



Missão Prática | Nível 2 | Mundo 3

Filipe Alvim Santos - 202208291325

Campus Polo Sulacap – RJ / Desenvolvimento Full Stack
Nível 2: Vamos manter as informações? – Turma: 22.3 – 3º Semestre

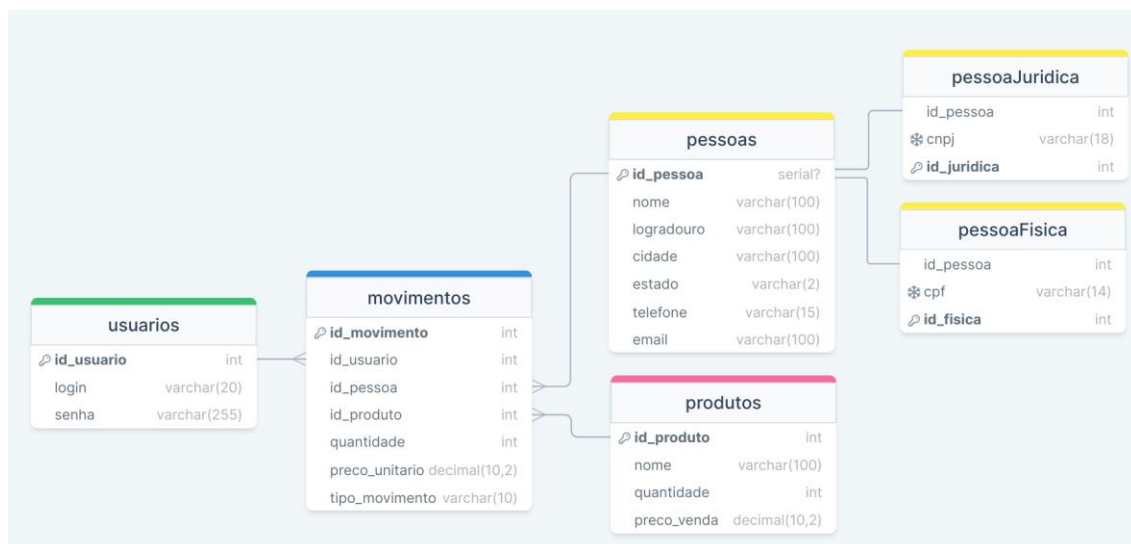
Objetivo da Prática

Os objetivos da prática são: Identificar os requisitos de um sistema e transformá-los no modelo adequado, utilizando ferramentas de modelagem de banco de dados relacionais. Explorar as sintaxes SQL na criação das estruturas do banco (DDL) e na consulta e manipulação dos dados (DML).

