

UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE EDUCAÇÃO DO PLANALTO NORTE – CEPLAN
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
BANCO DE DADOS

JEAN FELIPE Kruk DE OLIVEIRA
WELINGTON BRAYER COPATTI

TRABALHO II: SISTEMA DE GESTÃO DE POSTO DE SAÚDE COMUNITÁRIO

São Bento do Sul

2026

INTRODUÇÃO.....	3
DESCRIÇÃO DO PROBLEMA.....	4
PROJETO CONCEITUAL.....	5

ENTIDADES.....	5
RELACIONAMENTOS.....	6
ATRIBUTOS.....	7
DIAGRAMA ENTIDADE RELACIONAMENTO ESTENDIDO (DEER).....	11
PROJETO LÓGICO RELACIONAL.....	12
CRIAÇÃO DO BANCO DE DADOS.....	13
SCRIPT.....	13
USUÁRIOS.....	16
INSERÇÃO DE REGISTROS (TESTES).....	16
DESENVOLVIMENTO DA APLICAÇÃO.....	18
MANIPULAÇÃO DA TABELA 'PACIENTES'.....	18
MANIPULAÇÃO DA TABELA 'ESPECIALIDADE'.....	23

INTRODUÇÃO

Com o nível de tecnologia que temos hoje no mundo, podemos cada vez mais proporcionar melhorias significativas em alguns setores de serviços em variadas áreas, em especial, que trataremos no trabalho, a área da saúde. O uso da tecnologia e de sistemas de informação nos possibilita o armazenamento, organização e análise de dados de uma forma mais eficiente. Além de contribuir para a tomada de decisões, com o intuito de melhorar a qualidade dos serviços prestados à nossa sociedade.

O atual trabalho tem como foco principal a área da saúde, mas especificamente, o setor dos postos de saúde comunitários, conhecidos como Unidades Básicas de Saúde (UBS), que desempenham um papel essencial no atendimento básico, na gestão e gerenciamento dessas informações. Tornar esses “postinhos”, como são chamados pelos cidadãos, em um lugar organizado e eficiente, é um fator decisivo para que consigamos garantir a qualidade, a agilidade e a segurança adequados nos atendimentos prestados à população. Entretanto, alguns desses estabelecimentos ainda têm dificuldades relacionadas ao controle de das informações, organização de processos, e acima de tudo, à integração dessas informações entre as diferentes áreas existentes nesse sistema de um município.

Tendo em vista essas informações se torna fundamental o desenvolvimento de soluções tecnológicas que auxiliem na estruturação e gerenciamento dessas informações. O presente trabalho tem como objetivo principal propor um projeto de um banco de dados para um sistema de gestão de posto de saúde comunitário, contemplando o contexto do problema, e então, apresentando desde o diagrama entidade relacionamento estendido até a definição do modelo lógico relacional.

A proposta apresentada busca representar de forma estruturada os principais elementos envolvidos no funcionamento de um posto de saúde, como pacientes, profissionais, consultas, agendamentos, prontuários, medicamentos e vacinas, bem como os relacionamentos existentes entre todos esses componentes. Seguindo a proposta, pretendemos demonstrar como a utilização de um banco de dados pode contribuir de forma benéfica para a melhoria da organização, atendimento e gestão dos serviços prestados nos postos de saúde.

DESCRIÇÃO DO PROBLEMA

É de conhecimento geral a importância que os postos de saúde comunitários têm no atendimento básico à população, sendo responsáveis por serviços como consultas médicas, vacinação, acompanhamento de pacientes e distribuição de medicamentos. Entretanto, muitos desses estabelecimentos ainda enfrentam dificuldades em relação à organização e gestão de informações, que acaba comprometendo a eficiência dos atendimentos e a qualidade dos serviços prestados.

Um dos principais problemas observados é a descentralização das informações de pacientes, como o histórico de consultas, prontuários e registros de vacinação que frequentemente não têm ligações, o que dificulta o acesso rápido das informações necessárias durante um atendimento. Isso pode acabar resultando em retrabalho, perda de dados, e em casos mais graves, resulta indiretamente em complicações da situação do paciente.

Atualmente, em muitos casos, processos de agendamento de consultas estão sendo realizados de forma manual, com ferramentas pouco eficientes, gerando conflitos de horários, atrasos, longas filas, etc... A falta de uso de um sistema adequado dificulta o acompanhamento dos pacientes e o planejamento do posto de saúde. Sem um sistema estruturado, torna-se difícil até registrar dados importantes dos profissionais de saúde, como suas especialidades, quais procedimentos eles realizaram e quais pacientes foram atendidos por eles. Esses problemas impactam diretamente na distribuição dos serviços e eficiência da equipe responsável.

Diante desses e outros cenários, se vê necessário um sistema informatizado e centralizado, que permita o armazenamento, organização e integração dessas informações. A solução proposta na elaboração deste trabalho se baseia no desenvolvimento de um banco de dados capaz de gerenciar entidades como pacientes, especialidades, profissionais, consultas, agendamentos, receitas, medicamentos, item_receita, lote_medicamento, fabricante, vacinas e item_estoque, lote_vacina, aplicação_vacina, além de estruturar corretamente os relacionamentos entre essas principais entidades e seus elementos.

