INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

CAMPUS CAMPINAS – IFSP

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ANA JÚLIA SESSO RAMALHO (CP3031861)

MILENA SOUZA BORGES SILVA (CP3029506)

PEDRO HENRIQUE PEREIRA DE ALMEIDA (CP3029352)

PROJETO DE EXTENSÃO: SISTEMA PARA CONTROLE DE VACINAÇÃO DE ANIMAIS (CVA)

Prof. Ms. Everton Meyer da Silva.

Disciplina: Banco de Dados II.

CAMPINAS - SP

2025

ANA JÚLIA SESSO RAMALHO (CP3031861)

MILENA SOUZA BORGES SILVA (CP3029506)

PEDRO HENRIQUE PEREIRA DE ALMEIDA (CP3029352)

PROJETO DE EXTENSÃO: SISTEMA PARA CONTROLE DE VACINAÇÃO DE ANIMAIS (CVA)

Prof. Ms. Everton Meyer da Silva.

Disciplina: Banco de Dados II.

CAMPINAS - SP

2025

SUMÁRIO:

1. INTRODUÇÃO:

O presente documento visa apresentar os resultados obtidos no desenvolvimento do sistema proposto pelo projeto de extensão intitulado “Sistema para Controle de Vacinação de Animais (CVA)”, elaborado no contexto do curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo – Campus Campinas (IFSP). Nesse sentido, a proposta foi orientada pelos conteúdos abordados na disciplina de Banco de Dados II, com foco na estruturação lógica, modelagem e implementação de um banco relacional funcional.

Tendo isso em vista, a equipe desenvolveu um sistema inicial voltado ao cadastro e controle vacinal de animais, destinado a pet shops e organizações não governamentais (ONG’s). Dessa forma, o sistema visa centralizar as informações dos animais e de seus respectivos atendimentos, oferecendo aos profissionais envolvidos na saúde animal, como veterinários e colaboradores dos estabelecimentos, uma ferramenta intuitiva e acessível para registro, consulta e gerenciamento das vacinas aplicadas.

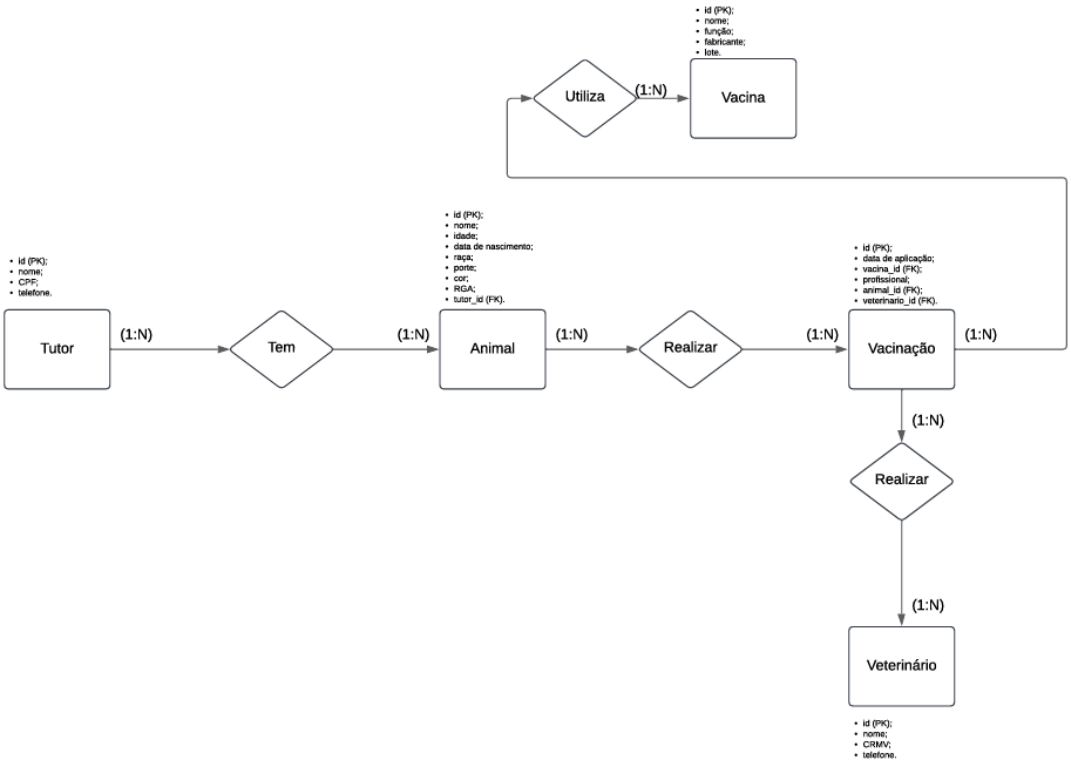
2. INTRODUÇÃO AO PROJETO:

