Modelagem e Implementação de um Banco de Dados para Eventos Universitários

Lucas Rodrigues
Universidade de Brasília (UnB)

Julho de 2025

Disciplina: Banco de Dados
Professor(a): Maristela Terto De Holanda

Sumário

1 Introdução

Este trabalho tem como objetivo demonstrar a aplicação dos conceitos fundamentais de modelagem de banco de dados por meio da elaboração de um Modelo Entidade-Relacionamento (MER), sua transposição para o Modelo Relacional (MR) e a implementação prática via script SQL. A proposta visa consolidar o entendimento teórico da disciplina, bem como a habilidade de projetar e estruturar bases de dados relacionais.

2 Modelo Entidade-Relacionamento (MER)

Disponível aqui.

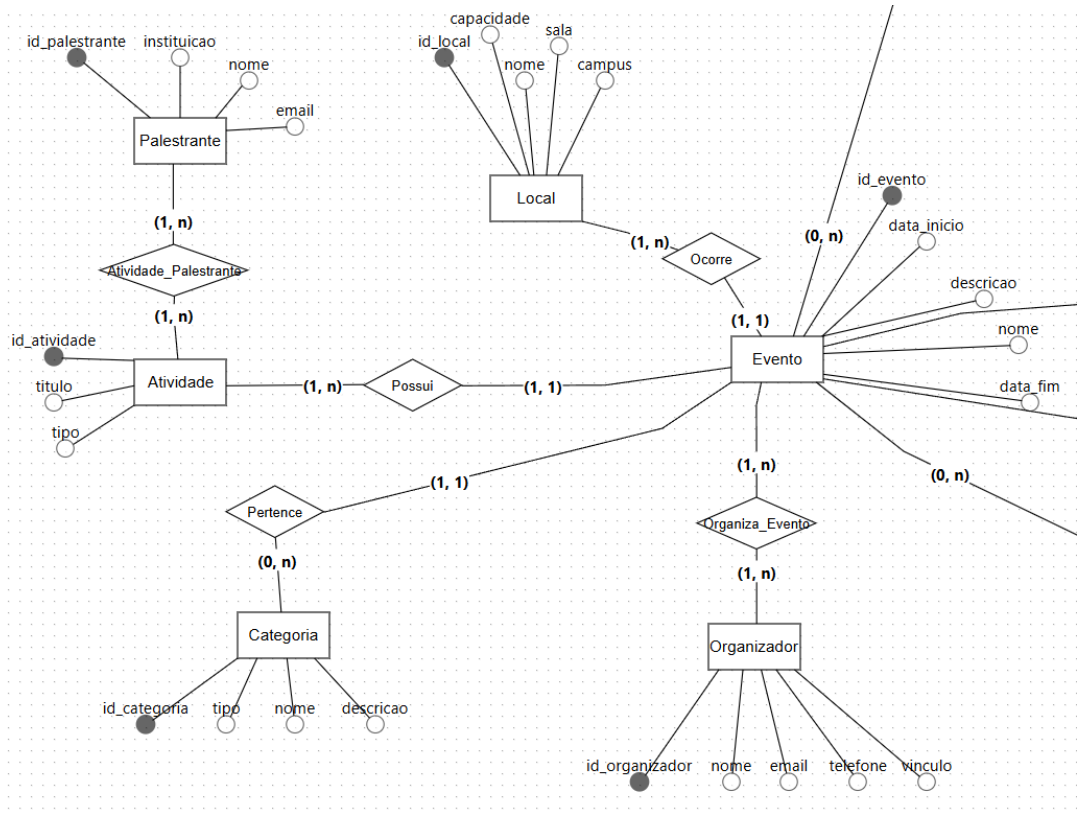

Figura 1: Parte 1 do Modelo Entidade-Relacionamento

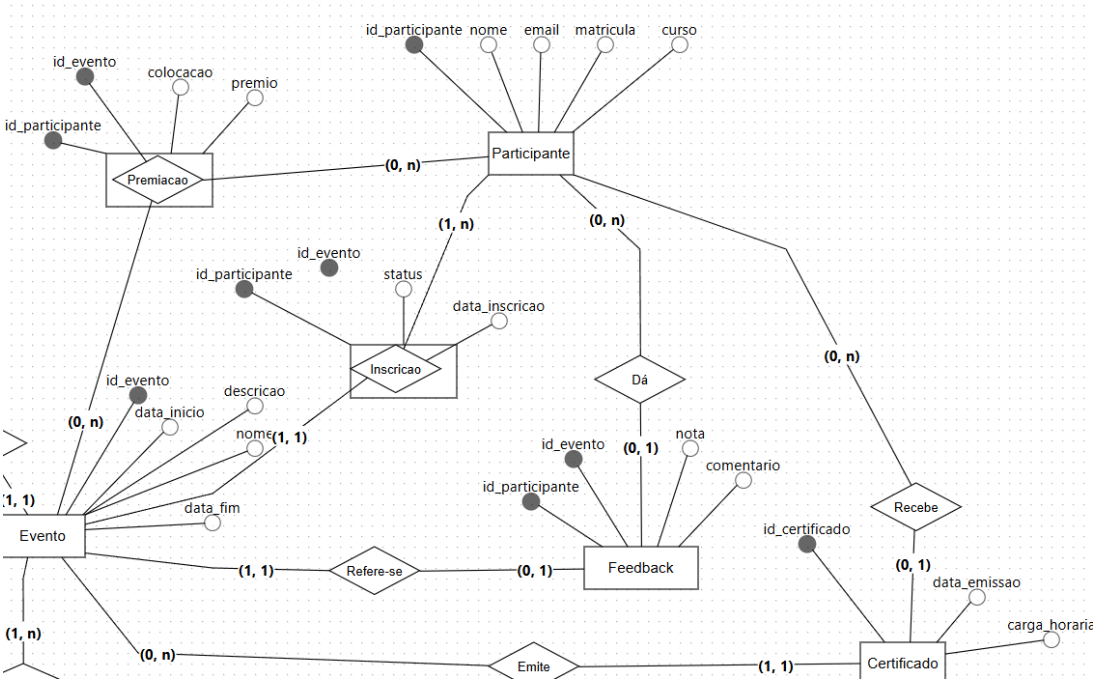

Figura 2: Parte 2 do Modelo Entidade-Relacionamento

3 Modelo Relacional (MR)

Disponível aqui.

