Universidade de São Paulo

ICMC - Instituto de Ciências Matemáticas e Computação

SSC0240 - Bases de Dados

Prof. Dra. Elaine Parros M. de Sousa

PAE: André Moreira Souza

ROTTEN LEMONS

UMA ALTERNATIVA PARA AVALIAÇÕES NO SPOTIFY

Natan Henrique Sanches 11795680

Álvaro José Lopes 10873365

Gabriel da Cunha Dertoni 11795717

São Carlos, 4 de julho de 2022

Sumário

1 Introdução

Um dos principais serviços de *streaming* de músicas utilizados na atualidade é o *Spotify*, que conta com uma grande variedade e quantidade de músicas, possuindo também uma grande gama de funcionalidades. Mesmo assim, algumas funcionalidades demandadas pelos usuários não estão disponíveis até o momento. Em particular, o *Spotify* não permite que usuários ou críticos deixem *reviews* em músicas, álbuns e artistas. Além disso, métodos de *reviews* e classificação são amplamente utilizados em outras mídias digitais, como no caso dos filmes e séries (através de plataformas como o *Rotten Tomatoes*). Por conta disso, existe uma demanda para tal serviço como demonstram publicações e comentários de usuários do *Spotify* no fórum oficial da plataforma (*Spofity community*[1]).

O sistema proposto atende à demanda existente permitindo a classificação e avaliação de músicas, álbuns e artistas da plataforma *Spotify*. Nesse sistema, o usuário dispõe de um mecanismo de avaliação de cinco estrelas, abrangendo também *tags* para classificação de acordo com gênero e temática, além de comentários de livre escrita. Com os dados das *tags*, serão levantadas estatísticas para a construção automática de *playlists* tematizadas, com uma gama de músicas fortemente relacionadas à *tag* escolhida. O usuário também pode se relacionar com outros usuários da plataforma na forma de seguidor, sendo notificado dos comentários e avaliações do seguido.

2 Modelo Entidade-Relacionamento

2.1 Levantamento de requisitos

Um **usuário** do sistema pode deixar uma **avaliação** (de 1 a 5 estrelas) para **músicas**, escolher ***tags*** (de no máximo 30 caracteres) classificadoras – como feliz, triste, nostálgica, etc. – para ela e deixar um **comentário** (com no máximo 300 caracteres). Cada usuário possui **nome de usuário** (com no mínimo 3 caracteres e no máximo 50 caracteres, consistindo de

[1]Pode ser observado em https://community.spotify.com/t5/Closed-Ideas/Social-Allow-Comments-on-Tracks-Albums/idi-p/1295893 e https://community.spotify.com/t5/Live-Ideas/Music-Personal-Rating-of-Music/idi-p/179102

caracteres alfanuméricos, *underscore* ou hífens), **sinalizador para críticos** (sim ou não) e um **sinalizador de cargo**, sendo os cargos: **usuário comum**, **moderador** e **administrador**. Os **moderadores** são responsáveis por garantir um ambiente agradável aos **usuários** e podem remover **comentários** e ***tags*** que não atendem às diretrizes da comunidade, além de banir/perdoar **usuários** (informando um motivo). Já os **administradores** possuem permissões ainda maiores, podendo gerenciar o sistema no geral e banir moderadores. O histórico de remoções e banimentos é salvo na base de dados. Além disso, cada **usuário** também possui um conjunto de informações pessoais, sendo eles **endereço de *e-mail*** (único), **data de nascimento** e **idade**. Ademais, **usuários** também podem seguir outros **usuários** e serão notificados de seus **comentários** e **avaliações**.

Um **artista** possui um **nome** (com no máximo 100 caracteres), um **gênero musical** principal (com no máximo 30 caracteres) e pode publicar vários **álbuns**. Cada **álbum** possui um **nome** (com no máximo 70 caracteres) e é composto por um conjunto de **músicas**. Artistas também podem colaborar na publicação de um **álbum** conjunto. Cada **música** possui um **nome** (com no máximo 70 caracteres), **duração** (em segundos) e o **álbum** a que está relacionado.