A implementação desse projeto busca trazer a melhoria geral do funcionamento dos postos de saúde, proporcionando mais agilidade e qualidade no atendimento, maior controle dos recursos e confiabilidade das informações, entre outros fatores que deixam a desejar atualmente. Dessa forma, o sistema contribuirá tanto para a melhoria das condições de trabalho dos profissionais quanto para a qualidade dos atendimentos oferecidos à comunidade.

PROJETO CONCEITUAL

ENTIDADES

- **Paciente:** Representa os usuários atendidos pelo posto de saúde, contendo seus dados pessoais e de contato.
- **Especialidade:** Representa as áreas de atuação médica (ex: Pediatria, Cardiologia) às quais os profissionais estão vinculados.
- **Profissional:** Armazena as informações dos profissionais de saúde que realizam atendimentos, como médicos e enfermeiros.
- **Agendamento:** Controla o agendamento prévio das consultas feitas pelos pacientes.
- **Consulta:** Registra os atendimentos realizados, incluindo data, horário, diagnóstico e observações.
- **Receita:** Registra as prescrições médicas geradas durante uma consulta.
- **Medicamento:** Contém os medicamentos disponíveis para prescrição e distribuição.
- **Item_Receita:** Tabela associativa que relaciona uma receita aos medicamentos prescritos nela, incluindo dose e quantidade.
- **Lote_Medicamento:** Identifica os lotes de fabricação dos medicamentos, permitindo rastreabilidade.
- **Item_Estoque:** Controla a quantidade e a validade de cada medicamento por lote disponível no posto.
- **Fabricante:** Representa as empresas responsáveis pela fabricação das vacinas.
- **Vacina:** Armazena informações sobre as vacinas disponíveis no posto de saúde.
- **Lote_Vacina:** Identifica os lotes de fabricação de cada vacina, com respectiva validade.
- **AplicacaoVacina:** Registra as vacinas aplicadas nos pacientes, incluindo data e dose.
- OBS: A entidade prontuário, presente na versão anterior, foi removida. Ela não faz parte do script SQL atual.

RELACIONAMENTOS

- **Paciente — realiza — Agendamento:** Um paciente pode realizar vários agendamentos, mas cada agendamento pertence a um único paciente.
- **Agendamento — resulta em — Consulta:** Cada agendamento pode gerar, no máximo, uma consulta.

- **Profissional — atende — Consulta:** Um profissional pode atender várias consultas, sendo cada consulta realizada por um único profissional.
- **Especialidade — classifica — Profissional:** Uma especialidade pode estar associada a vários profissionais, mas cada profissional possui uma única especialidade.
- **Consulta — gera — Receita:** Cada consulta pode gerar, no máximo, uma receita.
- **Receita — contém — Medicamento (via Item_Receita):** Uma receita pode conter vários medicamentos, e um medicamento pode estar presente em várias receitas.
- **Medicamento — possui — Lote_Medicamento (via Item_Estoque):** Um medicamento pode ter vários lotes em estoque, e cada registro de estoque identifica a quantidade e validade de um lote específico.
- **Fabricante — fabrica — Vacina:** Um fabricante pode produzir várias vacinas, mas cada vacina pertence a um único fabricante.
- **Vacina — possui — Lote_Vacina:** Uma vacina pode ter vários lotes, cada um com validade própria.
- **Paciente — recebe — AplicacaoVacina:** Um paciente pode receber várias aplicações de vacina.
- **AplicacaoVacina — utiliza — Lote_Vacina:** Cada aplicação utiliza um lote específico de vacina.

ATRIBUTOS

- Paciente:
 - id_paciente: Identificador único do paciente
 - nome: Nome completo do paciente
 - cpf: Documento de identificação do paciente (11 dígitos)
 - data_nascimento: Data de nascimento
 - telefone: Número para contato
 - endereco: Endereço residencial

- Especialidade:
 - id_especialidade: Identificador único da especialidade
 - descricao: Nome da área de atuação
 - Profissional:
 - id_profissional: Identificador único do profissional
 - nome: Nome completo
 - crm: Registro profissional
 - telefone: Contato do profissional
 - id_especialidade: Especialidade à qual o profissional pertence

- Agendamento:
 - id_agendamento: Identificador do agendamento
 - data: Data marcada
 - hora: Horário marcado
 - status: Situação do agendamento (AGENDADO, CONFIRMADO, CANCELADO, REALIZADO)
 - id_paciente: Paciente que realizou o agendamento

- Consulta:
 - id_consulta: Identificador da consulta
 - data: Data do atendimento
 - hora: Horário do atendimento
 - motivo: Motivo da consulta
 - diagnostico: Diagnóstico realizado
 - observacoes: Informações adicionais
 - id_profissional: Profissional responsável pelo atendimento
 - id_agendamento: Agendamento que originou a consulta

- Receita:
 - id_receita: Identificador da receita
 - data: Data de emissão
 - instrucoes: Orientações de uso
 - id_consulta: Consulta que originou a receita

- Medicamento:
 - id_medicamento: Identificador do medicamento
 - nome: Nome do medicamento
 - descricao: Descrição do medicamento

- Item_Receita:
 - id_item_receita: Identificador do item da receita
 - dose: Dosagem recomendada
 - quantidade: Quantidade prescrita
 - id_receita: Receita à qual o item pertence
 - id_medicamento: Medicamento prescrito