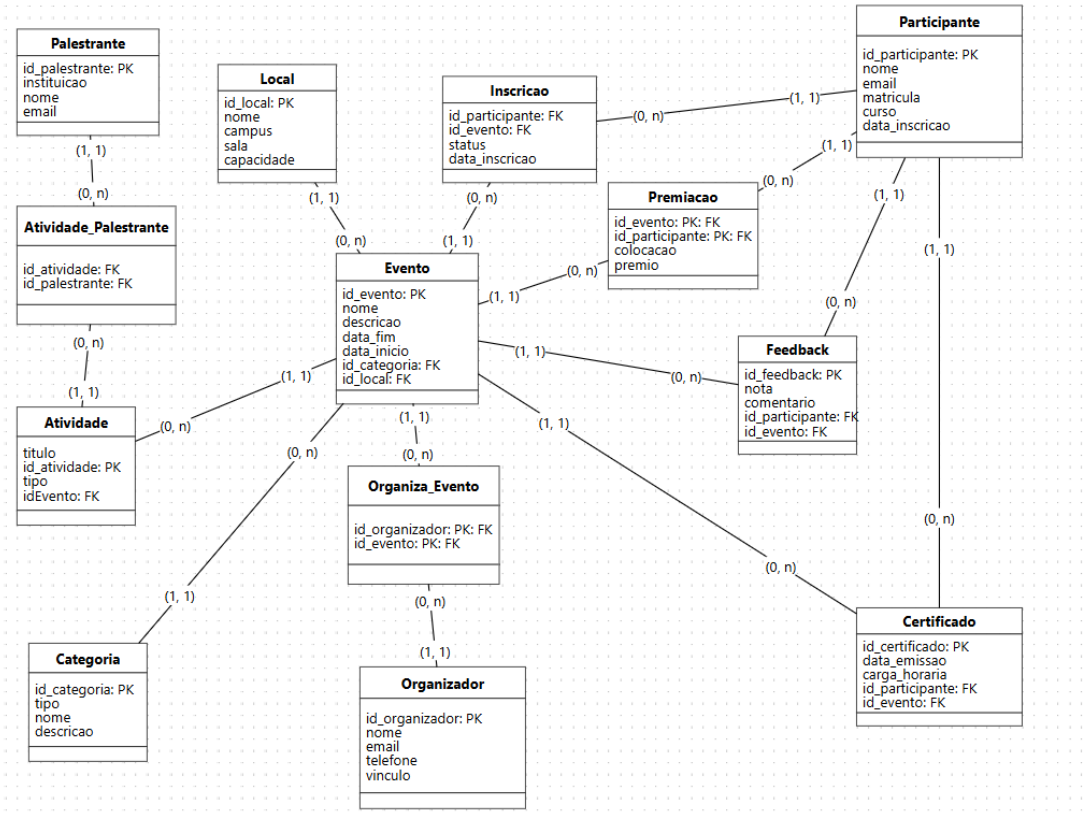

Figura 3: Modelo Relacional

4 Script SQL de Criação do Banco

Script completo disponível aqui.

```
CREATE TABLE Categoria (
    id_categoria INT PRIMARY KEY AUTO_INCREMENT,
    tipo VARCHAR(50) NOT NULL,
    nome VARCHAR(100) NOT NULL UNIQUE,
    descricao VARCHAR(255)
);

CREATE TABLE Local (
    id_local INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    campus VARCHAR(100) NOT NULL,
    sala VARCHAR(50) NOT NULL,
    capacidade INT NOT NULL
);

CREATE TABLE Evento (
    id_evento INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    descricao VARCHAR(255),
    data_inicio DATE NOT NULL,
    data_fim DATE NOT NULL,
    id_categoria INT NOT NULL,
    id_local INT NOT NULL,
    foto LONGBLOB,
    FOREIGN KEY (id_categoria) REFERENCES Categoria(id_categoria),
    FOREIGN KEY (id_local) REFERENCES Local(id_local)
);

CREATE TABLE Atividade (
    id_atividade INT PRIMARY KEY AUTO_INCREMENT,
    titulo VARCHAR(100) NOT NULL,
    tipo VARCHAR(50) NOT NULL,
    id_evento INT NOT NULL,
    FOREIGN KEY (id_evento) REFERENCES Evento(id_evento)
);

CREATE TABLE Organizador (
    id_organizador INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    telefone VARCHAR(20) NOT NULL,
    vinculo VARCHAR(50) NOT NULL
);

CREATE TABLE Participante (
```

```
    id_participante INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    matricula VARCHAR(20) NOT NULL UNIQUE,
    curso VARCHAR(100) NOT NULL,
    data_inscricao DATE NOT NULL
);

CREATE TABLE Palestrante (
    id_palestrante INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    instituicao VARCHAR(100) NOT NULL
);

CREATE TABLE Certificado (
    id_certificado INT PRIMARY KEY AUTO_INCREMENT,
    data_emissao DATE NOT NULL,
    carga_horaria FLOAT NOT NULL,
