Universidade Federal de São Carlos
Bacharelado em Ciência da Computação
Disciplina: Bancos de Dados
Prof(a). Dr(a). Sahudy Montenegro González

Projeto Prático – Aplicativo de Avaliação de Séries
Grupo 18 – Tema: Avaliação e Acompanhamento de Séries

Participantes:
• *Renan Suana Grothe Garcia – RA: 822469*

Data de Entrega: *19/02/2025*

Versão Final

1. Índice

---

2. Descrição do problema

O aplicativo de avaliação de séries foi desenvolvido com o intuito de oferecer aos usuários um sistema de banco de dados para acompanhar suas séries favoritas, realizar avaliações de episódios e consultar dados agregados que possibilitem uma análise detalhada das produções televisivas. Entre os principais objetivos estão:

- *Registrar e gerenciar informações dos usuários, incluindo dados pessoais e avaliações realizadas.*
- *Organizar séries em temporadas e episódios, garantindo a integridade dos dados e sentido lógico nas informações contidas no banco de dados.*
- *Permitir a realização de consultas que envolvam agregações, como a contagem de avaliações por temporada, a identificação de séries com maior duração total de episódios e outras consultas que poderiam ser visualizadas em uma interface web.*

2.1. Consultas

Cada consulta foi definida com base no minimundo e associada a um integrante do grupo durante a fase 2 do projeto. Na fase atual, implementamos-as em um arquivo separado, que contém detalhes sobre as definições das consultas, e exemplo de output com os dados que nós criamos para inserir.

- ***Consulta 1:*** *Quantas séries não possuem avaliações em nenhum episódio? (Responsável: Renan)*

Unset

```
''' sql
file: views_and_triggers/createviews.sql

 148   | CREATE OR REPLACE VIEW Series_Sem_Avaliacao AS
 149   |     SELECT s.nome
 150   |     FROM Serie s
 151   |     WHERE s.nome NOT IN (
 152   |         SELECT DISTINCT e.serie_nome
 153   |         FROM Episodio e
 154   |         JOIN Avaliacao a
 155   |           ON e.serie_nome = a.serie_nome
 156   |          AND e.num_episodio = a.num_episodio
 157   |          AND e.num_temporada = a.num_temporada
 158   |     );
'''
```

- ***Consulta 2:*** *Quantos episódios possuem duração acima da média de todos os episódios cadastrados? (Responsável: Lucas)*

Unset

```
''' sql
file: views_and_triggers/createviews.sql

 228   | CREATE VIEW Episodios_Acima_Media AS
 229   |     WITH media_duracao AS (
 230   |         SELECT AVG(duracao) as duracao_media
 231   |         FROM Episodio
 232   |     )
 233   |     SELECT COUNT(*) as total_episodios
 234   |     FROM Episodio, media_duracao
 235   |     WHERE Episodio.duracao > media_duracao.duracao_media;
'''
```

- ***Consulta 3:*** *Quantos usuários avaliaram pelo menos 10 episódios? (Responsável: Lucas)*

Unset

```
 ''' sql
file: views_and_triggers/createviews.sql

 249   | CREATE VIEW Usuarios_Min_Dez_Avaliacoes AS
 250   |     SELECT
 251   |         u.nome,
 252   |         u.email,
 253   |         COUNT(*) as total_avaliacoes
 254   |     FROM
 255   |         Usuario u
 256   |         JOIN Avaliacao a ON u.email = a.email_usuario
 257   |     GROUP BY
 258   |         u.email, u.nome
 259   |     HAVING
 260   |         COUNT(*) >= 10
 261   |     ORDER BY
 262   |         total_avaliacoes DESC;
'''
```

- ***Consulta 4:*** *Qual a série cujo conjunto de episódios possui a maior duração total? (Responsável: Renan)*

Unset

```
''' sql
file: views_and_triggers/createviews.sql

  70   | CREATE OR REPLACE VIEW Minutos_Por_Temporada AS
  71   |     SELECT s.nome, SUM(e.duracao) AS duracao_total
  72   |     FROM Serie s
  73   |     JOIN Episodio e ON s.nome = e.serie_nome
  74   |     GROUP BY s.nome
  75   |     HAVING SUM(e.duracao) = (
  76   |         SELECT MAX(total_duracao)
  77   |         FROM (
  78   |          SELECT SUM(e.duracao) AS total_duracao
  79   |          FROM Episodio
  80   |          GROUP BY serie_nome
  81   |         ) AS duracoes
  82   |     )
  83   |     ORDER BY duracao_total DESC;
'''
```