2.1. SISTEMA PARA CONTROLE DE VACINAÇÃO DE ANIMAIS (CVA):

O presente projeto visa o desenvolvimento de um sistema inicial voltado para o cadastro de vacinação de animais, para pet shops e ONG's. A ideia e desenvolvimento faz parte do projeto de extensão, integrando disciplinas do curso de Tecnologia em Análise e Desenvolvimento de Sistemas pelo Instituto Federal de Educação, Ciência e Tecnologia de São Paulo Campus Campinas (IFSP). O sistema permite coletar informações do animal para que se tenha um lugar único e centrado para o veterinário ou profissional que irá gerenciar as vacinas. Oferecendo assim, informações claras e de fácil acesso, buscando a facilitação do gerenciamento de vacinas e a melhoria em relação ao atendimento aos usuários.

2.2. MODELAGEM DE DADOS:

Abaixo segue o diagrama de entidade-relacionamento (DER) do projeto desenvolvido pela equipe:



2.3. ESTRUTURA DO BANCO DE DADOS:

A estrutura do banco de dados se deu por meio da linguagem de programação MySQL. Nesse sentido, segue abaixo os scripts de criação com as principais funcionalidades das tabelas do sistema CVA:

```sql
CREATE TABLE tb_veterinario(
    id_veterinario INT NOT NULL AUTO_INCREMENT, -- Identificador único do veterinário
    nome VARCHAR(30), -- Nome do veterinário
    usuario VARCHAR(30) UNIQUE, -- Nome de usuário exclusivo para login
    senha VARCHAR(100), -- Senha criptografada
    PRIMARY KEY(id_veterinario) -- Chave primária
);
```

```sql
CREATE TABLE tb_tutor(
    id_tutor INT NOT NULL AUTO_INCREMENT, -- Identificador único do tutor
    nome VARCHAR(30), -- Nome do tutor
    email VARCHAR(40) UNIQUE, -- Email exclusivo para login/comunicação
    senha VARCHAR(255),
    telefone VARCHAR(15), -- Telefone de contato
    PRIMARY KEY (id_tutor) -- Chave primária
);
```

```sql
CREATE TABLE tb_animal(
    id_animal INT NOT NULL AUTO_INCREMENT, -- Identificador único do animal
    nome VARCHAR(30), -- Nome do animal
    especie VARCHAR(20), -- Espécie do animal (ex: cão, gato)
    PRIMARY KEY(id_animal) -- Chave primária
);
```

```sql
CREATE TABLE tb_vacina (
    id_vacina INT NOT NULL AUTO_INCREMENT, -- Identificador único da vacina
    nome VARCHAR(20) NOT NULL, -- Nome da vacina
    lote VARCHAR(50) NOT NULL, -- Número do lote da vacina
    validade DATE NOT NULL, -- Data de validade da vacina
    PRIMARY KEY (id_vacina) -- Chave primária
);
```

```sql
CREATE TABLE tb_funcionario(
    id_funcionario INT NOT NULL AUTO_INCREMENT, -- Identificador único do funcionário
    nome VARCHAR(50), -- Nome do funcionário
    email VARCHAR(50) UNIQUE, -- Email exclusivo
    senha VARCHAR(255), -- Senha de acesso
    cargo VARCHAR(50), -- Cargo ocupado (ex: recepcionista, técnico)
    PRIMARY KEY(id_funcionario) -- Chave primária
);
```

```sql
CREATE TABLE tb_ani_tut(
    id_animal INT NOT NULL, -- ID do animal
    id_tutor INT NOT NULL, -- ID do tutor
    FOREIGN KEY(id_animal) REFERENCES tb_animal(id_animal), -- Chave estrangeira para animal
    FOREIGN KEY(id_tutor) REFERENCES tb_tutor(id_tutor) -- Chave estrangeira para tutor
);
```

```sql
CREATE TABLE tb_agendamento(
