



Universidade Federal do Ceará

Relatório de Projeto de Banco de Dados

Discentes: Tailan de Souza Oliveira (553381)

Kaio Davidson Santos Sampaio (554698)

Docente: Livia Almada Cruz

Disciplina: Fundamentos de Banco de Dados (FBD)

Título: Sistema de Gerenciamento de Remédios

07 de Abril de 2024

Nota ABNT: Este relatório segue as normas da Associação Brasileira de Normas Técnicas (ABNT).

Conteúdo

1	Introdução	1
2	Requisitos	1
2.1	Requisitos Funcionais	2
2.2	Requisitos de Relatórios	4
3	Desenvolvimento	4
3.1	Entidades, Atributos e Relacionamentos	4
3.2	Entidade: Remédio	4
3.3	Entidade: Fornecedor	5
3.4	Entidade: Pedido	6
3.5	Entidade: Cliente	6
3.6	Entidade: Funcionário	7
3.7	Entidade: Estoque	7
4	Diagrama ER/EER	8
5	Delegação de Atividades	8
6	Conclusão	8
7	Introdução	10
8	Estrutura do Banco de Dados	10
8.1	Script para criação do banco de dados	10
8.2	Populando as tabelas de maneira não aleatória	15
8.3	Ver todas as tabelas com os dados informados;	20
8.4	Logo abaixo modelo relacional do projeto de gerenciamento de remédios	22
9	O banco de dados é estruturado com as seguintes tabelas principais:	22
9.1	Tabela <code>remedio</code>	22
9.2	Tabela <code>genericos</code>	22
9.3	Tabela <code>tarjas</code>	23
9.4	Tabela <code>de_marcas</code>	23
9.5	Tabela <code>pedido</code>	23
9.6	Tabela <code>esta_incluido</code>	24
9.7	Tabela <code>fornecedor</code>	24
9.8	Tabela <code>solicitacao</code>	25

9.9	Tabela <code>funcionario</code>	25
9.10	Tabela <code>realiza</code>	25
9.11	Tabela <code>estoque</code>	26
9.12	Tabela <code>possui</code>	26
9.13	Tabela <code>cliente</code>	26
9.14	Tabela <code>telefone</code>	27
9.15	Tabela <code>historico_compra</code>	27
9.16	Tabela <code>faz_parte</code>	27
9.17	Tabela <code>tem</code>	28
10	Como Funciona um CRUD	28
10.1	Imagem original sem nenhuma implementação (Tabela remédios)	29
10.2	Create (Criar)	29
10.3	Após usar o CREATE	32
10.4	Read (Ler)	32
10.5	Resultando após usar o READ	35
10.6	Antes de utilizar o UPDATE	35
10.7	Update (Atualizar)	35
10.8	Após usar o UPDATE	39
10.9	Delete (Excluir)	39
11	Conclusão	40
12	Referências	43
13	Link do Repositório	43
14	Atualizações em relação a ultima entrega	1
14.1	Sugestões Lívia Almada	1
14.2	Mudanças realizadas por Tailan e por Kaio	1
15	CRUD da Tabela Remédio	1
15.1	Comando para instalar as dependências	1
15.2	Importações	1
15.3	Configuração com o banco de dados(Mudar os parâmentros conforme o seu BD)	2
15.4	CRUD da aplicação	2
15.5	Interface da aplicação	3
16	CRUD da tabela cliente	5
16.1	Comando para instalar as dependências	5
16.2	Configuração banco de dados	6

16.3 CRUD da tabela cliente	6
17 Consulta da tabela Cliente	10
18 Consulta da tabela Remédio	12
19 Link do Repositório	14

1 Introdução

Neste relatório, delineamos o projeto de banco de dados para um sistema abrangente de gerenciamento de medicamentos. Este sistema foi concebido com o objetivo primordial de aprimorar o controle e a gestão de informações essenciais relacionadas a clientes, produtos (remédios), vendas e funcionários, direcionado especificamente para farmácias ou estabelecimentos similares. Através da implementação deste sistema, buscamos oferecer uma solução eficiente para otimizar processos, garantindo um gerenciamento fluido e preciso das operações diárias, desde o controle de estoque até o acompanhamento das transações e interações com clientes e fornecedores.

2 Requisitos

A seguir, apresentamos os requisitos do sistema de gerenciamento de remédios:

- Entidades Principais
 1. Cliente
 2. Remédio
 3. Fornecedor
 4. Funcionário
 5. Estoque
 6. Pedido
- Relacionamentos
 1. Fornecedor-Pedido
 2. Funcionário-Pedido
 3. Remédio-Pedido
 4. Estoque-Remédio
 5. Remédio-Cliente
 6. Cliente-Histórico_Compras
 7. Histórico_Compras-Remédio
- Atributos
 1. Cliente:

- Nome, cpf, Telefone, Endereço, Número da Casa, Bairro, Cidade, Rua.
- 2. Remédio: Atributos e subgrupos
 - Genéricos, Tarjas, Cores da Tarjas, Outras Categorias, de Marca, nome, descrição, id
- 3. Fornecedor:
 - Nome de Empresa, Email, Endereço, Número, Rua, Bairro, Cidade, Estado, CNPJ.
- 4. Funcionário:
 - Salário, ID Funcionário, Cargo, Nome.
- 5. Estoque:
 - Quantidade no Estoque, ID_estoque, Data de Entrada no Estoque, Data de validade, Unidade de Medida, Unidade, Cartelas, Caixas, Frascos
- 6. Pedido:
 - Data do Pedido, Quantidade de Remédios Solicitados, Status do Pedido, ID Pedido, Código de Rastreamento.
- Chaves Primárias e Estrangeiras
 - Consulte as chaves primárias e estrangeiras conforme mencionadas no documento original.
- Restrições de Integridade
 - Restrições de integridade referencial para garantir que as chaves estrangeiras estejam sempre relacionadas a uma chave primária existente.
 - Restrições de domínio para garantir que os valores dos atributos estejam dentro de um intervalo aceitável (por exemplo, preço não pode ser negativo).

2.1 Requisitos Funcionais

- **RF001 - Cadastro de Medicamentos:**
 - **Descrição:** Permitir o cadastro, edição, remoção e listagem de medicamentos. Cada medicamento deve ter um nome e descrição.
 - **Restrição:** O nome do medicamento deve ser único no sistema.

- **Subgrupos:**
 - * **Genéricos:**
 - **Descrição:** Medicamentos genéricos produzidos por diferentes laboratórios.
 - * **De Marca:**
 - **Descrição:** Medicamentos de marca registrada produzidos por empresas específicas.
- **RF002 - Registro de Fornecedores:**
 - **Descrição:** Possibilitar o registro de novos fornecedores, incluindo informações como nome da empresa, CNPJ, endereço e contato.
 - **Restrição:** O CNPJ do fornecedor deve ser único no sistema.
- **RF003 - Registros de Pedidos:**
 - **Descrição:** Permitir que os funcionários registrem os pedidos de medicamentos feitos pelos clientes, incluindo detalhes como a data do pedido, quantidade solicitada, status do pedido e cliente associado.
 - **Restrição:** Deve ser possível associar vários medicamentos a um único pedido.
- **RF004 - Controle de Estoque:**
 - **Descrição:** Manter um registro atualizado do estoque de medicamentos, incluindo a quantidade disponível, data de entrada, data de validade e unidade de medida.
 - **Restrição:** Deve ser possível visualizar o estoque disponível para cada medicamento.
- **RF005 - Registro de Clientes:**
 - **Descrição:** Permitir o registro de novos clientes, incluindo informações como nome, CPF, telefone, endereço e histórico de compras.
 - **Restrição:** O CPF do cliente deve ser único no sistema.
- **RF006 - Geração de Relatórios:**
 - **Descrição:** Gerar relatórios sobre vendas, estoque, clientes, fornecedores e pedidos para auxiliar na gestão do estabelecimento.

- **Restrição:** Os relatórios devem ser facilmente acessíveis e customizáveis.
- **RF007 - Autenticação de Usuários:**
 - **Descrição:** Implementar um sistema de autenticação de usuários para garantir que apenas funcionários autorizados possam acessar as funcionalidades do sistema.
 - **Restrição:** Cada usuário deve ter um login único e uma senha segura para acesso.

2.2 Requisitos de Relatórios

- **RR001 - Relatório de Medicamentos Mais Vendidos:**
 - **Descrição:** Gerar um relatório listando os medicamentos mais vendidos dentro de um período específico, mostrando a quantidade de vendas realizadas para cada medicamento.
- **RR002 - Relatório de Estoque Mínimo:**
 - **Descrição:** Criar um relatório que liste os medicamentos que estão com estoque abaixo de um nível mínimo pré-definido, indicando a necessidade de reabastecimento.
- **RR003 - Relatório de Vendas por Cliente:**
 - **Descrição:** Fornecer um relatório detalhado das vendas realizadas para cada cliente, incluindo o total de compras, medicamentos adquiridos e datas das transações.

3 Desenvolvimento

3.1 Entidades, Atributos e Relacionamentos

A seguir, apresentamos as principais entidades, seus atributos e relacionamentos para o Sistema de Gerenciamento de Remédios:

3.2 Entidade: Remédio

- id (identificador único)
- Nome

- Descrição

Subgrupo da entidade remédio:

- Entidade adicional: genérico
- Entidade adicional: de marca
- Entidade adicional: outras categorias
- Entidade adicional: Tarja

Subgrupo da entidade remédio da entidade Tarja:

- Atributo adicional: Cores da tarja

Relacionamentos:

- Um pedido pode conter vários remédios (Relacionamento 1:N)
- Um remédio pode estar presente em vários pedidos (Relacionamento N:1)

3.3 Entidade: Fornecedor

- Nome da Empresa
- E-mail
- CNPJ
- Endereço
- Número
- Rua
- Bairro
- Cidade
- Estado

Relacionamentos:

- Um fornecedor pode fornecer vários pedidos (Relacionamento 1:N)
- Um pedido é feito para um único fornecedor (Relacionamento N:1)

3.4 Entidade: Pedido

- ID do Pedido (identificador único)
- Data do Pedido
- Quantidade de remédios solicitados
- Status do pedido
- Código de rastreamento

Relacionamentos:

- Um remédio pode estar associado a vários pedidos (Relacionamento N:1)
- Um pedido pode conter vários remédios (Relacionamento N:1)

3.5 Entidade: Cliente

- Nome
- CPF
- Telefone
- Endereço
- Número da casa
- Bairro
- Cidade
- Rua

Relacionamentos:

- Um cliente pode ter vários históricos de compras (Relacionamento 1:N)
- Cada compra é associada a um único cliente (Relacionamento 1:1)
- Cada cliente pode comprar vários remédios (Relacionamento N:M)
- Cada remédio pode ser comprado por vários clientes (Relacionamento N:M)

3.6 Entidade: Funcionário

- ID do Funcionário (identificador único)
- Nome
- Cargo
- Salário

Relacionamentos:

- Um funcionário pode ser responsável por vários pedidos (Relacionamento 1:N)
- Cada pedido é atribuído a um único funcionário (por exemplo, quem fez o pedido, quem processou, etc.).