- Lote_Medicamento:
 - id_lote_medicamento: Identificador do lote
 - numero_lote: Código do lote de fabricação
 - Item_Estoque:
 - id_item_estoque: Identificador do registro de estoque
 - quantidade: Quantidade disponível
 - validade: Data de validade
 - id_medicamento: Medicamento correspondente
 - id_lote_medicamento: Lote correspondente

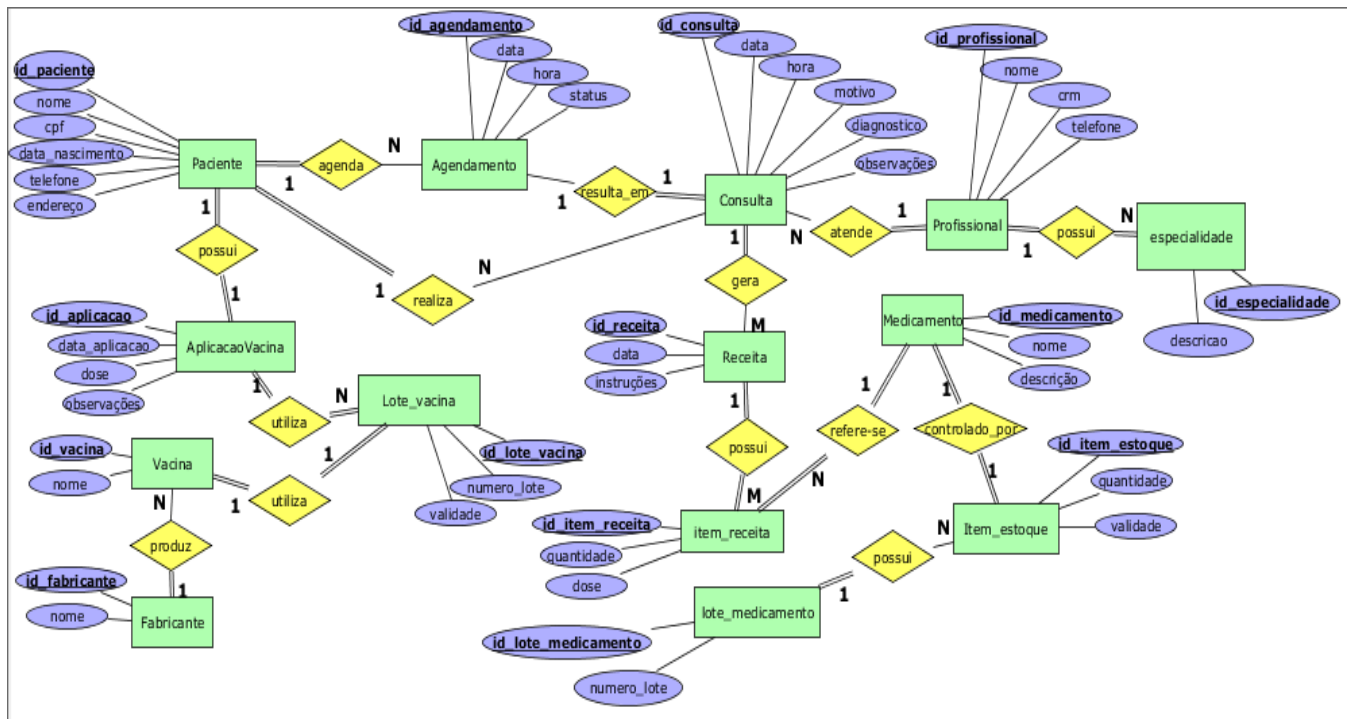
- Fabricante:
 - id_fabricante: Identificador do fabricante
 - nome: Nome da empresa fabricante

- Vacina:
 - id_vacina: Identificador da vacina
 - nome: Nome da vacina
 - id_fabricante: Fabricante responsável pela vacina

