

Universidade Federal da Paraíba Banco de Dados I

Projeto final - Cinema Sauro

Professor:

Marcelo Iury de Sousa Oliveira

Alunos:

Itamar de Paiva Rocha Filho	- 20180026510
João Pedro Vasconcelos Teixeira	- 20180028453
João Wallace Lucena Lins	- 20180027213

1. Tabelas

Tabelas propostas para a aplicação.

FILMES: Cada linha representa um filme transmitido no cinema

- id_filme PK SERIAL: ID do filme
- nome TEXT: Nome do filme
- categoria TEXT: Categoria do filme (e.g. Ação, Comédia, Drama)
- censura TEXT: Tipo de censura do filme (e.g. Livre, 14 anos, 16 anos, 18 anos)
- atores_principais TEXT: Texto com uma lista dos atores principais do filme (e.g. Tom Cruise, Tom Hanks, Tom Holland)
- duracao TIME: Texto com a duração do filme (e.g. 2h15min)
- produtora TEXT: Empresa Produtora do filme
- nacional BOOL: Indica se um filme é nacional ou não
- descricao TEXT: Descrição do filme

SALAS: Cada linha representa uma sala do cinema

- id_sala PK SERIAL (1 até 45): ID da sala (Total de salas = 45)
- capacidade INTEGER: Capacidade de pessoas em cada sala

SESSÕES: Cada linha representa uma sessão de filme

- id_sessao PK SERIAL: ID da sessão
- id_sala FK SERIAL: ID da sala onde ocorre a sessão
- id_filme FK SERIAL: ID do filme que vai passar na sessão
- total_vendido INTEGER: total de ingressos vendidos
- data DATE: data da sessão
- dia_da_semana TEXT: Dia da semana da sessão
- tempo_inicio TIMESTAMP: Horário de início da sessão
- tempo_final TIMESTAMP: Horário final da sessão
- valor_inteira NUMERIC: Valor de uma inteira nessa sessão

CLIENTES: Cada linha representa um cliente do cinema

- id_cliente PK SERIAL: ID do cliente
- nome TEXT: Nome do cliente
- CPF TEXT (OPTIONAL): CPF do cliente
- Telefone TEXT (OPTIONAL): Telefone do cliente

ITENS: Cada linha representa uma oferta na lanchonete do cinema

- id_item PK SERIAL: ID da oferta
- descricao TEXT: Descrição da oferta
- valor_item NUMERIC: descrição da oferta

PEDIDOS: Cada linha representa um pedido feito na lanchonete por um cliente no final da compra

- id_pedido SERIAL: ID da oferta comprada
- id_item FK SERIAL: ID da oferta
- id_compra FK SERIAL: ID da compra