3.7 Entidade: Estoque

- Data de validade
- Data de entrada no estoque
- Quantidade no estoque
- Unidade de medida
- Frascos
- Caixas
- Cartelas
- Unidade

Relacionamentos:

- Um estoque pode conter vários tipos de remédios em diferentes quantidades (Relacionamento 1:N)
- Um remédio pode estar presente em vários estoques, cada um com sua própria quantidade (Relacionamento N:1)

4 Diagrama ER/EER

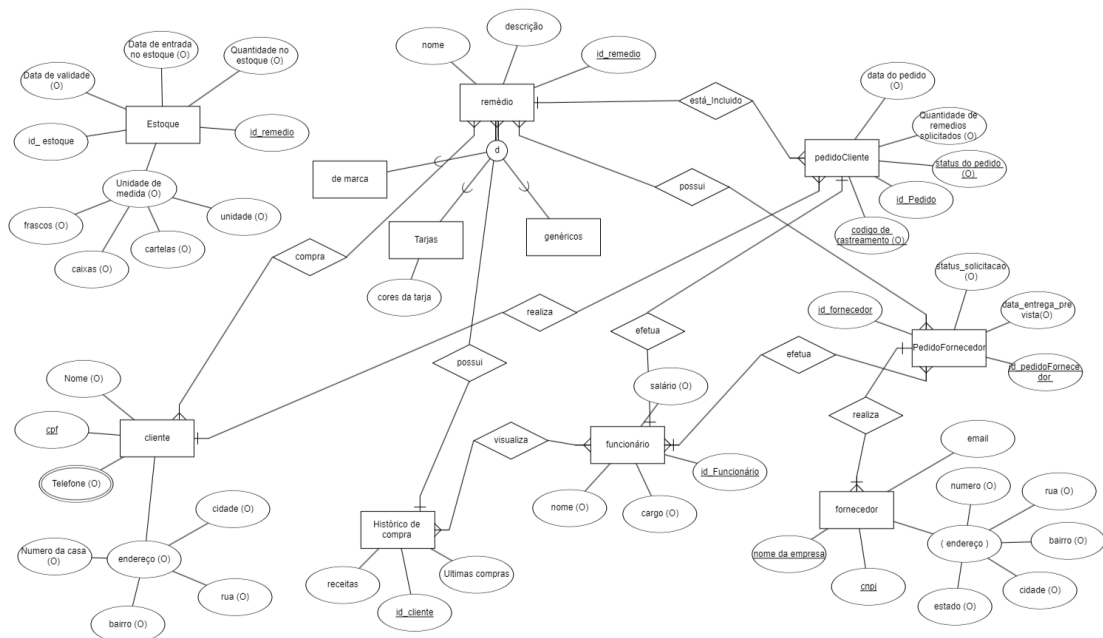


Figura 1: Diagrama ER/EER do Sistema de Gerenciamento de Remédios

5 Delegação de Atividades

6 Conclusão

O projeto de banco de dados para o Sistema de Gerenciamento de remédios abrangeu a identificação das principais entidades e seus atributos, bem como a delegação de atividades entre os membros da equipe. Através dessa análise,

Membro da Equipe	Atividade
Tailan	Desenvolvimento do modelo EER
Tailan	Análise final da etapa 1 do modelo EER
Kaio	Levantamento de requisitos do Sistema
Kaio	Documentação de Requisitos

Tabela 1: Delegação de Atividades

foi possível estabelecer uma estrutura sólida para o desenvolvimento do sistema, visando facilitar o controle e a gestão de informações relacionadas a clientes, produtos, vendas e funcionários.

[12pt, a4paper]abntex2
 graphicx longtable lscape
 [alf]abntex2cite
 geometry top=3cm, bottom=2cm, left=2cm, right=2cm

Relatório de Projeto: Gerenciamento de Remédios

18 de setembro de 2024

Resumo

Este relatório apresenta o desenvolvimento de um sistema para o gerenciamento de remédios, detalhando as principais tabelas do banco de dados e as operações CRUD implementadas na aplicação. O sistema utiliza o Flask para a construção da API e PostgreSQL como sistema de gerenciamento de banco de dados.

Conteúdo

7 Introdução

O projeto tem como objetivo criar uma aplicação para o gerenciamento de remédios, com funcionalidades de CRUD para diferentes tabelas do banco de dados. A aplicação é construída utilizando Flask para a API e PostgreSQL para o banco de dados.

8 Estrutura do Banco de Dados

8.1 Script para criação do banco de dados

```
CREATE SCHEMA farmacia;  
SET SCHEMA 'farmacia';  
  
-- Tabela Remédio  
CREATE TABLE remedio (  
    id_remedio SERIAL PRIMARY KEY,
```

```

        nome VARCHAR(255) NOT NULL,
        descricao TEXT NOT NULL
    );

-- Tabela Genéricos
CREATE TABLE generico (
    id_generico SERIAL PRIMARY KEY,
    id_remedio INTEGER NOT NULL,
    CONSTRAINT fk_generico_remedio FOREIGN KEY (id_remedio)
    REFERENCES remedio (id_remedio) ON DELETE CASCADE
);

-- Tabela Tarjas
CREATE TABLE tarja (
    id_tarja SERIAL PRIMARY KEY,
    id_remedio INTEGER NOT NULL,
    cores_tarjas VARCHAR(255) NOT NULL,
    CONSTRAINT fk_tarja_remedio FOREIGN KEY (id_remedio)
    REFERENCES remedio (id_remedio) ON DELETE CASCADE
);

-- Tabela Marcas
CREATE TABLE marca (
    id_marca SERIAL PRIMARY KEY,
    id_remedio INTEGER NOT NULL,
    nome_marca VARCHAR(255) NOT NULL,
    CONSTRAINT fk_marca_remedio FOREIGN KEY (id_remedio)
    REFERENCES remedio (id_remedio) ON DELETE CASCADE
);

-- Tabela Cliente
CREATE TABLE cliente (
    id_cliente SERIAL PRIMARY KEY,
    nome_cliente VARCHAR(255) NOT NULL,
    cpf CHAR(11) CHECK (char_length(cpf) = 11) NOT NULL UNIQUE,
    endereco_ rua VARCHAR(255),
    endereco_numero INTEGER,
    endereco_bairro VARCHAR(255),
    endereco_cidade VARCHAR(255),
    endereco_estado CHAR(2) -- Adicionada coluna para estado
);

```

```

-- Tabela Telefone
CREATE TABLE telefone (
    id_telefone SERIAL PRIMARY KEY,
    id_cliente INTEGER NOT NULL,
    numero_telefone VARCHAR(20) UNIQUE,
    CONSTRAINT fk_telefone_cliente FOREIGN KEY (id_cliente)
    REFERENCES cliente (id_cliente) ON DELETE CASCADE
);

-- Tabela Fornecedor
CREATE TABLE fornecedor (
    id_fornecedor SERIAL PRIMARY KEY,
    nome_empresa VARCHAR(255) UNIQUE NOT NULL,
    cnpj CHAR(14) NOT NULL UNIQUE CHECK (char_length(cnpj) = 14),
    email VARCHAR(255),
    endereco_rua VARCHAR(255),
    endereco_numero INTEGER,
    endereco_bairro VARCHAR(255),
    endereco_cidade VARCHAR(255),
    endereco_estado CHAR(2) NOT NULL
);

-- Tabela Pedidos de Clientes
CREATE TABLE pedido_cliente (
    id_pedido SERIAL PRIMARY KEY,
    id_cliente INTEGER NOT NULL,
    data_pedido DATE NOT NULL,
    qtd_rem_solic INTEGER NOT NULL,
    cod_rastr VARCHAR(50) UNIQUE,
    status_pedido VARCHAR(50) NOT NULL CHECK (status_pedido IN ('EM PRODUÇÃO', '
    CONSTRAINT fk_pedido_cliente FOREIGN KEY (id_cliente)
    REFERENCES cliente (id_cliente) ON DELETE CASCADE
);

-- Tabela Pedidos de Fornecedores
CREATE TABLE pedido_fornecedor (
    id_pedido_fornecedor SERIAL PRIMARY KEY,
    id_fornecedor INTEGER NOT NULL,
    data_solicitacao DATE NOT NULL,
    data_entrega_prevista DATE,

```



```

        status_solicitacao VARCHAR(50) NOT NULL,
        CONSTRAINT fk_pedido_fornecedor FOREIGN KEY (id_fornecedor)
        REFERENCES fornecedor (id_fornecedor) ON DELETE CASCADE
    );

-- Tabela Relacional entre Remédio e Pedido de Cliente
CREATE TABLE pedido_cliente_remedio (
    id_pedido INTEGER NOT NULL,
    id_remedio INTEGER NOT NULL,
    PRIMARY KEY (id_pedido, id_remedio),
    CONSTRAINT fk_pedido_cliente_remedio FOREIGN KEY (id_pedido)
    REFERENCES pedido_cliente (id_pedido) ON DELETE CASCADE,
    CONSTRAINT fk_remedio_pedido_cliente FOREIGN KEY (id_remedio)
    REFERENCES remedio (id_remedio) ON DELETE CASCADE
);

-- Tabela Relacional entre Remédio e Pedido de Fornecedor
CREATE TABLE pedido_fornecedor_remedio (
    id_pedido_fornecedor INTEGER NOT NULL,
    id_remedio INTEGER NOT NULL,
    PRIMARY KEY (id_pedido_fornecedor, id_remedio),
    CONSTRAINT fk_pedido_fornecedor_remedio FOREIGN KEY (id_pedido_fornecedor)
    REFERENCES pedido_fornecedor (id_pedido_fornecedor) ON DELETE CASCADE,
    CONSTRAINT fk_remedio_pedido_fornecedor FOREIGN KEY (id_remedio)
    REFERENCES remedio (id_remedio) ON DELETE CASCADE
);

-- Tabela Funcionário
CREATE TABLE funcionario (
    id_funcionario SERIAL PRIMARY KEY,
    nome_funcionario VARCHAR(255) NOT NULL, -- Mudança no nome da coluna para s
    cargo VARCHAR(255) NOT NULL,
    salario NUMERIC(10, 2) NOT NULL
);

-- Tabela Relacional entre Funcionário e Pedido de Cliente
CREATE TABLE realiza_cliente (
    id_funcionario INTEGER NOT NULL,
    id_pedido INTEGER NOT NULL,
    PRIMARY KEY (id_funcionario, id_pedido),
    CONSTRAINT fk_realiza_cliente_funcionario FOREIGN KEY (id_funcionario)

```

```

REFERENCES funcionario (id_funcionario) ON DELETE CASCADE,
CONSTRAINT fk_realiza_cliente_pedido FOREIGN KEY (id_pedido)
REFERENCES pedido_cliente (id_pedido) ON DELETE CASCADE
);

-- Tabela Relacional entre Funcionário e Pedido de Fornecedor
CREATE TABLE realiza_fornecedor (
    id_funcionario INTEGER NOT NULL,
    id_pedido_fornecedor INTEGER NOT NULL,
    PRIMARY KEY (id_funcionario, id_pedido_fornecedor),
    CONSTRAINT fk_realiza_fornecedor_funcionario FOREIGN KEY (id_funcionario)
REFERENCES funcionario (id_funcionario) ON DELETE CASCADE,
CONSTRAINT fk_realiza_fornecedor_pedido FOREIGN KEY (id_pedido_fornecedor)
REFERENCES pedido_fornecedor (id_pedido_fornecedor) ON DELETE CASCADE
);

-- Tabela Estoque
CREATE TABLE estoque (
    id_estoque SERIAL PRIMARY KEY,
    id_remedio INTEGER NOT NULL,
    data_entrega_stq DATE,
    qtd_stq INTEGER NOT NULL,
    data_vali_stq DATE NOT NULL,
    medida_frascos INTEGER,
    medida_caixas INTEGER,
    medida_cartelas INTEGER,
    medida_unidade INTEGER,
    CONSTRAINT fk_estoque_remedio FOREIGN KEY (id_remedio)
REFERENCES remedio (id_remedio) ON DELETE CASCADE
);

-- Tabela Histórico de Compra
CREATE TABLE historico_compra (
    id_historico SERIAL PRIMARY KEY,
    receitas TEXT, -- Ajustado para permitir textos maiores
    ult_compra DATE, -- Mudança para tipo DATE, dado o contexto de datas
    id_cliente INTEGER NOT NULL,
    CONSTRAINT fk_historico_cliente FOREIGN KEY (id_cliente)
REFERENCES cliente (id_cliente) ON DELETE CASCADE
);

