
Universidade Federal do Amazonas
Instituto de Computação
Engenharia de Software
Introdução a Banco de Dados

# SISTEMA DE STREAMING DE VÍDEOS

## (TRABALHO PRÁTICO FINAL)

**ESTUDANTES:**

Yago Lobato
Nathã Barbosa

yagobrlobato@icomp.ufam.edu.br
NathaBarbosa@icomp.ufam.edu.br

**PROFESSOR:**

[Moisés Carvalho]

# Sumário

# 1 Introdução

O setor de entretenimento digital tem vivenciado uma transformação significativa com a consolidação das plataformas de *streaming* de vídeo. Para sustentar a operação desses serviços em escala global, é fundamental a implementação de sistemas de gerenciamento de banco de dados (SGBD) robustos, capazes de armazenar e processar grandes volumes de informações relacionadas a catálogos, assinantes e métricas de consumo.

Neste contexto, o presente trabalho prático tem como objetivo projetar e implementar a camada de dados para uma nova plataforma de *streaming* voltada ao mercado global. O problema proposto exige a modelagem de um sistema capaz de gerir entidades complexas, incluindo o controle de assinaturas com histórico de alterações de planos, a gestão de catálogos de filmes com restrições de disponibilidade regional e o monitoramento detalhado das sessões de visualização dos usuários para fins analíticos.

O experimento realizado consiste na criação de um banco de dados relacional MySQL, estruturado para atender aos requisitos de negócio estipulados, tais como a capacidade de gerar relatórios sobre filmes mais assistidos por região e a análise de preferências dos usuários. Para validar a eficiência do modelo e das consultas SQL desenvolvidas, foi adotada uma abordagem de geração de dados sintéticos. Utilizando a linguagem Python e bibliotecas especializadas como `Faker`, foi desenvolvido um *script* de população automática, simulando um cenário realista de uso com dados massivos, sem comprometer a segurança de dados reais.

A infraestrutura do experimento foi isolada através da tecnologia de contêineres Docker, garantindo a reprodutibilidade do ambiente de testes. O relatório a seguir detalha a modelagem adotada, o processo de geração de dados e a análise das consultas obrigatórias para a validação dos requisitos do sistema.

# 2 Modelo Conceitual e Esquema Relacional

## 2.1 Análise de Requisitos

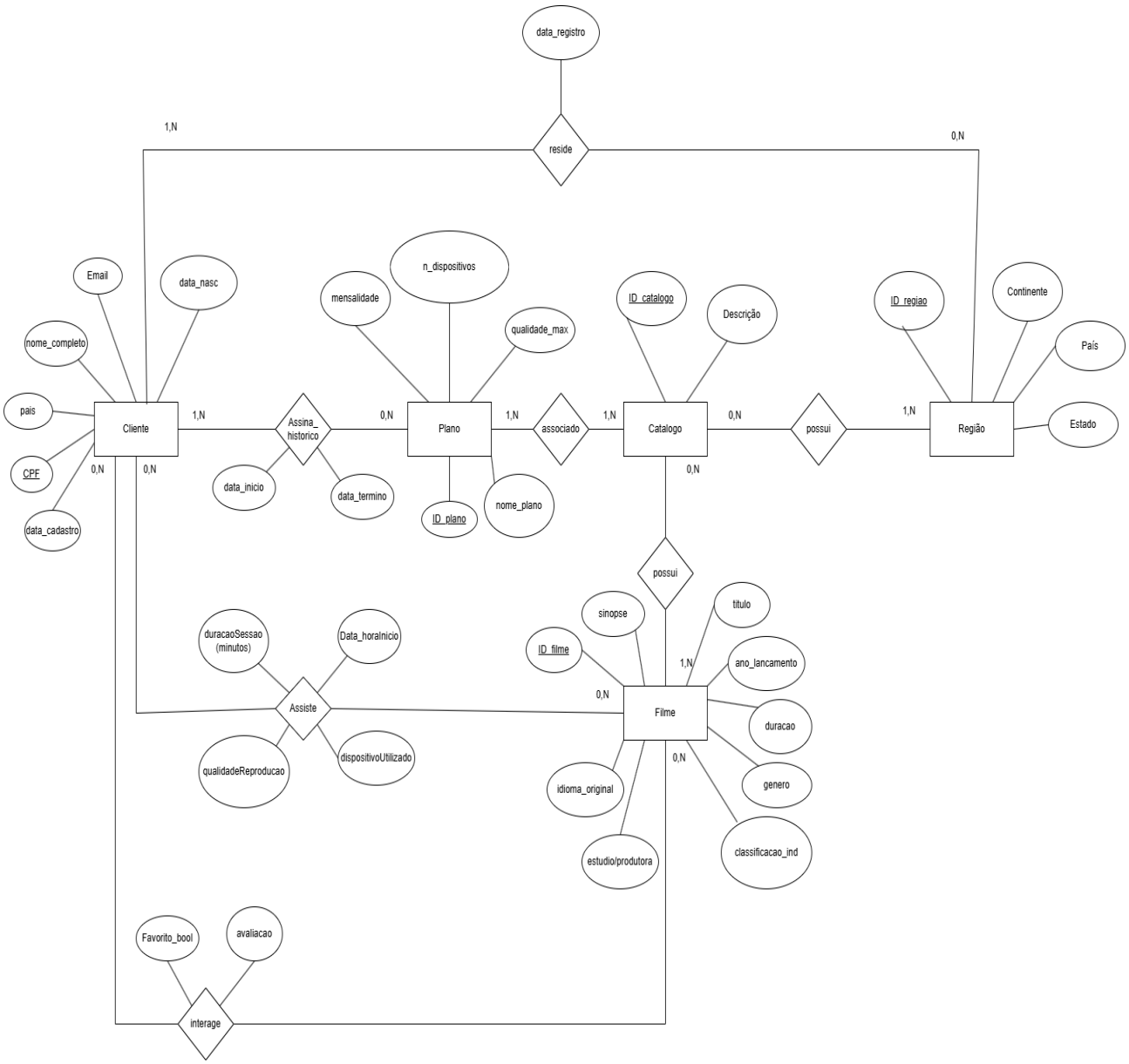
O projeto de banco de dados para a plataforma de *streaming* de vídeos requer uma estrutura capaz de suportar tanto operações transacionais (OLTP) quanto consultas analíticas (OLAP). O sistema deve gerenciar o ciclo de vida dos assinantes, a disponibilidade de conteúdo global e o monitoramento detalhado do consumo de mídia. Com base na descrição do problema, destacam-se as seguintes entidades e requisitos informacionais:

- **Cliente**: É necessário armazenar dados demográficos (como data de nascimento, país de origem e região residêncial) para segmentação, além da data de cadastro. O sistema deve manter o histórico de interações, incluindo filmes marcados como favoritos e avaliações atribuídas.
- **Plano de Assinatura**: Os planos variam em preço, qualidade de reprodução (SD, HD, 4K, etc) e limite de dispositivos simultâneos. Um requisito crítico é a manutenção do histórico de assinaturas, registrando as datas de início e fim de cada plano contratado pelo cliente para permitir auditoria de mudanças.
- **Filmes**: A entidade central do catálogo deve conter metadados técnicos e artísticos, incluindo título, ano, duração, classificação indicativa, estúdio e idioma original. Deve-se prever a categorização por gênero e o armazenamento de sinopses.
- **Disponibilidade Regional (Catálogo)**: O acervo não é uniforme globalmente. O banco de dados deve modelar a restrição de que um filme pode estar disponível em várias regiões, mas cada região possui seu próprio catálogo exclusivo.
- **Sessão de Visualização**: Para monitoramento de tráfego e geração de relatórios, é mandatório registrar cada visualização, contabilizando a data/hora de início, duração em minutos, dispositivo utilizado e a qualidade real da transmissão.

## 2.2 Modelo

Dadas as características que precisam ser atendidas para uma boa e eficiente implementação desse banco de dados, apresentamos abaixo o Diagrama Entidade-Relacionamento Estendido (EER) desenvolvido para a solução.

Figura 1 – Diagrama ER do Sistema




Fonte: Elaborado pelos autores.

## 2.3 Esquema Relacional

O mapeamento do modelo conceitual resultou nas seguintes tabelas normalizadas. As chaves primárias estão sublinhadas e as chaves estrangeiras indicadas com #.

- **Cliente** (CPF, nome_completo, email, data_nasc, pais, data_cadastro)
- **Plano** (ID_plano, nome_plano, mensalidade, n_dispositivos, qualidade_max)
- **Assina_Historico** (#CPF_cliente, #ID_plano, data_inicio, data_termino)

- **Catalogo** (ID_Catalogo, descrição)
- **Plano_Catalogo** (#ID_plano, #ID_Catalogo)
- **Regiao** (ID_regiao, Continente, País, Estado)
- **Regiao_residencia** (#CPF, #ID_regiao, data_registro)
- **Catalogo_Regional** (#ID_Catalogo, #ID_regiao)
- **Filme** (ID_filme, titulo, ano_lancamento, duracao, genero, classificacao_ind, produtora, idioma_original, sinopse)
- **Disponibilidade_Filme** (#ID_filme, #ID_Catalogo)
- **Sessao_Visualizacao** (#CPF_cliente, #ID_filme, data_hora_inicio, duracao_sessao, dispositivo_utilizado, qualidade_reproducao)
- **Preferencia_Cliente** (#CPF_cliente, #ID_filme, favorito_bool, avaliacao, data_interacao)

**Chaves Estrangeiras (Integridade Referencial):**

- Assina_Historico [#CPF_cliente] = Cliente [CPF]
- Assina_Historico [#ID_plano] = Plano [ID_plano]
- Plano_Catalogo [#ID_plano] = Plano [ID_plano]
- Plano_Catalogo [#ID_Catalogo] = Catalogo [ID_Catalogo]
- Catalogo_Regional [#ID_Catalogo] = Catalogo [ID_Catalogo]
- Catalogo_Regional [#ID_regiao] = Regiao [ID_regiao]
- Disponibilidade_Filme [#ID_filme] = Filme [ID_filme]
- Disponibilidade_Filme [#ID_Catalogo] = Catalogo [ID_Catalogo]
- Sessao_Visualizacao [#CPF_cliente] = Cliente [CPF]
- Sessao_Visualizacao [#ID_filme] = Filme [ID_filme]
- Preferencia_Cliente [#CPF_cliente] = Cliente [CPF]
- Preferencia_Cliente [#ID_filme] = Filme [ID_filme]

# 3 Scripts SQL (DDL)

Abaixo seguem os scripts para a criação da estrutura do banco de dados.

Listing 3.1 – Script DDL Completo - Criação das Tabelas

```sql
-- Desabilita verifica  o temporaria
SET FOREIGN_KEY_CHECKS = 0;

-- 1. Limpeza
DROP TABLE IF EXISTS Preferencia_Cliente;
DROP TABLE IF EXISTS Sessao_Visualizacao;
DROP TABLE IF EXISTS Disponibilidade_Filme;
DROP TABLE IF EXISTS Filme;
DROP TABLE IF EXISTS Catalogo_Regional;
DROP TABLE IF EXISTS Regiao_residencia;
DROP TABLE IF EXISTS Regiao;
DROP TABLE IF EXISTS Plano_Catalogo;
DROP TABLE IF EXISTS Catalogo;
DROP TABLE IF EXISTS Assina_Historico;
DROP TABLE IF EXISTS Plano;
DROP TABLE IF EXISTS Cliente;

-- 2. Tabelas Fortes
CREATE TABLE Cliente (
    CPF CHAR(11) PRIMARY KEY,
    nome_completo VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    data_nasc DATE NOT NULL,
    pais VARCHAR(50) NOT NULL,
    data_cadastro DATE NOT NULL
);

CREATE TABLE Plano (
    ID_plano INT AUTO_INCREMENT PRIMARY KEY,
    nome_plano VARCHAR(50) NOT NULL,
    mensalidade DECIMAL(10, 2) NOT NULL,
    n_dispositivos INT NOT NULL,
    qualidade_max VARCHAR(20) NOT NULL
);

CREATE TABLE Catalogo (
    ID_Catalogo INT AUTO_INCREMENT PRIMARY KEY,
    descricao VARCHAR(100) NOT NULL
);