As ***tags*** são compostas e identificadas pelo seu **nome** e sua relevância é determinada de acordo com a frequência que são utilizadas por **música**. Cada **música** pode ser dita fortemente relacionada com uma ***tag***, se ela foi repetidamente atribuída à **música** por diferentes **usuários**. Uma ***tag*** gera uma ***playlist*** que será composta por uma lista de **músicas**. Toda ***playlist*** deve obrigatoriamente estar associada a uma ***tag*** existente.

Os **comentários de música**, **álbum** e **artista** são feitos por **usuários** e possuem seu **conteúdo** e **data/hora de publicação**, se relacionando diretamente com **música**, **álbum** e **artista** respectivamente.

2.2 Principais funcionalidades

As funcionalidades são diversas para cada tipo de usuário (comum, moderador ou administrador). Dentre elas, podemos citar:

- **Usuário**
  - Inserção, alteração ou remoção de avaliações em músicas através de um sistema de avaliação cinco estrelas.
  - Inserção, alteração ou remoção de *tags* em músicas para classificação;
  - Inserção, alteração ou remoção de comentários de livre escrita em músicas, álbuns e artistas;
  - Seguir outros usuários de preferência;
  - Navegação entre as avaliações e os comentários mais recentes de determinada música, artista ou álbum;
  - Pesquisa por determinado usuário, com base em seu nome de usuário;
  - Salvamento de *playlists* através do mecanismo de 'Curtir'.
- **Moderador**
  - Alteração ou remoção de avaliações, comentários e *tags* que não sigam as diretrizes da comunidade;
  - Banimento e perdão a usuários que descumpram as regras da comunidade;
  - Listagem de usuários banidos;
  - Concebimento do sinalizador de crítico para usuários reconhecidos como tal.
- **Administrador**
  - Inserção, modificação ou remoção de registros de músicas, álbuns e artistas;
  - Listagem de moderadores;
  - Concebimento do cargo de moderador para usuário.
  - Geração de *playlists* baseadas em *tags*.

2.3 Análise dos ciclos presentes no MER

- **Ciclo: Artista → Álbum**

  Há a presença de um ciclo entre os entidades Artista e Álbum, ao serem considerados os relacionamentos "Publica" e "Participa". Esse ciclo pode ser problemático, uma vez que representa um ciclo de dependência (o relacionamento "Participa" depende da existência do álbum, que possui existência restrita ao relacionamento "Publica"). Há necessidade desse fato ser tratado posteriormente, visto que não foi detectada alternativa para esse ciclo. A dependência se estende a principalmente dois casos:

  - Permite que a participação de um artista seja atribuída a um álbum não existente.
  - Permite a possibilidade do artista responsável por um álbum entrar como participante de seu próprio.

- **Ciclo: Usuário → Música**

  Há a presença de um ciclo entre as entidades Usuário e Música, ao serem considerados os relacionamentos "Avalia" e "Comenta". Esse ciclo não pôde ser quebrado, uma vez que ambos representam ações distintas do Usuário em Música e, portanto, necessita da utilização de dois relacionamentos diferentes entre ambas entidades.

- **Ciclo: Usuário → Música → Álbum**

  Há a presença de um ciclo entre as entidades Usuário, Música e Álbum, ao serem considerados os relacionamentos "Avalia", "Possui"e "Comenta". Esse ciclo não pôde ser quebrado, uma vez que é permitido que Usuário realize um comentário em Álbum e, além disso, Música e Álbum precisam estar relacionados pela dependência existencial de Música.

- **Ciclo: Usuário → Música → Álbum → Artista**

  Há a presença de um ciclo entre as entidades Usuário, Música, Álbum e Artista, ao serem considerados os relacionamentos "Avalia", "Possui", "Publica"(também se aplica a "Participa") e "Comenta". Esse ciclo não pôde ser quebrado, uma vez que é permitido que Usuário realize um comentário em Artista e, além disso, Música e Álbum precisam

estar relacionados pela dependência existencial de Música, bem como Álbum e Artista pelo mesmo motivo.