```

8.2 Populando as tabelas de maneira não aleatória

```
-- Inserir dados na tabela Remédio
INSERT INTO remedio (nome, descricao) VALUES
('Paracetamol', 'Analgésico e antipirético utilizado para dor e febre.'),
('Amoxicilina', 'Antibiótico usado para tratar infecções bacterianas.'),
('Omeprazol', 'Inibidor da bomba de prótons para tratamento de úlceras e refluxo'),
('Ibuprofeno', 'Anti-inflamatório não esteroide para dor e febre.'),
('Dipirona', 'Analgésico e antipirético para dor e febre.'),
('Azitromicina', 'Antibiótico para infecções respiratórias e de pele.'),
('Lorazepam', 'Ansiolítico para tratamento de ansiedade e insônia.'),
('Metformina', 'Antidiabético para controle da glicose no sangue.'),
('Cetoconazol', 'Antifúngico para infecções por fungos.'),
('Cloridrato de Tramadol', 'Analgésico para tratamento de dor intensa.');
```

```
-- Inserir dados na tabela Genéricos
INSERT INTO generico (id_remedio) VALUES
(1), -- Paracetamol
(2), -- Amoxicilina
(3), -- Omeprazol
(4), -- Ibuprofeno
(5), -- Dipirona
(6), -- Azitromicina
(7), -- Lorazepam
(8), -- Metformina
(9), -- Cetoconazol
(10); -- Cloridrato de Tramadol
```

```
-- Inserir dados na tabela Tarjas
INSERT INTO tarja (id_remedio, cores_tarjas) VALUES
(1, 'Amarela'), -- Paracetamol
(2, 'Vermelha'), -- Amoxicilina
(3, 'Azul'), -- Omeprazol
(4, 'Verde'), -- Ibuprofeno
(5, 'Preta'), -- Dipirona
(6, 'Rosa'), -- Azitromicina
(7, 'Laranja'), -- Lorazepam
(8, 'Cinza'), -- Metformina
(9, 'Branca'), -- Cetoconazol
(10, 'Vinho'); -- Cloridrato de Tramadol
```

```

-- Inserir dados na tabela Marcas
INSERT INTO marca (id_remedio, nome_marca) VALUES
(1, 'Marca A'), -- Paracetamol
(2, 'Marca B'), -- Amoxicilina
(3, 'Marca C'), -- Omeprazol
(4, 'Marca D'), -- Ibuprofeno
(5, 'Marca E'), -- Dipirona
(6, 'Marca F'), -- Azitromicina
(7, 'Marca G'), -- Lorazepam
(8, 'Marca H'), -- Metformina
(9, 'Marca I'), -- Cetoconazol
(10, 'Marca J'); -- Cloridrato de Tramadol

-- Inserir dados na tabela Cliente
INSERT INTO cliente (nome_cliente, cpf, endereco_rua, endereco_numero, endereco_cidade, endereco_estado) VALUES
('João Silva', '12345678901', 'Rua A', 123, 'Bairro X', 'Cidade Y', 'SP'),
('Maria Oliveira', '10987654321', 'Rua B', 456, 'Bairro Y', 'Cidade Z', 'RJ'),
('Pedro Santos', '32165498700', 'Rua C', 789, 'Bairro Z', 'Cidade W', 'MG'),
('Ana Lima', '65432198700', 'Rua D', 101, 'Bairro A', 'Cidade V', 'SP'),
('Lucia Costa', '98765432100', 'Rua E', 202, 'Bairro B', 'Cidade U', 'BA'),
('Carlos Almeida', '01234567890', 'Rua F', 303, 'Bairro C', 'Cidade T', 'PE'),
('Fernanda Silva', '19876543210', 'Rua G', 404, 'Bairro D', 'Cidade S', 'SC'),
('Roberto Oliveira', '12378945600', 'Rua H', 505, 'Bairro E', 'Cidade R', 'PR'),
('Juliana Pereira', '45612378900', 'Rua I', 606, 'Bairro F', 'Cidade Q', 'DF'),
('Ricardo Martins', '78945612300', 'Rua J', 707, 'Bairro G', 'Cidade P', 'GO');

-- Inserir dados na tabela Telefone
INSERT INTO telefone (id_cliente, numero_telefone) VALUES
(1, '123456789'), -- João Silva
(2, '987654321'), -- Maria Oliveira
(3, '111222333'), -- Pedro Santos
(4, '444555666'), -- Ana Lima
(5, '777888999'), -- Lucia Costa
(6, '000111222'), -- Carlos Almeida
(7, '333444555'), -- Fernanda Silva
(8, '666777888'), -- Roberto Oliveira
(9, '999000111'), -- Juliana Pereira
(10, '222333444'); -- Ricardo Martins

-- Inserir dados na tabela Fornecedor
INSERT INTO fornecedor (nome_empresa, cnpj, email, endereco_rua, endereco_numero, endereco_cidade, endereco_estado) VALUES

```

```
( 'Farmácia Central', '12345678000195', 'contato@farmaciacentral.com.br', 'Rua C',
( 'Laboratório Saúde', '09876543000187', 'suporte@laboratoriosauade.com.br', 'Rua
( 'Distribuidora Farma', '23456789000198', 'vendas@distribuidorafarma.com.br', 'R
( 'Medicamentos & Cia', '34567890000101', 'info@medicamentoscia.com.br', 'Rua F',
( 'Farmácia Popular', '45678901000112', 'contato@farmaciapopular.com.br', 'Rua G'
( 'Farma Mais', '56789012000123', 'atendimento@farmamais.com.br', 'Rua H', 505, '
( 'Laboratório Vida', '67890123000134', 'suporte@laboratoriodvida.com.br', 'Rua I
( 'Saúde em Casa', '78901234000145', 'contato@saudeemcasa.com.br', 'Rua J', 707,
( 'Distribuidora de Medicamentos', '89012345000156', 'vendas@distribuidoramedicam
( 'Medicamentos Brasil', '90123456000167', 'info@medicamentosbrasil.com.br', 'Rua
```

```
-- Inserir dados na tabela Pedidos de Clientes
```

```
INSERT INTO pedido_cliente (id_cliente, data_pedido, qtd_rem_solic, cod_rastr, s
(1, '2024-09-01', 2, 'COD1234', 'EM PRODUÇÃO'),
(2, '2024-09-05', 1, 'COD5678', 'A CAMINHO'),
(3, '2024-09-07', 3, 'COD9101', 'ENTREGUE'),
(4, '2024-09-10', 1, 'COD1121', 'EM PRODUÇÃO'),
(5, '2024-09-12', 2, 'COD3141', 'A CAMINHO'),
(6, '2024-09-15', 1, 'COD5161', 'ENTREGUE'),
(7, '2024-09-17', 4, 'COD7181', 'EM PRODUÇÃO'),
(8, '2024-09-20', 2, 'COD9202', 'A CAMINHO'),
(9, '2024-09-22', 3, 'COD2232', 'ENTREGUE'),
(10, '2024-09-25', 1, 'COD3243', 'EM PRODUÇÃO');
```

```
-- Inserir dados na tabela Pedidos de Fornecedores
```

```
INSERT INTO pedido_fornecedor (id_fornecedor, data_solicitacao, data_entrega_pre
(1, '2024-09-01', '2024-09-10', 'PENDENTE'),
(2, '2024-09-03', '2024-09-12', 'PENDENTE'),
(3, '2024-09-05', '2024-09-15', 'PENDENTE'),
(4, '2024-09-07', '2024-09-17', 'PENDENTE'),
(5, '2024-09-10', '2024-09-20', 'PENDENTE'),
(6, '2024-09-12', '2024-09-22', 'PENDENTE'),
(7, '2024-09-15', '2024-09-25', 'PENDENTE'),
(8, '2024-09-17', '2024-09-27', 'PENDENTE'),
(9, '2024-09-20', '2024-09-30', 'PENDENTE'),
(10, '2024-09-22', '2024-10-02', 'PENDENTE');
```

```
-- Inserir dados na tabela Relacional entre Remédio e Pedido de Cliente
```

```
INSERT INTO pedido_cliente_remedio (id_pedido, id_remedio) VALUES
(1, 1), -- Paracetamol no Pedido 1
(1, 2), -- Amoxicilina no Pedido 1
```

```

(2, 3), -- Omeprazol no Pedido 2
(3, 4), -- Ibuprofeno no Pedido 3
(3, 5), -- Dipirona no Pedido 3
(4, 6), -- Azitromicina no Pedido 4
(5, 7), -- Lorazepam no Pedido 5
(6, 8), -- Metformina no Pedido 6
(7, 9), -- Cetoconazol no Pedido 7
(8, 10); -- Cloridrato de Tramadol no Pedido 8

-- Inserir dados na tabela Relacional entre Remédio e Pedido de Fornecedor
INSERT INTO pedido_fornecedor_remedio (id_pedido_fornecedor, id_remedio) VALUES
(1, 1), -- Paracetamol no Pedido de Fornecedor 1
(1, 2), -- Amoxicilina no Pedido de Fornecedor 1
(2, 3), -- Omeprazol no Pedido de Fornecedor 2
(3, 4), -- Ibuprofeno no Pedido de Fornecedor 3
(4, 5), -- Dipirona no Pedido de Fornecedor 4
(5, 6), -- Azitromicina no Pedido de Fornecedor 5
(6, 7), -- Lorazepam no Pedido de Fornecedor 6
(7, 8), -- Metformina no Pedido de Fornecedor 7
(8, 9), -- Cetoconazol no Pedido de Fornecedor 8
(9, 10); -- Cloridrato de Tramadol no Pedido de Fornecedor 9

-- Inserir dados na tabela Funcionário
INSERT INTO funcionario (nome_funcionario, cargo, salario) VALUES
('Carlos Mendes', 'Farmacêutico', 5000.00),
('Ana Souza', 'Atendente', 3000.00),
('Luiz Costa', 'Gerente', 7000.00),
('Juliana Martins', 'Auxiliar Administrativo', 3500.00),
('Ricardo Almeida', 'Farmacêutico', 4800.00),
('Patrícia Lima', 'Atendente', 3200.00),
('Eduardo Silva', 'Supervisor', 6000.00),
('Mariana Oliveira', 'Farmacêutico', 4900.00),
('Fernanda Santos', 'Auxiliar de Farmácia', 3300.00),
('Gustavo Pereira', 'Atendente', 3100.00);

-- Inserir dados na tabela Relacional entre Funcionário e Pedido de Cliente
INSERT INTO realiza_cliente (id_funcionario, id_pedido) VALUES
(1, 1), -- Carlos Mendes realizou o Pedido 1
(2, 2), -- Ana Souza realizou o Pedido 2
(3, 3), -- Luiz Costa realizou o Pedido 3
(4, 4), -- Juliana Martins realizou o Pedido 4

```

```

(5, 5), -- Ricardo Almeida realizou o Pedido 5
(6, 6), -- Patrícia Lima realizou o Pedido 6
(7, 7), -- Eduardo Silva realizou o Pedido 7
(8, 8), -- Mariana Oliveira realizou o Pedido 8
(9, 9), -- Fernanda Santos realizou o Pedido 9
(10, 10); -- Gustavo Pereira realizou o Pedido 10

-- Inserir dados na tabela Relacional entre Funcionário e Pedido de Fornecedor
INSERT INTO realiza_fornecedor (id_funcionario, id_pedido_fornecedor) VALUES
(1, 1), -- Carlos Mendes realizou o Pedido de Fornecedor 1
(2, 2), -- Ana Souza realizou o Pedido de Fornecedor 2
(3, 3), -- Luiz Costa realizou o Pedido de Fornecedor 3
(4, 4), -- Juliana Martins realizou o Pedido de Fornecedor 4
(5, 5), -- Ricardo Almeida realizou o Pedido de Fornecedor 5
(6, 6), -- Patrícia Lima realizou o Pedido de Fornecedor 6
(7, 7), -- Eduardo Silva realizou o Pedido de Fornecedor 7
(8, 8), -- Mariana Oliveira realizou o Pedido de Fornecedor 8
(9, 9), -- Fernanda Santos realizou o Pedido de Fornecedor 9
(10, 10); -- Gustavo Pereira realizou o Pedido de Fornecedor 10

-- Inserir dados na tabela Estoque
INSERT INTO estoque (id_remedio, data_entrega_stq, qtd_stq, data_vali_stq, medid
(1, '2024-09-01', 100, '2025-09-01', 10, 5, 20, 100), -- Paracetamol
(2, '2024-09-02', 200, '2025-09-02', 20, 10, 30, 200), -- Amoxicilina
(3, '2024-09-03', 150, '2025-09-03', 15, 8, 25, 150), -- Omeprazol
(4, '2024-09-04', 180, '2025-09-04', 18, 9, 28, 180), -- Ibuprofeno
(5, '2024-09-05', 160, '2025-09-05', 16, 7, 24, 160), -- Dipirona
(6, '2024-09-06', 140, '2025-09-06', 14, 6, 22, 140), -- Azitromicina
(7, '2024-09-07', 170, '2025-09-07', 17, 8, 26, 170), -- Lorazepam
(8, '2024-09-08', 190, '2025-09-08', 19, 10, 29, 190), -- Metformina
(9, '2024-09-09', 120, '2025-09-09', 12, 5, 21, 120), -- Cetoconazol
(10, '2024-09-10', 130, '2025-09-10', 13, 6, 23, 130); -- Cloridrato de Tramadol

-- Inserir dados na tabela Histórico de Compra
INSERT INTO historico_compra (receitas, ult_compra, id_cliente) VALUES
('Receita de Paracetamol', '2024-08-20', 1), -- João Silva
('Receita de Amoxicilina', '2024-08-22', 2), -- Maria Oliveira
('Receita de Omeprazol', '2024-08-24', 3), -- Pedro Santos
('Receita de Ibuprofeno', '2024-08-26', 4), -- Ana Lima
('Receita de Dipirona', '2024-08-28', 5), -- Lucia Costa
('Receita de Azitromicina', '2024-08-30', 6), -- Carlos Almeida

```

```

('Receita de Lorazepam', '2024-09-01', 7), -- Fernanda Silva
('Receita de Metformina', '2024-09-03', 8), -- Roberto Oliveira
('Receita de Cetoconazol', '2024-09-05', 9), -- Juliana Pereira
('Receita de Cloridrato de Tramadol', '2024-09-07', 10); -- Ricardo Martins