INGRESSOS: Cada linha representa um ingresso comprado por um cliente

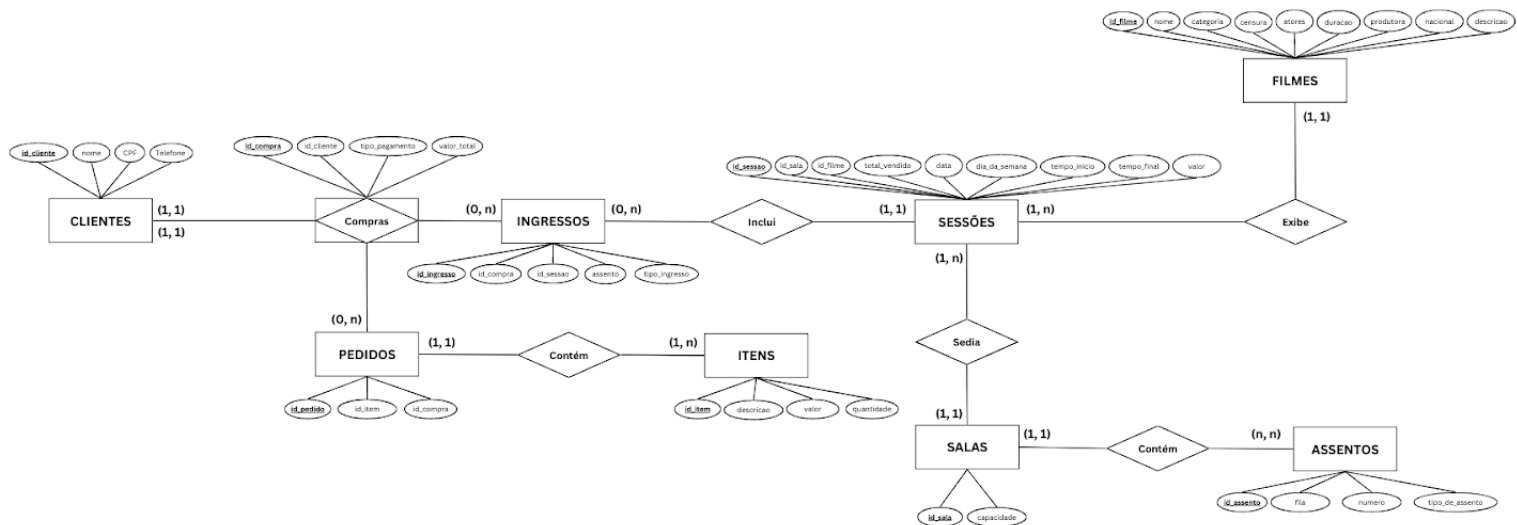
- id_ingresso PK SERIAL: ID do ingresso
- id_compra FK SERIAL: ID da compra
- id_sessao FK SERIAL: ID da sessão
- assento TEXT: Localizador do assento comprado no ingresso
- tipo_ingresso TEXT: Categoria do ingresso, que determina desconto no valor do ingresso (flamenguista: isento, infantil: 25%, estudante: meia, idoso: meia, adulto: inteira)

COMPRAS: Cada linha representa uma compra feita no cinema

- id_compra PK SERIAL: ID da compra
- id_cliente FK SERIAL: ID do cliente que fez a compra
- tipo_pagamento TEXT: Forma de pagamento utilizada
- valor_total NUMERIC: Valor total da compra

2. Diagrama E-R

O diagrama E-R desenvolvido para servir como modelo para esse trabalho pode ser encontrado no arquivo [do canva linkado aqui](#).



Os Clientes se relacionam com os Ingressos e Pedidos através da tabela Compras, e a relação se dá da seguinte maneira: um cliente realiza uma compra (ou **n** compras), esta que pode conter pedidos (que contém Itens da lanchonete) e ingressos, o id da compra é armazenada em ambas tabelas Ingressos e Pedidos. Os ingressos dizem respeito a uma sessão, cada uma que exibe um filme e é sediada em uma sala, e a mesma dispõe de vários assentos.

3. Normalização (3FN)

Quanto à primeira forma normal, nosso projeto a atende, mas com uma pequena ressalva: dependendo do ponto de vista, o campo atores_principais da tabela Filmes pode se tratar de um campo multivalorado, entretanto, como não é um ponto central de nossa aplicação, iremos considerá-lo apenas como um mero texto.

E como pode ser observado na seção 1 deste trabalho, nossas tabelas não possuem atributos com dependências transitivas com relação às suas chaves primárias, fazendo com que atendam à 3ª forma normal e, conseqüentemente, também à 2ª.

4. Queries

Nesta consulta, nós recuperamos os dados dos filmes e as sessões associadas a cada filme, incluindo dados da sala em que será exibida e dos ingressos já comprados para podermos fazer a visualização dos assentos disponíveis na sessão.

```
query get_filmes {  
  app_filmes {  
    sessoes {  
      id_sessao  
      data_sessao  
      dia_da_semana  
      tempo_inicio  
      valor_inteira  
      sala {  
        capacidade  
        id_sala  
      }  
      ingressos {  
        assento  
      }  
    }  
    nome  
    id_filme  
    censura  
    categoria  
  }  
}
```

