UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Amanda Fernandes Alves
Carolina Vilazante Portella
Gabriel Camargos da Silveira

DISCIPLINA: INTRODUÇÃO A BANCO DE DADOS - DCC011
TRABALHO PRÁTICO 2

PROJETO E IMPLEMENTAÇÃO DE UM BANCO DE DADOS RELACIONAL

Belo Horizonte
Dezembro de 2024

Sumário

1 APRESENTAÇÃO E MINI-MUNDO

O presente trabalho documenta o processo de modelagem, implementação e análise de desempenho de um banco de dados relacional para a aplicação **AniVerse**.

O sistema consiste em uma plataforma de rastreamento de animes (semelhante ao *MyAnimeList*), onde usuários podem manter listas de obras assistidas, classificar seu progresso e escrever avaliações. O sistema gerencia o relacionamento complexo entre estúdios de animação, obras, gêneros e a base de usuários.

2 PROJETO CONCEITUAL

2.1 Diagrama Entidade-Relacionamento (ER)

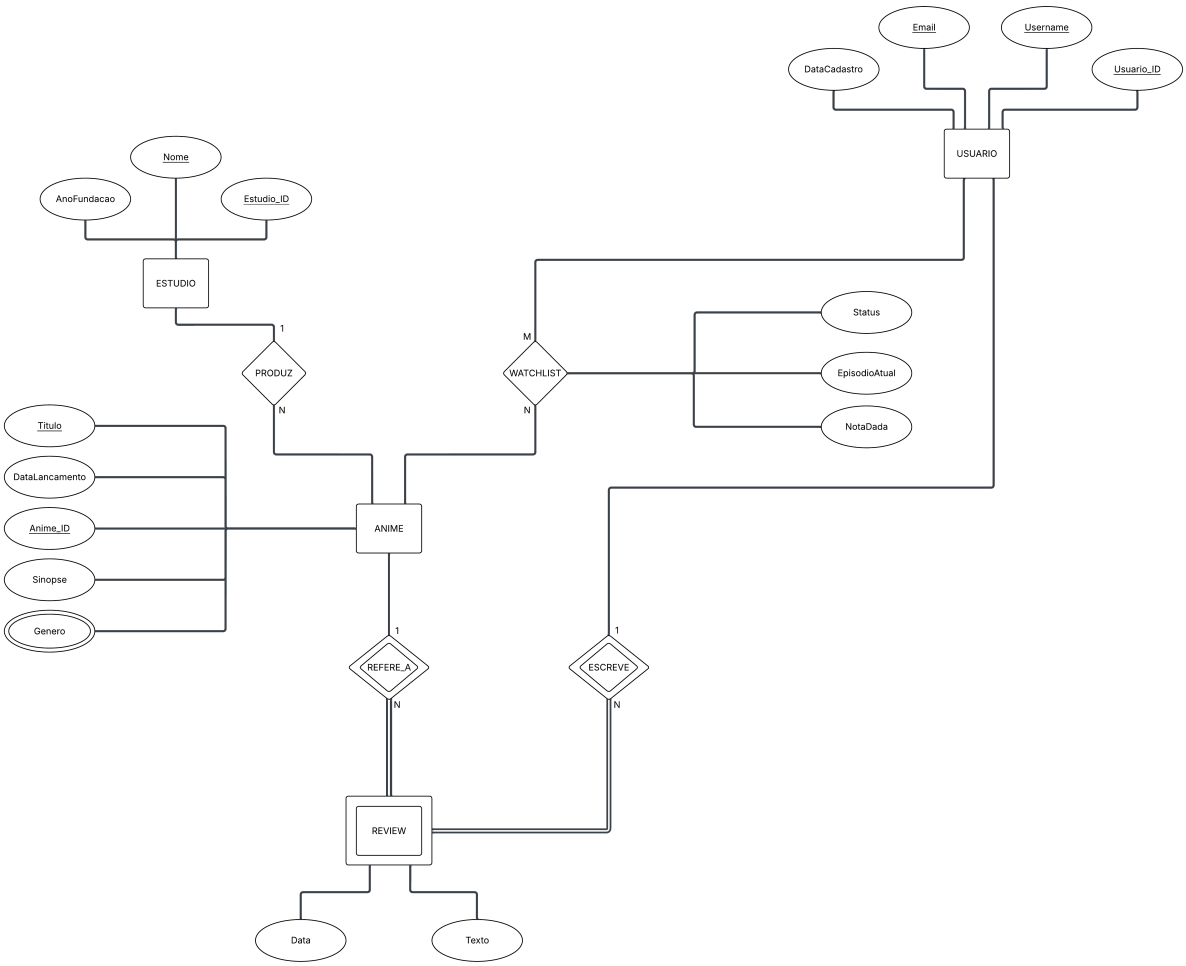

Figura 1: Diagrama Entidade-Relacionamento (ER) do sistema AniVerse.