- ***Consulta 5:*** *Qual o ano em que foram lançados mais episódios? (Responsável: Carol)*

```
Unset
''' sql
file: views_and_triggers/createviews.sql

 122   | CREATE OR REPLACE VIEW Ano_Com_Mais_Lançamentos AS
 123   |     SELECT DATE_PART('year', data_estreia) AS ano, COUNT(*) AS
total_ep
       | isodios
 124   |     FROM Episodio
 125   |     GROUP BY DATE_PART('year', data_estreia)
 126   |     HAVING COUNT(*) = (
 127   |         SELECT MAX(total)
 128   |         FROM (
 129   |          SELECT COUNT(*) AS total
 130   |          FROM Episodio
 131   |          GROUP BY DATE_PART('year', data_estreia)
 132   |         ) AS contagens
 133   |     );
'''
```

- ***Consulta 6:*** *Quantos episódios receberam a maior nota de avaliação para cada série? (Responsável: Renan)*

```
Unset
''' sql
file: views_and_triggers/createviews.sql

 174   | CREATE OR REPLACE VIEW Episodios_Com_Maior_Nota_Dada AS
 175   |     WITH MaxNotas AS (  -- Encontra a maior nota dada para cada
série
 176   |         SELECT serie_nome, MAX(nota) AS max_nota
 177   |         FROM Avaliacao
 178   |         GROUP BY serie_nome
 179   |     )
 180   |     SELECT
 181   |         e.serie_nome,
 182   |         COUNT(DISTINCT (e.num_temporada, e.num_episodio)) AS
num_episod
       | ios_max_nota
 183   |     FROM Episodio e
 184   |     JOIN Avaliacao a
```

```
    ON e.serie_nome = a.serie_nome
    AND e.num_temporada = a.num_temporada
    AND e.num_episodio = a.num_episodio
JOIN MaxNotas m
    ON a.serie_nome = m.serie_nome
    AND a.nota = m.max_nota
GROUP BY e.serie_nome
ORDER BY num_episodios_max_nota DESC;
'''
```

Obs: Durante a realização das consultas, tornou-se claro que haveriam outras consultas mais relevantes do que as descritas pela fase 2 do nosso projeto, então adicionamos querys mais específicas na primeira parte do arquivo em *./views_and_triggers/createviews.sql*. *Desta forma, demonstramos a construção de mais tabelas úteis para a avaliação.*

Ex: Usuario_Com_Idade, Series_Com_Avaliacao, Avaliacoes_Por_Temporada, Serie_Com_Temporadas.

---

3. Projeto conceitual

Nesta fase, foi elaborado um Modelo Entidade-Relacionamento (MER) que reflete os requisitos do sistema. As principais entidades identificadas foram:

- ***Usuário:*** *Armazena informações como e-mail (identificador), nome, data de nascimento, idade (calculada) e gênero.*
- ***Série:*** *Contém dados sobre o nome, sinopse, país de origem, status (em andamento ou finalizada) e gêneros (multivalorado).*
- ***Temporada:*** *Identificada pelo número da temporada e associada a uma série, com atributos como ano de lançamento e número de episódios.*
- ***Episódio:*** *Relacionado a uma temporada, com informações sobre nome, número, duração, data de estreia e classificação indicativa.*
- ***Avaliação:*** *Representa a avaliação de um episódio por um usuário, contendo nota, comentário e data da avaliação.*

Além disso, para atender à normalização e às restrições do minimundo, foram criadas tabelas auxiliares para atributos multivalorados (por exemplo, Gênero da Série e Status da Série) e foram definidas restrições de integridade (como limites de datas e valores permitidos). Uma tabela de metadados foi confeccionada para documentar os tipos e restrições de cada atributo, garantindo clareza no projeto conceitual.

4. Projeto lógico

O mapeamento do MER para o modelo relacional foi realizado seguindo as regras discutidas em sala de aula. As principais decisões adotadas incluem:

- *A transformação dos atributos multivalorados em relações separadas (por exemplo, a criação das tabelas Serie_Genero e Serie_Status) para atender à Primeira Forma Normal (1FN).*
- *A criação de chaves primárias e estrangeiras que garantem a unicidade e a integridade referencial entre as tabelas.*
- *A inclusão de restrições de integridade (CHECK, NOT NULL, UNIQUE) para assegurar que os dados inseridos respeitem os limites definidos pelo domínio.*

O modelo relacional resultante foi analisado quanto à 3FN, concluindo-se que todas as relações atendem à Terceira Forma Normal, uma vez que cada atributo não-chave depende unicamente da chave primária e não existem dependências transitivas.