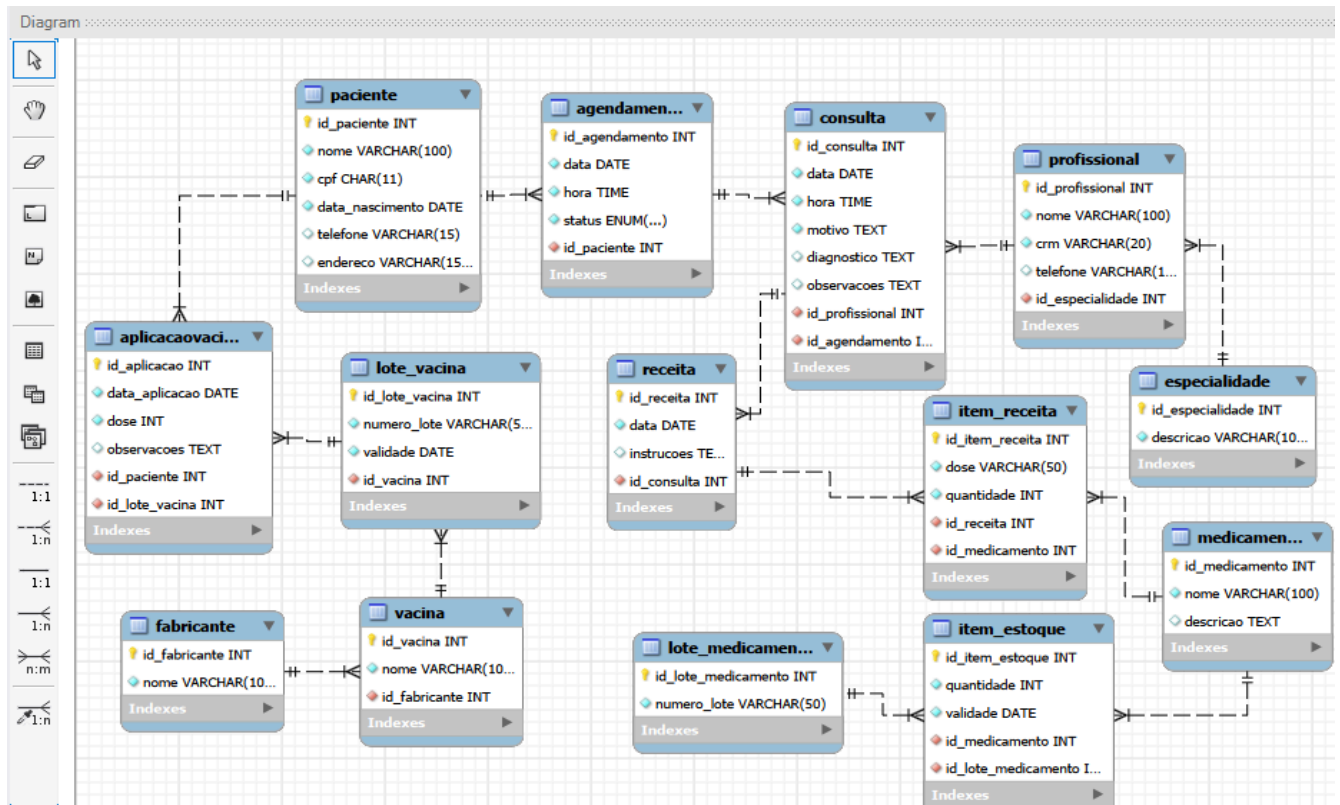
- Lote_Vacina:
 - id_lote_vacina: Identificador do lote
 - numero_lote: Código do lote de fabricação
 - validade: Data de validade do lote
 - id_vacina: Vacina correspondente ao lote

- AplicacaoVacina:
 - id_aplicacao: Identificador da aplicação
 - data_aplicacao: Data em que foi aplicada
 - dose: Número da dose
 - observacoes: Informações adicionais
 - id_paciente: Paciente que recebeu a aplicação
 - id_lote_vacina: Lote da vacina utilizado na aplicação

DIAGRAMA ENTIDADE RELACIONAMENTO ESTENDIDO (DEER)



PROJETO LÓGICO RELACIONAL



CRIAÇÃO DO BANCO DE DADOS

SCRIPT

#Criação do banco de dados:

CREATE DATABASE ProjetoPosto;

USE ProjetoPosto;

#Criação das tabelas:

CREATE TABLE Paciente (

id_paciente INT AUTO_INCREMENT PRIMARY KEY,

nome VARCHAR(100) NOT NULL,

cpf CHAR(11) NOT NULL UNIQUE CHECK (CHAR_LENGTH(cpf) = 11),

data_nascimento DATE NOT NULL CHECK (data_nascimento >= '1900-01-01'),

telefone VARCHAR(15) CHECK (telefone IS NULL OR CHAR_LENGTH(telefone)
BETWEEN 10 AND 15),

endereco VARCHAR(150)

);

CREATE TABLE Especialidade (

id_especialidade INT AUTO_INCREMENT PRIMARY KEY,

descricao VARCHAR(100) NOT NULL UNIQUE

);

CREATE TABLE Profissional (

id_profissional INT AUTO_INCREMENT PRIMARY KEY,

nome VARCHAR(100) NOT NULL,

crm VARCHAR(20) NOT NULL UNIQUE CHECK (CHAR_LENGTH(crm) >= 4),

telefone VARCHAR(15) CHECK (telefone IS NULL OR CHAR_LENGTH(telefone)
BETWEEN 10 AND 15),

id_especialidade INT NOT NULL,

FOREIGN KEY (id_especialidade) REFERENCES
Especialidade(id_especialidade)

);

CREATE TABLE Agendamento (

id_agendamento INT AUTO_INCREMENT PRIMARY KEY,

data DATE NOT NULL,

hora TIME NOT NULL,

status ENUM('AGENDADO', 'CONFIRMADO', 'CANCELADO', 'REALIZADO') NOT
NULL,

id_paciente INT NOT NULL,

FOREIGN KEY (id_paciente) REFERENCES Paciente(id_paciente)
);