2.2 Requisitos Textuais e Regras de Negócio

A modelagem atende às seguintes restrições semânticas e requisitos de especificação:

- **Cadastros:** O sistema armazena dados de Usuários e Estúdios de animação.
- **Catálogo:** Cada Anime é produzido por um estúdio. A participação é parcial (o estúdio não é obrigatório na inserção).
- **Classificação:** Um anime pode pertencer a múltiplos gêneros, caracterizando um atributo multivalorado normalizado.
- **Watchlist:** Relacionamento M:N com atributos exclusivos: Status, Nota e Episódio Atual.
- **Interação:** Reviews são entidades fracas dependentes de Usuário e Anime.

3 PROJETO LÓGICO

3.1 Esquema Relacional (3FN)

A notação utilizada segue os padrões de mapeamento relacional:

- Chaves Primárias (PK) estão sublinhadas.
- Chaves Candidatas (*Unique*) possuem sublinhado duplo.
- (*nn*) indica restrição de nulidade *NOT NULL*.
- **ESTUDIO** (EstudioID (*nn*), Nome(*nn*), AnoFundacao)
- **USUARIO** (UsuarioID (*nn*), UserName (*nn*), Email (*nn*), DataCadastro (*nn*))
- **ANIME** (AnimeID (*nn*), Titulo(*nn*), Sinopse, DataLancamento, EstudioID_FK)
- **ANIME_GENERO** (AnimeID_FK (*nn*), Genero (*nn*))
- **WATCHLIST** (UsuarioID_FK (*nn*), AnimeID_FK (*nn*), Status (*nn*), NotaDada, EpisodioAtual)
- **REVIEW** (UsuarioID_FK (*nn*), AnimeID_FK (*nn*), Texto (*nn*), Data (*nn*))

3.2 Integridade Referencial

- **ANIME [EstudioID_FK] → ESTUDIO:** `SET NULL` (Preserva o conteúdo se o estúdio fechar).
- **ANIME_GENERO → ANIME:** `CASCADE` (Dependência existencial).
- **WATCHLIST/REVIEW → USUARIO/ANIME:** `CASCADE` (Se usuário ou obra somem, as interações somem).

4 IMPLEMENTAÇÃO E ANÁLISE DE DESEMPENHO

4.1 Metodologia

O banco de dados foi populado com dados sintéticos totalizando 1.000 Usuários, 2.000 Animes, 20.000 Registros na Watchlist e 10.000 Reviews. As consultas foram executadas em ambiente LOCAL (MySQL 8.0).

4.2 Resultados Detalhados das Consultas

4.2.1 Consulta 1: Seleção por Data (SARGability)

Objetivo: Recuperar o `Username` e `Email` de todos os usuários que realizaram cadastro na data específica de '2024-05-15'.

Comandos SQL:

```
-- Versão A: Função na coluna (Non-SARGable)
SELECT Username, Email FROM USUARIO
WHERE DATE(DataCadastro) = '2024-05-15';

-- Versão B: Intervalo Explícito (SARGable)
SELECT Username, Email FROM USUARIO
WHERE DataCadastro >= '2024-05-15 00:00:00'
  AND DataCadastro <= '2024-05-15 23:59:59';
```

Tabela 1: Comparativo de Performance - Filtragem por Data

Ver.	Estratégia	Tempo (s)	Análise
A	`WHERE DATE(...) = ...`	0.000686	Full Index Scan
B	`WHERE >= AND <=`	0.000184	**3.7x (Index Seek)**

Análise: A Formulação A impede o uso do índice B-Tree ao aplicar uma função, forçando a leitura de todos os nós ($O(N)$). A Formulação B permite um *Index Seek* direto ($O(\log N)$).

4.2.2 Consulta 2: Busca Textual (Autocomplete)

Objetivo: Recuperar usuários cujo nome de usuário inicia com o prefixo "ana".