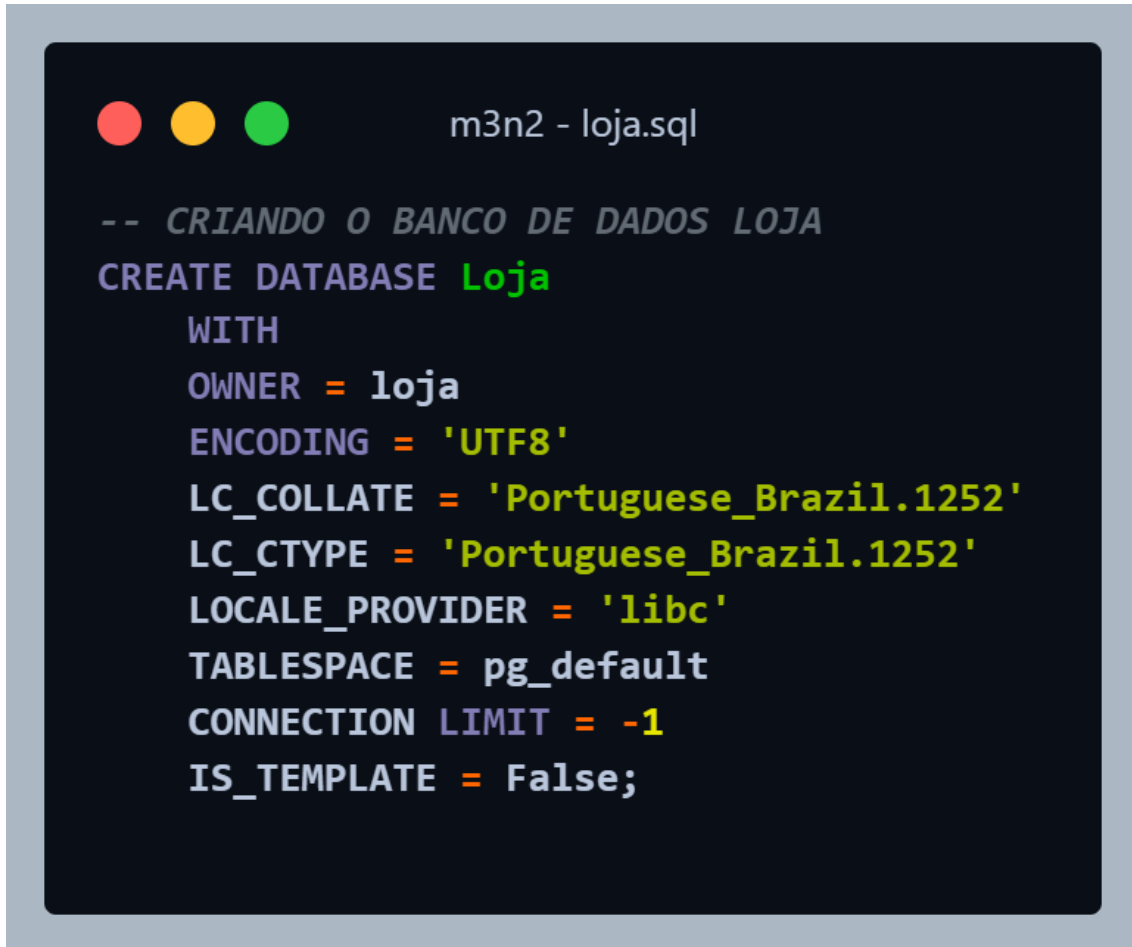
Sendo o objetivo final é vivenciar a experiência de modelar e implementar um banco de dados para um sistema simples, através da sintaxe SQL, no meu caso no banco de dados PostgreSQL.

1º Procedimento– Criando o Banco de Dados

- Modelagem do Banco de dados “loja”:



- Criando o Banco de Dados “loja” no PostgreSQL:



```
m3n2 - loja.sql

-- CRIANDO O BANCO DE DADOS LOJA
CREATE DATABASE Loja
WITH
  OWNER = loja
  ENCODING = 'UTF8'
  LC_COLLATE = 'Portuguese_Brazil.1252'
  LC_CTYPE = 'Portuguese_Brazil.1252'
  LOCALE_PROVIDER = 'libc'
  TABLESPACE = pg_default
  CONNECTION LIMIT = -1
  IS_TEMPLATE = False;
```

- Criando as tabelas (usuarios, produtos, pessoas, pessoaFisica, pessoaJuridica e movimentos) no banco de dados “loja”:



m3n2 -

-- CRIANDO A TABELA USUARIOS

```
CREATE TABLE usuarios (  
    id_usuario INT PRIMARY KEY,  
    login VARCHAR(20) NOT NULL,  
    senha VARCHAR(255) NOT NULL  
);
```



m3n2 - Loja_DB.sql

-- CRIANDO A TABELA PRODUTOS

```
CREATE TABLE produtos (  
    id_produto INT PRIMARY KEY,  
    nome VARCHAR(100) NOT NULL,  
    quantidade INTEGER NOT NULL,  
    preco_venda DECIMAL(10, 2) NOT NULL,  
);
```



m3n2 - Loja_DB.sql

-- CRIANDO A TABELA PESSOAS

```
CREATE TABLE pessoas (  
    id_pessoa SERIAL PRIMARY KEY,  
    nome VARCHAR(100) NOT NULL,  
    logradouro VARCHAR(100) NOT NULL,  
    cidade VARCHAR(100) NOT NULL,  
    estado VARCHAR(2) NOT NULL,  
    telefone VARCHAR(15) NOT NULL,  
    email VARCHAR(100) NOT NULL,  
);
```



m3n2 - loja.sql

-- CRIANDO A TABELA PESSOA FISICA

```
CREATE TABLE pessoaFisica (  
    id_pessoa INT NOT NULL,  
    cpf VARCHAR(14) NOT NULL UNIQUE,  
    CONSTRAINT pessoa_fisica_FK FOREIGN KEY (id_pessoa) REFERENCES pessoas(id_pessoa) ON DELETE CASCADE ON UPDATE CASCADE  
);
```