```

```sql
CREATE TABLE Regiao (
    ID_regiao INT AUTO_INCREMENT PRIMARY KEY,
    Continente VARCHAR(50) NOT NULL,
    Pais VARCHAR(50) NOT NULL,
    Estado VARCHAR(50) NOT NULL
);

CREATE TABLE Filme (
    ID_filme INT AUTO_INCREMENT PRIMARY KEY,
    titulo VARCHAR(150) NOT NULL,
    ano_lancamento INT NOT NULL,
    duracao INT NOT NULL,
    genero VARCHAR(50),
    classificacao_ind VARCHAR(10),
    produtora VARCHAR(100),
    idioma_original VARCHAR(50),
    sinopse TEXT
);

-- 3. Tabelas Associativas
CREATE TABLE Assina_Historico (
    CPF_cliente CHAR(11),
    ID_plano INT,
    data_inicio DATE NOT NULL,
    data_termino DATE,
    PRIMARY KEY (CPF_cliente, ID_plano, data_inicio),
    FOREIGN KEY (CPF_cliente) REFERENCES Cliente(CPF),
    FOREIGN KEY (ID_plano) REFERENCES Plano(ID_plano)
);

CREATE TABLE Plano_Catalogo (
    ID_plano INT,
    ID_Catalogo INT,
    PRIMARY KEY (ID_plano, ID_Catalogo),
    FOREIGN KEY (ID_plano) REFERENCES Plano(ID_plano),
    FOREIGN KEY (ID_Catalogo) REFERENCES Catalogo(ID_Catalogo)
);

CREATE TABLE Regiao_residencia (
    CPF CHAR(11),
    ID_regiao INT,
    data_registro DATE NOT NULL,
    PRIMARY KEY (CPF, ID_regiao),
    FOREIGN KEY (CPF) REFERENCES Cliente(CPF),
    FOREIGN KEY (ID_regiao) REFERENCES Regiao(ID_regiao)
);

```

```sql
CREATE TABLE Catalogo_Regional (
    ID_Catalogo INT,
    ID_regiao INT,
    PRIMARY KEY (ID_Catalogo, ID_regiao),
    FOREIGN KEY (ID_Catalogo) REFERENCES Catalogo(ID_Catalogo),
    FOREIGN KEY (ID_regiao) REFERENCES Regiao(ID_regiao)
);

CREATE TABLE Disponibilidade_Filme (
    ID_filme INT,
    ID_Catalogo INT,
    PRIMARY KEY (ID_filme, ID_Catalogo),
    FOREIGN KEY (ID_filme) REFERENCES Filme(ID_filme),
    FOREIGN KEY (ID_Catalogo) REFERENCES Catalogo(ID_Catalogo)
);

CREATE TABLE Sessao_Visualizacao (
    CPF_cliente CHAR(11),
    ID_filme INT,
    data_hora_inicio DATETIME NOT NULL,
    duracao_sessao INT NOT NULL,
    dispositivo_utilizado VARCHAR(50),
    qualidade_reproducao VARCHAR(20),
    PRIMARY KEY (CPF_cliente, ID_filme, data_hora_inicio),
    FOREIGN KEY (CPF_cliente) REFERENCES Cliente(CPF),
    FOREIGN KEY (ID_filme) REFERENCES Filme(ID_filme)
);

CREATE TABLE Preferencia_Cliente (
    CPF_cliente CHAR(11),
    ID_filme INT,
    favorito_bool BOOLEAN DEFAULT FALSE,
    avaliacao INT CHECK (avaliacao BETWEEN 1 AND 5),
    data_interacao DATETIME DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (CPF_cliente, ID_filme),
    FOREIGN KEY (CPF_cliente) REFERENCES Cliente(CPF),
    FOREIGN KEY (ID_filme) REFERENCES Filme(ID_filme)
);

SET FOREIGN_KEY_CHECKS = 1;
```

# 4 Consultas Obrigatórias (DML)

Nesta seção, apresentamos as consultas SQL desenvolvidas. Para cada item, apresentamos a descrição em linguagem natural, a expressão em Álgebra Relacional e o respectivo código SQL.

## Consulta 1: Disponibilidade Regional

**Descrição:** Listar os títulos distintos de todos os filmes que estão disponíveis em catálogos associados a regiões cujo país é o "Y".
**Álgebra Relacional:**

$$\pi_{titulo}(\sigma_{Pais='Brasil'}(Filme \bowtie Disponib \bowtie Catalogo \bowtie Cat\_Reg \bowtie Regiao))$$

Listing 4.1 – Listar títulos disponíveis na região Brasil

```sql
SELECT DISTINCT f.titulo
FROM Filme f
JOIN Disponibilidade_Filme df ON f.ID_filme = df.ID_filme
JOIN Catalogo c ON df.ID_Catalogo = c.ID_Catalogo
JOIN Catalogo_Regional cr ON c.ID_Catalogo = cr.ID_Catalogo
JOIN Regiao r ON cr.ID_regiao = r.ID_regiao
WHERE r.Pais = 'Brasil';
```

## Consulta 2: Favoritos do Cliente

**Descrição:** Recuperar o título e a data de interação dos filmes que o cliente "Fulano de Tal"marcou como favoritos, ordenados do mais recente para o mais antigo.
**Álgebra Relacional:**

$$\pi_{titulo,data}(\sigma_{nome='Fulano' \wedge favorito=true}(Filme \bowtie Pref\_Cli \bowtie Cliente))$$

Listing 4.2 – Filmes favoritos do cliente específico

```sql
SELECT f.titulo, pc.data_interacao
FROM Filme f
JOIN Preferencia_Cliente pc ON f.ID_filme = pc.ID_filme
JOIN Cliente c ON pc.CPF_cliente = c.CPF
WHERE c.nome_completo = 'Fulano de Tal'
  AND pc.favorito_bool = TRUE
ORDER BY pc.data_interacao DESC;
```

## Consulta 3: Melhores Comédias

**Descrição:** Listar os títulos e a nota média dos filmes do gênero "Comédia"que possuem uma avaliação média igual ou superior a 4.
**Álgebra Relacional:**

$$\pi_{titulo,avg}(\sigma_{avg\geq 4}({}_{titulo}\Im_{AVG(aval)}(\sigma_{genero='Comedia'}(Filme \bowtie Pref\_Cli))))$$

Listing 4.3 – Filmes de Comédia com avaliação alta

```sql
SELECT f.titulo, AVG(pc.avaliacao) as media_nota
FROM Filme f
JOIN Preferencia_Cliente pc ON f.ID_filme = pc.ID_filme
WHERE f.genero = 'Comedia'
GROUP BY f.ID_filme, f.titulo
HAVING AVG(pc.avaliacao) >= 4;
```

## Consulta 4: Filmes sem Visualização (Diferença)

**Descrição:** Encontrar os títulos dos filmes que **não** tiveram nenhuma sessão de visualização registrada no período de 01/08/2025 a 31/08/2025.
**Álgebra Relacional:**

$$\pi_{titulo}(Filme) - \pi_{titulo}(Filme \bowtie \sigma_{01/08\leq data\leq 31/08}(Sessao))$$

Listing 4.4 – Filmes sem visualização num período

```sql
SELECT f.titulo
FROM Filme f
WHERE NOT EXISTS (
    SELECT 1
    FROM Sessao_Visualizacao s
    WHERE s.ID_filme = f.ID_filme
      AND s.data_hora_inicio BETWEEN '2025-08-01' AND '2025-08-31'
);
```

## Consulta 5: Relatório de Consumo por Filme

**Descrição:** Apresentar o título, a contagem total de visualizações e a soma de minutos assistidos para filmes disponíveis no Brasil durante Novembro de 2025.