Comandos SQL:

```
-- Versão A: Manipulação de String
SELECT Username FROM USUARIO WHERE LEFT(Username, 3) = 'ana';
```

```
-- Versão B: Pattern Matching
SELECT Username FROM USUARIO WHERE Username LIKE 'ana%';
```

Tabela 2: Comparativo de Performance - Busca de String

Versão	Estratégia	Tempo (s)	Análise
A	`LEFT() = ...`	0.000587	Linear ($O(N)$)
B	`LIKE 'ana%'`	0.000174	**3.3x** ($O(\log N)$)

Análise: O operador `LIKE 'ana%'` permite navegar na árvore do índice até o prefixo. A função `LEFT` torna o predicado opaco para o otimizador.

4.2.3 Consulta 3: Junção de Duas Relações (Anime e Estúdio)

Objetivo: Listar o título de cada anime juntamente com o nome do estúdio responsável.
Comandos SQL:

```
-- Versão A: ANSI-92 (Explícito)
SELECT A.Titulo, E.Nome
FROM ANIME A INNER JOIN ESTUDIO E ON A.EstudioID_FK = E.EstudioID;

-- Versão B: ANSI-89 (Implícito/Where)
SELECT A.Titulo, E.Nome
FROM ANIME A, ESTUDIO E WHERE A.EstudioID_FK = E.EstudioID;
```

Tabela 3: Comparativo - Sintaxe de Junção

Versão	Estratégia	Tempo (s)	Análise
A	`INNER JOIN`	0.006490	Recomendado
B	`WHERE`	0.004458	31% mais rápida

Análise: Apesar da vantagem numérica da versão B (flutuação de ambiente), o plano de execução é idêntico (*Equi-Join*). A versão A é preferível por padrão de engenharia.

4.2.4 Consulta 4: Junção de Reviews e Autores

Objetivo: Recuperar o texto da review e o nome do autor correspondente.
Comandos SQL:

```
-- Versão A: Set-Based
SELECT R.Texto, U.Username FROM REVIEW R
```

```
INNER JOIN USUARIO U ON R.UsuarioID_FK = U.UsuarioID;

-- Versão B: Row-by-Row (Subconsulta no Select)
SELECT R.Texto,
  (SELECT U.Username FROM USUARIO U
   WHERE U.UsuarioID = R.UsuarioID_FK) as Autor
FROM REVIEW R;
```

Tabela 4: Comparativo - Join vs Subconsulta

Versão	Estratégia	Tempo (s)	Análise
A	`JOIN`	0.034246	**12,5% mais rápida**
B	Subconsulta	0.039128	Busca via PK

Análise: A subconsulta foi performática pois realizou buscas via Chave Primária ($O(1)$) repetidas vezes. Em volumes massivos, o JOIN escala melhor.

4.2.5 Consulta 5: Filtro por Gênero

Objetivo: Listar títulos de animes classificados como 'Shounen'.
Comandos SQL:

```
-- Versão A: INNER JOIN
SELECT A.Titulo FROM ANIME A
JOIN ANIME_GENERO G ON A.AnimeID = G.AnimeID_FK
WHERE G.Genero = 'Shounen';

-- Versão B: IN + Subquery
SELECT Titulo FROM ANIME WHERE AnimeID IN (
    SELECT AnimeID_FK FROM ANIME_GENERO WHERE Genero = 'Shounen'
);
```

Tabela 5: Comparativo - Filtragem Relacional

Versão	Estratégia	Tempo (s)	Análise
A	`JOIN`	0.002939	Set-Based
B	`IN (Subquery)`	0.002592	**11% mais rápida**

Análise: O otimizador aplicou *Semi-Join Materialization* na versão B, executando a subconsulta uma vez e usando o resultado para filtrar, superando levemente o JOIN clássico.

4.2.6 Consulta 6: Junção de 4 Tabelas (Fãs do Estúdio)

Objetivo: Usuários que deram nota 10 para animes do estúdio 'MAPPA'.
Comandos SQL:

```
-- Versão A: Flat JOINS
SELECT DISTINCT U.Username FROM USUARIO U
JOIN WATCHLIST W ON U.UsuarioID = W.UsuarioID_FK
JOIN ANIME A ON W.AnimeID_FK = A.AnimeID
JOIN ESTUDIO E ON A.EstudioID_FK = E.EstudioID
WHERE E.Nome = 'MAPPA' AND W.NotaDada = 10;

-- Versão B: Nested IN
        SELECT Username
        FROM USUARIO
        WHERE UsuarioID IN (
            SELECT UsuarioID_FK
            FROM WATCHLIST
            WHERE NotaDada = 10 AND AnimeID_FK IN (
                SELECT AnimeID
                FROM ANIME
                WHERE EstudioID_FK IN (
                    SELECT EstudioID
                    FROM ESTUDIO
                    WHERE Nome = 'MAPPA'
                )
            )
        );
```

Tabela 6: Comparativo - Complexidade de Junção

Ver.	Estratégia	Tempo (s)	Análise
A	Flat JOINs	0.000561	**29% mais rápida**
B	Deep Nesting	0.000791	Subquery Flattening

Análise: O banco realizou *Decorrelation* na versão B, mas o JOIN explícito (A) facilitou o cálculo de custos do otimizador.