    id_participante INT NOT NULL,
    id_evento INT NOT NULL,
    FOREIGN KEY (id_participante) REFERENCES Participante(id_participante),
    FOREIGN KEY (id_evento) REFERENCES Evento(id_evento)
);

CREATE TABLE Feedback (
    id_feedback INT PRIMARY KEY AUTO_INCREMENT,
    nota INT NOT NULL CHECK (nota >= 0 AND nota <= 10),
    comentario VARCHAR(255),
    id_participante INT NOT NULL,
    id_evento INT NOT NULL,
    FOREIGN KEY (id_participante) REFERENCES Participante(id_participante),
    FOREIGN KEY (id_evento) REFERENCES Evento(id_evento)
);

CREATE TABLE Organiza_Evento (
    id_organizador INT NOT NULL,
    id_evento INT NOT NULL,
    PRIMARY KEY (id_organizador, id_evento),
    FOREIGN KEY (id_organizador) REFERENCES Organizador(id_organizador),
    FOREIGN KEY (id_evento) REFERENCES Evento(id_evento)
);

CREATE TABLE Premiacao (
    id_evento INT NOT NULL,
    id_participante INT NOT NULL,
    colocacao INT NOT NULL,
    premio FLOAT NOT NULL,
```

```
    PRIMARY KEY (id_evento, id_participante),
    FOREIGN KEY (id_evento) REFERENCES Evento(id_evento),
    FOREIGN KEY (id_participante) REFERENCES Participante(id_participante)
);

CREATE TABLE Inscricao (
    id_participante INT NOT NULL,
    id_evento INT NOT NULL,
    status VARCHAR(50) NOT NULL,
    data_inscricao DATE NOT NULL,
    PRIMARY KEY (id_participante, id_evento),
    FOREIGN KEY (id_participante) REFERENCES Participante(id_participante),
    FOREIGN KEY (id_evento) REFERENCES Evento(id_evento)
);

CREATE TABLE Atividade_Palestrante (
    id_atividade INT NOT NULL,
    id_palestrante INT NOT NULL,
    PRIMARY KEY (id_atividade, id_palestrante),
    FOREIGN KEY (id_atividade) REFERENCES Atividade(id_atividade),
    FOREIGN KEY (id_palestrante) REFERENCES Palestrante(id_palestrante)
);
```