**Álgebra Relacional:**

$$_{titulo}\Im_{COUNT(sessao),SUM(duracao)}(\sigma_{Pais='Brasil' \wedge data \in Nov}(Filme \bowtie ... \bowtie Sessao))$$

Listing 4.5 – Total de views e minutos no Brasil

```sql
SELECT
    f.titulo,
    -- Conta quantas sessoes existem para este filme no periodo
    COUNT(s.data_hora_inicio) AS total_visualizacoes,
    -- Soma a duracao das sessoes (se for NULL, retorna 0)
    COALESCE(SUM(s.duracao_sessao), 0) AS total_minutos
FROM Filme f

JOIN (
    SELECT DISTINCT df.ID_filme
    FROM Disponibilidade_Filme df
    JOIN Catalogo_Regional cr ON df.ID_Catalogo = cr.ID_Catalogo
    JOIN Regiao r ON cr.ID_regiao = r.ID_regiao
    WHERE r.Pais = 'Brasil' -- Troque 'Brasil' pela Regi o Y desejada
) filmes_na_regiao ON f.ID_filme = filmes_na_regiao.ID_filme
-- 2. Agora fazemos o LEFT JOIN limpo com as sess es filtradas por data
LEFT JOIN Sessao_Visualizacao s
    ON f.ID_filme = s.ID_filme
    AND s.data_hora_inicio >= '2025-11-01 00:00:00'
    AND s.data_hora_inicio <= '2025-11-30 23:59:59' -- Filtro do m s M/
    AAAA
GROUP BY f.ID_filme, f.titulo
ORDER BY total_visualizacoes DESC;
```

## Consulta 6: Consumo por Gênero

**Descrição:** Calcular o total de horas que o cliente "Fulano de Tal"passou assistindo conteúdo em 2025, agrupado por gênero.

**Álgebra Relacional:**

$$_{genero}\Im_{SUM(dur)/60}(\sigma_{nome='Fulano' \wedge ano=2025}(Sessao \bowtie Filme \bowtie Cliente))$$

Listing 4.6 – Total de horas assistidas por gênero

```sql
SELECT
    f.genero,
    -- Soma os minutos e divide por 60 para obter horas, arredondando para 2 casas decimais
    ROUND(SUM(s.duracao_sessao) / 60, 2) AS total_horas_assistidas
FROM Sessao_Visualizacao s
JOIN Filme f ON s.ID_filme = f.ID_filme
JOIN Cliente c ON s.CPF_cliente = c.CPF
WHERE c.nome_completo = 'Anne Chen' -- Substitua pelo nome do cliente "X"
  AND s.data_hora_inicio BETWEEN '2025-01-01 00:00:00' AND '2025-12-31 23:59:59' -- Per odo informado
GROUP BY f.genero
ORDER BY total_horas_assistidas DESC;
```

## Consulta 7: Métricas de Planos

**Descrição:** Listar os nomes dos planos oferecidos, a quantidade total de clientes atualmente ativos (aqueles cuja data de término na assinatura é nula) e a média do número máximo de dispositivos permitidos para cada plano.

**Álgebra Relacional:**

$$_{nome\_plano}\Im_{COUNT(cpf),AVG(n\_disp)}(\sigma_{dt\_termino=NULL}(Plano \bowtie Assina\_Historico))$$

Listing 4.7 – Clientes ativos e média de dispositivos

```sql
SELECT
    p.nome_plano,
    COUNT(ah.CPF_cliente) AS qtd_clientes_ativos,
    -- Como todos do mesmo plano t m o mesmo limite, a m dia    o
   pr prio limite.
    -- Usamos AVG ou MAX para garantir que o GROUP BY funcione
   corretamente.
    ROUND(AVG(p.n_dispositivos), 0) AS media_dispositivos_max
FROM Plano p
JOIN Assina_Historico ah ON p.ID_plano = ah.ID_plano
WHERE ah.data_termino IS NULL -- Filtra apenas assinaturas ativas (sem
   data de fim)
GROUP BY p.ID_plano, p.nome_plano
ORDER BY qtd_clientes_ativos DESC;
```

## Consulta 8: Ranking de Popularidade

**Descrição:** Identificar os 5 filmes com maior tempo total de exibição (soma da duração das sessões) assistidos por clientes que residem no Brasil, considerando apenas o último mês.

**Álgebra Relacional:**

$$\pi_{titulo,total}({}_{titulo}\Im_{SUM(duracao)}(\sigma_{Pais='Brasil'}(Sessao \bowtie Filme \bowtie Cliente \bowtie Regiao)))$$

Listing 4.8 – Top 5 filmes mais assistidos no Brasil

```sql
SELECT
    f.titulo,
    SUM(s.duracao_sessao) AS total_minutos_assistidos,
    COUNT(*) AS total_sessoes
FROM Sessao_Visualizacao s
JOIN Filme f ON s.ID_filme = f.ID_filme
JOIN Cliente c ON s.CPF_cliente = c.CPF
JOIN Regiao_residencia rr ON c.CPF = rr.CPF
JOIN Regiao r ON rr.ID_regiao = r.ID_regiao
WHERE r.Pais = 'Brasil'
  AND s.data_hora_inicio >= DATE_SUB(NOW(), INTERVAL 1 MONTH)
GROUP BY f.ID_filme, f.titulo
ORDER BY total_minutos_assistidos DESC, total_sessoes DESC
LIMIT 5;
```

## Consulta 9: Análise de Qualidade de Streaming

**Descrição:** Calcular a distribuição (contagem e percentual) das diferentes qualidades de reprodução (como '4K', 'HD') utilizadas nas sessões de visualização do cliente "Fulano de Tal".