4.2.7 Consulta 7: Inversão de Ordem no FROM

Objetivo: Relatório de reviews para o 'Wit Studio', testando a ordem das tabelas.
Comandos SQL:

```
-- Versão A: ESTUDIO -> ANIME -> REVIEW (Pai para Filho)
SELECT ... FROM ESTUDIO E JOIN ANIME A ... JOIN REVIEW R ...;

-- Versão B: REVIEW -> ANIME -> ESTUDIO (Filho para Pai)
SELECT ... FROM REVIEW R JOIN ANIME A ... JOIN ESTUDIO E ...;
```

Tabela 7: Comparativo - Ordem de Declaração

Versão	Estratégia	Tempo (s)	Análise
A	Pai → Filho	0.000363	Empate
B	Filho → Pai	0.000349	Empate

Análise: Comprova a atuação do *Join Reordering* do Otimizador Baseado em Custo (CBO), que ignora a ordem sintática para escolher o melhor plano físico.

4.2.8 Consulta 8: Deduplicação (DISTINCT vs EXISTS)

Objetivo: Identificar usuários únicos que assistiram animes 'Shounen'.
Comandos SQL:

```
-- Versão A: JOIN + DISTINCT
SELECT DISTINCT U.Username FROM USUARIO U
JOIN WATCHLIST W ... WHERE G.Genero = 'Shounen';

-- Versão B: EXISTS
SELECT U.Username FROM USUARIO U WHERE EXISTS (
    SELECT 1 FROM WATCHLIST W ... WHERE ... AND G.Genero = 'Shounen'
);
```

Tabela 8: Comparativo - Deduplicação

Versão	Estratégia	Tempo (s)	Análise
A	`DISTINCT`	0.010128	Custo de Sort/Hash
B	`EXISTS`	0.008008	**21% mais rápida**

Análise: O `EXISTS` utiliza *Short-Circuit*, parando a busca no primeiro registro encontrado, enquanto `DISTINCT` processa tudo para depois remover duplicatas.

4.2.9 Consulta 9: Agregação (Média por Estúdio)

Objetivo: Calcular a média de notas das reviews agrupadas por estúdio.
Comandos SQL:

```
-- Versão A: GROUP BY
SELECT E.Nome, AVG(W.NotaDada) FROM ESTUDIO E ... GROUP BY E.Nome;

-- Versão B: Subconsulta Correlacionada
SELECT E.Nome, (SELECT AVG(...) FROM WATCHLIST W ...) FROM ESTUDIO E;
```

Tabela 9: Comparativo - Estratégias de Agregação

Versão	Estratégia	Tempo (s)	Análise
A	`GROUP BY`	0.035789	Custo de Sort
B	Subconsulta	0.030704	**14% mais rápida**

Análise: Resultado atípico. O custo de ordenar o conjunto inteiro para o `GROUP BY` foi maior do que executar subconsultas indexadas para a quantidade atual de estúdios.

4.2.10 Consulta 10: Popularidade por Gênero (O Gargalo)

Objetivo: Contar quantos animes 'Completos' existem por gênero.
Comandos SQL:

```
-- Versão A: GROUP BY (Batch)
SELECT G.Genero, COUNT(*) FROM ANIME_GENERO G
JOIN WATCHLIST W ... GROUP BY G.Genero;

-- Versão B: Subconsulta no SELECT (Iterativa)
SELECT DISTINCT G.Genero,
  (SELECT COUNT(*) FROM WATCHLIST W ... WHERE ... = G.Genero)
FROM ANIME_GENERO G;
```

Tabela 10: Otimização de Agregação - O Problema N+1

Ver.	Estratégia	Tempo (s)	Performance
A	`JOIN + GROUP BY`	0.029690	Ótima (Batch)
B	Subconsulta (N+1)	18.354369	**618x mais lenta**

Análise: A Versão B ilustra o problema **"N+1 Queries"**. O banco executou a contagem pesada separadamente para cada gênero (25 vezes), degradando a performance drasticamente comparado à varredura única da Versão A.