    id_agendamento INT NOT NULL AUTO_INCREMENT, -- Identificador único do agendamento
    id_veterinario INT NOT NULL, -- ID do veterinário responsável
    id_animal INT NOT NULL, -- ID do animal atendido
    horario DATETIME NOT NULL, -- Data e hora do agendamento
    situacao VARCHAR(10), -- Situação do atendimento (ex: "realizado", "pendente")
    PRIMARY KEY(id_agendamento), -- Chave primária
    FOREIGN KEY(id_veterinario) REFERENCES tb_veterinario(id_veterinario), -- FK para veterinário
    FOREIGN KEY(id_animal) REFERENCES tb_animal(id_animal) -- FK para animal (corrigido aqui)
);
```

```sql
CREATE TABLE tb_vacinacao(
    id_vacinacao INT NOT NULL AUTO_INCREMENT, -- Identificador único da vacinação
    id_agendamento INT NOT NULL, -- ID do agendamento onde a vacina foi aplicada
    id_vacina INT NOT NULL, -- ID da vacina aplicada
    data_aplicacao DATE, -- Data de aplicação da vacina
    PRIMARY KEY(id_vacinacao), -- Chave primária
    FOREIGN KEY(id_agendamento) REFERENCES tb_agendamento(id_agendamento), -- FK para agendamento
    FOREIGN KEY(id_vacina) REFERENCES tb_vacina(id_vacina) -- FK para vacina
);
```

```sql
CREATE TABLE tb_historico(
    id_historico INT NOT NULL AUTO_INCREMENT, -- Identificador único do histórico
    id_vacinacao INT NOT NULL, -- ID da vacinação realizada
    id_animal INT NOT NULL, -- ID do animal vacinado
    PRIMARY KEY (id_historico), -- Chave primária
    FOREIGN KEY (id_vacinacao) REFERENCES tb_vacinacao (id_vacinacao), -- FK para vacinação
    FOREIGN KEY (id_animal) REFERENCES tb_animal (id_animal) -- FK para animal
);
```

```sql
CREATE TABLE tb_lancamento_vacina(
    id_lancamento INT NOT NULL AUTO_INCREMENT, -- Identificador do lançamento
    id_funcionario INT, -- ID do funcionário que fez o lançamento
    id_vacina INT, -- ID da vacina lançada
    PRIMARY KEY(id_lancamento), -- Chave primária
    FOREIGN KEY (id_funcionario) REFERENCES tb_funcionario(id_funcionario), -- FK para funcionário
    FOREIGN KEY (id_vacina) REFERENCES tb_vacina(id_vacina) -- FK para vacina
);
```

2.4. DESCRIÇÃO DAS TABELAS:

a) Tabela Veterinário: Armazena as informações relacionadas ao veterinário.

- id_veterinario: Identificador único do veterinário (chave primária).
- nome: Nome completo do veterinário.
- usuario: Nome de usuário exclusivo.
- senha: Senha criptografada para acesso.

b) Tabela Tutor: Armazena as informações referentes aos tutores dos animais.

- id_tutor: Identificador único do tutor (chave primária).
- nome: Nome do tutor.
- email: Email exclusivo.
- senha: Senha de acesso.
- telefone: Telefone de contato.

c) Tabela Animal: Armazena as informações relacionadas aos animais dos tutores.

- id_animal: Identificador do animal (chave primária).
- nome: Nome do animal.
- especie: Espécie (ex: cão, gato).

d) Tabela Vacina: Armazena as informações referentes as vacinas que serão aplicadas nos animais.

- id_vacina: Identificador da vacina (chave primária).

- nome: Nome da vacina.
- lote: Lote da vacina.
- validade: Data de validade.

e) **Tabela Funcionário:** Armazena as informações referentes aos funcionários da organização.