m3n2 - loja.sql

-- CRIANDO A TABELA PESSOA JURIDICA

```
CREATE TABLE pessoaJuridica (  
    id_pessoa INT NOT NULL,  
    cnpj VARCHAR(18) NOT NULL UNIQUE,  
    CONSTRAINT pessoa_juridica_FK FOREIGN KEY (id_pessoa) REFERENCES pessoas(id_pessoa) ON DELETE CASCADE ON UPDATE CASCADE  
);
```

```

m3n2 - Loja_DB.sql

-- CRIANDO A TABELA MOVIMENTOS
CREATE TABLE movimentos (
    id_movimento INT PRIMARY KEY,
    id_usuario INT NOT NULL,
    id_pessoa INT NOT NULL,
    id_produto INT NOT NULL,
    quantidade INTEGER NOT NULL,
    preco_unitario DECIMAL(10, 2) NOT NULL,
    tipo_movimento VARCHAR(10) NOT NULL CHECK (tipo_movimento IN ('E', 'S')),
    CONSTRAINT "movimentos_id_usuario_fk"
        FOREIGN KEY(id_usuario) REFERENCES usuarios(id_usuario),
    CONSTRAINT "movimentos_id_pessoa_fk"
        FOREIGN KEY(id_pessoa) REFERENCES pessoas(id_pessoa) ON DELETE CASCADE,
    CONSTRAINT "movimentos_id_produto_fk"
        FOREIGN KEY(id_produto) REFERENCES produtos(id_produto)
);

```

- Criando a Sequence "pessoa_id_seq":

```

m3n2 - loja.sql

CREATE SEQUENCE pessoa_id_seq;

ALTER TABLE pessoas
    ALTER COLUMN id_pessoa SET DEFAULT nextval('pessoa_id_seq');

ALTER TABLE pessoaFisica
    ALTER COLUMN id_pessoa SET DEFAULT nextval('pessoa_id_seq');

ALTER TABLE pessoaJuridica
    ALTER COLUMN id_pessoa SET DEFAULT nextval('pessoa_id_seq');

```

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

Resposta: Na cardinalidade 1x1 (um para um), cada linha em uma tabela está associada a uma única linha em outra tabela. Por exemplo, em um banco de dados de uma empresa, um funcionário tem um número de segurança social e cada número de segurança social pertence a um único funcionário. Isso é implementado colocando a chave estrangeira em uma das tabelas que se refere à chave primária da outra tabela.

Na cardinalidade 1xN (um para muitos), uma linha em uma tabela pode estar associada a várias linhas em outra tabela. Por exemplo, um departamento tem muitos funcionários. Isso é implementado colocando a chave estrangeira na tabela 'muitos' (funcionários) que se refere à chave primária da tabela 'um' (departamento).

Na cardinalidade NxN (muitos para muitos), várias linhas em uma tabela podem estar associadas a várias linhas em outra tabela. Por exemplo, um aluno pode se matricular em várias disciplinas e uma disciplina pode ter vários alunos. Isso é implementado usando uma tabela de junção (ou tabela de ligação) que contém chaves estrangeiras que se referem às chaves primárias de ambas as tabelas.

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

Resposta: A herança em bancos de dados relacionais pode ser representada principalmente de três maneiras:

Tabela por hierarquia: Nesse método, uma tabela é criada para cada classe na hierarquia. Cada tabela contém campos para seus atributos específicos, além de chaves estrangeiras para modelar o relacionamento de herança.

Tabela única: Aqui, uma única tabela é usada para representar todas as classes na hierarquia. A tabela contém todos os campos de todas as classes, e cada linha inclui os atributos relevantes para sua classe específica.

Tabela por subclasse: Nesse método, uma tabela é criada para cada classe e subclasse. Cada tabela contém apenas os campos específicos para essa classe ou subclasse, e as chaves estrangeiras são usadas para vincular subclasses às suas superclasses.