- **Ciclos: Usuário → Música → Tag → Playlist**

  Há possíveis ciclos formados entre as entidades Usuário, Música, Tag e Playlist, ao serem considerados os relacionamentos "Classifica por", "Curte", "Gera"e "Contém". Não foram encontradas alternativas para estes, visto a necessidade do relacionamento ternário entre Usuário, Música e Tag para permitir a classificação por *tags* de músicas. De forma análoga, há a necessidade da entidade Playlist estar relacionada com as entidades Usuário (visto que usuários podem 'Curtir' *playlists* – e como ela precisa existir, há a presença de um ciclo de dependência com o relacionamento "Gera"), Tag (visto sua origem a partir desta) e Música (uma vez que a *playlist* contém músicas – novamente com outro ciclo de dependência com o relacionamento "Gera").

- **Ciclo: Moderador → Comum**

  Há a presença de um ciclo de dependência entre as entidades 'Moderador' e 'Comum', através dos relacionamentos 'Bane' e 'Perdoa'. Não foram encontradas alternativas para esse ciclo, visto que o ato de banir/perdoar um usuário são ações distintas e que devem ser separadas. O ciclo representa dependência, uma vez que somente usuários banidos podem ser perdoados. Se não tratado, esse ciclo permite que um usuário que não está banido seja perdoado (não possui sentido lógico).

- **Ciclo: Usuário**

  Há a presença de um ciclo de dependência na entidade 'Usuário', através do auto-relacionamento 'Segue'. O ciclo não pode ser removido, uma vez que isso implicaria na exclusão desta ação. O ciclo representa dependência, uma vez que um usuário não pode seguir a si mesmo.