Tipo-Entidade	Atributo	Tipo	Restrição
Usuário	nome	Monovalorado	obrigatório
	data de nascimento	Monovalorado	obrigatório, > que 01/01/1910
	idade	Calculado	obrigatório, deriva-se da data de nascimento
	gênero	Monovalorado	obrigatório, [mulher, homem, prefiro não informar]
	senha	Monovalorado	obrigatório
	email	**Identificador**	**obrigatório**
Série	nome	Identificador	obrigatório
	sinopse	Monovalorado	obrigatório
	número de temporadas	Calculado	obrigatório, = à maior numeração da temporada
	status	Monovalorado	obrigatório, [em andamento, finalizada]
	país de origem	Monovalorado	obrigatório, (link)
	gênero	Multivalorado	obrigatório, (gêneros listados na tabela 3)
Temporada	numeração da	Identificador	obrigatório

	temporada		
	ano de lançamento	Monovalorado	obrigatório, > que o ano de 1910
	número de episódios	Monovalorado	obrigatório, > que 0
	nome da série	Identificador	obrigatório (pertence a série)

Pais	nome	Monovalorado	obrigatório
	id	Identificador	obrigatório (gerado pelo SQL)

Avaliacao	id	Identificador	obrigatório
	email_usuario	**Identificador**	**obrigatório**
	serie_nome	**Identificador**	**obrigatório**
	num_temporada	**Identificador**	**obrigatório**
	num_episodio	**Identificador**	**obrigatório**
	nota	Monovalorado	obrigatório, (entre 0 e 5)
	data da avaliação	Monovalorado	obrigatório, (posterior ao lançamento do episódio
	comentário	Monovalorado	opcional

---

5. Projeto físico de banco de dados

Esta seção descreve a implementação física do banco de dados utilizando PostgreSQL.

- ***Criação do Banco de Dados:***

  Os scripts de criação das tabelas encontram-se no arquivo setup.sql. Estes scripts implementam o modelo lógico, definindo as tabelas (Usuário, Série, Temporada, Episódio, Avaliação, etc.) e as respectivas restrições de integridade (chaves primárias, estrangeiras, CHECKs e UNIQUE). Além disso, houve a criação de tabelas auxiliares que contém os valores para campos que podem ter múltiplas categorias (como serie.genero) ou campos monovalorados que tem uma lista finita de valores permitidos (como em Serie_status)

- ***Políticas de Integridade:***

  As restrições de integridade foram implementadas tanto na DDL (através de cláusulas NOT NULL, CHECK, etc.) quanto por meio de triggers. Por exemplo, a trigger_verifica_data_estreia garante que a data de estreia dos episódios seja compatível com o ano de lançamento da temporada, e a trigger trigger_verifica_genero_serie assegura que somente gêneros permitidos sejam associados a uma série. Há mais triggers disponíveis para garantir o cumprimento de condições estabelecidas no projeto teórico.

- ***Alimentação do Banco de Dados:***

  Os scripts de inserção de séries no banco de dados (disponíveis na pasta scripts_insercao) foram desenvolvidos através de web-scraping, para permitir a testagem das restrições e das consultas para que os dados inseridos reflitam cenários reais de uso e torne a compreensão das consultas mais fácil. Outras inserções, como a de usuários e avaliações, foram criadas com valores aleatórios para testagem, mas mantendo episódios sem avaliação,e outros exemplos úteis para a demonstração de consultas. Para montar o banco de dados, execute todas as inserções na pasta scripts_insercao.

- ***Triggers e condições:***

  As triggers estão disponíveis no arquivo ./views_and_triggers/triggers_funcoes.sql e cada uma delas contém uma documentação identificando qual a sua função. Foram desenvolvidas restrições com triggers para a inserção de dados: verifica_data_estreia, verifica_genero_serie, check_serie_genero_exists (para não deletar um gênero e permitir que uma série não contenha um valor neste campo), verifica_data_avaliacao e verifica_classificacao_avaliacao.

6. Especificação de consultas em álgebra relacional e SQL

As consultas definidas foram implementadas utilizando views que consolidam dados provenientes de múltiplas tabelas. Cada consulta possui uma descrição textual, a representação em Álgebra Relacional (AR) e sua solução em SQL. Exemplos:

- ***Consulta: Idade do Usuário***

  Descrição: Calcula a idade do usuário baseado na data de nascimento dele.