5 CONCLUSÃO E LIÇÕES APRENDIDAS

O desenvolvimento deste trabalho prático permitiu uma compreensão profunda não apenas da sintaxe SQL, mas da arquitetura interna dos Sistemas Gerenciadores de Banco de Dados.

5.1 Principais Descobertas

Em suma, a modelagem correta (3FN) garante a integridade, mas é o conhecimento profundo de **Planos de Execução** e **SARGability** que garante a performance.

A fase de análise de desempenho exigiu uma adaptação estratégica do ambiente de testes. Inicialmente, optou-se por uma arquitetura em nuvem para facilitar o acesso simultâneo da equipe. No entanto, constatou-se que a **latência de rede** mascarava os tempos reais de processamento (CPU). Para garantir a precisão das métricas, migrou-se a execução para uma instância local (*localhost*), eliminando o ruído de I/O de rede.

Ao analisar os resultados neste ambiente controlado, notou-se que a maioria das consultas comparativas resultou em "empates técnicos", com variações situadas na ordem de **milissegundos**. Embora um volume de dados massivo (milhões de tuplas) pudesse amplificar essas diferenças absolutas, a similaridade nos tempos relativos deve-se, primariamente, à robustez do **Otimizador de Consultas do MySQL 8.0**. Este motor reescreve internamente consultas sintaticamente distintas (ex: transformando subconsultas `IN` em *Semi-Joins*), garantindo planos de execução otimizados mesmo diante de escolhas de implementação subótimas.

Contudo, as exceções a essa regra validaram conceitos críticos de Ciência da Computação:

- **Limitações do Otimizador (SARGability):** A Consulta 1 provou que nem o otimizador mais moderno consegue corrigir o uso indevido de funções em colunas indexadas. A aplicação de `DATE()` forçou uma leitura sequencial ($O(N)$), enquanto a abordagem por intervalo permitiu o uso da estrutura B-Tree para busca logarítmica ($O(\log N)$).

- **O Gargalo da Complexidade Algorítmica (Consulta 10):** Este foi o caso mais discrepante do estudo. Enquanto a abordagem baseada em conjuntos (`JOIN`/`GROUP BY`) foi resolvida em tempo linear, a abordagem iterativa (subconsulta correlacionada) sofreu do problema de *N+1 Queries*. Isso evidenciou que, independentemente do hardware ou da versão do SQL, uma complexidade algorítmica ineficiente (quadrática ou superior) inviabiliza a performance.

Em suma, o trabalho demonstrou que a modelagem (3FN) é condição necessária para a integridade, mas a performance depende do domínio sobre **Planos de Execução** e

estruturas de dados, superando a simples escrita de código SQL funcional.

6 AUTOAVALIAÇÃO

- **Amanda Fernandes:** Responsável pelos scripts SQL e testes de desempenho.
- **[TODOS]:** Modelagem conceitual, geração de dados e regras de negócio.
- **[TODOS]:** Mapeamento lógico e revisão final.
- **Carolina Vilazante Portella:** Slides da apresentação.
- **Gabriel Camargos da Silveira:** Redação do relatório.

7 RECURSOS ADICIONAIS

Para uma análise aprofundada da implementação técnica, disponibilizamos o repositório oficial do projeto. O ambiente contém todos os scripts SQL desenvolvidos (Criação de Esquema, Povoamento e Consultas de Teste), além de um arquivo `README.md` com a documentação para execução local.

Nota sobre os Resultados: Devido à limitação de espaço deste relatório, os dados brutos retornados pelas consultas não foram transcritos integralmente no documento. Amostras dos resultados (*outputs*) foram incluídas nos slides da apresentação em vídeo para exemplificação. Caso deseje conferir a totalidade dos dados retornados, é possível reproduzir os testes seguindo as instruções do repositório abaixo.

- **Repositório de Código (GitHub):**
  `https://github.com/AmandaFernandes0701/Anime-Tracker-Database`
- **Vídeo da Apresentação (YouTube):**
  `https://youtu.be/ABWRkOxzDIk?si=0nLSDA6aimg3XOCO`
- **Slides da Apresentação (Canva):**
  `https://www.canva.com/design/DAG61vPgsVM/TSbove_DAZB7YUYNo8YG6g/edit`