5 Consultas em Álgebra Relacional

Neste item, apresentam-se cinco consultas escritas em álgebra relacional, cada uma envolvendo pelo menos três tabelas do banco de dados.

5.1 Consulta 1

Objetivo: Listar nome e data de eventos que acontecerão no próximo mês.

$$\pi_{\text{nome, data\_inicio, data\_fim}}\Big(\sigma_{\text{data\_inicio}\geq\text{CURRENT\_DATE} \,\wedge\, \text{data\_inicio}\leq\text{ADD\_MONTHS(CURRENT\_DATE, 1)}}(\text{Evento})\Big)$$

5.2 Consulta 2

Objetivo: Mostrar todos os eventos que ocorrerão no auditório principal.

$$\pi_{\text{Evento.nome, Evento.data\_inicio}}\Big(\text{Evento} \bowtie \sigma_{\text{nome='Auditório Principal'}}(\text{Local})\Big)$$

5.3 Consulta 3

Objetivo: Listar eventos que tiveram mais de 100 participantes inscritos.

$$\pi_{\text{Evento.nome, contagem}}\Big($$

2: Evento ⋈
3: $\gamma_{\text{id\_evento;COUNT(id\_participante)}\rightarrow\text{contagem}}(\text{Inscricao})$ ⋈
4: $\sigma_{\text{contagem}>100}$
5:)

5.4 Consulta 4

Objetivo: Listar todos os eventos com suas respectivas categorias e locais.

1: $\pi_{\text{Evento.nome, Categoria.nome}\rightarrow\text{categoria,Local.nome}\rightarrow\text{local}}($
2: Evento ⋈ Categoria ⋈ Local
3:)

5.5 Consulta 5

Objetivo: Mostrar o nome do organizador e os eventos que ele organiza.

1: $\pi_{\text{Organizador.nome, Evento.nome}}($
2: Organizador ⋈ Organiza_Evento ⋈ Evento
3:)

6 Avaliação das Formas Normais

Nesta seção, demonstramos como o projeto atende às três primeiras formas normais, usando como exemplo as tabelas **Evento**, **Categoria**, **Local**, **Atividade** e **Organizador**.

6.1 Tabela Única Inicial

Para ilustrar o processo de normalização, simulamos inicialmente uma única tabela unindo dados das cinco tabelas, conforme a Tabela 1:

Tabela 1: Tabela inicial desnormalizada

id_evento	nome_evento	titulo_atividade	nome_organizador	nome_categoria	nome_local
1	Evento XPTO	Palestra ABC	João Silva	Congresso	Auditório A
1	Evento XPTO	Palestra DEF	João Silva	Congresso	Auditório A
1	Evento XPTO	Palestra ABC	Maria Souza	Congresso	Auditório A

6.2 Primeira Forma Normal (1FN)

Na tabela acima, cada célula armazena apenas um valor atômico, portanto atende à 1FN. No entanto, há repetição de dados, o que indica problemas em formas superiores.

6.3 Segunda Forma Normal (2FN)

Problema: Atributos como nome_evento, nome_categoria e nome_local dependem apenas de parte da chave composta (id_evento, titulo_atividade, nome_organizador).

Solução: Criar tabelas separadas:

- Evento(id_evento, nome, descricao, data_inicio, data_fim, id_categoria, id_local)
- Categoria(id_categoria, nome)
- Local(id_local, nome, campus, sala, capacidade)
- Atividade(id_atividade, titulo, tipo, id_evento)
- Organizador(id_organizador, nome, email, telefone, vinculo)
- Organiza_Evento(id_organizador, id_evento)

6.4 Terceira Forma Normal (3FN)

Problema: Atributos como nome_categoria ou nome_local poderiam permanecer em Evento, causando dependências transitivas.