  AR:

  π (email, nome, data_nascimento, idade ← (current_year − year(data_nascimento)), senha)(Usuario)

SQL:

```
Unset
''' sql
   7   | CREATE OR REPLACE VIEW Usuario_Com_Idade AS
   8   |     SELECT
   9   |         email,
  10   |         nome,
  11   |         data_nascimento,
  12   |         date_part('year', age(current_date, data_nascimento))::int
AS idade,
  13   |         senha
  14   |     FROM Usuario;
'''
```

- ***Consulta: Séries com pelo menos uma avaliação***

  Descrição:

  AR:

  π Serie.nome (

  Serie ⋈(Serie.nome = Episodio.serie_nome)

  (Episodio ⋈(Episodio.serie_nome = Avaliacao.serie_nome ∧

  Episodio.num_episodio = Avaliacao.num_episodio ∧

  Episodio.num_temporada = Avaliacao.num_temporada)

Avaliacao)

)

SQL:

```
Unset
''' sql
  18   | CREATE OR REPLACE VIEW Series_Com_Avaliacao AS
  19   |     SELECT DISTINCT s.nome
  20   |     FROM Serie s
  21   |     JOIN Episodio e ON s.nome = e.serie_nome
  22   |     JOIN Avaliacao a
  23   |       ON e.serie_nome = a.serie_nome
  24   |      AND e.num_episodio = a.num_episodio
  25   |      AND e.num_temporada = a.num_temporada;
'''
```

Outras consultas seguem a mesma lógica, utilizando junções, agregações e as funções requisitadas: COUNT, SUM, AVG e MAX. Uma definição de todas elas pode ser vista em views_and_triggers/createviews.sql.

---

7. Triggers

Para garantir a consistência dos dados, foram implementadas duas triggers e uma view, além das já mencionadas:

1. ***trigger_verifica_data_estreia:***
   – Função verifica_data_estreia valida que a data de estreia do episódio seja igual ou posterior a 01/01 do ano de lançamento da temporada.
2. ***trigger_verifica_genero_serie:***
   – Função verifica_genero_serie assegura que somente gêneros pré-definidos (action, comedy, drama, sci-fi, romance, thriller, horror) sejam associados a uma série.
3. ***trigger_check_serie_genero_exists:***
   – Impede a exclusão do único gênero associado à série, garantindo que cada série mantenha pelo menos um gênero cadastrado.
4. ***trigger_verifica_data_avaliacao:***
   – Função verifica_data_avaliacao verifica se a data da avaliação é posterior à data em que o episódio estreou.

5. ***trigger_verifica_classificacao_avaliacao:***
   – Função verifica_classificacao_avaliacao verifica se a classificação indicativa é menor ou igual à idade do usuário.

---

8. Considerações finais

O desenvolvimento do aplicativo de avaliação de séries demonstrou a importância da integração entre o modelo conceitual, lógico e físico para garantir a integridade e a consistência dos dados. A utilização de triggers e views não só reforçou as restrições impostas pelo minimundo, como também facilitou a execução de consultas complexas, agregando valor ao sistema. Durante o desenvolvimento, desafios como a correta definição dos limites temporais e a manipulação de atributos multivalorados foram superados, o que evidencia o aprendizado e a aplicação prática dos conceitos de bancos de dados. Futuras melhorias poderão incluir a otimização das consultas para grandes volumes de dados e a implementação de mecanismos adicionais de logging e monitoramento.

---

9. Scripts para a construção do banco de dados

Fonte: https://github.com/renangrothe/Sistema-De-Avaliacao-de-Series/

A estrutura do projeto consiste em scripts simples (ddl = setup.sql) e as views, triggers e inserções devem ser adicionadas na ordem especificada no README.md do projeto no GitHub.

---

10. Referências

- *IMDb – Fonte de informações sobre séries e filmes.*
- *Documentação do PostgreSQL – Disponível em: https://www.postgresql.org/docs/*
- *Letterboxd – Referência para sistemas de avaliação e gerenciamento de séries.*
- *StackOverflow – Diversas discussões e respostas que auxiliaram na implementação.*

    - *https://stackoverflow.com/questions/3093214/drop-all-triggers-from-postgres-db*
    - *https://unix.stackexchange.com/questions/32907/what-characters-do-i-need-to-escape-when-using-sed-in-a-sh-script* *(para corrigir a inserção)*
    - *https://stackoverflow.com/questions/3602450/where-does-postgresql-store-configuration-conf-files*
- *https://neon.tech/postgresql/postgresql-tutorial/postgresql-serial*