**Álgebra Relacional:**

$$_{qualidade}\Im_{COUNT(sessao)}(\sigma_{nome='Fulano'}(Sessao \bowtie Cliente))$$

Listing 4.9 – Distribuição de qualidade de reprodução

```sql
SELECT
    s.qualidade_reproducao,
    COUNT(*) AS qtd_sessoes,
    (COUNT(*) * 100.0 / (
        SELECT COUNT(*)
        FROM Sessao_Visualizacao s2
        JOIN Cliente c2 ON s2.CPF_cliente = c2.CPF
        WHERE c2.nome_completo = 'Fulano de Tal'
    )) AS percentual
FROM Sessao_Visualizacao s
JOIN Cliente c ON s.CPF_cliente = c.CPF
WHERE c.nome_completo = 'Fulano de Tal'
GROUP BY s.qualidade_reproducao;
```

## Consulta 10: Migração de Planos (Churn/Upgrade)

**Descrição:** Listar os clientes que alteraram seu plano de assinatura nos últimos 3 meses, exibindo o nome do cliente, o plano anterior, o plano novo e as datas relevantes.
**Álgebra Relacional:**

$$\pi_{nome,planos}(\sigma_{data \geq 3meses}(Assina\_Hist_{antigo} \bowtie_{CPF} Assina\_Hist_{novo}))$$

Listing 4.10 – Clientes que mudaram de plano

```sql
SELECT
    c.nome_completo,
    p_antigo.nome_plano AS plano_anterior,
    p_novo.nome_plano AS plano_novo,
    antigo.data_inicio AS inicio_anterior,
    antigo.data_termino AS termino_anterior,
    novo.data_inicio AS inicio_novo
FROM Assina_Historico antigo
JOIN Assina_Historico novo
    ON antigo.CPF_cliente = novo.CPF_cliente
    AND antigo.data_termino = novo.data_inicio
JOIN Cliente c ON antigo.CPF_cliente = c.CPF
JOIN Plano p_antigo ON antigo.ID_plano = p_antigo.ID_plano
JOIN Plano p_novo ON novo.ID_plano = p_novo.ID_plano
WHERE antigo.data_termino >= DATE_SUB(NOW(), INTERVAL 3 MONTH)
  AND p_antigo.ID_plano <> p_novo.ID_plano;
```

# 5 Populando o BD com Dados Sintéticos

## 5.1 Script de População do Banco de Dados

O script tem como objetivo automatizar a geração de dados sintéticos e inseri-los no banco de dados MySQL. Ele foi desenvolvido utilizando bibliotecas como `Faker`, `TQDM`, `mysql-connector-python` e `dotenv`, permitindo realismo na geração de dados, controle modular e execução otimizada em lotes.

### 5.1.1 Estrutura Geral do Script

O script é organizado em três blocos principais:

- **Configuração e Setup**
- **Funções Auxiliares**
- **Funções de População**

Ao final, uma função `main()` controla o fluxo de execução, garantindo atomicidade via transações, tratamento de erros e validações antes de inserir dados massivos.

### 5.1.2 Configuração e Carregamento de Dependências

Primeiramente, o script importa bibliotecas responsáveis pela conexão com o banco, geração de dados e gerenciamento do ambiente. Uma estrutura de dicionário chamada `DB_CONFIG` define os parâmetros do MySQL, suportando variáveis via `dotenv`:

```
DB_CONFIG = {
    'host': os.getenv('DB_HOST', 'localhost'),
    'database': os.getenv('DB_NAME', 'meu_banco_trabalho'),
    'user': os.getenv('DB_USER', 'aluno'),
    'password': os.getenv('DB_PASSWORD', 'senha_segura'),
    'port': int(os.getenv('DB_PORT', 3306))
}
```

O objeto `Faker` é configurado com locale brasileiro para padronização inicial.

A estratégia de povoamento do banco de dados foi implementada em Python, utilizando uma arquitetura modular que separa a infraestrutura de conexão, a geração de dados sintéticos e a orquestração de transações. O script organiza-se nas seguintes camadas funcionais:

#### 5.1.2.1 Infraestrutura e Utilitários

As funções auxiliares garantem a comunicação com o SGBD e a integridade dos dados gerados, abstraindo complexidades repetitivas:

**`criar_conexao()`:** Estabelece a interface com o banco MySQL utilizando parâmetros seguros carregados via variáveis de ambiente.

```
def criar_conexao():
    try:
        conn = mysql.connector.connect(**DB_CONFIG)
        if conn.is_connected():
            return conn
    except Error as e:
        print(f"Erro de conexão: {e}")
    return None
```

**`resetar_banco()`:** Automatiza a reconstrução do esquema relacional (DDL), garantindo um ambiente limpo a cada execução.

```
def resetar_banco(cursor):
    print("Recriando estrutura do banco...")
    nome_arquivo = 'BD_schema.sql'

    if not os.path.exists(nome_arquivo):
        print(f"ERRO CRÍTICO: O arquivo '{nome_arquivo}' não foi
        ↪  encontrado!")
        exit()

    with open(nome_arquivo, 'r', encoding='utf-8') as f:
        sql_file = f.read()
        commands = sql_file.split(';')
        for command in tqdm(commands, desc="Executando DDL"):
            if command.strip():
                try:
                    cursor.execute(command)
                except Error as e:
                    print(f"Erro SQL ignorado: {e}")
```

**`inserir_em_lotes()`:** Otimiza a performance de escrita realizando *bulk inserts*, essencial para grandes volumes de dados, com feedback visual de progresso.

```
def inserir_em_lotes(cursor, sql, dados, tamanho_lote=5000,
↪  descricao="Inserindo"):
    """Função para inserir grandes quantidades de dados em pedaços
    ↪  menores com barra de progresso"""
    total = len(dados)

    for i in tqdm(range(0, total, tamanho_lote), desc=f"{descricao}",
    ↪  leave=False):
        lote = dados[i:i + tamanho_lote]
        cursor.executemany(sql, lote)
```

**`gerar_cpf()` e `escolher_idioma()`:** Funções de suporte à biblioteca *Faker* para garantir unicidade em identificadores e distribuição probabilística realista de atributos culturais.

```
def gerar_cpf():
    """Gera um CPF fictício de 11 dígitos"""
    return str(fake.unique.random_number(digits=11, fix_len=True))
```

```
def escolher_idioma():
    IDIOMAS = {
        "Inglês": 0.45,
        "Espanhol": 0.10,
        "Português": 0.10,
        "Francês": 0.05,
        "Alemão": 0.04,
        "Italiano": 0.04,
        "Japonês": 0.05,
        "Coreano": 0.05,
        "Chinês (Mandarim)": 0.05,
        "Hindi": 0.03
    }

    r = random.random()
    acumulado = 0
```

```
    for idioma, peso in IDIOMAS.items():
        acumulado += peso
        if r <= acumulado:
            return idioma
    return "Inglês"
```

#### 5.1.2.2 Geração de Entidades e Relacionamentos

O processo de população respeita a ordem de dependência das chaves estrangeiras, dividindo-se em dados estáticos (domínio) e dados dinâmicos (volumétricos):