2.4 Diagrama do Modelo Entidade-Relacionamento

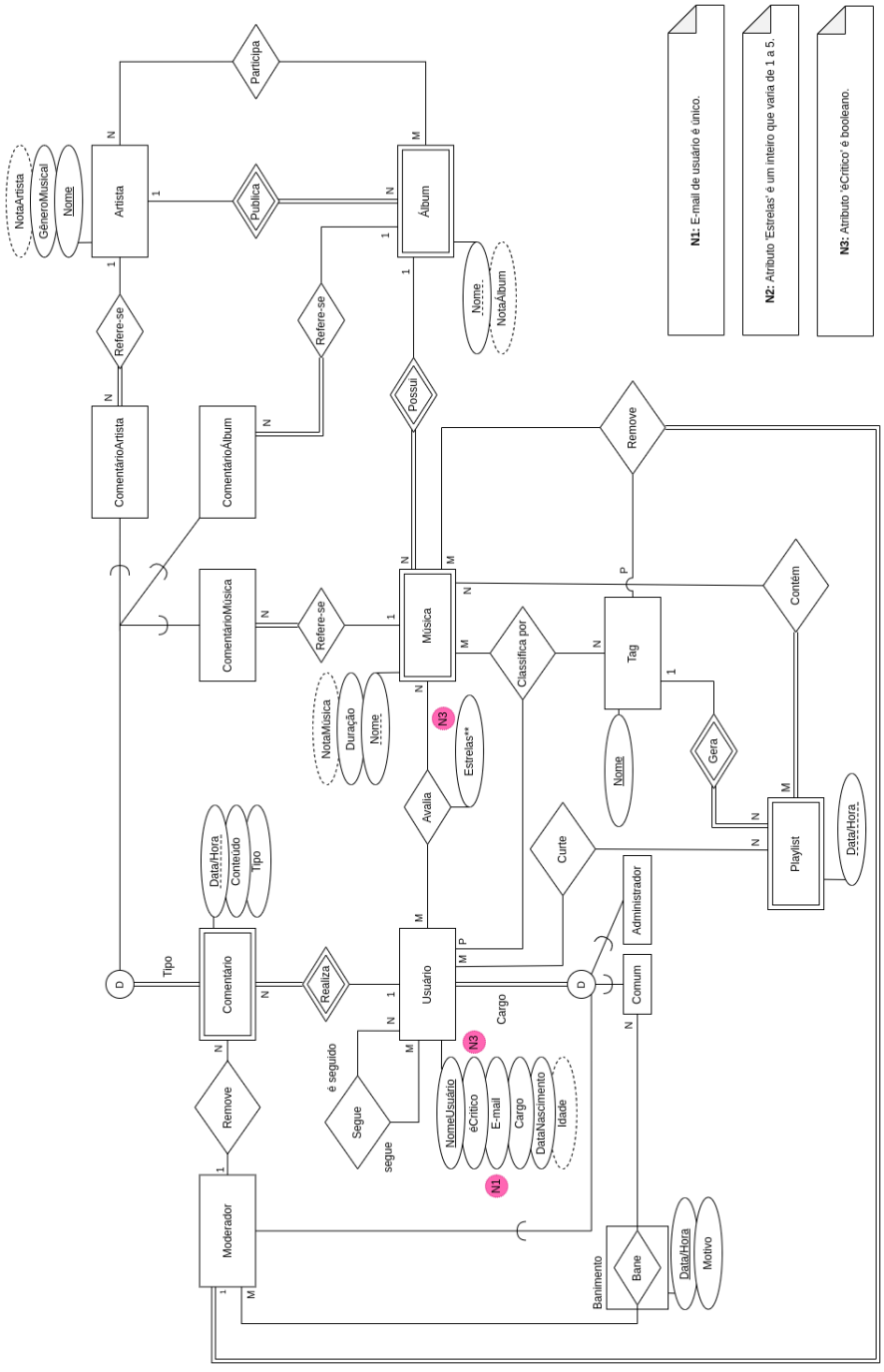

Figura 1: Diagrama MER do projeto Rotten Lemons

2.5 Mudanças relacionadas à Primeira Entrega

Apesar de inconsistências não terem sido encontradas por parte do monitor da disciplina, o grupo achou prudente realizar algumas modificações na primeira etapa do projeto. A motivação para essas modificações leva em consideração a maior familiaridade do grupo em relação ao primeiro assunto de modelagem atualmente. Assim, segue que:

- **Modificação nos atributos de 'Usuário':** Foi inserido o atributo 'DataNascimento' no conjunto de entidades 'Usuário', fazendo com que 'Idade' se tornasse um atributo derivado.
- **Remodelagem das entidades de comentário**: As entidades 'ComentarioArtista', 'ComentarioAlbum' e 'ComentarioMusica' agora são entidades fortes, especializações de uma entidade fraca geral 'Comentário', ao invés de agregações.
- **Relacionamento 'Modifica acesso' foi substituído:** O relacionamento 'Modifica acesso' entre 'Moderador' e 'Comum' foi substituído pelo relacionamento "Bane", que gera uma agregação com a data de banimento e o período de banimento.
- **Inserção do relacionamento 'Remove' entre 'Moderador' e 'Tag':** Notou-se que a ausência desse relacionamento gerava inconsistência entre o modelo e o levantamento de requisitos realizado na seção 2.1.

3 Modelo Relacional

Nesta seção, introduziremos o modelo relacional da base de dados proposta, que possui como função relacionar as diversas tabelas existentes a nível físico. Além de apenas construir o esquema, iremos também discutir sobre os diversos mapeamentos que foram feitos, evidenciando suas vantagens e desvantagens.

Na imagem do esquema (seção 3.1), as tabelas são representadas por blocos e sendo relacionadas a partir das setas.