- id_funcionario: Identificador do funcionário (chave primária).
- nome: Nome completo.
- email: Email exclusivo.
- senha: Senha de acesso.
- cargo: Cargo ocupado.

f) **Tabela Animal com tutor:** Associa o ID de um tutor ao de um animal.

- id_animal: Identificador do animal (chave estrangeira).
- id_tutor: Identificador do tutor (chave estrangeira).

g) **Tabela Agendamento:** Armazena as informações relacionadas aos agendamentos gerados pelo veterinário.

- id_agendamento: Identificador do agendamento (chave primária).
- id_veterinario: Veterinário responsável (chave estrangeira).
- id_animal: Animal atendido (chave estrangeira).
- horario: Data e hora do agendamento.
- situacao: Situação do atendimento (pendente, realizado).

h) **Tabela Vacinação:** Armazena as informações referentes ao processo de vacinação.

- id_vacinacao: Identificador da vacinação (chave primária).
- id_agendamento: Agendamento relacionado (chave estrangeira).
- id_vacina: Vacina aplicada (chave estrangeira).
- data_aplicacao: Data da aplicação.

i) **Tabela Histórico:** Armazena as informações provenientes do histórico de vacinação dos animais.

- id_historico: Identificador do histórico (chave primária).
- id_vacinacao: Vacinação registrada (chave estrangeira).
- id_animal: Animal correspondente (chave estrangeira).

j) **Tabela Vacina:** Armazena as informações relacionadas as vacinas cadastradas no sistema.

- id_lancamento: Identificador do lançamento (chave primária).
- id_funcionario: Funcionário que lançou (chave estrangeira).
- id_vacina: Vacina lançada (chave estrangeira).

2.5. CONSULTA E FUNCIONALIDADES:

2.5.1. DESCRIÇÃO DAS CONSULTAS:

a) Através dessa consulta, pode-se conferir o histórico de vacinação de um animal específico.

```
SELECT a.nome AS animal, v.nome AS vacina, h.id_historico, vac.data_aplicacao
FROM tb_historico h
JOIN tb_vacinacao vac ON h.id_vacinacao = vac.id_vacinacao
JOIN tb_animal a ON h.id_animal = a.id_animal
JOIN tb_vacina v ON vac.id_vacina = v.id_vacina
WHERE a.nome = 'Nina';
```

b) Através dessa consulta, pode-se listar os agendamentos pendentes.

```sql
SELECT ag.id_agendamento, a.nome AS animal, ag.horario, ag.situacao
FROM tb_agendamento ag
JOIN tb_animal a ON ag.id_animal = a.id_animal
WHERE ag.situacao = 'pendente';
```

c) Através dessa consulta, pode-se verificar a quantidade de vacinas aplicadas por veterinário.

```sql
SELECT v.nome AS veterinario, COUNT(*) AS total_aplicadas
FROM tb_agendamento ag
JOIN tb_veterinario v ON ag.id_veterinario = v.id_veterinario
JOIN tb_vacinacao vac ON ag.id_agendamento = vac.id_agendamento
GROUP BY v.nome;
```

2.5.2. DESCRIÇÃO DAS FUNCIONALIDADES:

a) Através dessa funcionalidade, *view*, apresenta-se o histórico complete de vacinação de todos os animais com nome e datas.

```sql
CREATE VIEW vw_historico_vacinacao AS
SELECT
    a.nome AS animal,
    v.nome AS vacina,
    vac.data_aplicacao
FROM tb_historico h
JOIN tb_vacinacao vac ON h.id_vacinacao = vac.id_vacinacao
JOIN tb_animal a ON h.id_animal = a.id_animal
JOIN tb_vacina v ON vac.id_vacina = v.id_vacina;
```

b) Através dessa funcionalidade, *view*, apresenta-se o total de vacinas lançadas por cada funcionário, juntamente com seu nome e cargo, contribuindo para o controle de entrada de vacinas no sistema.

```sql
CREATE VIEW vw_vacinas_lancadas_funcionario AS
SELECT
    f.nome AS 'Nome do Funcionário',
    f.cargo AS 'Cargo',
    COUNT(lv.id_lancamento) AS 'Total de Lançamentos'
FROM tb_lancamento_vacina lv
JOIN tb_funcionario f ON lv.id_funcionario = f.id_funcionario
GROUP BY f.id_funcionario, f.nome, f.cargo;
SELECT * FROM vw_vacinas_lancadas_funcionario;
```