```

8.3 Ver todas tabelas com os dados informados;

```

-- Selecionar dados da tabela Remédio
SELECT * FROM remedio;

-- Selecionar dados da tabela Genéricos
SELECT * FROM generico;

-- Selecionar dados da tabela Tarjas
SELECT * FROM tarja;

-- Selecionar dados da tabela Marcas
SELECT * FROM marca;

-- Selecionar dados da tabela Cliente
SELECT * FROM cliente;

-- Selecionar dados da tabela Telefone
SELECT * FROM telefone;

-- Selecionar dados da tabela Fornecedor
SELECT * FROM fornecedor;

-- Selecionar dados da tabela Pedidos de Clientes
SELECT * FROM pedido_cliente;

-- Selecionar dados da tabela Pedidos de Fornecedores
SELECT * FROM pedido_fornecedor;

-- Selecionar dados da tabela pedido_cliente_remedio (relacional entre Pedido Cl
SELECT * FROM pedido_cliente_remedio;

-- Selecionar dados da tabela pedido_fornecedor_remedio (relacional entre Pedido

```



```
SELECT * FROM pedido_fornecedor_remedio;

-- Selecionar dados da tabela Funcionário
SELECT * FROM funcionario;

-- Selecionar dados da tabela realiza_cliente (relacional entre Funcionário e Pe
SELECT * FROM realiza_cliente;

-- Selecionar dados da tabela realiza_fornecedor (relacional entre Funcionário e
SELECT * FROM realiza_fornecedor;

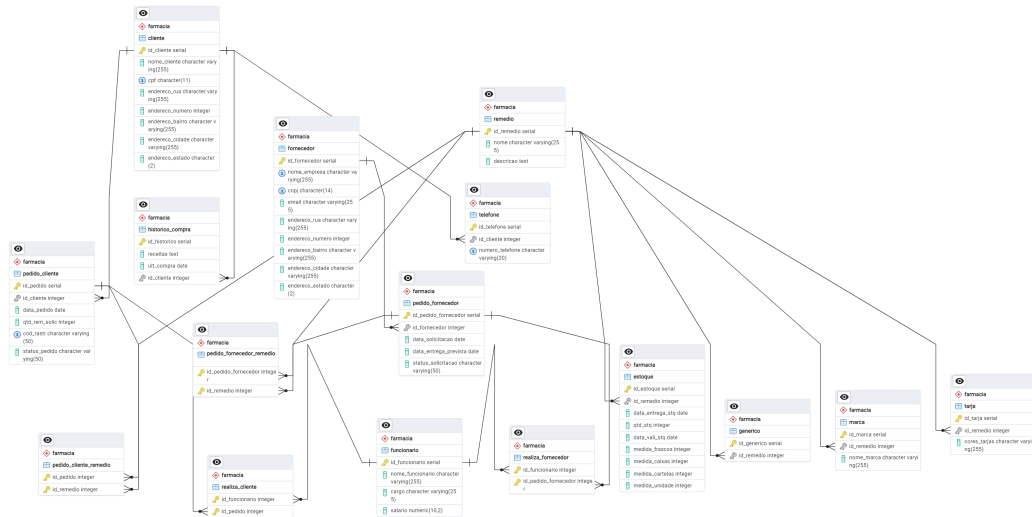
-- Selecionar dados da tabela Estoque
SELECT * FROM estoque;

-- Selecionar dados da tabela possui (relacional entre Estoque e Remédio)
SELECT * FROM possui;

-- Selecionar dados da tabela Histórico de Compra
SELECT * FROM historico_compra;

-- Selecionar dados da tabela faz_parte (relacional entre Histórico de Compra e
SELECT * FROM faz_parte;
```

8.4 Logo abaixo modelo relacional do projeto de gerenciamento de remédios



9 O banco de dados é estruturado com as seguintes tabelas principais:

9.1 Tabela remedio

- **Descrição:** Armazena informações sobre os remédios disponíveis.
- **Chave Primária:** id_remedio
- **Atributos:**
 - id_remedio: Identificador único para cada remédio.
 - nome: Nome do remédio.
 - descricao: Descrição do remédio.

9.2 Tabela genericos

- **Descrição:** Armazena informações sobre remédios genéricos.
- **Chave Primária:** id_generico
- **Herança:** Esta tabela não herda diretamente da tabela remedio, mas está relacionada a ela através da chave estrangeira id_remedio.

- **Atributos:**
 - `id_generico`: Identificador único para cada remédio genérico.
 - `id_remedio`: Identificador do remédio ao qual o genérico está associado.

9.3 Tabela `tarjas`

- **Descrição:** Armazena informações sobre as tarjas dos remédios.
- **Chave Primária:** `id_tarjas`
- **Atributos:**
 - `id_tarjas`: Identificador único para cada tarja.
 - `id_remedio`: Identificador do remédio associado à tarja.
 - `cores_tarjas`: Cores das tarjas do remédio.
- **Relacionamento:** Relacionada à tabela `remedio` através de `id_remedio`.

9.4 Tabela `de_marcas`

- **Descrição:** Armazena informações sobre marcas dos remédios.
- **Chave Primária:** `id_marcas`
- **Atributos:**
 - `id_marcas`: Identificador único para cada marca.
 - `id_remedio`: Identificador do remédio associado à marca.
- **Relacionamento:** Relacionada à tabela `remedio` através de `id_remedio`.

9.5 Tabela `pedido`

- **Descrição:** Armazena informações sobre os pedidos de remédios.
- **Chave Primária:** `id_pedido`
- **Atributos:**
 - `id_pedido`: Identificador único para cada pedido.
 - `cod_rastr`: Código de rastreamento do pedido.

- status_pedido: Status atual do pedido (EM PRODUÇÃO, A CAMINHO, ENTREGUE).
- data_pedido: Data em que o pedido foi realizado.
- qtd_rem_solic: Quantidade de remédios solicitados.

9.6 Tabela esta_incluido

- **Descrição:** Relaciona remédios a pedidos.
- **Chave Primária:** (id_remedio, id_pedido)
- **Atributos:**
 - id_remedio: Identificador do remédio.
 - id_pedido: Identificador do pedido.
- **Relacionamento:** Relaciona a tabela `remedio` com a tabela `pedido`.

9.7 Tabela fornecedor

- **Descrição:** Armazena informações sobre fornecedores.
- **Chave Primária:** id_fornecedor
- **Atributos:**
 - id_fornecedor: Identificador único para cada fornecedor.
 - nome_empresa: Nome da empresa fornecedora.
 - cnpj: CNPJ da empresa fornecedora.
 - email: E-mail da empresa fornecedora.
 - endereco_rua: Rua do endereço do fornecedor.
 - endereco_numero: Número do endereço do fornecedor.
 - endereco_bairro: Bairro do endereço do fornecedor.
 - endereco_cidade: Cidade do endereço do fornecedor.
 - endereco_estado: Estado do endereço do fornecedor.

9.8 Tabela solicitacao

- **Descrição:** Relaciona pedidos a fornecedores.
- **Chave Primária:** (id_pedido, id_fornecedor)
- **Atributos:**
 - id_pedido: Identificador do pedido.
 - id_fornecedor: Identificador do fornecedor.
- **Relacionamento:** Relaciona a tabela pedido com a tabela fornecedor.

9.9 Tabela funcionario

- **Descrição:** Armazena informações sobre funcionários.
- **Chave Primária:** id_funcionario
- **Atributos:**
 - id_funcionario: Identificador único para cada funcionário.
 - nome: Nome do funcionário.
 - cargo: Cargo do funcionário.
 - salario: Salário do funcionário.

9.10 Tabela realiza

- **Descrição:** Relaciona funcionários a pedidos.
- **Chave Primária:** (id_funcionario, id_pedido)
- **Atributos:**
 - id_funcionario: Identificador do funcionário.
 - id_pedido: Identificador do pedido.
- **Relacionamento:** Relaciona a tabela funcionario com a tabela pedido.

9.11 Tabela estoque

- **Descrição:** Armazena informações sobre o estoque de remédios.
- **Chave Primária:** `id_estoque`
- **Atributos:**
 - `id_estoque`: Identificador único para cada registro de estoque.
 - `data_entrega_stq`: Data de entrega do estoque.
 - `qtd_stq`: Quantidade disponível no estoque.
 - `data_vali_stq`: Data de validade do estoque.
 - `medida_frascos`: Quantidade de frascos.
 - `medida_caixas`: Quantidade de caixas.
 - `medida_cartelas`: Quantidade de cartelas.
 - `medida_unidade`: Quantidade de unidades.

9.12 Tabela possui

- **Descrição:** Relaciona o estoque aos remédios.
- **Chave Primária:** (`id_estoque`, `id_remedio`)
- **Atributos:**
 - `id_estoque`: Identificador do estoque.
 - `id_remedio`: Identificador do remédio.
- **Relacionamento:** Relaciona a tabela `estoque` com a tabela `remedio`.

9.13 Tabela cliente

- **Descrição:** Armazena informações sobre os clientes.
- **Chave Primária:** `id_cliente`
- **Atributos:**
 - `id_cliente`: Identificador único para cada cliente.
 - `nome_cliente`: Nome do cliente.
 - `cpf`: CPF do cliente.

- `endereco_rua`: Rua do endereço do cliente.
- `endereco_numero`: Número do endereço do cliente.
- `endereco_bairro`: Bairro do endereço do cliente.
- `endereco_cidade`: Cidade do endereço do cliente.

9.14 Tabela `telefone`

- **Descrição:** Armazena números de telefone dos clientes.
- **Chave Primária:** `id_telefone`
- **Atributos:**
 - `id_telefone`: Identificador único para cada número de telefone.
 - `id_cliente`: Identificador do cliente associado ao telefone.
 - `numero_telefone`: Número do telefone.
- **Relacionamento:** Relacionada à tabela `cliente` através de `id_cliente`.

9.15 Tabela `historico_compra`

- **Descrição:** Armazena informações sobre o histórico de compras dos clientes.
- **Chave Primária:** `id_historico`
- **Atributos:**
 - `id_historico`: Identificador único para cada histórico de compra.
 - `receitas`: Informações sobre receitas.
 - `ult_compra`: Informações sobre a última compra.
 - `id_cliente`: Identificador do cliente associado ao histórico.
- **Relacionamento:** Relacionada à tabela `cliente` através de `id_cliente`.

9.16 Tabela `faz_parte`

- **Descrição:** Relaciona o histórico de compras com os remédios.
- **Chave Primária:** (`id_historico`, `id_remedio`)
- **Atributos:**

- `id_historico`: Identificador do histórico de compras.
- `id_remedio`: Identificador do remédio.
- **Relacionamento:** Relaciona a tabela `historico_compra` com a tabela `remedio`.

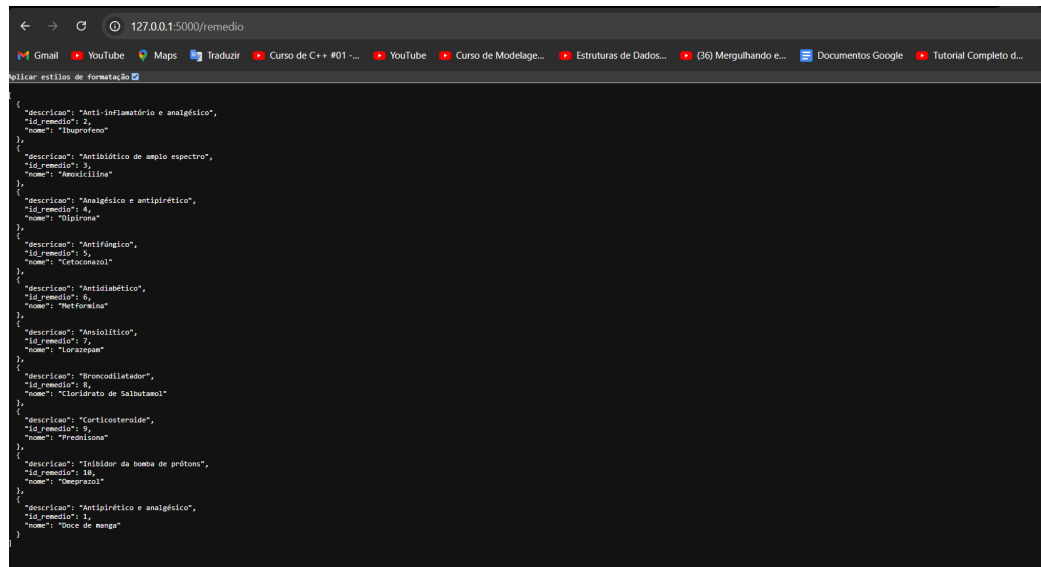
9.17 Tabela `tem`

- **Descrição:** Relaciona o histórico de compras com os clientes.
- **Chave Primária:** (`id_historico`, `id_cliente`)
- **Atributos:**
 - `id_historico`: Identificador do histórico de compras.
 - `id_cliente`: Identificador do cliente.
- **Relacionamento:** Relaciona a tabela `historico_compra` com a tabela `cliente`.

10 Como Funciona um CRUD

CRUD é um acrônimo para as operações básicas de gerenciamento de dados: Create (Criar), Read (Ler), Update (Atualizar) e Delete (Excluir). Aqui estão exemplos de como cada operação é realizada utilizando Flask e PostgreSQL.