3.1 Esquema Relacional

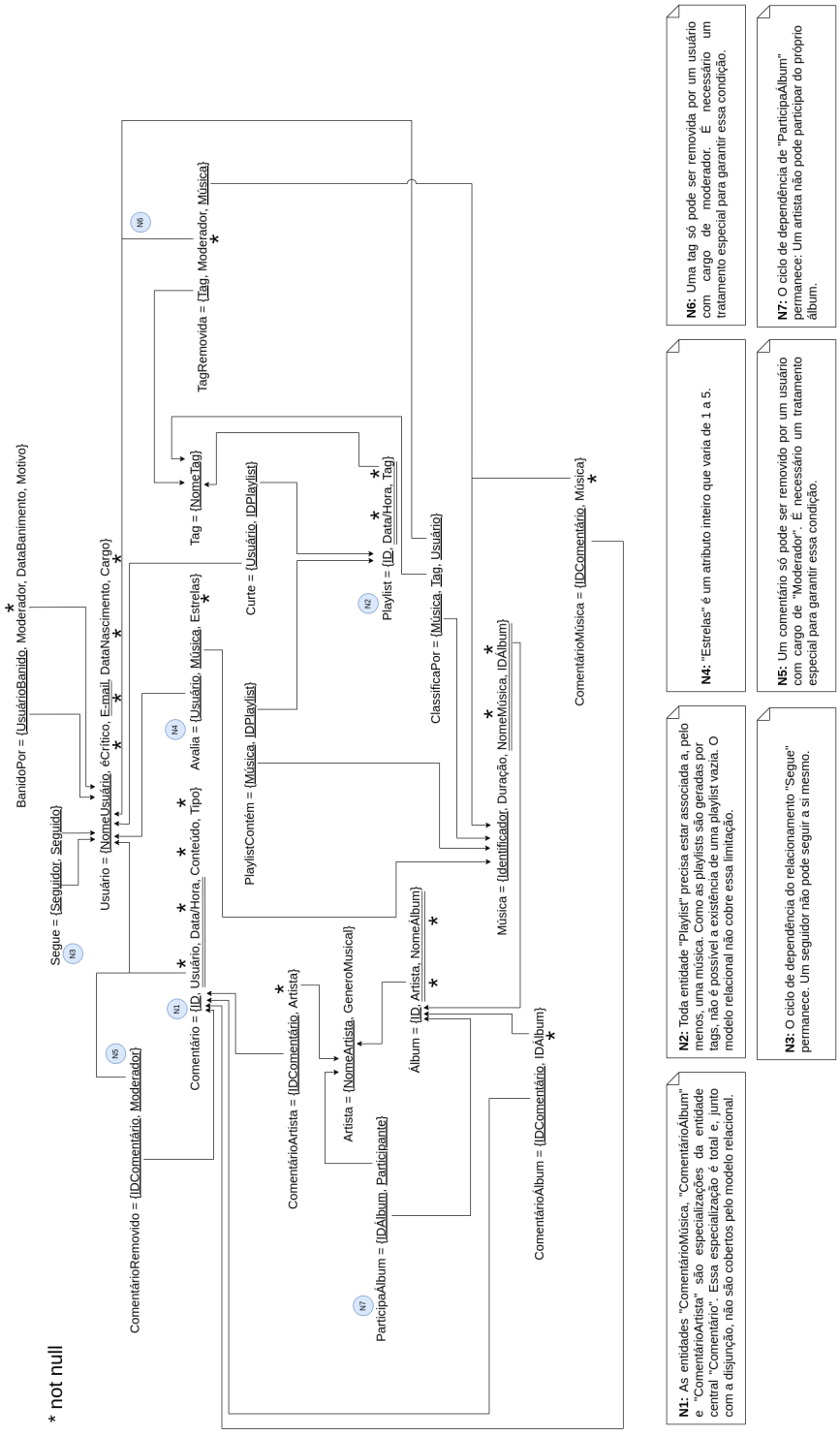

Figura 2: Esquema relacional do projeto Rotten Lemons

3.2 Discussão sobre os mapeamentos propostos

1. **Especialização da entidade 'Usuário'**

   - **Solução adotada:** Foi mapeada em apenas uma tabela ('Usuário'), sem a presença de uma tabela para cada especialização. Achou-se viável esse mapeamento, uma vez que os relacionamentos entre as especializações de 'Usuário' são poucos e, além disso, estes não possuem atributos.
   - **Vantagens:** Dado que as especializações não possuem atributos (i.e. todos os dados estão concentrados na entidade 'Usuário'), a criação de apenas uma tabela economiza recursos computacionais de armazenamento e processamento. O mapeamento cobre a especialização total.
   - **Desvantagens:** Dado que as especializações possuem relacionamentos, é requerido tratamento posterior a nível de aplicação para preservação da integridade. Além disso, é necessário tratamento posterior para garantir que o atributo 'Cargo' possuirá apenas valores válidos.
   - **Alternativas:** Uma alternativa seria a criação de uma tabela para cada especialização, que eliminaria a necessidade de tratamento a nível de aplicação. Por outro lado, seria mais custoso computacionalmente em questão de memória.

2. **Especialização da entidade 'Comentário'**

   - **Solução adotada:** Foi criada uma tabela para cada especialização de 'Comentário', uma vez que cada uma se relaciona com entidades distintas. Essa tabela, apesar de parecer redundante, foi utilizada posteriormente para mapear os relacionamentos 'Refere-se' (item 15).
   - **Vantagens:** O fato de cada especialização possuir uma tabela distinta permite que seja mais fácil mapear os relacionamento 'Refere-se', presentes em cada especialização.
   - **Desvantagens:** A presença de múltiplas tabelas encarece o custo computacional

de memória utilizado. Além disso, essa maneira de mapear não cobre a especialização total de 'Comentário', presente no Modelo Entidade-Relacionamento, e também não garante que a especialização total seja disjunta.