c) Através dessa funcionalidade, *view*, apresenta-se uma visão detalhada de cada agendamento, incluindo: nome do tutor, animal, espécie, veterinário responsável, data, horário e situação do atendimento. Facilitando assim, o acompanhamento completo dos agendamentos.

```sql
CREATE VIEW vw_agendamentos_detalhados AS
SELECT
    ag.id_agendamento AS 'Id do Agendamento',
    t.nome AS 'Tutor',
    a.nome AS 'Nome do Animal',
    a.especie AS 'Espécie do Animal',
    ag.horario AS 'Horário',
    v.nome AS 'Veterinário',
    ag.situacao AS 'Situação',
    t.email AS 'E-mail do Tutor',
    t.telefone AS 'Telefone do Tutor'
FROM tb_agendamento ag
JOIN tb_animal a ON ag.id_animal = a.id_animal
JOIN tb_ani_tut at ON a.id_animal = at.id_animal
JOIN tb_tutor t ON at.id_tutor = t.id_tutor
JOIN tb_veterinario v ON ag.id_veterinario = v.id_veterinario;
SELECT * FROM vw_agendamentos_detalhados;
```

d) Através dessa funcionalidade, *view*, apresenta-se somente os agendamentos que já foram realizados, permitindo assim, filtrar os atendimentos concluídos com informações completas do tutor, animal e profissional envolvido.

```
CREATE VIEW vw_agendamentos_realizados AS
    SELECT
    ag.id_agendamento AS 'Id do Agendamento',
    t.nome AS 'Tutor',
    a.nome AS 'Nome do Animal',
    a.especie AS 'Espécie do Animal',
    ag.horario AS 'Horário',
    v.nome AS 'Veterinário',
    t.email AS 'E-mail do Tutor',
    t.telefone AS 'Telefone do Tutor'
FROM tb_agendamento ag
JOIN tb_animal a ON ag.id_animal = a.id_animal
JOIN tb_ani_tut at ON a.id_animal = at.id_animal
JOIN tb_tutor t ON at.id_tutor = t.id_tutor
JOIN tb_veterinario v ON ag.id_veterinario = v.id_veterinario
WHERE ag.situacao = 'realizado';
SELECT * FROM vw_agendamentos_realizados;
```

e) Através dessa funcionalidade, view, apresenta-se apenas os agendamentos que ainda estão pendentes. Tornando prático, o gerenciamento das consultas e vacinas que ainda precisam ser realizadas.

```
CREATE VIEW vw_agendamentos_pendentes AS
    SELECT
    ag.id_agendamento AS 'Id do Agendamento',
    t.nome AS 'Tutor',
    a.nome AS 'Nome do Animal',
    a.especie AS 'Espécie do Animal',
    ag.horario AS 'Horário',
    v.nome AS 'Veterinário',
    t.email AS 'E-mail do Tutor',
    t.telefone AS 'Telefone do Tutor'
FROM tb_agendamento ag
JOIN tb_animal a ON ag.id_animal = a.id_animal
JOIN tb_ani_tut at ON a.id_animal = at.id_animal
JOIN tb_tutor t ON at.id_tutor = t.id_tutor
JOIN tb_veterinario v ON ag.id_veterinario = v.id_veterinario
WHERE ag.situacao = 'pendente';
SELECT * FROM vw_agendamentos_pendentes;
```

f) Através dessa funcionalidade, *view*, apresenta-se contabilização do total de vacinas aplicadas por cada veterinário, permitindo, desse modo, a análise do desempenho e carga de trabalho da equipe de atendimento.

```
CREATE VIEW vw_vacinas_aplicadas_por_veterinario AS
    SELECT
        v.nome AS 'Veterinário',
        COUNT(*) AS 'Total de Vacinas Aplicadas'
    FROM tb_agendamento ag
    JOIN tb_veterinario v ON ag.id_veterinario = v.id_veterinario
    JOIN tb_vacinacao vac ON ag.id_agendamento = vac.id_agendamento
    GROUP BY v.id_veterinario, v.nome;
SELECT * FROM vw_vacinas_aplicadas_por_veterinario;
```