1. **Dados de Referência:** As funções `popular_planos()`, `popular_regioes()` e `popular_catalogos()` inicializam as tabelas de domínio fixo, essenciais para a validação das regras de negócio.

```
def popular_planos(cursor):
    print("Inserindo Planos...")
    planos = [
        ("Mobile", 19.90, 1, "480p"),
        ("Básico", 29.90, 2, "1080p"),
        ("Padrão", 45.90, 3, "4K"),
        ("Premium", 59.90, 4, "4K+HDR")
    ]
    cursor.executemany("INSERT INTO Plano (nome_plano, mensalidade,
    ↪  n_dispositivos, qualidade_max) VALUES (%s, %s, %s, %s)", planos)
```

```
def popular_regioes(cursor):
    print("Inserindo Regiões...")

    regioes = [
        # América do Sul
        ("América do Sul", "Brasil", "São Paulo"),
        ("América do Sul", "Brasil", "Amazonas"),
        ("América do Sul", "Brasil", "Bahia"),
        ("América do Sul", "Argentina", "Buenos Aires"),
        ("América do Sul", "Chile", "Santiago"),
        ("América do Sul", "Colômbia", "Bogotá"),
        # América do Norte
        ("América do Norte", "Estados Unidos", "California"),
        ("América do Norte", "Estados Unidos", "Texas"),
        ("América do Norte", "Estados Unidos", "Nova York"),
```

```
        ("América do Norte", "Canadá", "Ontario"),
        ("América do Norte", "Canadá", "Quebec"),
        ("América do Norte", "México", "Jalisco"),
        # Europa
        ("Europa", "França", "Île-de-France"),
        ("Europa", "Alemanha", "Baviera"),
        ("Europa", "Reino Unido", "Inglaterra"),
        ("Europa", "Portugal", "Lisboa"),
        ("Europa", "Espanha", "Catalunha"),
        ("Europa", "Itália", "Lombardia"),
        # Ásia
        ("Ásia", "Japão", "Tóquio"),
        ("Ásia", "China", "Pequim"),
        ("Ásia", "Índia", "Maharashtra"),
        ("Ásia", "Coreia do Sul", "Seul"),
        ("Ásia", "Arábia Saudita", "Riyadh"),
        ("Ásia", "Indonésia", "Java Ocidental"),
        # África
        ("África", "Nigéria", "Lagos"),
        ("África", "África do Sul", "Gauteng"),
        ("África", "Egito", "Cairo"),
        ("África", "Quênia", "Nairóbi"),
        ("África", "Marrocos", "Casablanca"),
        # Oceania
        ("Oceania", "Austrália", "Nova Gales do Sul"),
        ("Oceania", "Austrália", "Victoria"),
        ("Oceania", "Nova Zelândia", "Auckland")
    ]

    cursor.executemany(
        "INSERT INTO Regiao (Continente, Pais, Estado) VALUES (%s, %s,
        ↪  %s)",
        regioes
    )
```

```python
def popular_catalogos(cursor):

    print("Inserindo Catálogos...")

    catalogos = [
        # Catálogos gerais
        ("Catálogo Global",),
        ("Catálogo Internacional",),
        ("Catálogo Latam",),
        ("Catálogo Europa",),
        ("Catálogo Ásia e Oceania",),
        # Conteúdos originais e exclusivos
        ("Originais Exclusivos",),
        ("Produções Independentes",),
        ("Conteúdos Premium",)
    ]

    cursor.executemany(
        "INSERT INTO Catalogo (descricao) VALUES (%s)",
        catalogos
    )
```

2. **Dados Sintéticos Volumosos:** As funções `popular_filmes()` e `popular_clientes()` geram o núcleo da massa de dados, simulando perfis variados e um acervo diversificado.

```python
def popular_filmes(cursor, qtd):
    print(f"Gerando {qtd} Filmes...")
    generos = [
        "Ação", "Aventura", "Comédia", "Comédia Romântica", "Drama",
        ↪  "Terror",
        "Suspense", "Sci-Fi", "Fantasia", "Documentário", "Biografia",
        "Romance", "Animação", "Musical", "Crime", "Guerra",
        ↪  "Histórico",
        "Western", "Mistério", "Esportes"
    ]
    classificacoes = ["Livre", "10", "12", "14", "16", "18"]
```

```
    dados = []
    # Loop de GERAÇÃO
    for _ in tqdm(range(qtd), desc="Gerando Filmes"):
        titulo = fake.catch_phrase().title()
        ano = random.randint(1990, 2024)
        duracao = random.randint(80, 180)
        genero = random.choice(generos)
        classif = random.choice(classificacoes)
        produtora = fake.company()
        idioma = escolher_idioma()
        sinopse = fake.text(max_nb_chars=200)

        dados.append((titulo, ano, duracao, genero, classif, produtora,
        ↪  idioma, sinopse))

    sql = """INSERT INTO Filme (titulo, ano_lancamento, duracao, genero,
    ↪  classificacao_ind, produtora, idioma_original, sinopse)
             VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"""
    inserir_em_lotes(cursor, sql, dados, descricao="Filmes")
```

```
def popular_clientes(cursor, qtd):
    PAISES_LOCAIS = {
        "Brasil": "pt_BR",
        "Estados Unidos": "en_US",
        "Canadá": "en_CA",
        "México": "es_MX",
        "Argentina": "es_AR",
        "Chile": "es_CL",
        "Portugal": "pt_PT",
        "Espanha": "es_ES",
        "Reino Unido": "en_GB",
        "França": "fr_FR",
        "Alemanha": "de_DE",
        "Itália": "it_IT",
        "Japão": "ja_JP",
        "China": "zh_CN",
        "Coreia do Sul": "ko_KR",
        "Índia": "en_IN",
        "Austrália": "en_AU"
    }
```

```
    print(f"Gerando {qtd} Clientes...")
    dados = []
    cpfs_gerados = []

    # Loop de GERAÇÃO
    for _ in tqdm(range(qtd), desc="Gerando Clientes"):
        # Sorteia país
        pais = choice(list(PAISES_LOCAIS.keys()))
        # Define o Faker com o locale adequado
        fake_local = Faker(PAISES_LOCAIS[pais])
        cpf = gerar_cpf()
        cpfs_gerados.append(cpf)
        nome = fake_local.name()
        email = f"{fake_local.user_name()}_{random.randint(1,999)}.
        ↪ {random.randint(1,999)}@email.com"
        nasc = fake_local.date_of_birth(minimum_age=16, maximum_age=70)
        cadastro = fake_local.date_between(start_date='-5y',
        ↪ end_date='today')
        dados.append((cpf, nome, email, nasc, pais, cadastro))

    sql = """
        INSERT INTO Cliente (CPF, nome_completo, email, data_nasc, pais,
        ↪ data_cadastro)
        VALUES (%s, %s, %s, %s, %s, %s)
    """
    inserir_em_lotes(cursor, sql, dados, descricao="Clientes")
    return cpfs_gerados
```