10.1 Imagem original sem nenhuma implementação (Tabela remédios)



10.2 Create (Criar)

Para adicionar um novo registro à tabela, você usa a operação POST. A seguir está um exemplo de código para adicionar um novo remédio:

Usando o POSTMAN para gerenciar esses dados;

Com o campo "POST" `http://localhost:5000/remedio/11`

no body;

```
{
  "descricao": "Antipirético e analgésico",
  "id_remedio": 11,
  "nome": "Manga doce"
}
```

HTTP <http://localhost:5000/remedio/11> Save Share

POST <http://localhost:5000/remedio/11> Send

Params Auth Headers (8) **Body** Scripts Tests Settings Cookies Beautify

raw JSON

```

1 {
2   .... "descricao": "Antipirético e analgésico",
3   .... "id_remédio": 11,
4   .... "nome": "Manga doce"
5 }

```

Gerenciamento_Remedios/postgres@PostgreSQL 16

Query Query History

```

1 select * from farmacia.remédio

```

Data Output Messages Notifications

	id_remédio [PK] integer	nome character varying (255)	descricao character varying (255)
1	2	Ibuprofeno	Anti-inflamatório e analgésico
2	3	Amoxicilina	Antibiótico de amplo espectro
3	4	Dipirona	Analgésico e antipirético
4	5	Cetoconazol	Antifúngico
5	6	Metformina	Antidiabético
6	7	Lorazepam	Ansiolítico
7	8	Cloridrato de Salbutamol	Broncodilatador
8	9	Prednisona	Corticosteroide
9	10	Omeprazol	Inibidor da bomba de prótons
10	1	Paracetamol	Antipirético e analgésico
11	11	Manga doce	Antipirético e analgésico

Total rows: 11 of 11 Query complete 00:00:00.084 Ln 1, Col 31

```

@app.route('/remedio', methods=['POST'])
def adicionar_remedio():
    conn = get_db_connection()
    if conn is None:
        return jsonify({"error": "Não foi possível conectar ao banco de dados"})

    data = request.json
    nome = data['nome']

```

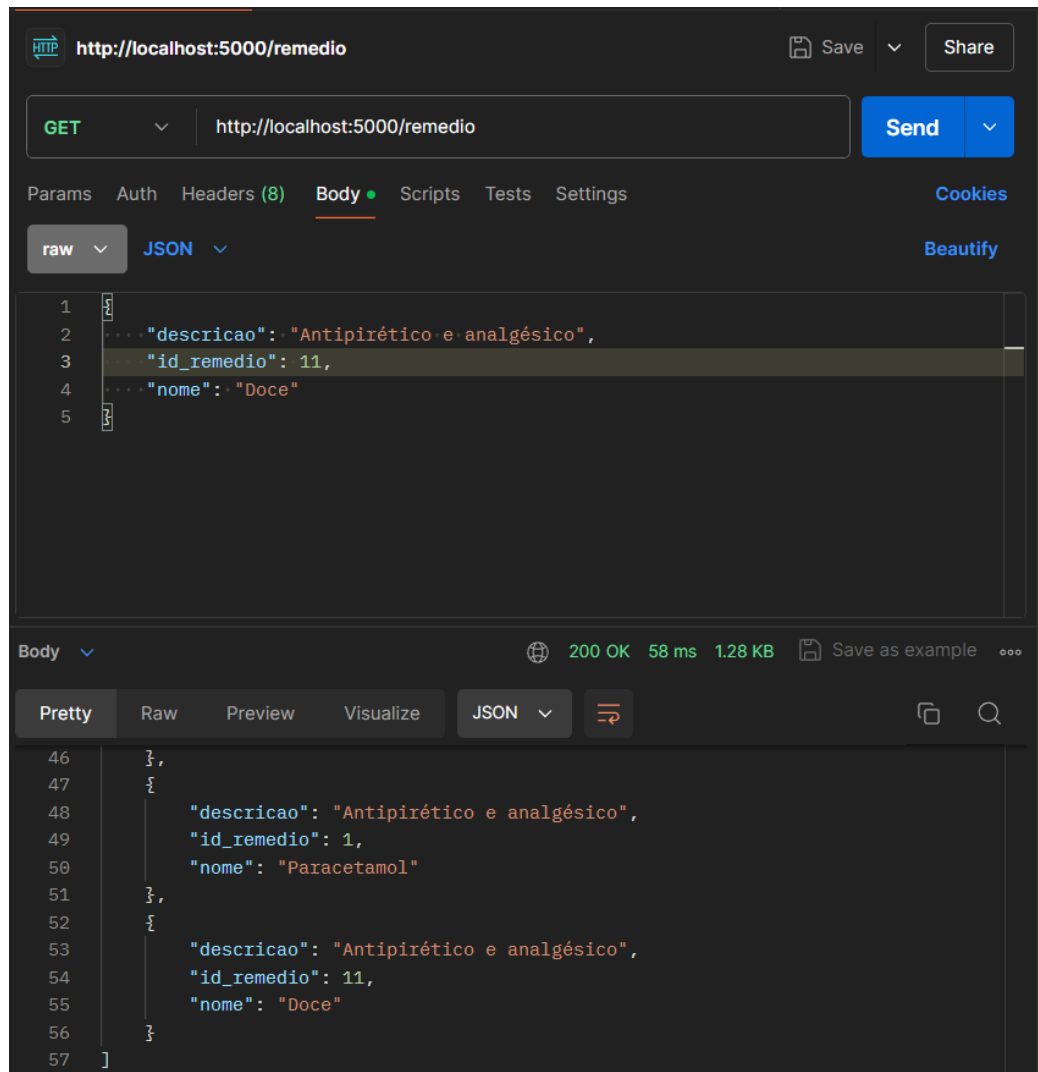
```

descricao = data['descricao']

try:
    with conn.cursor() as cur:
        cur.execute(
            "INSERT INTO farmacia.remedio (nome, descricao) VALUES (%s, %s) "
            "(nome, descricao)
        )
        id_remedio = cur.fetchone()[0]
        conn.commit()
        return jsonify({"id_remedio": id_remedio}), 201
except Exception as e:
    return jsonify({"error": str(e)}), 500
finally:
    conn.close()

```

10.3 Após usar o CREATE



10.4 Read (Ler)

Para obter dados existentes, você usa a operação GET. Aqui está um exemplo de código para listar todos os remédios:

Usando o POSTMAN para gerenciar esses dados;
Com o campo "GET" `http://localhost:5000/remedio/11`

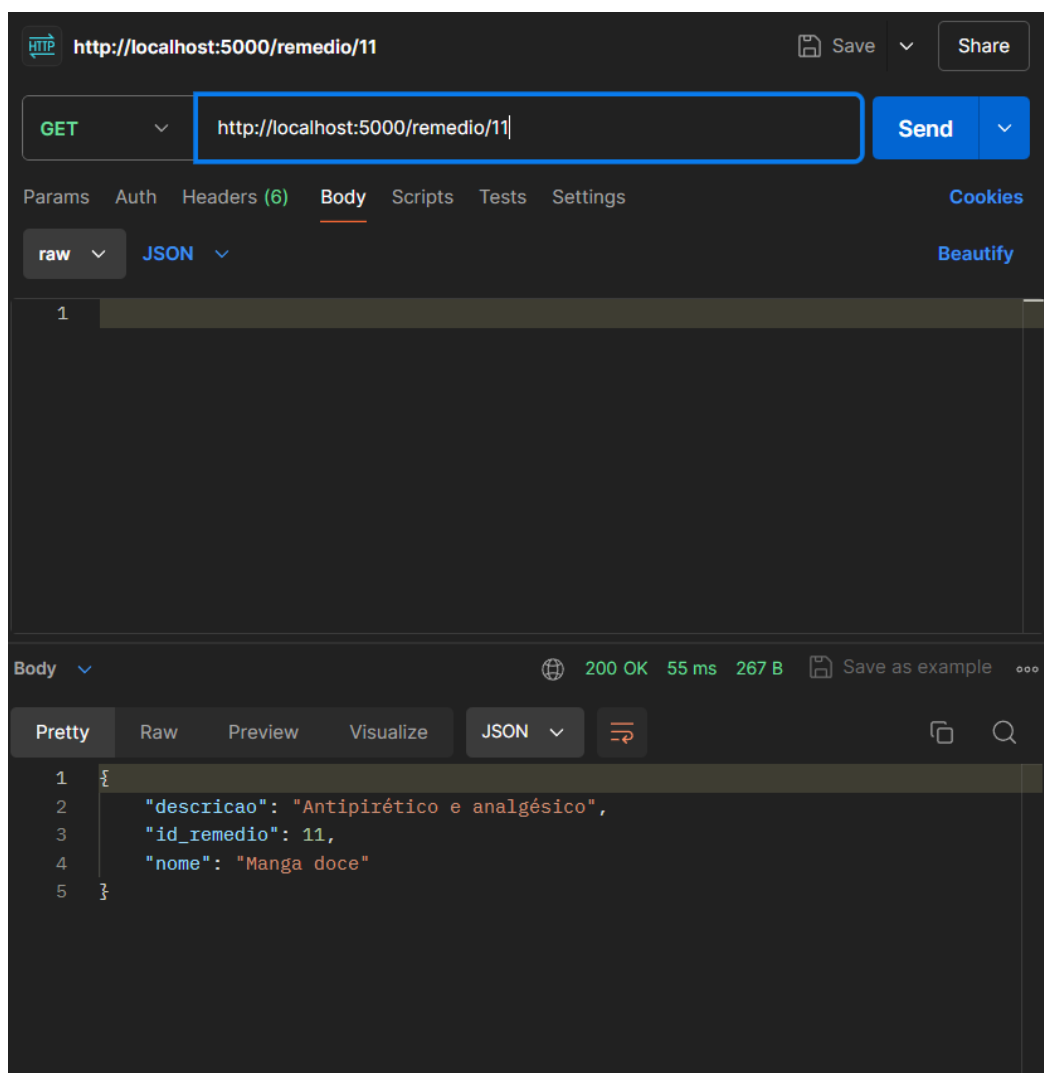
```
@app.route('/remedio', methods=['GET'])
def listar_remedios():
```

```

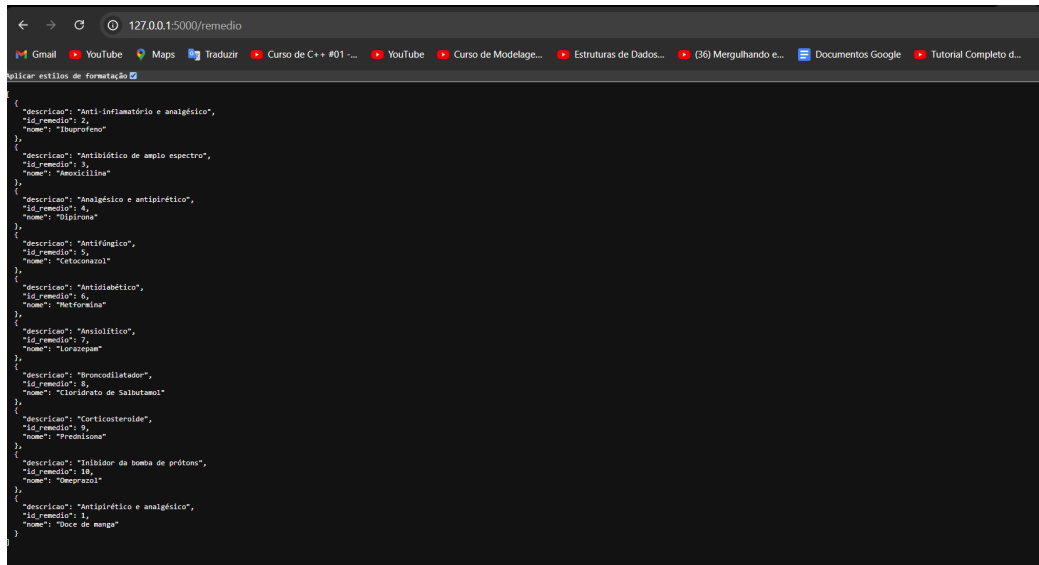
conn = get_db_connection()
if conn is None:
    return jsonify({"error": "Não foi possível conectar ao banco de dados"})

try:
    with conn.cursor() as cur:
        cur.execute("SELECT * FROM farmacia.remedio;")
        dados = cur.fetchall()
        colunas = [desc[0] for desc in cur.description]
        resultados = [dict(zip(colunas, linha)) for linha in dados]
        return jsonify(resultados)
except Exception as e:
    return jsonify({"error": str(e)}), 500
finally:
    conn.close()