g) Através dessa funcionalidade, *procedure*, o banco lista todas as vacinas aplicadas em um animal específico por meio de seu nome.

```
DELIMITER //
CREATE PROCEDURE sp_vacinas_por_animal(IN nome_animal VARCHAR(30))
BEGIN
    SELECT
        a.nome AS animal,
        v.nome AS vacina,
        vac.data_aplicacao
    FROM tb_animal a
    JOIN tb_agendamento ag ON ag.id_animal = a.id_animal
    JOIN tb_vacinacao vac ON ag.id_agendamento = vac.id_agendamento
    JOIN tb_vacina v ON vac.id_vacina = v.id_vacina
    WHERE a.nome = nome_animal;
END;
//
DELIMITER ;
```

h) Através dessa funcionalidade, *procedure*, o banco lista todas as vacinas aplicadas em um animal específico com base em seu ID, incluindo o nome do animal, a vacina administrada e a data de aplicação.

```
DELIMITER //
CREATE PROCEDURE historico_vacinas_animal(IN p_id_animal INT)
BEGIN
    SELECT
        a.nome AS 'Animal',
        v.nome AS 'Vacina',
        vac.data_aplicacao AS 'Data de Aplicação'
    FROM tb_animal a
    JOIN tb_agendamento ag ON ag.id_animal = a.id_animal
    JOIN tb_vacinacao vac ON ag.id_agendamento = vac.id_agendamento
    JOIN tb_vacina v ON vac.id_vacina = v.id_vacina
    WHERE a.id_animal = p_id_animal;
END;
//
```

i) Através dessa funcionalidade, *procedure*, o banco retorna todos os animais que pertencem a um tutor específico, permitindo consultar rapidamente a relação de animais sob responsabilidade de um determinado tutor.

```
DELIMITER ;
CALL historico_vacinas_animal(3);
DELIMITER //
-- seleciona todos animais de um tutor especifico que for coloacado no parametro
CREATE PROCEDURE listar_animais_por_tutor(IN id_tutor1 INT) -- insere como parâmetro o id_tutor que quero saber a lista de animais
BEGIN
    SELECT a.id_animal, a.nome, a.especie -- seleciona nome, especie e o id do animail
    FROM tb_animal a JOIN tb_ani_tut an ON a.id_animal = an.id_animal
    WHERE an.id_tutor = id_tutor1; -- porcura na o tabela tb_ani_tutor quais animais pertence ao tutor que foi inserido como parâmetro
END //
DELIMITER ;
CALL listar_animais_por_tutor(1);
```

j) Através dessa funcionalidade, *procedure*, o banco realiza o agendamento de um atendimento, inserindo os dados do veterinário, animal, data e situação no sistema de forma automática e segura.

```
DELIMITER //
-- cadastra um novo agendamento recebendo como parametro o  id_animal, id_veterinario, horario e situação
CREATE PROCEDURE agendar_atendimento( IN id_veterinario1 INT, IN id_animal1 INT, IN horario1 DATETIME, IN situacao1 VARCHAR(10))

BEGIN
INSERT INTO tb_agendamento ( id_veterinario, id_animal, horario, situacao) -- insere automaticamente o valores do parâmetro na tabela de agen
VALUES ( id_veterinario1, id_animal1, horario1, situacao1);

END //
DELIMITER ;

CALL agendar_atendimento(2, 5, '2025-07-01 10:30:00', 'pendente');
```