3. **Relacionamentos Complexos:** A função `popular_associativas_e_historico()` é responsável pela complexidade relacional, criando o histórico de assinaturas, distribuindo filmes em catálogos regionais e gerando sessões de visualização coerentes com os clientes ativos.

```
def popular_associativas_e_historico(cursor, cpfs):
  print("Criando relacionamentos dinâmicos...")
  # Carrega IDs para uso aleatório
  cursor.execute("SELECT ID_plano FROM Plano")
  ids_plano = [r[0] for r in cursor.fetchall()]

  cursor.execute("SELECT ID_regiao FROM Regiao")
  ids_regiao = [r[0] for r in cursor.fetchall()

  cursor.execute("SELECT ID_filme, duracao FROM Filme")
  filmes = cursor.fetchall()

  cursor.execute("SELECT ID_Catalogo FROM Catalogo")
  ids_catalogo = [r[0] for r in cursor.fetchall()]

  if not ids_plano or not ids_regiao or not filmes or not ids_catalogo:
      print("ERRO: Tabelas base vazias. Popule-as primeiro.")
      return
  # 1. Clientes, Regiões e Assinaturas
  residencia_data = []
  assinatura_data = []
  for cpf in tqdm(cpfs, desc="Associando Dados"):
      regiao = choice(ids_regiao)
      residencia_data.append((cpf, regiao,
      ↪  fake.date_between(start_date='-3y')))
      if random.random() > 0.10:  # 90% assinam
          plano_atual = choice(ids_plano)
          inicio = fake.date_between(start_date='-2y', end_date='-6m')
          # Primeira assinatura
          assinatura_data.append((cpf, plano_atual, inicio, None))
          # 30% dos clientes trocam de plano pelo menos uma vez
          if random.random() < 0.30:
              fim = fake.date_between(start_date=inicio, end_date='-3m')
              # Atualiza a primeira assinatura com data de término
              assinatura_data[-1] = (cpf, plano_atual, inicio, fim)
              # Novo plano diferente do primeiro
              novo_plano = choice([p for p in ids_plano if p !=
              ↪  plano_atual])
              novo_inicio = fim  # começa exatamente quando o outro termina

              assinatura_data.append((cpf, novo_plano, novo_inicio, None))
```

```
inserir_em_lotes(cursor, "INSERT INTO Regiao_residencia (CPF, ID_regiao,
↪  data_registro) VALUES (%s, %s, %s)", residencia_data,
↪  descricao="Residências")
inserir_em_lotes(cursor, "INSERT INTO Assina_Historico (CPF_cliente,
↪  ID_plano, data_inicio, data_termino) VALUES (%s, %s, %s, %s)",
↪  assinatura_data, descricao="Assinaturas")

# 2. Distribuição de Catálogos (Regiões e Planos)
print()#Quebra de linha para melhor visualização
cat_reg_data = []
plano_cat_data = []

for id_cat in tqdm(ids_catalogo, desc="Distribuindo Catálogos"):
    regioes_sorteadas = sample(ids_regiao, k=random.randint(2,
    ↪  len(ids_regiao)))
    for reg in regioes_sorteadas:
        cat_reg_data.append((id_cat, reg))

for id_plano in ids_plano:
    catalogos_sorteados = sample(ids_catalogo, k=random.randint(1,
    ↪  len(ids_catalogo)))
    for cat in catalogos_sorteados:
        plano_cat_data.append((id_plano, cat))

inserir_em_lotes(cursor, "INSERT IGNORE INTO Catalogo_Regional
↪  (ID_Catalogo, ID_regiao) VALUES (%s, %s)", cat_reg_data,
↪  descricao="Catálogos Regionais")
inserir_em_lotes(cursor, "INSERT IGNORE INTO Plano_Catalogo (ID_plano,
↪  ID_Catalogo) VALUES (%s, %s)", plano_cat_data, descricao="Planos x
↪  Catálogos")

# 3. Disponibilidade de Filmes
print()#Quebra de linha para melhor visualização
```

```
disp_data = []
for f in tqdm(filmes, desc="Distribuindo Filmes"):
    cats_filme = sample(ids_catalogo, k=random.randint(1, min(3,
    ↪  len(ids_catalogo))))
    for cat in cats_filme:
        disp_data.append((f[0], cat))
inserir_em_lotes(cursor, "INSERT IGNORE INTO Disponibilidade_Filme
↪  (ID_filme, ID_Catalogo) VALUES (%s, %s)", disp_data,
↪  descricao="Disponibilidade")

# 4. Sessões e Preferências
print()#Quebra de linha para melhor visualização
sessoes_data = []
prefs_data = []
for cpf in tqdm(cpfs, desc="Gerando Histórico"):
    qtd_sessoes = random.randint(5, 40)
    for _ in range(qtd_sessoes):
        filme = choice(filmes)
        id_filme, duracao_total = filme[0], filme[1]
        inicio = fake.date_time_between(start_date='-6m', end_date='now')
        progresso = random.randint(5, duracao_total)
        dispositivo = choice(["TV", "Celular", "Web", "Tablet"])
        qualidade = choice(["HD", "4K"])
        sessoes_data.append((cpf, id_filme, inicio, progresso,
        ↪  dispositivo, qualidade))
        if progresso > duracao_total * 0.8 and random.random() > 0.5:
            prefs_data.append((cpf, id_filme, choice([True, False]),
            ↪  random.randint(1, 5), inicio))

inserir_em_lotes(cursor, "INSERT IGNORE INTO Sessao_Visualizacao
↪  (CPF_cliente, ID_filme, data_hora_inicio, duracao_sessao,
↪  dispositivo_utilizado, qualidade_reproducao) VALUES (%s, %s, %s, %s,
↪  %s, %s)", sessoes_data, descricao="Sessões")
inserir_em_lotes(cursor, "INSERT IGNORE INTO Preferencia_Cliente
↪  (CPF_cliente, ID_filme, favorito_bool, avaliacao, data_interacao)
↪  VALUES (%s, %s, %s, %s, %s)", prefs_data, descricao="Avaliações")

print(f" Sessões inseridas: {len(sessoes_data)}")
print(f" Avaliações inseridas: {len(prefs_data)}")
print("Dados gerados com maior realismo e variabilidade!")
```

#### 5.1.2.3 Orquestração da Execução

A função principal (`main`) gerencia o ciclo de vida da operação sob uma única transação atômica (ACID). O fluxo de execução segue as etapas: (1) estabelecimento de conexão, (2) início da transação, (3) execução sequencial das funções de população e, finalmente, (4) a persistência definitiva via *commit* ou reversão total via *rollback* em caso de falhas, garantindo a consistência do banco.

