado por operador:*

```
SELECT
 u.nome_usuario,
 SUM(m.quantidade_produto * m.valor_unitario) AS valor_total
FROM
 movimentacao_venda m
 INNER JOIN usuario u ON m.id_usuario = u.id
GROUP BY
 u.nome_usuario;
```

usuario 1

SELECT u.nome_usuario, SUM(m.quantidade_produto * m.valor_u | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	nome_usuario	valor_total
1	op2	54

- *Valor médio de venda por produto, utilizando média ponderada:*

```
SELECT
 p.nome AS produto,
 AVG(m.valor_unitario * m.quantidade_produto) / AVG(m.quantidade_produto) AS valor_medio
FROM
 movimentacao_compra m
 INNER JOIN produto p ON m.id_produto = p.id
GROUP BY
 p.nome;
```

produto 1

SELECT p.nome AS produto, AVG(m.valor_unitario * m.quantidade | Insira uma expressão SQL para filtrar os resultados (use Ctrl+Espaço)

	produto	valor_medio
1	Laranja	2

Análises

a. Quais as diferenças no uso de *sequence* e *identity*?

R: Uma sequence é um objeto de banco de dados que gera uma série de valores únicos e consecutivos. Ela pode ser utilizada em qualquer tabela e pode ser compartilhada entre várias tabelas. Além disso, uma sequence pode ser restartada ou alterada seu valor inicial.

Identity é uma propriedade de uma coluna que gera automaticamente um valor único e consecutivo quando uma linha é inserida na tabela. Ela é específica para uma coluna e não pode ser compartilhada entre tabelas.

Uma sequence é mais flexível e pode ser utilizada em várias tabelas, enquanto uma identity é mais simples e específica para uma coluna.

b. Qual a importância das chaves estrangerias para a consistência do banco?

R: Evitar a inserção de dados inconsistentes, por exemplo: uma chave estrangeira verifica se o valor inserido existe na tabela relacionada e manter a integridade referencial, uma chave estrangeira garante que os dados relacionados sejam consistentes e atualizados corretamente.

c. Quais operadores do SQL pertencem à álgebra relacional e quais são definidos no cálculo relacional?

R: Os operadores do SQL pertencentes a algebra relacional são: SELECT, PROJECT, JOIN, UNION, INTERSECT, EXCEPT.

Os operadores de Cálculo Relacional são os aritméticos (+, -, *, /), comparativos (=, <, >, <=, >=) e lógicos (AND, OR, NOT).

d. Como é feito o agrupamento em consultas, e qual requisito é obrigatório?

R: O agrupamento em consultas é feito utilizando a cláusula GROUP BY, que agrupa as linhas de uma tabela com base em uma ou mais colunas. O requisito obrigatório para o agrupamento é que todas as colunas selecionadas devem ser incluídas na cláusula GROUP BY ou ser utilizadas em uma função de agregação (e.g. SUM, AVG, MAX, MIN).

Conclusão

A atividade é extremamente útil para que possamos praticar os conceitos de modelagem e comandos SQL. Ao longo dos procedimentos precisei retornar e transformar várias vezes os modelos que havia definido por entender algumas relações melhores do que antes. A necessidade de descobrir quais comandos promovem herança da forma correta ao que era necessário ser utilizado e a liberdade de construir o modelo da forma que preferimos também ajuda a simular necessidades de um projeto real, onde precisamos refletir e descobrir como implementar nossas ideias em código. A liberdade de poder utilizar um gerenciador que já temos experiência (no caso desse projeto, o DBeaver) também facilita o foco na atividade e não no aprendizado de uma nova ferramenta.

Em resumo, desenvolvi uma compreensão sólida de como modelar e implementar uma base de dados para um sistema simples, o que é uma habilidade valiosa em muitas áreas da tecnologia. Além disso, a experiência prática em como trabalhar com bases de dados relacionais e sintaxe SQL, o que é fundamental para muitas aplicações empresariais e pessoais.