```

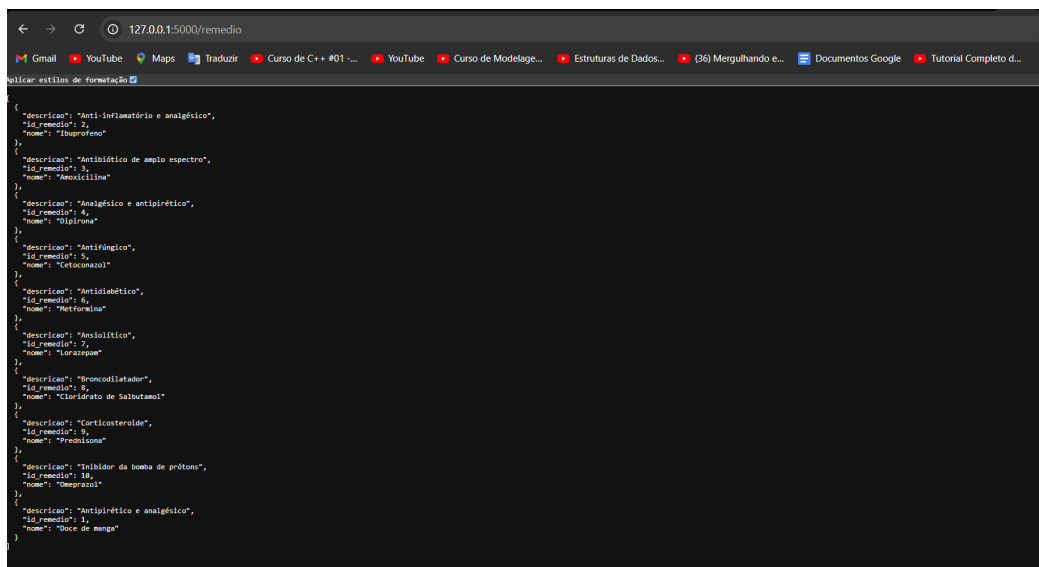


10.5 Resultando após usar o READ



```
{
  "descricao": "Anti-inflamatório e analgésico",
  "id_remedio": 2,
  "nome": "Ibuprofeno"
},
{
  "descricao": "Antibiótico de amplo espectro",
  "id_remedio": 3,
  "nome": "Amoxicilina"
},
{
  "descricao": "Analgésico e antipirético",
  "id_remedio": 4,
  "nome": "Dipirona"
},
{
  "descricao": "Antifúngico",
  "id_remedio": 5,
  "nome": "Cetoconazol"
},
{
  "descricao": "Antidiabético",
  "id_remedio": 6,
  "nome": "Metformina"
},
{
  "descricao": "Ansiolítico",
  "id_remedio": 7,
  "nome": "Lorazepam"
},
{
  "descricao": "Broncodilatador",
  "id_remedio": 8,
  "nome": "Cloridrato de Salbutamol"
},
{
  "descricao": "Corticosteroide",
  "id_remedio": 9,
  "nome": "Prednisona"
},
{
  "descricao": "Inibidor da bomba de prótons",
  "id_remedio": 10,
  "nome": "Omeprazol"
},
{
  "descricao": "Antipirético e analgésico",
  "id_remedio": 1,
  "nome": "Doce de manga"
}
}
```

10.6 Antes de utilizar o UPDATE



```
{
  "descricao": "Anti-inflamatório e analgésico",
  "id_remedio": 2,
  "nome": "Ibuprofeno"
},
{
  "descricao": "Antibiótico de amplo espectro",
  "id_remedio": 3,
  "nome": "Amoxicilina"
},
{
  "descricao": "Analgésico e antipirético",
  "id_remedio": 4,
  "nome": "Dipirona"
},
{
  "descricao": "Antifúngico",
  "id_remedio": 5,
  "nome": "Cetoconazol"
},
{
  "descricao": "Antidiabético",
  "id_remedio": 6,
  "nome": "Metformina"
},
{
  "descricao": "Ansiolítico",
  "id_remedio": 7,
  "nome": "Lorazepam"
},
{
  "descricao": "Broncodilatador",
  "id_remedio": 8,
  "nome": "Cloridrato de Salbutamol"
},
{
  "descricao": "Corticosteroide",
  "id_remedio": 9,
  "nome": "Prednisona"
},
{
  "descricao": "Inibidor da bomba de prótons",
  "id_remedio": 10,
  "nome": "Omeprazol"
},
{
  "descricao": "Antipirético e analgésico",
  "id_remedio": 1,
  "nome": "Doce de manga"
}
}
```

10.7 Update (Atualizar)

Para atualizar um registro existente, você usa a operação PUT. Veja um exemplo para atualizar um remédio:

Usando o POSTMAN para gerenciar esses dados;

Com o campo "PUT" `http://localhost:5000/remedio/11`
no body;

```
{
  "descricao": "Antipirético e analgésico",
  "id_remedio": 11,
  "nome": "Doce de manga"
}
```

Gerenciamento Remedios/postgres@PostgreSQL 16

Query Query History

```
1 select * from farmacia.remedio
```

Data Output Messages Notifications

	id_remedio [PK] integer	nome character varying (255)	descricao character varying (255)
1	2	Ibuprofeno	Anti-inflamatório e analgésico
2	3	Amoxicilina	Antibiótico de amplo espectro
3	4	Dipirona	Analgésico e antipirético
4	5	Cetoconazol	Antifúngico
5	6	Metformina	Antidiabético
6	7	Lorazepam	Ansiolítico
7	8	Cloridrato de Salbutamol	Broncodilatador
8	9	Prednisona	Corticosteroide
9	10	Omeprazol	Inibidor da bomba de prótons
10	1	Paracetamol	Antipirético e analgésico
11	11	Manga doce	Antipirético e analgésico

Total rows: 11 of 11 Query complete 00:00:00.084 Ln 1, Col 31

HTTP <http://localhost:5000/remedio/11> Save Share

PUT <http://localhost:5000/remedio/11> Send

Params Auth Headers (8) **Body** Scripts Tests Settings Cookies

raw JSON Beautify

```
1 {
2   ... "descricao": "Antipirético e analgésico",
3   ... "id_remedio": 11,
4   ... "nome": "Doce de manga"
5 }
```

Body 200 OK 55 ms 217 B Save as example

Pretty Raw Preview Visualize JSON Copy Search

```
1 {
2   "message": "Operação bem-sucedida!"
3 }
```

Query			
Query History			
1 select * from farmacia.remedio			
Data Output			
Messages			
Notifications			
	id_remedio [PK] integer	nome character varying (255)	descricao character varying (255)
1	2	Ibuprofeno	Anti-inflamatório e analgésico
2	3	Amoxicilina	Antibiótico de amplo espectro
3	4	Dipirona	Analgésico e antipirético
4	5	Cetoconazol	Antifúngico
5	6	Metformina	Antidiabético
6	7	Lorazepam	Ansiolítico
7	8	Cloridrato de Salbutamol	Broncodilatador
8	9	Prednisona	Corticosteroide
9	10	Omeprazol	Inibidor da bomba de prótons
10	1	Paracetamol	Antipirético e analgésico
11	11	Doce de manga	Antipirético e analgésico

```

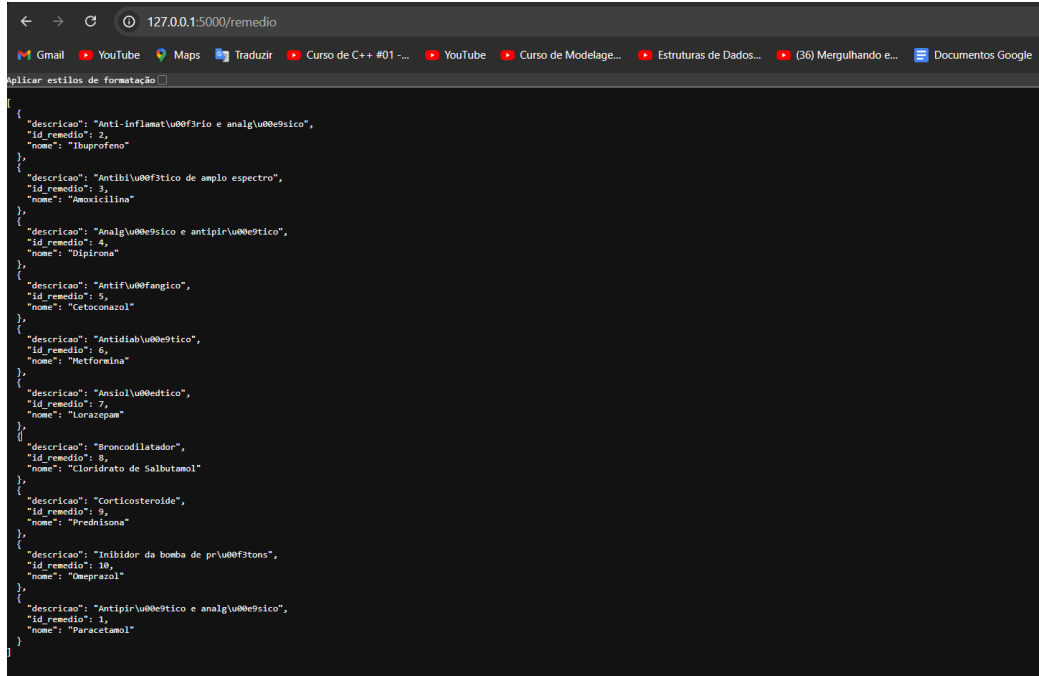
@app.route('/remedio/<int:id>', methods=['PUT'])
def atualizar_remedio(id):
    conn = get_db_connection()
    if conn is None:
        return jsonify({"error": "Não foi possível conectar ao banco de dados"})

    data = request.json
    nome = data.get('nome')
    descricao = data.get('descricao')

    try:
        with conn.cursor() as cur:
            if nome:
                cur.execute("UPDATE farmacia.remedio SET nome = %s WHERE id_remed= %s")
            if descricao:
                cur.execute("UPDATE farmacia.remedio SET descricao = %s WHERE id_remed= %s")
            conn.commit()
            return jsonify({"status": "Remédio atualizado com sucesso"})
    except Exception as e:
        return jsonify({"error": str(e)}), 500
    finally:
        conn.close()

```

10.8 Após usar o UPDATE



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/remedio'. The browser's address bar also shows several tabs: Gmail, YouTube, Maps, Traduzir, Curso de C++ #01 ~..., YouTube, Curso de Modelage..., Estruturas de Dados..., (36) Mergulhando e..., and Documentos Google. The main content area of the browser displays a JSON array of medicine records. The records are as follows:

```
{
  "descricao": "Anti-inflamat\u00f3rio e analg\u00e9sico",
  "id_remedio": 2,
  "nome": "Ibuprofeno"
},
{
  "descricao": "Antib\u00edotico de amplo espectro",
  "id_remedio": 3,
  "nome": "Amoxicilina"
},
{
  "descricao": "Analg\u00e9sico e antipir\u00e9tico",
  "id_remedio": 4,
  "nome": "Dipirona"
},
{
  "descricao": "Antif\u00fongico",
  "id_remedio": 5,
  "nome": "Cetoconazol"
},
{
  "descricao": "Antidiab\u00e9tico",
  "id_remedio": 6,
  "nome": "Metformina"
},
{
  "descricao": "Ansio\u00e9tico",
  "id_remedio": 7,
  "nome": "Lorazepam"
},
{
  "descricao": "Broncodilatador",
  "id_remedio": 8,
  "nome": "Cloridrato de Salbutamol"
},
{
  "descricao": "Corticoester\u00f3ide",
  "id_remedio": 9,
  "nome": "Prednisona"
},
{
  "descricao": "Inibidor da bomba de pr\u00f3tons",
  "id_remedio": 10,
  "nome": "Omeprazol"
},
{
  "descricao": "Antipir\u00e9tico e analg\u00e9sico",
  "id_remedio": 1,
  "nome": "Paracetamol"
}
```