## 5.2 Volumes de Dados Gerados

Para simular um ambiente de produção realista (*Big Data*), o script de povoamento foi configurado para gerar os seguintes dados:

- **3.000 Clientes** ativos com dados demográficos variados (Brasil, EUA, Europa, etc.).
- **1.000 Filmes** cadastrados com metadados completos (gênero, duração, produtora).
- Aproximadamente **2.700 Assinaturas** ativas ou encerradas (com taxa simulada de 90% da base).
- Cerca de **67.000 Sessões de Visualização** (média de 22 sessões por usuário, variando entre 5 e 40).
- Aproximadamente **7.000 Avaliações de Usuários** (geradas apenas quando o conteúdo foi assistido acima de 80% da duração total).
- **32 Regiões Globais** distribuídas em cinco continentes.
- **Distribuição Dinâmica**: relacionamento N:N aleatório entre Clientes, Catálogos, Regiões e Planos para evitar padrões repetitivos.

# 6 Manual de Execução e Uso

Este apêndice contém as instruções originais do projeto para configuração de ambiente, população do banco de dados e acesso às ferramentas.

## 6.1 Visão Geral

Este projeto cria e popula um banco de dados MySQL para um serviço de Streaming de Vídeo utilizando Docker e Python. O projeto contempla:

- Infraestrutura Docker para o Sistema Gerenciador de Banco de Dados (SGBD).
- Script de população automática com dados sintéticos realistas (biblioteca Faker).
- Configuração segura de credenciais.
- Consultas SQL analíticas para validação.

## 6.2 Dependências

O projeto foi testado com as seguintes versões das bibliotecas Python (listadas no arquivo `requirements.txt`):

- `mysql-connector-python==9.5.0`
- `python-dotenv==1.2.1`
- `Faker>=30.0.0`
- `tqdm>=4.67.0`

## 6.3 Estrutura do Projeto

A organização dos arquivos no repositório segue a estrutura abaixo:

```
Trabalho_IBD
|-- docker-compose.yml  # Sobe a infraestrutura do banco (MySQL)
|-- population_script.py #Script de popula  o de dados
|-- BD_schema.sql    # Script SQL com a estrutura do banco
|-- consultas_SQL_IBD.sql  # Script SQL com as consultas
|-- requirements.txt   # Lista de bibliotecas Python
|-- .env.example         # Modelo de vari veis de ambiente
|-- .env                 # Suas senhas reais
```

## 6.4 Configuração de Segurança (.env)

Este projeto utiliza variáveis de ambiente para não expor senhas no código fonte. Antes de rodar, é necessário configurar o ambiente local:

1. Localize o arquivo `.env.example` na raiz do projeto.
2. Faça uma cópia dele e renomeie-a para `.env`.
3. Preencha as variáveis com as configurações desejadas (senha, porta, usuário).

**Exemplo de conteúdo do arquivo `.env`:**

```
MYSQL_ROOT_PASSWORD=root
MYSQL_USER=aluno
MYSQL_PASSWORD=aluno123
MYSQL_DATABASE=trabalho_ibd
MYSQL_PORT=3307
```

## 6.5 Passo 1: Subindo o Banco com Docker

Certifique-se de ter o **Docker Desktop** instalado e em execução.

1. Abra o terminal na pasta do projeto.
2. Execute o comando para subir o container:

```
docker-compose up -d
```

Este comando baixa a imagem oficial do MySQL 8.0 e cria o container em segundo plano. Para verificar se o banco subiu corretamente, execute:

```
docker ps
```

Você deve ver o container `trabalho_ibd_mysql` com status **Up**.

## 6.6 Passo 2: Populando o Banco com Python

Com o banco de dados em execução, execute o script de população. Ele irá criar as tabelas (baseado no `BD_schema.sql`) e inserir milhares de registros de dados falsos para teste.

1. **Instale as dependências:**

```
pip install -r requirements.txt
```

2. **Execute o script:**

```
python population_script.py
```

Aguarde a barra de progresso finalizar. Se tudo correr bem, será exibida uma mensagem de sucesso ao final do processo.

## 6.7 Passo 3: Acessando via MySQL Workbench

Para visualizar os dados e rodar as consultas SQL, utilize o MySQL Workbench (ou outro cliente de sua preferência).

1. Abra o **MySQL Workbench**.
2. Clique no ícone (+) ao lado de "MySQL Connections".
3. Configure a nova conexão com os dados definidos no seu arquivo `.env`:

| **Campo** | **Valor** |
|---|---|
| Hostname | `localhost` |
| Port | `3307` (ou a porta definida no `.env`) |
| Username | `aluno` |
| Password | Clique em *Store in Vault* e digite sua senha |

4. Teste a conexão e clique em OK para salvar.

## 6.8 Solução de Problemas Comuns

- **Erro de conexão no Python:** Verifique se a porta definida no arquivo `.env` é a mesma que o Docker está usando (confira com `docker ps`).
- **Erro "Port already allocated":** Isso significa que a porta 3307 já está em uso. Mude a porta no arquivo `.env` para `3308` ou `3309` e reinicie o Docker.

# 7 Conclusão

O desenvolvimento deste trabalho prático permitiu a aplicação integrada dos conceitos de modelagem de dados e engenharia de software para solucionar as demandas de uma plataforma de *streaming* de vídeo global.

A etapa de modelagem (conceitual e lógica) provou-se eficaz ao mapear requisitos complexos de negócio, garantindo a integridade referencial em cenários críticos como a gestão de disponibilidade regional de catálogos e o controle temporal do histórico de assinaturas. A normalização do esquema relacional assegurou uma estrutura livre de redundâncias, apta a suportar o crescimento da base de dados.

Do ponto de vista técnico, a implementação da infraestrutura via Docker, aliada ao desenvolvimento de um algoritmo em Python para geração de dados sintéticos, foi fundamental para a validação do experimento. Esta abordagem permitiu simular um ambiente de alta volumetria (com milhares de clientes, sessões e avaliações), aproximando o trabalho acadêmico de um cenário real de *Big Data* e garantindo a variabilidade necessária para testes de estresse.

Por fim, a execução das consultas analíticas (OLAP) e transacionais demonstrou a capacidade do sistema em transformar dados brutos em informações estratégicas. O banco de dados implementado respondeu satisfatoriamente a todas as questões de negócio propostas — desde o monitoramento de *churn* até a análise de preferências de consumo — validando a robustez da arquitetura proposta.