CREATE TABLE Consulta (

id_consulta INT AUTO_INCREMENT PRIMARY KEY,

```

data DATE NOT NULL,
hora TIME NOT NULL,
motivo TEXT NOT NULL,
diagnostico TEXT,
observacoes TEXT,
id_profissional INT NOT NULL,
id_agendamento INT NOT NULL UNIQUE,

FOREIGN KEY (id_profissional) REFERENCES Profissional(id_profissional),
FOREIGN KEY (id_agendamento) REFERENCES
Agendamento(id_agendamento)
);

CREATE TABLE Receita (
id_receita INT AUTO_INCREMENT PRIMARY KEY,
data DATE NOT NULL,
instrucoes TEXT,
id_consulta INT NOT NULL UNIQUE,
FOREIGN KEY (id_consulta) REFERENCES Consulta(id_consulta)
);

CREATE TABLE Medicamento (
id_medicamento INT AUTO_INCREMENT PRIMARY KEY,
nome VARCHAR(100) NOT NULL,
descricao TEXT
);

CREATE TABLE Item_Receita (
id_item_receita INT AUTO_INCREMENT PRIMARY KEY,
dose VARCHAR(50) NOT NULL,
quantidade INT NOT NULL CHECK (quantidade > 0),
id_receita INT NOT NULL,
id_medicamento INT NOT NULL,

FOREIGN KEY (id_receita) REFERENCES Receita(id_receita),
FOREIGN KEY (id_medicamento) REFERENCES Medicamento(id_medicamento)
);

CREATE TABLE Lote_Medicamento (
id_lote_medicamento INT AUTO_INCREMENT PRIMARY KEY,
numero_lote VARCHAR(50) NOT NULL UNIQUE
);

CREATE TABLE Item_Estoque (
id_item_estoque INT AUTO_INCREMENT PRIMARY KEY,
quantidade INT NOT NULL CHECK (quantidade >= 0),
validade DATE NOT NULL CHECK (validade > '2000-01-01'),

```

```

id_medicamento INT NOT NULL,
id_lote_medicamento INT NOT NULL,

FOREIGN KEY (id_medicamento) REFERENCES Medicamento(id_medicamento),
FOREIGN KEY (id_lote_medicamento) REFERENCES
Lote_Medicamento(id_lote_medicamento)
);

CREATE TABLE Fabricante (
    id_fabricante INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL UNIQUE
);

CREATE TABLE Vacina (
    id_vacina INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    id_fabricante INT NOT NULL,

    FOREIGN KEY (id_fabricante) REFERENCES Fabricante(id_fabricante)
);

CREATE TABLE Lote_Vacina (
    id_lote_vacina INT AUTO_INCREMENT PRIMARY KEY,
    numero_lote VARCHAR(50) NOT NULL UNIQUE,
    validade DATE NOT NULL CHECK (validade > '2000-01-01'),
    id_vacina INT NOT NULL,

    FOREIGN KEY (id_vacina) REFERENCES Vacina(id_vacina)
);

CREATE TABLE AplicacaoVacina (
    id_aplicacao INT AUTO_INCREMENT PRIMARY KEY,
    data_aplicacao DATE NOT NULL,
    dose INT NOT NULL CHECK (dose > 0),
    observacoes TEXT,
    id_paciente INT NOT NULL,
    id_lote_vacina INT NOT NULL,

    FOREIGN KEY (id_paciente) REFERENCES Paciente(id_paciente),
    FOREIGN KEY (id_lote_vacina) REFERENCES Lote_Vacina(id_lote_vacina)
);

```

USUÁRIOS

#Criação e definição de permissões para os usuários:

```
CREATE USER 'administrador'@'localhost' IDENTIFIED BY 'admin';  
GRANT ALL PRIVILEGES ON ProjetoPosto.* TO 'administrador'@'localhost';
```

```
CREATE USER 'funcionario'@'localhost' IDENTIFIED BY 'funcionario';  
GRANT SELECT, INSERT ON ProjetoPosto.* TO 'funcionario'@'localhost';
```

```
FLUSH PRIVILEGES;
```

INSERÇÃO DE REGISTROS (TESTES)

#Inserção de registros nas tabelas (na sequência de criação):

```
INSERT INTO Especialidade (descricao) VALUES  
(('Clínica Geral'), ('Pediatria'), ('Ginecologia'), ('Cardiologia'), ('Dermatologia'),  
(('Ortopedia'));
```

```
INSERT INTO Profissional (nome, crm, telefone, id_especialidade) VALUES  
(('Dr. Carlos Mendes', 'CRM12345', '47999990001', 1),  
(('Dra. Fernanda Souza', 'CRM23456', '47999990002', 2),  
(('Dra. Juliana Alves', 'CRM34567', '47999990003', 3),  
(('Dr. Ricardo Lima', 'CRM45678', '47999990004', 4),  
(('Dra. Patricia Rocha', 'CRM56789', '47999990005', 5),  
(('Dr. Eduardo Santos', 'CRM67890', '47999990006', 6);
```

```
INSERT INTO Paciente (nome, cpf, data_nascimento, telefone, endereco) VALUES  
(('Ana Beatriz Costa', '12345678901', '1990-05-14', '47988880001', 'Rua das Flores,  
123'),  
(('Bruno Henrique Silva', '23456789012', '1985-11-02', '47988880002', 'Av. Brasil,  
456'),  
(('Camila Ferreira', '34567890123', '2000-02-20', '47988880003', 'Rua XV de  
Novembro, 789'),  
(('Diego Martins', '45678901234', '1978-07-30', '47988880004', 'Rua Sete de  
Setembro, 321'),  
(('Elaine Cristina Oliveira', '56789012345', '1995-09-09', '47988880005', 'Rua Rio  
Branco, 654'),  
(('Fabio Rodrigues', '67890123456', '1960-12-25', '47988880006', 'Rua Getúlio Vargas,  
987');
```

```
INSERT INTO Agendamento (data, hora, status, id_paciente) VALUES  
(('2026-07-10', '08:00:00', 'REALIZADO', 1),  
(('2026-07-10', '09:00:00', 'REALIZADO', 2),  
(('2026-07-11', '10:30:00', 'REALIZADO', 3),  
(('2026-07-11', '14:00:00', 'REALIZADO', 4),  
(('2026-07-12', '08:30:00', 'REALIZADO', 5),  
(('2026-07-12', '15:00:00', 'REALIZADO', 6),  
(('2026-07-15', '09:00:00', 'AGENDADO', 1),
```