10.9 Delete (Excluir)

Para excluir um registro, voc\u00ea usa a opera\u00e7\u00e3o DELETE. Aqui est\u00e1 um exemplo para excluir um rem\u00e9dio:

```
@app.route('/remedio/<int:id>', methods=['DELETE'])
def excluir_remedio(id):
    conn = get_db_connection()
    if conn is None:
        return jsonify({"error": "N\u00e3o foi poss\u00edvel conectar ao banco de dados"})

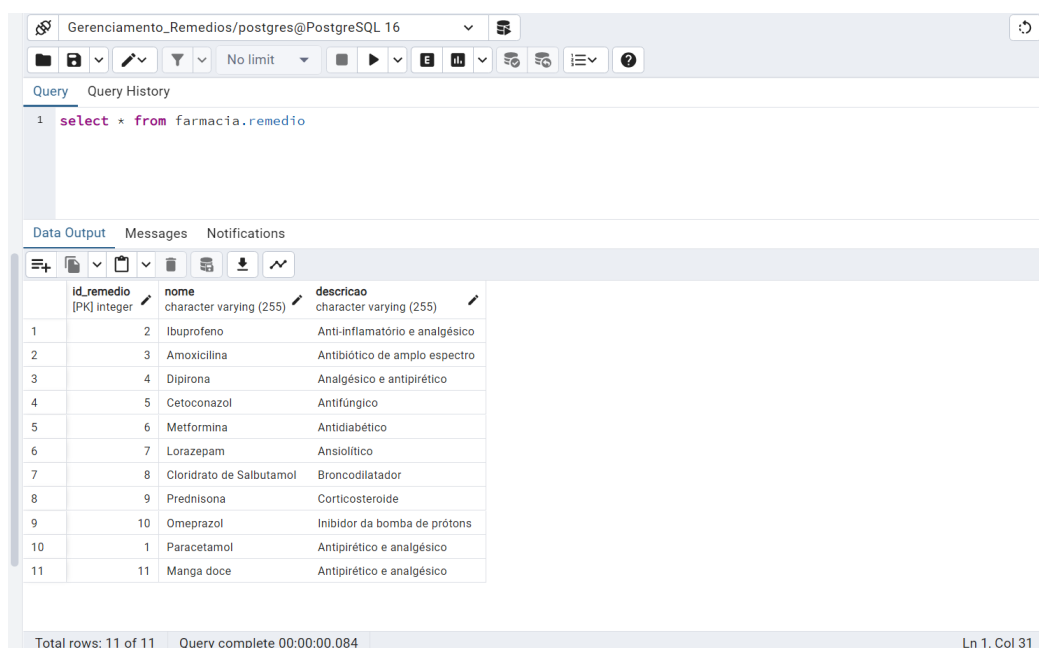
    try:
        with conn.cursor() as cur:
            cur.execute("DELETE FROM farmacia.remedio WHERE id_remedio = %s;", (id,))
            conn.commit()
            return jsonify({"status": "Rem\u00e9dio exclu\u00eddo com sucesso"})
    except Exception as e:
        return jsonify({"error": str(e)}), 500
    finally:
        conn.close()
```

11 Conclusão

O sistema de gerenciamento de remédios foi implementado com sucesso, com uma API robusta e uma estrutura de banco de dados eficiente. As operações CRUD permitem um gerenciamento completo dos dados de remédios, fornecedores, e outros aspectos relacionados.

Usando o POSTMAN para gerenciar esses dados;

Com o campo "DELETE" `http://localhost:5000/remedio/11`



The screenshot shows a web interface for PostgreSQL. At the top, the connection is 'Gerenciamento_Remédios/postgres@PostgreSQL 16'. Below the connection bar, there's a 'Query' tab with the SQL query: `select * from farmacia.remedio`. The 'Data Output' tab is active, displaying a table with 11 rows. The table has three columns: `id_remédio` (integer), `nome` (character varying (255)), and `descricao` (character varying (255)). The data includes various medications like Ibuprofeno, Amoxicilina, Dipirona, Cetoconazol, Metformina, Lorazepam, Cloridrato de Salbutamol, Prednisona, Omeprazol, Paracetamol, and Manga doce. At the bottom, it shows 'Total rows: 11 of 11' and 'Query complete 00:00:00.084'.

	<code>id_remédio</code> [PK] integer	<code>nome</code> character varying (255)	<code>descricao</code> character varying (255)
1	2	Ibuprofeno	Anti-inflamatório e analgésico
2	3	Amoxicilina	Antibiótico de amplo espectro
3	4	Dipirona	Analgésico e antipirético
4	5	Cetoconazol	Antifúngico
5	6	Metformina	Antidiabético
6	7	Lorazepam	Ansiolítico
7	8	Cloridrato de Salbutamol	Broncodilatador
8	9	Prednisona	Corticosteroide
9	10	Omeprazol	Inibidor da bomba de prótons
10	1	Paracetamol	Antipirético e analgésico
11	11	Manga doce	Antipirético e analgésico

Total rows: 11 of 11 Query complete 00:00:00.084 Ln 1, Col 31

HTTP <http://localhost:5000/remedio/11> Save Share

DELETE ▼ <http://localhost:5000/remedio/11> Send ▼

Params Auth Headers (8) **Body** • Scripts Tests Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2   ... "descricao": "Antipirético e analgésico",
3   ... "id_remedio": 11,
4   ... "nome": "Doce de manga"
5 }
```

Body ▼ 200 OK 76 ms 217 B Save as example ...

Pretty Raw Preview Visualize JSON ▼ ≡ 📄 🔍

```
1 {
2   "message": "Operação bem-sucedida!"
3 }
```

Gerenciamento_Medios/postgres@PostgreSQL 16

Query Query History

```
1 select * from farmacia.remedio;
```

Data Output Messages Notifications

	id_remedio [PK] integer	nome character varying (255)	descricao character varying (255)
1	2	Ibuprofeno	Anti-inflamatório e analgésico
2	3	Amoxicilina	Antibiótico de amplo espectro
3	4	Dipirona	Analgésico e antipirético
4	5	Cetoconazol	Antifúngico
5	6	Metformina	Antidiabético
6	7	Lorazepam	Ansiolítico
7	8	Cloridrato de Salbutamol	Broncodilatador
8	9	Prednisona	Corticosteroide
9	10	Omeprazol	Inibidor da bomba de prótons
10	1	Paracetamol	Antipirético e analgésico

Tarefa	Quem Realizou
Implementação de Atributos Multivalorados	Kaio
Implementação da Hierarquia de Herança	Kaio
Prototipagem do SQL do banco de dados	Kaio
Modelo Relacional	Tailan de Souza
Modelagem e Implementação do Banco de Dados Final	Tailan de Souza
Popular tabelas	Tailan de Souza
Testar Aplicação	Tailan de Souza
Desenvolvimento e Configuração do Projeto	Tailan de Souza
Criação de Documentos e Relatórios	Tailan de Souza
Trabalho com GitHub	Tailan de Souza
CRUD da Aplicação	Tailan de Souza
Revisão Geral	Tailan de Souza

12 Referências

Referências

- [1] Elmasri, R., & Navathe, S. B. (2024). *Fundamentals of Database Systems*. Editora ABC.

13 Link do Repositório

https://github.com/Naliat/trabalho_fbd

Relatório de Projeto: Aplicação em Python

18 de setembro de 2024

14 Atualizações em relação a ultima entrega

14.1 Sugestões Lívia Almada

1 - id do remédio na tabela estoque
2 -excluir tabel possui (vinculação entre estoque e remédio). Cada linha da tabela estoque é relativa a um único remédio.
3 - Incluir id do cliente no Historico de compra e excluir tabela tem que vincula historico de compra ao cliente
4 - Explicar estratégia para atributos multivalorados e herança

14.2 Mudanças realizadas por Tailan e por Kaio

formulação do sql
foi feita a divisão da tabela pedido em "pedidoCliente" e "pedidoFornecedor"
criação da tabela pedidoCliente e pedidoFornecedor;
relacionamento da tabela remédio com pedidoFornecedor e com pedidoCliente
relacionamento da tabela funcionario com pedidoCliente e com pedidoFornecedor
relacionamento da funcionario com histórico de compra
retirado relacionamenro da tabela histórico de compra e cliente
retirado do relacionamento da tabela estoque com remédio
reirada da tabela outras categorias
inclusão do id_remedio em estoque

15 CRUD da Tabela Remédio

15.1 Comando para instalar as dependências

```
pip install pandas panel sqlalchemy psycpg2-binary
```

15.2 Importações

```
import pandas as pd  
import panel as pn  
from sqlalchemy import create_engine, text  
from sqlalchemy.exc import SQLAlchemyError  
import psycpg2
```

15.3 Configuração com o banco de dados(Mudar os parâmetros conforme o seu BD)

```
db_config = {
    'user': 'postgres',
    'password': '1412',
    'host': 'localhost',
    'port': '5432',
    'database': 'Gerenciamento_Remedios'
}
```

```
engine = create_engine(f"postgresql+psycopg2://{db_config['user']}:{db_config['password']}@{db_config['host']}:{db_config['port']}/{db_config['database']}")
con = engine.connect()
```

15.4 CRUD da aplicação

```
def read_remedios(id_remedio=None, nome=None):
    try:
        if id_remedio is not None:
            query = text("SELECT * FROM farmacia.remedio WHERE id_remedio = :id_remedio")
            df = pd.read_sql(query, con, params={"id_remedio": id_remedio})
        elif nome is not None:
            query = text("SELECT * FROM farmacia.remedio WHERE nome ILIKE :nome")
            df = pd.read_sql(query, con, params={"nome": f"%{nome}%"})
        else:
            query = "SELECT * FROM farmacia.remedio"
            df = pd.read_sql(query, con)
        return df
    except SQLAlchemyError as e:
        return str(e)

def create_remedio(nome, descricao):
    try:
        query = text("INSERT INTO farmacia.remedio (nome, descricao) VALUES (:nome, :descricao)")
        con.execute(query, {"nome": nome, "descricao": descricao})
        con.commit()
        return "Remédio adicionado com sucesso!"
    except SQLAlchemyError as e:
        return str(e)
```

```

def update_remedio(id_remedio, nome, descricao):
    try:
        query = text("UPDATE farmacia.remedio SET nome = :nome, descricao = :descricao")
        con.execute(query, {"id_remedio": id_remedio, "nome": nome, "descricao": descricao})
        con.commit()
        return "Remédio atualizado com sucesso!"
    except SQLAlchemyError as e:
        return str(e)

def delete_remedio(id_remedio):
    try:
        query = text("DELETE FROM farmacia.remedio WHERE id_remedio = :id_remedio")
        con.execute(query, {"id_remedio": id_remedio})
        con.commit()
        return "Remédio excluído com sucesso!"
    except SQLAlchemyError as e:
        return str(e)

```

15.5 Interface da aplicação

```

pn.extension()

def update_table(id_remedio=None, nome=None):
    df = read_remedios(id_remedio, nome)
    table.value = df

def on_filter_button_click(event):
    id_filter = id_filter_input.value if id_filter_input.value else None
    nome_filter = nome_filter_input.value if nome_filter_input.value else None
    update_table(id_remedio=id_filter, nome=nome_filter)
    output_pane.object = "Tabela atualizada com base no filtro."

id_filter_input = pn.widgets.IntInput(name='Filtrar por ID', width=150)
nome_filter_input = pn.widgets.TextInput(name='Filtrar por Nome', placeholder='Nome')
filter_button = pn.widgets.Button(name='Aplicar Filtro', button_type='primary')

```

```
filter_button.on_click(on_filter_button_click)
```

```
nome_input = pn.widgets.TextInput(name='Nome', placeholder='Nome do Remédio', width=150)
descricao_input = pn.widgets.TextInput(name='Descrição', placeholder='Descrição do Remédio', width=150)
create_button = pn.widgets.Button(name='Adicionar Remédio', button_type='primary')
```

```
def on_create_button_click(event):
    message = create_remedio(nome_input.value, descricao_input.value)
    update_table()
    output_pane.object = message
```

```
create_button.on_click(on_create_button_click)
```

```
id_input = pn.widgets.IntInput(name='ID do Remédio', width=150)
update_nome_input = pn.widgets.TextInput(name='Novo Nome', placeholder='Novo Nome do Remédio', width=150)
update_descricao_input = pn.widgets.TextInput(name='Nova Descrição', placeholder='Nova Descrição do Remédio', width=150)
update_button = pn.widgets.Button(name='Atualizar Remédio', button_type='warning')
```

```
def on_update_button_click(event):
    message = update_remedio(id_input.value, update_nome_input.value, update_descricao_input.value)
    update_table()
    output_pane.object = message
```

```
update_button.on_click(on_update_button_click)
```

```
delete_id_input = pn.widgets.IntInput(name='ID do Remédio para Excluir', width=150)
delete_button = pn.widgets.Button(name='Excluir Remédio', button_type='danger')
```

```
def on_delete_button_click(event):
    message = delete_remedio(delete_id_input.value)
    update_table()
    output_pane.object = message
```

```
delete_button.on_click(on_delete_button_click)
```

```

table = pn.widgets.DataFrame(read_remedios(), name='Tabela de Remédios', width=600)
output_pane = pn.pane.Markdown("Mensagens aparecerão aqui", width=600)

layout = pn.Column(
    pn.pane.Markdown("# Gerenciamento de Remédios", sizing_mode='stretch_width'),
    pn.Row(
        pn.Column(id_filter_input, nome_filter_input, filter_button),
        sizing_mode='stretch_width'
    ),
    pn.Row(
        pn.Column(nome_input, descricao_input, create_button),
        sizing_mode='stretch_width'
    ),
    pn.Row(
        pn.Column(id_input, update_nome_input, update_descricao_input, update_button),
        sizing_mode='stretch_width'
    ),
    pn.Row(
        pn.Column(delete_id_input, delete_button),
        sizing_mode='stretch_width'
    ),
    table,
    output_pane,
    sizing_mode='stretch_width'
)
layout.show()

```

16 CRUD da tabela cliente

16.1 Comando para instalar as dependências