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

Resposta: O pgAdmin é uma ferramenta de administração para PostgreSQL que melhora a produtividade ao oferecer uma interface gráfica intuitiva para gerenciar objetos de banco de dados. Ele suporta várias versões do PostgreSQL e é multiplataforma, funcionando em Windows, Linux e macOS. Além disso, possui uma ferramenta de consulta poderosa com destaque de sintaxe, exibição gráfica de planos de consulta, um depurador de linguagem procedural, uma ferramenta de diferença de esquema para gerenciar diferenças entre esquemas e uma ferramenta ERD para projetar e documentar esquemas. A documentação extensa do pgAdmin também ajuda os usuários a entenderem e usar efetivamente a ferramenta.

2º Procedimento – Alimentando a Base

- Inserindo usuários no Banco de dados:

```
m3n2 - insertions.sql

insert into usuarios (id_usuario, login, senha)
values (1, 'op1', 'op1'), (2, 'op2', 'op2');
```

	id_usuario [PK] integer	login character varying (20)	senha character varying (255)
1	1	op1	op1
2	2	op2	op2

- Inserindo produtos no Banco de dados:

```
m3n2 - insertions.sql

insert into produtos (id_produto, nome, quantidade, preco_venda)
values
  (1, 'Banana', 100, 5.00),
  (2, 'Laranja', 500, 2.00),
  (3, 'Manga', 800, 4.00),
  (4, 'Uva', 200, 6.00),
  (5, 'Abacaxi', 300, 7.00);
```

	id_produto [PK] integer	nome character varying (100)	quantidade integer	preco_venda numeric (10,2)
1	1	Banana	100	5.00
2	2	Laranja	500	2.00
3	3	Manga	800	4.00
4	4	Uva	200	6.00
5	5	Abacaxi	300	7.00

- Inserindo pessoas físicas no Banco de dados:

```
m3n2 - insertions.sql

SELECT nextval('pessoa_id_seq');

INSERT INTO pessoas (id_pessoa, nome, logradouro, cidade, estado, telefone, email)
VALUES (1, 'Joao', 'Rua 12, casa 3, Quitanda', 'Riacho do Sul', 'PA', '1111-1111', 'joao@riacho.com');

INSERT INTO pessoaFisica (id_pessoa, cpf)
VALUES (1, '11111111111');

```

	id_pessoa integer	nome character varying (100)	logradouro character varying (100)	cidade character varying (100)	estado character varying (2)	telefone character varying (15)	email character varying (100)	id_pessoa integer	cpf character varying (14)
1	1	Joao	Rua 12, casa 3, Quitanda	Riacho do Sul	PA	1111-1111	joao@riacho.com	1	11111111111

- Inserindo pessoas jurídicas no Banco de dados:

```
m3n2 - insertions.sql

SELECT nextval('pessoa_id_seq');

INSERT INTO pessoas (id_pessoa, nome, logradouro, cidade, estado, telefone, email)
VALUES (2, 'JJC', 'Rua 11, Centro', 'Riacho do Norte', 'PA', '1212-1212', 'jjc@riacho.com');

INSERT INTO pessoaJuridica (id_pessoa, cnpj)
VALUES (2, '222222222222');

```

	id_pessoa integer	nome character varying (100)	logradouro character varying (100)	cidade character varying (100)	estado character varying (2)	telefone character varying (15)	email character varying (100)	id_pessoa integer	cnpj character varying (18)
1	2	JJC	Rua 11, Centro	Riacho do Norte	PA	1212-1212	jjc@riacho.com	2	222222222222

- Inserindo movimentos no Banco de dados:

```
m3n2 - Loja_DB.sql


-- CRIANDO A TABELA MOVIMENTOS
CREATE TABLE movimentos (
  id_movimento INT PRIMARY KEY,
  id_usuario INT NOT NULL,
  id_pessoa INT NOT NULL,
  id_produto INT NOT NULL,
  quantidade INTEGER NOT NULL,
  preco_unitario DECIMAL(10, 2) NOT NULL,
  tipo_movimento VARCHAR(10) NOT NULL CHECK (tipo_movimento IN ('E', 'S')),
  CONSTRAINT "movimentos_id_usuario_fk"
    FOREIGN KEY(id_usuario) REFERENCES usuarios(id_usuario),
  CONSTRAINT "movimentos_id_pessoa_fk"
    FOREIGN KEY(id_pessoa) REFERENCES pessoas(id_pessoa) ON DELETE CASCADE,
  CONSTRAINT "movimentos_id_produto_fk"
    FOREIGN KEY(id_produto) REFERENCES produtos(id_produto)
);