Aqui, recuperamos todos os itens da lanchonete e seus respectivos valores.

```
query get_itens {  
  app_itens {  
    descricao  
    valor_item  
  }  
}
```

Nesta query, nós inserimos o novo cliente na tabela app_clientes, e caso já haja um cliente com o CPF cadastrado e seus outros dados estejam divergentes, atualizamos seus dados.

```
mutation upsert_clientes {  
  insert_app_clientes_one(object: {  
    {id_cliente: "{cpf}", nome: "{nome}", telefone: "{telefone}"},  
    on_conflict: {constraint: clientes_pkey, update_columns: [nome, telefone]}) {  
    id_cliente  
    nome  
    telefone  
  }  
}
```

A query abaixo faz a inserção da compra na tabela app_compras:

```
mutation insert_compra {
```

```

        insert_app_compras_one(object: {id_cliente: "{cpf}", tipo_pagamento:
"{forma_de_pagamento}", valor_total: valor}) {
    id_compra
    id_cliente
    tipo_pagamento
    valor_total
  }
}

```

Para o painel de gerenciamento, utilizamos esta query para recuperar os dados dos filmes que contém dados adicionais comparada à query "get_filmes", como os atores principais, a produtora e a descrição do filme:

```

query get_filmes_gerencia {
  app_filmes {
    id_filme
    nome
    atores_principais
    categoria
    censura
    duracao
    produtora
    nacional
    descricao
  }
}

```

Para a deleção de filmes, fazemos uso da seguinte query:

```

mutation delete_filme {
  delete_app_filmes_by_pk(id_filme: {id}) {
    id_filme
  }
}

```

Para o cadastro de filmes, fazemos uso da seguinte query:

```

mutation insert_filme {
  insert_app_filmes_one(object: {nome: "{nome}", categoria: "{categoria}",
censura: "{censura}", atores_principais: "{atores_principais}", duracao:
"{duracao}", produtora: "{produtora}", nacional: {nacional}, descricao:
"{descricao}") {
    nome
    produtora
    nacional
    id_filme
    duracao
    descricao
    censura
    categoria
    atores_principais
  }
}

```

Assim como a query 'get_filmes_gerencia', temos uma query com mais informações relacionadas à cada sessão, por exemplo o total de ingressos vendidos para aquela sessão

```

query get_sesoes {
  app_sesoes {
    id_sessao
    data_sessao
    dia_da_semana
    tempo_final
    tempo_inicio
    total_vendido
    valor_inteira
    filme {
      nome
      atores_principais
      categoria
      censura
      descricao
    }
    sala {
      capacidade
      id_sala
    }
  }
}

```

Para a deleção de sessões, temos:

```

mutation delete_sessao {
  delete_app_sesoes_by_pk(id_sessao: id) {
    id_sessao
  }
}

```

Para o cadastro de sessões, temos:

```

mutation insert_sessao {
  table(objects: {id_sala: id_sala, id_filme: id_filme, total_vendido:
total_vendido, data_sessao: "data_sessao", dia_da_semana: "dia_da_semana",
tempo_inicio: "tempo_inicio", tempo_final: "tempo_final", valor_inteira:
valor_inteira}) {
    returning {
      id_sala
      id_filme
      total_vendido
      dia_da_semana
      tempo_inicio
      tempo_final
      valor_inteira
    }
  }
}

```

5. Arquivos SQL

Nessa seção colocaremos os arquivos SQL de criação das tabelas e inserção de dados mockeados originais. Colocaremos uma quantidade reduzida de exemplos inseridos devido ao espaço.