('2026-07-16', '11:00:00', 'CANCELADO', 3);

INSERT INTO Consulta (data, hora, motivo, diagnostico, observacoes, id_profissional, id_agendamento) VALUES
('2026-07-10', '08:00:00', 'Dor de cabeça recorrente', 'Enxaqueca', 'Paciente orientado a retornar em 30 dias', 1, 1),
('2026-07-10', '09:00:00', 'Consulta de rotina pediátrica', 'Desenvolvimento normal', NULL, 2, 2),
('2026-07-11', '10:30:00', 'Check-up ginecológico anual', 'Sem alterações', NULL, 3, 3),
('2026-07-11', '14:00:00', 'Dor no peito', 'Suspeita de arritmia', 'Encaminhado para exames complementares', 4, 4),
('2026-07-12', '08:30:00', 'Lesão de pele suspeita', 'Dermatite de contato', NULL, 5, 5),
('2026-07-12', '15:00:00', 'Dor no joelho após queda', 'Entorse leve', 'Repouso recomendado por 7 dias', 6, 6);

INSERT INTO Receita (data, instrucoes, id_consulta) VALUES
('2026-07-10', 'Tomar 1 comprimido a cada 8 horas por 5 dias', 1),
('2026-07-10', 'Tomar 1 comprimido a cada 6 horas se necessário', 2),
('2026-07-11', 'Uso conforme orientação médica', 3),
('2026-07-11', 'Uso contínuo conforme prescrição cardiológica', 4),
('2026-07-12', 'Aplicar pomada 2x ao dia por 10 dias', 5),
('2026-07-12', 'Repouso e uso de anti-inflamatório por 7 dias', 6);

INSERT INTO Medicamento (nome, descricao) VALUES
('Dipirona Sódica', 'Analgésico e antitérmico'),
('Paracetamol', 'Analgésico e antitérmico'),
('Amoxicilina', 'Antibiótico de amplo espectro'),
('Losartana', 'Anti-hipertensivo'),
('Betametasona Creme', 'Corticoide tópico'),
('Ibuprofeno', 'Anti-inflamatório não esteroide');)

INSERT INTO Item_Receita (dose, quantidade, id_receita, id_medicamento) VALUES
('500mg', 20, 1, 1),
('750mg', 12, 2, 2),
('500mg', 21, 3, 3),
('50mg', 30, 4, 4),
('aplicação local', 1, 5, 5),
('400mg', 10, 6, 6);

INSERT INTO Lote_Medicamento (numero_lote) VALUES
('LOTE-001'),
('LOTE-002'),
('LOTE-003'),
('LOTE-004'),
('LOTE-005'),
('LOTE-006');

DESENVOLVIMENTO DA APLICAÇÃO

MANIPULAÇÃO DA TABELA 'PACIENTES'

```
import mysql.connector
from mysql.connector import Error

def mensagem_duplicidade(erro):
    texto = str(erro).lower()
    if "cpf" in texto:
        return "Não foi possível registrar o paciente, pois já existe um paciente cadastrado com esse CPF."
    if "telefone" in texto:
        return "Não foi possível registrar o paciente, pois já existe um paciente cadastrado com esse telefone."
    return "Não foi possível salvar: valor duplicado em um campo único."

class Paciente:

    def __init__(self, nome, cpf, data_nascimento, telefone, endereco, id_paciente=None):
        self.id_paciente = id_paciente
        self.nome = nome
        self.cpf = cpf
        self.data_nascimento = data_nascimento
        self.telefone = telefone
        self.endereco = endereco

    def __str__(self):
        return (f"[{self.id_paciente}] {self.nome} | CPF: {self.cpf} | "
                f"Nascimento: {self.data_nascimento} | Telefone: {self.telefone} | "
                f"Endereço: {self.endereco}")

class BancoDados:

    def __init__(self, host="localhost", user="administrador",
                 password="admin", database="ProjetoPosto"):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
```

```
self.conexao = None
```

```
def conectar(self):
    try:
        self.conexao = mysql.connector.connect(
            host=self.host,
            user=self.user,
            password=self.password,
            database=self.database
        )
        print("Conexão com o banco de dados bem-sucedida.")
    except Error as erro:
        print(f"Erro ao se conectar no banco de dados: {erro}")
        self.conexao = None
```

```
def desconectar(self):
    if self.conexao:
        self.conexao.close()
        print("A conexão com o MySQL foi fechada.")
```

```
def inserir(self, paciente):
    cursor = self.conexao.cursor()
    sql = ("INSERT INTO Paciente (nome, cpf, data_nascimento, telefone,
    endereco) "
           "VALUES (%s, %s, %s, %s, %s)")
    valores = (paciente.nome, paciente.cpf, paciente.data_nascimento,
               paciente.telefone, paciente.endereco)
    try:
        cursor.execute(sql, valores)
        self.conexao.commit()
        print(f"{cursor.rowcount} registro(s) inserido(s).")
    except Error as erro:
        self.conexao.rollback()
        if erro.errno == 1062:
            print(mensagem_duplicidade(erro))
        else:
            print(f"Erro ao inserir paciente: {erro}")
    finally:
        cursor.close()
```

```
def listar(self):
    cursor = self.conexao.cursor()
    cursor.execute(
        "SELECT id_paciente, nome, cpf, data_nascimento, telefone, endereco "
```

```

        "FROM Paciente"
    )
    resultado = cursor.fetchall()
    cursor.close()

    pacientes = []
    for id_paciente, nome, cpf, data_nascimento, telefone, endereco in resultado:
        pacientes.append(Paciente(nome, cpf, data_nascimento, telefone, endereco,
id_paciente))
    return pacientes