- **Alternativas:** Uma alternativa de mapeamento seria a compressão de todas as especializações em uma tabela só, como foi feito no tópico 1. Essa alternativa não foi considerada, por conta de que dificultaria muito o mapeamento dos relacionamentos 'Refere-se'.

3. **Relacionamento 'Segue' entre usuários – N:M**

   - **Solução adotada:** O mapeamento foi feito através da criação de uma nova tabela, 'Segue', que armazena o usuário seguidor e o usuário seguido como chave composta.
   - **Vantagens:** A solução permite que o relacionamento do tipo "muitos-para-muitos" seja preservado.
   - **Desvantagens:** A criação de uma nova tabela aumenta o custo de memória necessário. Além disso, esse método não cobre a participação total presente no relacionamento.
   - **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

4. **Relacionamentos 'Remove' entre Moderador e Comentário – 1:N**

   - **Solução adotada:** Houve a criação da tabela "ComentárioRemovido", uma vez que a proporção de comentários removidos são minoria comparados com a totalidade. Por isso, justifica-se a criação dessa nova tabela.
   - **Vantagens:** A cardinalidade do relacionamento é preservada, mesmo em uma tabela separada. Com a criação de uma nova tabela, evita-se a permanência de valores nulos nos atributos de "Comentário".
   - **Desvantagens:** Há maior uso de memória com a criação de uma nova tabela.

- **Alternativas:** Uma alternativa seria o mapeamento de um atributo "Removido-Por" dentro da entidade "Comentário". Essa alternativa não foi utilizada, uma vez que acumula muitos valores nulos nesse novo atributo.

5. **Relacionamentos 'Bane' entre Moderador e Comum – N:M**

   - **Solução adotada:** Foi criada uma nova tabela para a agregação "Banimento", tendo como chave a composição "Usuário + Moderador + Data do Banimento".
   - **Vantagens:** Permite que o histórico de banimentos seja armazenado.
   - **Desvantagens:** A criação de uma nova tabela necessita de maior uso de memória.
   - **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