filmes.sql

```
-- change database and user
\connect datalake datalakeuser

-- Cria tabela de filmes
CREATE TABLE app.filmes (
    id_filme serial PRIMARY KEY,
    nome text,
    categoria text,
    censura text,
    atores_principais text,
    duracao time,
    produtora text,
    nacional boolean,
    descricao text NULL
);

-- Mock
INSERT INTO app.filmes VALUES (
    1, 'Eurotrip', 'comedia', '16', 'Michael Cera, etc', '01:50:00',
    'MGM', false, 'Um filme muito alto astral sobre amigos viajando na europa'
);

INSERT INTO app.filmes VALUES (
    6, 'Racionais: Das Ruas de São Paulo pro Mundo', 'Documentário', 14, 'Mano Brown,
Ice Blue, Edi Rock, KL Jay', '01:56:00', 'NETFLIX', true, 'Gravado ao longo de trinta
anos, o documentário do Racionais MCs segue o grupo mais influente do rap nacional,
desde sua criação. Mostrando a origem e ascensão do grupo, o documentário traz
[...]'
);
```

salas.sql

```
-- change database and user
\connect datalake datalakeuser

-- Cria tabela de filmes
CREATE TABLE app.salas (
    id_sala serial PRIMARY KEY CHECK (id_sala < 46),
    capacidade integer
);

INSERT INTO app.salas VALUES (1, 64);
INSERT INTO app.salas VALUES (2, 96);
INSERT INTO app.salas VALUES (3, 128);
INSERT INTO app.salas VALUES (4, 160);
INSERT INTO app.salas VALUES (5, 192);
```


sessoes.sql

```
-- change database and user
\connect datalake datalakeuser

-- Cria tabela de sessoes
CREATE TABLE app.sessoes (
    id_sessao serial PRIMARY KEY,
    id_sala serial,
    id_filme serial,
    total_vendido integer,
    data_sessao date,
    dia_da_semana text,
    tempo_inicio time,
    tempo_final time,
    valor_inteira numeric,

    CONSTRAINT FK_sessoes_salas FOREIGN KEY(id_sala)
        REFERENCES salas(id_sala)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT FK_sessoes_filmes FOREIGN KEY(id_filme)
        REFERENCES filmes(id_filme)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

INSERT INTO app.sessoes VALUES (
    1, 1, 1, 1, '2022/12/06', 'Terça-feira', '18:00:00',
    '20:00:00', 34.90
);

INSERT INTO app.sessoes VALUES (
    2, 5, 6, 7, '2022/12/06', 'Terça-feira', '16:00:00',
    '18:00:00', 34.90
);

INSERT INTO app.sessoes VALUES (
    3, 2, 6, 0, '2022/12/06', 'Terça-feira', '20:00:00',
    '22:00:00', 34.90
);

INSERT INTO app.sessoes VALUES (
    4, 2, 6, 0, '2022/12/07', 'Quarta-feira', '20:00:00',
    '22:00:00', 34.90
);

INSERT INTO app.sessoes VALUES (
    5, 3, 3, 0, '2022/12/06', 'Terça-feira', '14:00:00',
    '16:00:00', 34.90
);

INSERT INTO app.sessoes VALUES (
    6, 1, 4, 0, '2022/12/06', 'Terça-feira', '16:00:00',
    '18:00:00', 34.90
);
```

```
);
INSERT INTO app.sesoes VALUES (
  7, 5, 5, 0, '2022/12/06', 'Quarta-feira', '16:00:00',
  '18:00:00', 34.90
);
INSERT INTO app.sesoes VALUES (
  8, 1, 2, 0, '2022/12/06', 'Terça-feira', '16:00:00',
  '18:00:00', 34.90
);
```

clientes.sql

```
-- change database and user
\connect datalake datalakeuser

-- Cria tabela de clientes
CREATE TABLE app.clientes (
  id_cliente varchar(14) PRIMARY KEY,
  nome text,
  telefone varchar(11)
);

INSERT INTO app.clientes VALUES (
  '999.333.222-92', 'Junior', '83988884444'
);

INSERT INTO app.clientes VALUES (
  '888.333.222-92', 'Aldemar', '83988884454'
);

INSERT INTO app.clientes VALUES (
  '777.333.222-92', 'Vladimir', '83988884455'
);
```