Solução: Remover esses dados da tabela Evento, deixando apenas as chaves estrangeiras:

```
CREATE TABLE Evento (
    id_evento INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100),
    descricao VARCHAR(255),
    data_inicio DATE,
    data_fim DATE,
    id_categoria INT,
    id_local INT,
    FOREIGN KEY (id_categoria) REFERENCES Categoria(id_categoria),
    FOREIGN KEY (id_local) REFERENCES Local(id_local)
);
```

Assim, cada atributo depende apenas da chave primária de sua tabela, sem dependências transitivas.

6.5 Conclusão

Após os ajustes, o banco encontra-se em **3FN**, garantindo redução de redundância e maior integridade dos dados.

7 View Implementada

Para a complexidade do trabalho, foi criada uma view que une participantes e eventos nos quais estão inscritos:

```
CREATE VIEW vw_resumo_eventos AS
SELECT
    e.id_evento,
    e.nome AS nome_evento,
    e.data_inicio,
    e.data_fim,
```

```
    c.nome AS categoria,
    l.campus,
    l.sala,
    COUNT(DISTINCT a.id_atividade) AS qtd_atividades,
    COUNT(DISTINCT i.id_participante) AS qtd_participantes
FROM Evento e
JOIN Categoria c ON e.id_categoria = c.id_categoria
JOIN Local l ON e.id_local = l.id_local
LEFT JOIN Atividade a ON e.id_evento = a.id_evento
LEFT JOIN Inscricao i ON e.id_evento = i.id_evento
GROUP BY
    e.id_evento,
    e.nome,
    e.data_inicio,
    e.data_fim,
    c.nome,
    l.campus,
    l.sala;
```

8 Procedure Implementada

Apresenta-se abaixo a procedure implementada, que premia automaticamente participantes de determinado evento caso tenham nota de feedback superior a 8.

```
DELIMITER $$

CREATE PROCEDURE sp_inserir_feedback(
    IN p_id_participante INT,
    IN p_id_evento INT,
    IN p_nota INT,
    IN p_comentario VARCHAR(255)
)
BEGIN
    -- Verifica se o participante está inscrito no evento
    IF NOT EXISTS (
        SELECT 1
        FROM Inscricao
        WHERE id_participante = p_id_participante
          AND id_evento = p_id_evento
    ) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Participante não inscrito neste evento.';
    ELSE
        -- Insere o feedback
        INSERT INTO Feedback (
            nota,
            comentario,
```

```
            id_participante,
            id_evento
        ) VALUES (
            p_nota,
            p_comentario,
            p_id_participante,
            p_id_evento
        );
    END IF;
END$$

DELIMITER ;
```

9 CRUD Implementado

O projeto implementa as funções básicas de CRUD para as tabelas **Evento**, **Atividade** e **Inscricao**. O código-fonte encontra-se no repositório:
https://github.com/lucasarod-br/pjdb

10 Diagrama da Camada de Persistência

Apresenta-se abaixo o diagrama que representa a interação entre a interface gráfica, a camada de persistência e o banco de dados:

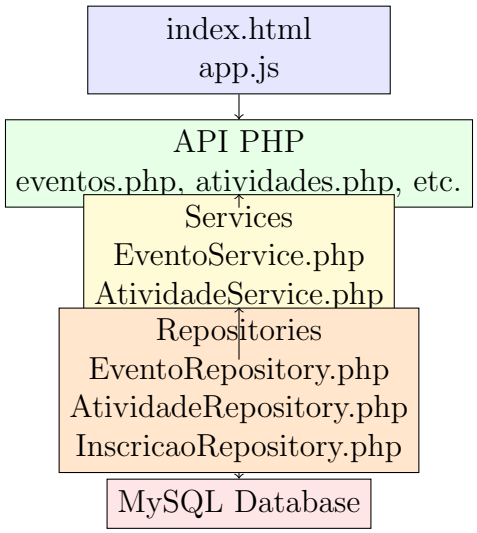

Figura 4: Diagrama da Arquitetura do Sistema CRUD

Figura 5: Diagrama da Camada de Persistência

11 Considerações Finais

Este trabalho permitiu consolidar o aprendizado dos conceitos teóricos da disciplina de Banco de Dados, incluindo modelagem, normalização, implementação prática em SQL, construção de views, procedures e desenvolvimento de um CRUD funcional. Os resultados obtidos demonstram a viabilidade do sistema proposto e seu potencial de aplicação real na gestão de eventos universitários.