6. **Relacionamento 'Realiza' entre Usuário e Comentário – 1:N**

   - **Solução adotada:** Na tabela 'Comentário', foi inserido um atributo 'Usuário', não nulo, que representa o usuário autor de determinado comentário. Isso permite com que o mesmo usuário seja responsável por vários comentários, mas que cada comentário possua apenas um autor.
   - **Vantagens:** A vantagem se dá, além de preservar a cardinalidade da relação, em cobrir a participação total de 'Comentário' no relacionamento, considerando que um comentário não pode ser cadastrado sem estar relacionado com um usuário.
   - **Desvantagens:** Não foram identificadas desvantagens em mapear desta maneira.
   - **Alternativas:** Uma alternativa seria a criação de uma tabela separada, que relacionasse cada comentário com seu respectivo usuário. No entanto, como a realização de comentários por parte dos usuários é bem recorrente, essa solução faria com que muitos dados redundantes precisassem ser replicados.

7. **Relacionamento 'Avalia' entre Usuário e Música – N:M**

   - **Solução adotada:** Foi criada uma nova tabela, 'Avalia', que armazena a relação entre o usuário e a música avaliada (chave composta), além do número de estrelas

dadas naquela avaliação. Isso permite com que uma mesma música possua diversos avaliadores, e que cada avaliador avalie diversas músicas.

- **Vantagens:** A cardinalidade do relacionamento é preservada.
- **Desvantagens:** A criação de uma nova tabela aumenta o custo de memória necessário.
- **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

8. **Relacionamento 'Classifica Por' entre Usuário, Música e Tag – M:N:P**

   - **Solução adotada:** Foi criada uma nova tabela, 'ClassificaPor', que armazena a relação entre Usuário, Música e Tag relacionada (todos como chave composta). Isso permite com que o par (Usuário, Música) esteja relacionado a diversas tags, o par (Usuário, Tag) esteja relacionado a várias músicas e, por fim, o par (Música, Tag) esteja relacionado a vários usuários.
   - **Vantagens:** A cardinalidade do relacionamento é preservada.
   - **Desvantagens:** A criação de uma nova tabela aumenta o custo de memória necessário.
   - **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

9. **Relacionamento 'Curte' entre Usuário e Playlist – N:M**

   - **Solução adotada:** Foi criada uma nova tabela, 'Curte', que relaciona Usuário e Playlist (chave composta). Isso permite com que um mesmo usuário curta várias playlists, e que uma mesma playlist seja curtida por vários usuários.
   - **Vantagens:** A cardinalidade do relacionamento é preservada.
   - **Desvantagens:** A criação de uma nova tabela aumenta o custo de memória necessário.
   - **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

10. **Relacionamento ‘Gera’ entre Tag e Playlist – 1:N**

    - **Solução adotada:** A entidade ‘Playlist’ foi interpretada como entidade forte (possuindo uma tabela própria), com um identificador sintético (justificado no item 18) como chave primária e, por consequência, a chave presente no MER foi utilizada como chave secundária.
    - **Vantagens:** A relação de entidade fraca de ‘Playlist’ é preservada, uma vez que cada registro de Playlist necessita da Tag relacionada. Além disso, essa maneira de mapeamento preserva a cardinalidade e a participação total do relacionamento.
    - **Desvantagens:** Não foram identificadas desvantagens em mapear desta maneira.
    - **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

11. **Relacionamento ‘Contém’ entre Playlist e Música – N:M**

    - **Solução adotada:** Foi criada uma nova tabela, ‘PlaylistContém’, que relaciona as chaves da entidade ‘Playlist’ com a chave da entidade ‘Música’ (no caso do modelo relacional, foi utilizado um identificador sintético, que será justificado a seguir) através de chaves estrangeiras. Todos os atributos citados e contidos na tabela formam uma chave composta, permitindo que uma mesma playlist possua várias músicas, e que uma mesma música esteja relacionada a várias playlists.
    - **Vantagens:** O mapeamento preserva a cardinalidade do relacionamento.
    - **Desvantagens:** Esse mapeamento não cobre a participação total presente em ‘Playlist’. Além disso, a criação de uma nova tabela aumenta o custo computacional de memória.
    - **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

12. **Entidades fracas Música e Álbum (de ‘Possui’ e ‘Publica’) – 1:N**

    - **Solução adotada:** As entidades fracas foram interpretadas como entidades fortes, possuindo tabelas próprias com um identificador sintético que as identifica

univocamente, como justificado nos itens 16 e 17, e as chaves primárias presentes no MER passaram a ser chaves secundárias.