k) Através dessa funcionalidade, *function*, o banco verifica se um determinado e-mail já foi cadastrado na tabela de tutores, retornando verdadeiro caso exista e falso se ainda não estiver registrado, o que auxilia na validação de cadastros.

```
DELIMITER //
-- procura se um email já foi cadastrado
CREATE FUNCTION email_existe(email1 VARCHAR(40)) -- insere como parâmetro o email que quero pesquisar
RETURNS BOOLEAN -- returna TRUE caso já foi cadastrado e FALSE se não foi cadastrado
DETERMINISTIC
BEGIN
    DECLARE totalp INT; -- variavel que armazena o total de vezes que aparece o email

    SELECT COUNT(*) INTO totalp -- conta o total de vezes que aparece o email
    FROM tb_tutor
    WHERE email = email1; -- compara se o email da tabela tutor é igual ao email inserido no parâmetro da função

    IF totalp > 0 THEN -- se o email já foi cadastrado, vai retornar verdadeiro
    RETURN TRUE;
    ELSE
    RETURN FALSE; -- se o email  não foi cadastrado ainda , vai retornar falso
    END IF;
END //
DELIMITER ;
SELECT email_existe('eduardo.ramos3@email.com');
```

l) Através dessa funcionalidade, *trigger*, o banco atualiza automaticamente a tabela de histórico de vacinações, por meio do novo registro de vacinação inserido. Vinculando assim, o animal vacinado ao registro recém-criado, sem necessidade de inserção manual.

```
DELIMITER //
CREATE TRIGGER after_update_vacinacao_update_historico
AFTER INSERT ON tb_vacinacao
FOR EACH ROW
BEGIN

 DECLARE  id_ani INT;
  -- seleciona o id_animal para ser inserido na tabela de histórico
 SELECT id_animal INTO id_ani
 FROM tb_agendamento
 WHERE id_agendamento=new.agendamento;

 -- insere na tabela histórico

INSERT INTO tb_historico ( id_vacinacao, id_animal) VALUES (new.id_vacinacao, id_ani);

END //
DELIMITER ;
```

m) Através dessa funcionalidade, *trigger*, o banco verifica, antes da inserção de um novo registro de vacinação, se a vacina está dentro da validade. Caso esteja vencida, impede a aplicação e emite uma mensagem de erro, garantindo a segurança do processo..

```
DELIMITER //
CREATE TRIGGER before_insert_vacinacao
BEFORE INSERT ON tb_vacinacao
FOR EACH ROW
BEGIN

   DECLARE valid DATE;

    -- Porcura a validade da vacina
    SELECT validade INTO valid
    FROM tb_vacina
    WHERE id_vacina = NEW.id_vacina;

    -- Verifica se está vencida e aparece erro se estiver vencida
    IF NEW.data_aplicacao > valid THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Erro: Vacina vencida. Aplicação não permitida.';
    END IF;
END//

DELIMITER ;
```

n) Através dessa funcionalidade, *transaction*, o banco executa um conjunto de operações encadeadas para registrar a aplicação de uma vacina. Dessa maneira, o processo envolve a inserção na tabela de vacinação, a recuperação do animal relacionado e a inserção no histórico – entretanto, se ocorrer algum erro durante essas etapas, todas as ações podem ser desfeitas com o uso de "ROLLBACK", garantindo assim, a integridade e consistência dos dados.

```
START TRANSACTION;
INSERT INTO tb_vacinacao (id_agendamento, id_vacina, data_aplicacao)
VALUES (1, 2, '2025-07-02');

SET @id_animal := (
    SELECT id_animal
    FROM tb_agendamento
    WHERE id_agendamento = 1
);

INSERT INTO tb_historico (id_vacinacao, id_animal)
VALUES (LAST_INSERT_ID(), @id_animal);
COMMIT;
```

3. CONCLUSÃO:

A partir do que foi retratado, concluí-se que, o banco de dados desenvolvido em MySQL permitiu representar a estrutura fundamental do sistema de vacinação, possibilitando ao grupo exercitar a lógica de banco relacional e refletir sobre as diversas abordagens de armazenamento e manipulação de dados em plataformas reais. Entretanto, houveram desafios que implicaram no envolvimento do script para com o sistema web, visto que, sua implementação não foi possível por conta de divergências de linguagens – por conta do ambiente de desenvolvimento da ferramenta estabelecida *Firebase*. Logo, a equipe limitou-se a utilizá-lo somente como apoio teórico e prático, servindo como base para a modelagem e organização dos dados, mesmo não sendo integrado à aplicação final.

.