itens.sql

```
-- change database and user
\connect datalake datalakeuser

-- Cria tabela de itens
CREATE TABLE app.itens (
  id_item serial PRIMARY KEY,
  descricao text,
  valor_item numeric
);

INSERT INTO app.itens VALUES (
  1, 'Bolo Fubica', 45.99
);

INSERT INTO app.itens VALUES (
  2, 'Caldo de Cana', 5.99
);
```

```

INSERT INTO app.itens VALUES (
    3, 'Suco de Fruta', 7.99
);

INSERT INTO app.itens VALUES (
    4, 'Pipoca P', 7.99
);

INSERT INTO app.itens VALUES (
    5, 'Pipoca M', 10.99
);

INSERT INTO app.itens VALUES (
    6, 'Pipoca G', 12.99
);

INSERT INTO app.itens VALUES (
    7, 'Refri 2L', 15.99
);

```

compras.sql

```

-- change database and user
\connect datalake datalakeuser

-- Cria tabela de compras
CREATE TABLE app.compras (
    id_compra serial PRIMARY KEY,
    id_cliente varchar(14),
    tipo_pagamento text,
    valor_total numeric,

    CONSTRAINT FK_compras_clientes FOREIGN KEY(id_cliente)
        REFERENCES clientes(id_cliente)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

INSERT INTO app.compras VALUES (
    1, '999.333.222-92', 'PIX', 81.89
);

INSERT INTO app.compras VALUES (
    2, '777.333.222-92', 'CARTÃO', 223.83
);

INSERT INTO app.compras VALUES (
    3, '888.333.222-92', 'CARTÃO', 14.29
);

```

ingressos.sql

```
-- change database and user
\connect datalake datalakeuser

-- Cria tabela de ingressos
CREATE TABLE app.ingressos (
    id_ingresso serial PRIMARY KEY,
    id_compra serial,
    id_sessao serial,
    assento text,
    tipo_ingresso text,

    CONSTRAINT FK_ingressos_compras FOREIGN KEY(id_compra)
        REFERENCES compras(id_compra)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT FK_ingressos_sesoes FOREIGN KEY(id_sessao)
        REFERENCES sessoes(id_sessao)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

INSERT INTO app.ingressos VALUES (
    1, 1, 1, 'A7', 'Adulto'
);

INSERT INTO app.ingressos VALUES (
    2, 2, 2, 'E2', 'Adulto'
);

INSERT INTO app.ingressos VALUES (
    3, 2, 2, 'E3', 'Adulto'
);

INSERT INTO app.ingressos VALUES (
    4, 2, 2, 'E4', 'Adulto'
);

INSERT INTO app.ingressos VALUES (
    5, 2, 2, 'E5', 'Adulto'
);

INSERT INTO app.ingressos VALUES (
    6, 2, 2, 'E6', 'Adulto'
);

INSERT INTO app.ingressos VALUES (
    7, 3, 2, 'D7', 'Flamenguista'
);

INSERT INTO app.ingressos VALUES (
    8, 3, 2, 'D8', 'Flamenguista'
);
```

pedidos.sql

```
-- change database and user
\connect datalake datalakeuser

-- Cria tabela de pedidos
CREATE TABLE app.pedidos (
    id_pedido serial PRIMARY KEY,
    id_item serial,
    id_compra serial,

    CONSTRAINT FK_pedidos_itens FOREIGN KEY(id_item)
        REFERENCES itens(id_item)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT FK_pedidos_compras FOREIGN KEY(id_compra)
        REFERENCES compras(id_compra)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

INSERT INTO app.pedidos VALUES (
    1, 1, 1
);

INSERT INTO app.pedidos VALUES (
    2, 6, 3
);

INSERT INTO app.pedidos VALUES (
    3, 7, 3
);

INSERT INTO app.pedidos VALUES (
    4, 6, 2
);
```

6. Implementação do projeto

Nosso projeto foi dividido em duas partes diferentes, a composição do ambiente e do app. O ambiente consistiu na criação de um banco de dados a partir do uso do Docker, visando tornar nosso projeto replicável em qualquer dispositivo. Para o SGBD, nossa escolha foi o PostgreSQL, que foi utilizado juntamente ao Hasura para possibilitar uma visualização facilitada dos dados e a recuperação dos dados através de queries do GraphQL.