```
pip install pandas panel sqlalchemy psycopg2-binary
```

16.2 Configuração banco de dados

```
import panel as pn
import pandas as pd
from sqlalchemy import create_engine, text
from sqlalchemy.exc import SQLAlchemyError
db_config = {
    'user': 'postgres',
    'password': '1412',
    'host': 'localhost',
    'port': '5432',
    'database': 'Gerenciamento_Remedios'
}
```

```
engine = create_engine(f"postgresql+psycopg2://{db_config['user']}:{db_config['p'
con = engine.connect()
```

16.3 CRUD da tabela cliente

```
def read_clientes(id_cliente=None, nome_cliente=None):
    try:
        if id_cliente is not None:
            query = text("SELECT * FROM farmacia.cliente WHERE id_cliente = :id_
            df = pd.read_sql(query, con, params={"id_cliente": id_cliente})
        elif nome_cliente is not None:
            query = text("SELECT * FROM farmacia.cliente WHERE nome_cliente ILIK
            df = pd.read_sql(query, con, params={"nome_cliente": f"%{nome_client
        else:
            query = "SELECT * FROM farmacia.cliente"
            df = pd.read_sql(query, con)
        return df
    except SQLAlchemyError as e:
        return str(e)

def create_cliente(nome_cliente, cpf, endereco_rua, endereco_numero, endereco_ba
    try:
```

```

query = text("""
    INSERT INTO farmacia.cliente
        (nome_cliente, cpf, endereco_rua, endereco_numero, endereco_bairro,
        VALUES (:nome_cliente, :cpf, :endereco_rua, :endereco_numero, :ender
    """)
con.execute(query, {
    "nome_cliente": nome_cliente,
    "cpf": cpf,
    "endereco_rua": endereco_rua,
    "endereco_numero": endereco_numero,
    "endereco_bairro": endereco_bairro,
    "endereco_cidade": endereco_cidade
})
con.commit()
return "Cliente adicionado com sucesso!"
except SQLAlchemyError as e:
    return str(e)

```

```

def update_cliente(id_cliente, nome_cliente, cpf, endereco_rua, endereco_numero,
try:
    query = text("""
        UPDATE farmacia.cliente
        SET nome_cliente = :nome_cliente, cpf = :cpf, endereco_rua = :endere
            endereco_numero = :endereco_numero, endereco_bairro = :endereco_
            endereco_cidade = :endereco_cidade
        WHERE id_cliente = :id_cliente
    """)
con.execute(query, {
    "id_cliente": id_cliente,
    "nome_cliente": nome_cliente,
    "cpf": cpf,
    "endereco_rua": endereco_rua,
    "endereco_numero": endereco_numero,
    "endereco_bairro": endereco_bairro,
    "endereco_cidade": endereco_cidade
})
con.commit()
return "Cliente atualizado com sucesso!"
except SQLAlchemyError as e:
    return str(e)

```

```

def delete_cliente(id_cliente):
    try:
        query = text("DELETE FROM farmacia.cliente WHERE id_cliente = :id_cliente")
        con.execute(query, {"id_cliente": id_cliente})
        con.commit()
        return "Cliente excluído com sucesso!"
    except SQLAlchemyError as e:
        return str(e)

def update_table(id_cliente=None, nome_cliente=None):
    df = read_clientes(id_cliente, nome_cliente)
    table.value = df

def on_filter_button_click(event):
    id_filter = id_filter_input.value if id_filter_input.value else None
    nome_filter = nome_filter_input.value if nome_filter_input.value else None
    update_table(id_cliente=id_filter, nome_cliente=nome_filter)
    output_pane.object = "Tabela atualizada com base no filtro."

id_filter_input = pn.widgets.IntInput(name='Filtrar por ID', width=150)
nome_filter_input = pn.widgets.TextInput(name='Filtrar por Nome', placeholder='Nome')
filter_button = pn.widgets.Button(name='Aplicar Filtro', button_type='primary')

filter_button.on_click(on_filter_button_click)

nome_input = pn.widgets.TextInput(name='Nome', placeholder='Nome do Cliente', width=150)
cpf_input = pn.widgets.TextInput(name='CPF', placeholder='CPF do Cliente', width=150)
endereco_rua_input = pn.widgets.TextInput(name='Endereço - Rua', placeholder='Rua', width=150)
endereco_numero_input = pn.widgets.IntInput(name='Endereço - Número', width=150)
endereco_bairro_input = pn.widgets.TextInput(name='Endereço - Bairro', placeholder='Bairro', width=150)
endereco_cidade_input = pn.widgets.TextInput(name='Endereço - Cidade', placeholder='Cidade', width=150)
create_button = pn.widgets.Button(name='Adicionar Cliente', button_type='primary')

def on_create_button_click(event):
    message = create_cliente(
        nome_input.value, cpf_input.value, endereco_rua_input.value,
        endereco_numero_input.value, endereco_bairro_input.value, endereco_cidade_input.value
    )

```



```

    )
    update_table()
    output_pane.object = message

create_button.on_click(on_create_button_click)

id_input = pn.widgets.IntInput(name='ID do Cliente', width=150)
update_nome_input = pn.widgets.TextInput(name='Novo Nome', placeholder='Novo Nome')
update_cpf_input = pn.widgets.TextInput(name='Novo CPF', placeholder='Novo CPF')
update_endereco_rua_input = pn.widgets.TextInput(name='Nova Rua', placeholder='Nova Rua')
update_endereco_numero_input = pn.widgets.IntInput(name='Novo Número', width=150)
update_endereco_bairro_input = pn.widgets.TextInput(name='Novo Bairro', placeholder='Novo Bairro')
update_endereco_cidade_input = pn.widgets.TextInput(name='Nova Cidade', placeholder='Nova Cidade')
update_button = pn.widgets.Button(name='Atualizar Cliente', button_type='warning')

def on_update_button_click(event):
    message = update_cliente(
        id_input.value, update_nome_input.value, update_cpf_input.value,
        update_endereco_rua_input.value, update_endereco_numero_input.value,
        update_endereco_bairro_input.value, update_endereco_cidade_input.value
    )
    update_table()
    output_pane.object = message

update_button.on_click(on_update_button_click)

delete_id_input = pn.widgets.IntInput(name='ID do Cliente para Excluir', width=150)
delete_button = pn.widgets.Button(name='Excluir Cliente', button_type='danger')

def on_delete_button_click(event):
    message = delete_cliente(delete_id_input.value)
    update_table()
    output_pane.object = message

delete_button.on_click(on_delete_button_click)

table = pn.widgets.DataFrame(read_clientes(), name='Tabela de Clientes', width=600)

```

```

output_pane = pn.pane.Markdown("Mensagens aparecerão aqui", width=600)

layout = pn.Column(
    pn.pane.Markdown("# Gerenciamento de Clientes", sizing_mode='stretch_width')
    pn.Row(
        pn.Column(id_filter_input, nome_filter_input, filter_button),
        sizing_mode='stretch_width'
    ),
    pn.Row(
        pn.Column(nome_input, cpf_input, endereco_rua_input, endereco_numero_inp
        sizing_mode='stretch_width'
    ),
    pn.Row(
        pn.Column(id_input, update_nome_input, update_cpf_input, update_endereco
        sizing_mode='stretch_width'
    ),
    pn.Row(
        pn.Column(delete_id_input, delete_button),
        sizing_mode='stretch_width'
    ),
    table,
    output_pane,
    sizing_mode='stretch_width'
)

layout.show()

```

17 Consulta da tabela Cliente

```

import panel as pn
import pandas as pd
from sqlalchemy import create_engine, text

```

```

db_config = {
    'user': 'postgres',
    'password': '1412',
    'host': 'localhost',

```

```

    'port': '5432',
    'database': 'Sistemarm'
}

```

```

engine = create_engine(f"postgresql+psycopg2://{db_config['user']}:{db_config['p
con = engine.connect()

```

```

def read_clientes(filtro_tipo=None, filtro_valor=None, columns=None):
    if columns is None:
        columns = ["*"]
    columns_str = ", ".join(columns)

```

```

    query = "SELECT " + columns_str + " FROM farmacia.cliente"

```

```

    if filtro_tipo:
        if filtro_tipo == "id":
            query += " WHERE id_cliente = :filtro_valor"
        elif filtro_tipo == "nome":
            query += " WHERE nome_cliente ILIKE :filtro_valor"
            filtro_valor = f"%{filtro_valor}%"
        elif filtro_tipo == "cpf":
            query += " WHERE cpf = :filtro_valor"
        elif filtro_tipo == "cidade":
            query += " WHERE endereco_cidade ILIKE :filtro_valor"
            filtro_valor = f"%{filtro_valor}%"
        elif filtro_tipo == "estado":
            query += " WHERE endereco_estado = :filtro_valor"

```

```

    df = pd.read_sql(text(query), con, params={"filtro_valor": filtro_valor})
    return df

```

```

def on_filter_button_click(event):
    filtro_tipo = filtro_tipo_input.value
    filtro_valor = filtro_valor_input.value
    columns = ["nome_cliente", "endereco_cidade", "cpf", "endereco_estado"] #
    df = read_clientes(filtro_tipo=filtro_tipo, filtro_valor=filtro_valor, colum
    table.value = df

```

```

filtro_tipo_input = pn.widgets.Select(name="Tipo de Filtro", options=["id", "nom
filtro_valor_input = pn.widgets.TextInput(name="Valor do Filtro", placeholder="D
filter_button = pn.widgets.Button(name="Aplicar Filtro", button_type="primary")

table = pn.widgets.DataFrame(pd.DataFrame(), name="Resultado da Consulta", width

filter_button.on_click(on_filter_button_click)

layout = pn.Column(
    pn.pane.Markdown("# Consulta de Clientes Dinâmica", sizing_mode='stretch_wid
    pn.Row(filtro_tipo_input, filtro_valor_input, filter_button),
    table
)

pn.extension()
layout.show()

```

18 Consulta da tabela Remédio

```

import panel as pn
import pandas as pd
from sqlalchemy import create_engine, text

db_config = {
    'user': 'postgres',
    'password': '1412',
    'host': 'localhost',
    'port': '5432',
    'database': 'Sistemarm'
}

engine = create_engine(f"postgresql+psycopg2://{db_config['user']}:{db_config['p
con = engine.connect()

```

```

def read_remedios(filtro_tipo=None, filtro_valor=None, columns=None):
    if columns is None:
        columns = ["*"]
    columns_str = ", ".join(columns)

    query = "SELECT " + columns_str + " FROM farmacia.remedio"

    if filtro_tipo:
        if filtro_tipo == "id":
            query += " WHERE id_remedio = :filtro_valor"
        elif filtro_tipo == "nome":
            query += " WHERE nome ILIKE :filtro_valor"
            filtro_valor = f"%{filtro_valor}%"

    df = pd.read_sql(text(query), con, params={"filtro_valor": filtro_valor})
    return df

def on_filter_button_click(event):
    filtro_tipo = filtro_tipo_input.value
    filtro_valor = filtro_valor_input.value
    columns = ["id_remedio", "nome", "descricao"]
    df = read_remedios(filtro_tipo=filtro_tipo, filtro_valor=filtro_valor, columns=columns)
    table.value = df

filtro_tipo_input = pn.widgets.Select(name="Tipo de Filtro", options=["id", "nome"])
filtro_valor_input = pn.widgets.TextInput(name="Valor do Filtro", placeholder="Digite o valor")
filter_button = pn.widgets.Button(name="Aplicar Filtro", button_type="primary")

table = pn.widgets.DataFrame(pd.DataFrame(), name="Resultado da Consulta", width=1000)

filter_button.on_click(on_filter_button_click)

layout = pn.Column(
    pn.pane.Markdown("# Consulta de Remédios Dinâmica", sizing_mode='stretch_width'),
    pn.Row(filtro_tipo_input, filtro_valor_input, filter_button),

```

```
    table  
)
```

```
pn.extension()  
layout.show()
```

19 Link do Repositório

<https://github.com/Naliat/EntregaFinalFBD>

Tarefa	Quem Realizou
Finalização do SQL	Tailan de Souza
Popular tabelas	Kaio
Crud da tabela Remédio	Kaio
Crud da tabela Cliente	Tailan de Souza
Modelo ER final	Kaio
Modelo ERD final	Kaio
Subir arquivos gitHub	Tailan de Souza
Relatório e documentação	Tailan de Souza
Aplicação realizando pelo menos 3 consultas no Banco de Dados	Tailan de Souza
Análise final	Tailan de Souza e Kaio
Correções sugeridos pela Livia	Tailan de Souza e Kaio