```

```

def buscar_por_id(self, id_paciente):
    cursor = self.conexao.cursor()
    cursor.execute(
        "SELECT id_paciente, nome, cpf, data_nascimento, telefone, endereco "
        "FROM Paciente WHERE id_paciente = %s",
        (id_paciente,)
    )
    registro = cursor.fetchone()
    cursor.close()

    if registro:
        return Paciente(registro[1], registro[2], registro[3],
            registro[4], registro[5], registro[0])
    return None

```

```

def atualizar(self, paciente):
    cursor = self.conexao.cursor()
    sql = ("UPDATE Paciente SET nome = %s, cpf = %s, data_nascimento = %s, "
        "telefone = %s, endereco = %s WHERE id_paciente = %s")
    valores = (paciente.nome, paciente.cpf, paciente.data_nascimento,
        paciente.telefone, paciente.endereco, paciente.id_paciente)
    try:
        cursor.execute(sql, valores)
        self.conexao.commit()
        print(f"{cursor.rowcount} registro(s) atualizado(s).")
    except Error as erro:
        self.conexao.rollback()
        if erro.errno == 1062:
            print(mensagem_duplicidade(erro))
        else:
            print(f"Erro ao atualizar registro de paciente: {erro}")
    finally:
        cursor.close()

```

```

def remover(self, id_paciente):

```

```

        cursor = self.conexao.cursor()
        try:
            cursor.execute(
                "DELETE FROM Paciente WHERE id_paciente = %s",
                (id_paciente,)
            )
            self.conexao.commit()
            print(f"{cursor.rowcount} registro(s) removido(s).")
        except Error as erro:
            self.conexao.rollback()
            if erro.errno == 1451:
                print("Não foi possível remover: existem agendamentos ou aplicações de
vacina "
                    "vinculados a esse paciente.")
            else:
                print(f"Erro ao remover paciente: {erro}")
        finally:
            cursor.close()

```

```

def menu_adicionar(bd):
    nome = input("Nome: ")
    cpf = input("CPF (11 dígitos): ")
    data_nascimento = input("Data de nascimento (AAAA-MM-DD): ")
    telefone = input("Telefone: ")
    endereco = input("Endereço: ")
    novo = Paciente(nome, cpf, data_nascimento, telefone, endereco)
    bd.inserir(novo)

```

```

def menu_listar(bd):
    for paciente in bd.listar():
        print(paciente)

```

```

def menu_buscar(bd):
    id_paciente = input("\nDigite o ID do paciente: ")
    paciente = bd.buscar_por_id(id_paciente)
    if paciente:
        print(paciente)
    else:
        print("\nO paciente não foi encontrado.")
    return paciente

```

```

def menu_atualizar(bd):
    paciente = menu_buscar(bd)

```

```

if paciente:
    print("\nDeixe em branco para manter o dado atual.\n")
    nome = input(f"Nome [{paciente.nome}]: ")
    cpf = input(f"CPF [{paciente.cpf}]: ")
    data_nascimento = input(f"Data de nascimento [{paciente.data_nascimento}]: ")
    telefone = input(f"Telefone [{paciente.telefone}]: ")
    endereco = input(f"Endereço [{paciente.endereco}]: ")

    paciente.nome = nome or paciente.nome
    paciente.cpf = cpf or paciente.cpf
    paciente.data_nascimento = data_nascimento or paciente.data_nascimento
    paciente.telefone = telefone or paciente.telefone
    paciente.endereco = endereco or paciente.endereco

    bd.atualizar(paciente)

```

```

def menu_remove(bd):
    id_paciente = input("\nID do paciente a remover: ")
    bd.remove(id_paciente)

```

```

def menu_principal():
    bd = BancoDados()
    bd.conectar()

```

```

if not bd.conexao:
    return

```

```

while True:
    print("\nCADASTRO DE PACIENTES\n")
    print("1 - Adicionar paciente")
    print("2 - Listar todos os pacientes")
    print("3 - Buscar paciente")
    print("4 - Atualizar cadastro de paciente")
    print("5 - Remover paciente")
    print("0 - Sair")