Já para a criação do app, utilizamos a biblioteca Streamlit, utilizando Python como linguagem de programação. Para fazermos as consultas ao banco de dados, utilizamos a biblioteca *requests* para conversar com o conteúdo armazenado, seja para escrita ou para leitura.

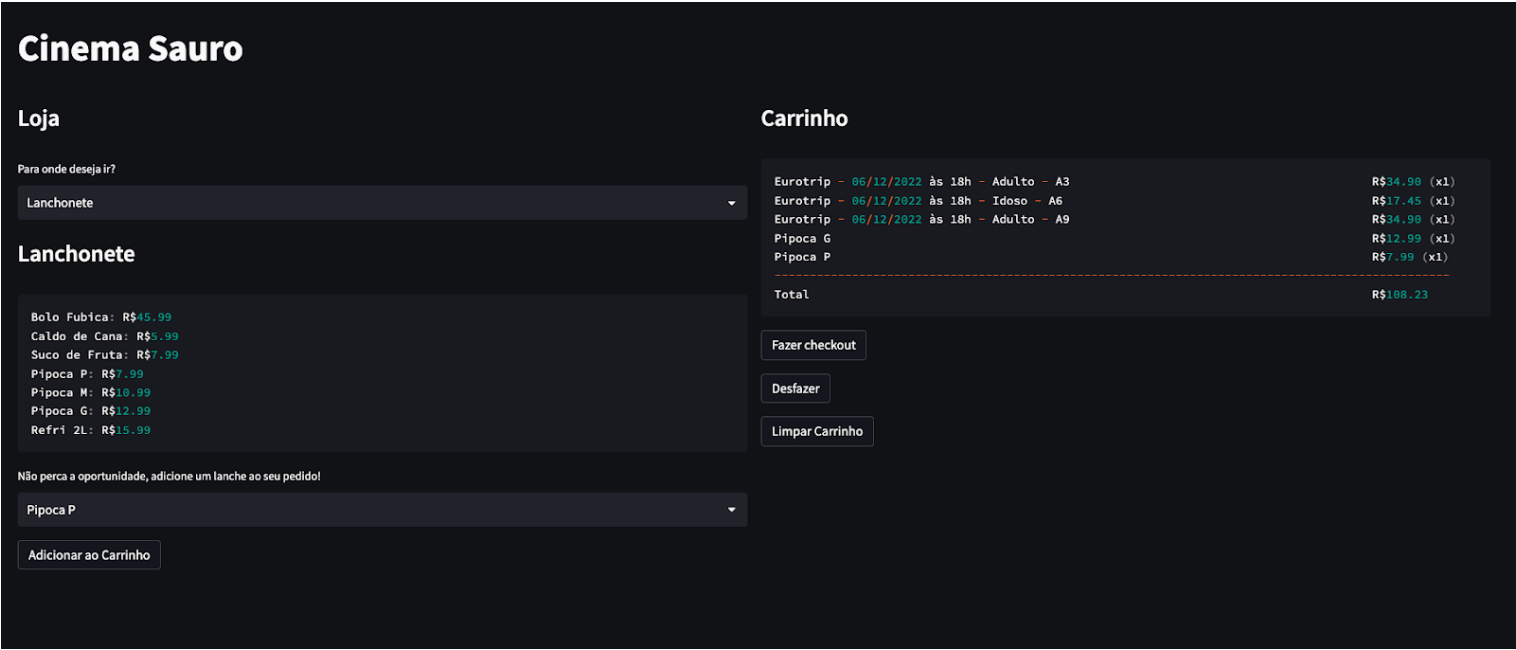


Imagem da loja: carrinho com ingressos e itens da lanchonete

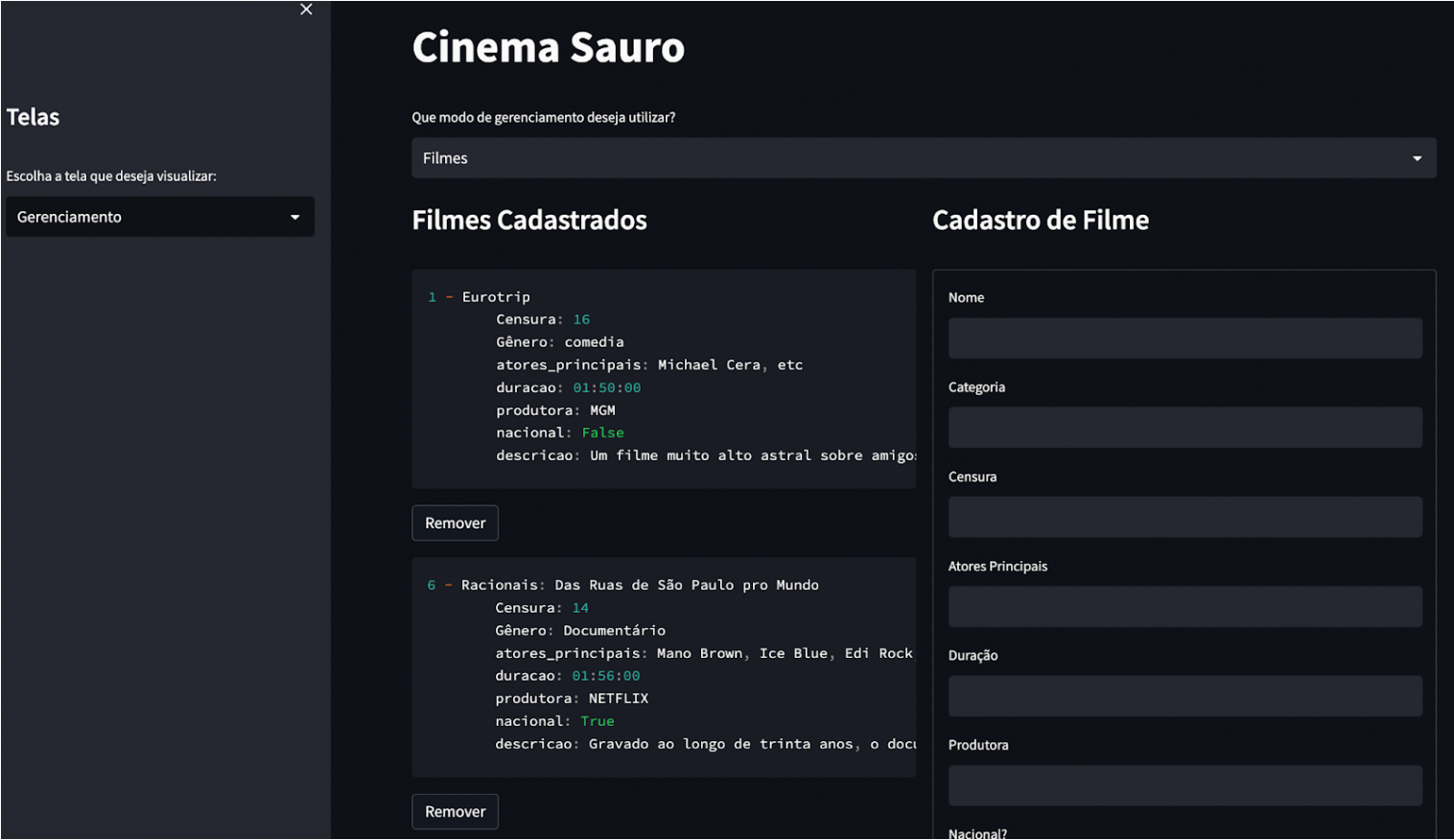


Imagem do gerenciamento: opções de deletar e inserir filmes e sessões