```

	id_movimento [PK] integer	id_usuario integer	id_pessoa integer	id_produto integer	quantidade integer	preco_unitario numeric (10,2)	tipo_movimento character varying (10)
1	1	1	2	1	20	4.00	S
2	2	1	2	2	15	2.00	S
3	3	2	2	5	30	7.00	S
4	4	1	4	4	50	6.00	E
5	5	2	4	3	10	5.00	E

- Consultas sobre os dados inseridos:

a) Dados completos de pessoas físicas.




```
m3n2 - consultas.sql

SELECT p.id_pessoa, p.nome, p.logradouro, p.cidade, p.estado, p.telefone, p.email, pf.id_pessoa, pf.cpf
FROM pessoas p
JOIN pessoa_fisica pf ON p.id_pessoa = pf.id_pessoa
```

	id_pessoa integer	nome character varying (100)	logradouro character varying (100)	cidade character varying (100)	estado character varying (2)	telefone character varying (15)	email character varying (100)	id_pessoa integer	cpf character varying (14)
1	1	Joao	Rua 12, casa 3, Quitanda	Riacho do Sul	PA	1111-1111	joao@riacho.com	1	111111111111

b) Dados completos de pessoas jurídicas.




```
m3n2 - consultas.sql

SELECT p.id_pessoa, p.nome, p.logradouro, p.cidade, p.estado, p.telefone, p.email, pj.id_pessoa, pj.cnpj
FROM pessoas p
JOIN pessoa_juridica pj ON p.id_pessoa = pj.id_pessoa
```

	id_pessoa integer	nome character varying (100)	logradouro character varying (100)	cidade character varying (100)	estado character varying (2)	telefone character varying (15)	email character varying (100)	id_pessoa integer	cnpj character varying (18)
1	2	JJC	Rua 11, Centro	Riacho do Norte	PA	1212-1212	jjc@riacho.com	2	22222222222222

c) Movimentos de entrada, com produto, fornecedor, quantidade, preço unitário e valor total.




```
m3n2 - consultas.sql

SELECT p.nome AS fornecedor, pr.nome AS produto, m.quantidade, m.preco_unitario, m.quantidade * m.preco_unitario AS valor_total
FROM movimentos m
JOIN produtos pr ON m.id_produto = pr.id_produto
JOIN pessoas p ON m.id_pessoa = p.id_pessoa
WHERE m.tipo_movimento = 'E';
```

	fornecedor character varying (100) 🔒	produto character varying (100) 🔒	quantidade integer 🔒	preco_unitario numeric (10,2) 🔒	valor_total numeric 🔒
1	JJC	Manga	10	5.00	50.00
2	JJC	Uva	50	6.00	300.00

d) Movimentos de saída, com produto, fornecedor, quantidade, preço unitário e valor total.




```
m3n2 - consultas.sql

SELECT p.nome AS comprador, pr.nome AS produto, m.quantidade, m.preco_unitario, m.quantidade * m.preco_unitario AS valor_total
FROM movimentos m
JOIN produtos pr ON m.id_produto = pr.id_produto
JOIN pessoas p ON m.id_pessoa = p.id_pessoa
WHERE m.tipo_movimento = 'S';
```

	fornecedor character varying (100) 🔒	produto character varying (100) 🔒	quantidade integer 🔒	preco_unitario numeric (10,2) 🔒	valor_total numeric 🔒
1	Joao	Banana	20	4.00	80.00
2	Joao	Laranja	15	2.00	30.00
3	Joao	Abacaxi	30	7.00	210.00

e) Valor total das entradas agrupadas por produto.



```
m3n2 - consultas.sql

SELECT pr.nome AS produto, SUM(m.quantidade * m.preco_unitario) AS valor_total
FROM movimentos m
JOIN produtos pr ON m.id_produto = pr.id_produto
WHERE m.tipo_movimento = 'E'
GROUP BY pr.nome;
```

	produto character varying (100) 🔒	valor_total numeric 🔒
1	Manga	50.00
2	Uva	300.00