```

```

opcao = input("\nEscolha uma opção: ")

```

```

if opcao == "1":
    menu_adicionar(bd)
elif opcao == "2":
    menu_listar(bd)
elif opcao == "3":
    menu_buscar(bd)
elif opcao == "4":
    menu_atualizar(bd)

```

```
elif opcao == "5":  
    menu_remove(bd)  
elif opcao == "0":  
    print("Encerrando...")  
    break  
else:  
    print("Opção inválida. Digite uma opção (válida) novamente.")
```

```
bd.desconectar()
```

```
if __name__ == "__main__":  
    menu_principal()
```

MANIPULAÇÃO DA TABELA 'ESPECIALIDADE'

```
import mysql.connector
from mysql.connector import Error

class Especialidade:
    def __init__(self, descricao, id_especialidade=None):
        self.id_especialidade = id_especialidade
        self.descricao = descricao

    def __str__(self):
        return "[%s] %s" % (self.id_especialidade, self.descricao)

class BancoDados:
    def __init__(self, host="localhost", user="administrador",
                 password="admin", database="ProjetoPosto"):
        self.host = host
        self.user = user
        self.password = password
        self.database = database
        self.conexao = None

    def conectar(self):
        try:
            self.conexao = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
            )
            print("Conexão com o banco de dados bem sucedida!")
        except Error as erro:
            print(f"Ocorreu um erro ao se conectar ao banco de dados: {erro}")
            self.conexao = None

    def desconectar(self):
        if self.conexao:
            self.conexao.close()
            print("A conexão com o MySQL foi fechada!")

    def inserir(self, especialidade):
        cursor = self.conexao.cursor()
        sql = "INSERT INTO Especialidade (descricao) VALUES (%s)"
        cursor.execute(sql, (especialidade.descricao,))
        self.conexao.commit()
        print("%s registro inserido." % cursor.rowcount)
        cursor.close()
```

```

def listar(self):
    cursor = self.conexao.cursor()
    cursor.execute("SELECT id_especialidade, descricao FROM Especialidade")
    resultado = cursor.fetchall()
    cursor.close()

    especialidades = []
    for id_especialidade, descricao in resultado:
        especialidades.append(Especialidade(descricao, id_especialidade))
    return especialidades

def buscar_por_id(self, id_especialidade):
    cursor = self.conexao.cursor()
    cursor.execute(
        "SELECT id_especialidade, descricao FROM Especialidade WHERE
id_especialidade = %s",
        (id_especialidade,)
    )
    registro = cursor.fetchone()
    cursor.close()

    if registro:
        return Especialidade(registro[1], registro[0])
    return None

def atualizar(self, especialidade):
    cursor = self.conexao.cursor()
    sql = "UPDATE Especialidade SET descricao = %s WHERE id_especialidade =
%s"
    cursor.execute(sql, (especialidade.descricao, especialidade.id_especialidade))
    self.conexao.commit()
    print("%s registro atualizado." % cursor.rowcount)
    cursor.close()

def remover(self, id_especialidade):
    cursor = self.conexao.cursor()
    cursor.execute(
        "DELETE FROM Especialidade WHERE id_especialidade = %s",
        (id_especialidade,)
    )
    self.conexao.commit()
    print("%s registro removido." % cursor.rowcount)
    cursor.close()

def menu_adicionar(bd):
    descricao = input("Descrição da especialidade: ")
    nova = Especialidade(descricao)
    bd.inserir(nova)

```

```

def menu_listar(bd):
    for especialidade in bd.listar():
        print(especialidade)

def menu_buscar(bd):
    id_especialidade = input("ID da especialidade: ")
    especialidade = bd.buscar_por_id(id_especialidade)
    if especialidade:
        print(especialidade)
    else:
        print("Especialidade não encontrada.")
    return especialidade

def menu_atualizar(bd):
    especialidade = menu_buscar(bd)
    if especialidade:
        nova_descricao = input("Nova descrição [%s]: " % especialidade.descricao)
        especialidade.descricao = nova_descricao
        bd.atualizar(especialidade)

def menu_remover(bd):
    id_especialidade = input("ID da especialidade a remover: ")
    bd.remover(id_especialidade)

def menu_principal():
    bd = BancoDados()
    bd.conectar()

    if not bd.conexao:
        return

    while True:
        print("\nMENU DE CADASTRO DE ESPECIALIDADES")
        print("1 - Adicionar especialidade")
        print("2 - Listar todas as especialidades")
        print("3 - Buscar por uma especialidade")
        print("4 - Atualizar especialidade existente")
        print("5 - Remover especialidade")
        print("0 - Sair do programa")
        opcao = input("Digite uma opção: ")

        if opcao == "1":
            menu_adicionar(bd)
        elif opcao == "2":
            menu_listar(bd)
        elif opcao == "3":
            menu_buscar(bd)

```

```
elif opcao == "4":
    menu_atualizar(bd)
elif opcao == "5":
    menu_remover(bd)
elif opcao == "0":
    break
else:
    print("Opção inválida.")
```

```
bd.desconectar()
```

```
if __name__ == "__main__":
    menu_principal()
```