f) Valor total das saídas agrupadas por produto.

```
m3n2 - consultas.sql

SELECT pr.nome AS produto, SUM(m.quantidade * m.preco_unitario) AS valor_total
FROM movimentos m
JOIN produtos pr ON m.id_produto = pr.id_produto
WHERE m.tipo_movimento = 'S'
GROUP BY pr.nome;
```

	produto character varying (100) 🔒	valor_total numeric 🔒
1	Abacaxi	210.00
2	Banana	80.00
3	Laranja	30.00

g) Operadores que não efetuaram movimentações de entrada (compra).

```
m3n2 - consultas.sql

SELECT u.login as operador
FROM usuarios u
LEFT JOIN movimentos m ON u.id_usuario = m.id_usuario AND m.tipo_movimento = 'E'
WHERE m.id_movimento IS NULL;
```

operador character varying (20) 🔒

h) Valor total de entrada, agrupado por operador

```
m3n2 - consultas.sql

SELECT u.login as operador, SUM(m.quantidade * m.preco_unitario) AS valor_total
FROM movimentos m
JOIN usuarios u ON m.id_usuario = u.id_usuario
WHERE m.tipo_movimento = 'E'
GROUP BY u.login;
```

	operador character varying (20) 🔒	valor_total numeric 🔒
1	op1	300.00
2	op2	50.00

i) Valor total de saída, agrupado por operador

```
m3n2 - consultas.sql

SELECT u.login as operador, SUM(m.quantidade * m.preco_unitario) AS valor_total
FROM movimentos m
JOIN usuarios u ON m.id_usuario = u.id_usuario
WHERE m.tipo_movimento = 'S'
GROUP BY u.login;
```

	operador character varying (20) 🔒	valor_total numeric 🔒
1	op1	110.00
2	op2	210.00

j) Valor médio de venda por produto, utilizando média ponderada.

```
m3n2 - consultas.sql

SELECT id_produto, CAST(SUM(preco_unitario * quantidade) / SUM(quantidade) AS DECIMAL(10, 2)) AS valor_medio_venda
FROM movimentos
WHERE tipo_movimento = 'S'
GROUP BY id_produto;
```

	id_produto integer 🔒	valor_medio_venda numeric (10,2) 🔒
1	1	4.00
2	2	2.00
3	5	7.00

a) Quais as diferenças no uso de sequence e identity?

Resposta: No PostgreSQL, sequence e identity são usados para gerar valores numéricos únicos. sequence é um objeto de banco de dados independente que pode ser usado por várias tabelas para gerar valores únicos. identity é uma

propriedade de coluna usada para gerar valores únicos para uma coluna específica em uma tabela. Enquanto sequence oferece mais flexibilidade e controle, identity é mais simples de usar, pois gera automaticamente valores durante as operações de inserção.

b) Qual a importância das chaves estrangeiras para a consistência do banco?

Resposta: As chaves estrangeiras são fundamentais para a consistência do banco de dados, pois garantem a integridade referencial, prevenindo dados inconsistentes, suportando operações em cascata e melhorando a qualidade dos dados. Elas asseguram que as relações entre as tabelas permaneçam consistentes e que apenas valores válidos sejam inseridos em uma coluna de chave estrangeira.

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

Resposta: Na álgebra relacional, os operadores correspondentes no SQL incluem SELECT (para a operação de seleção), PROJECT (para a operação de projeção), UNION (para a operação de união), SET DIFFERENCE (para a operação de diferença de conjuntos) e CROSS PRODUCT (para a operação de produto cartesiano).

O cálculo relacional, por outro lado, é uma linguagem de consulta declarativa. Não especifica uma sequência de operações para realizar a consulta, mas descreve o resultado desejado. O SQL é baseado em grande parte no cálculo relacional de tuplas. Portanto, muitas das cláusulas do SQL, como WHERE, podem ser vistas como parte do cálculo relacional.

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

Resposta: O agrupamento em consultas SQL é feito usando a cláusula GROUP BY, que agrupa linhas que têm os mesmos valores nas colunas especificadas. É obrigatório usar uma função de agregação, como COUNT, SUM, AVG, MAX ou MIN, para resumir os dados de cada grupo.

Conclusão

Foi uma missão muito importante, pois aprendi a modelar um banco de dados, o que facilitou demais a implantação do mesmo através da DDL e a manipulação através da DML, inserindo novos dados ao banco de dados e recuperar os dados inseridos.