- **Vantagens:** Esse mapeamento preserva a cardinalidade do relacionamento e cobre as participações totais das entidades fracas, com relação ao seu respectivo *owner*.
- **Desvantagens:** A criação de novas tabelas exige mais recursos computacionais de memória.
- **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

13. **Entidade fraca Comentário (de 'Realiza') – 1:N**

- **Solução adotada:** A entidade 'Comentário' foi interpretada como entidade forte (possuindo uma tabela própria), com a chave primária dessa tabela sendo um identificador sintético (justificado no item 17) e, por consequência, a chave presente no MER foi utilizada como chave secundária.
- **Vantagens:** A relação de entidade fraca de 'Comentário' é preservada, uma vez que cada registro de comentário necessita do Usuário relacionado. Além disso, essa maneira de mapeamento preserva a cardinalidade e participação total do relacionamento.
- **Desvantagens:** Não foram identificadas desvantagens em mapear desta maneira.
- **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

14. **Relacionamento 'Participa' entre Artista e Álbum – N:M**

- **Solução adotada:** Foi criada uma nova tabela, 'ParticipaÁlbum', que armazena o identificador relativo a determinado álbum e nome do artista participante, todos como chave composta. Isso permite com que um mesmo álbum possua, além do artista responsável, diversos artistas participantes; de maneira análoga permite que um mesmo artista participe de diversos álbuns.
- **Vantagens:** A cardinalidade do relacionamento é preservada.

- **Desvantagens:** O ciclo de dependência desse relacionamento com 'Publica' não é coberto, uma vez que o artista responsável por um álbum pode entrar como participante do seu próprio. Além disso, a criação de uma nova tabela aumenta o custo computacional de memória.
- **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

15. **Relacionamentos 'Refere-se' das especializações de Comentário – 1:N**

- **Solução adotada:** A respectiva entidade relacionada com 'Refere-se' foi colocada dentro da tabela de cada especialização de comentário. Por exemplo, o nome da música relacionada a determinado comentário foi colocada dentro da tabela 'ComentárioMúsica', através de uma chave estrangeira. O mesmo foi feito para as tabelas de 'ComentárioÁlbum' e 'ComentárioArtista'. Isso permite com que um comentário esteja relacionado a apenas uma entidade (Artista/Álbum/Música), mas que uma dessas entidades esteja relacionado a diversos comentários.
- **Vantagens:** A cardinalidade da relação é preservada, além de cobrir a participação total das especializações de comentário.
- **Desvantagens:** Não foram identificadas desvantagens em mapear desta maneira.
- **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

16. **Identificador sintético para a tabela 'Música'**

- **Solução adotada:** Dado que a entidade 'Música' possui muitos relacionamentos, sua chave primária inicial (Artista + Álbum + NomeMúsica) tornou-se chave secundária composta e não nula, e foi substituída por um identificador sintético que ocupa a posição de chave primária.
- **Vantagens:** A chave de música não dependerá de *strings* de tamanho variável, além de que não será mais necessário replicar 3 atributos nas tabelas que se relacionam com 'Música' através de chave estrangeira.
- **Desvantagens:** Ocorreu a inserção de um atributo sem significado semântico.

- **Alternativas:** A alternativa foi descrita no tópico de 'Solução adotada'.

17. **Identificador sintético para a tabela 'Álbum', 'Playlist' e 'Comentário'**

    - **Solução adotada:** Dado que todas as tabelas utilizavam de *strings* de tamanho variável para compor a chave primária, optou-se pela criação de um identificador sintético que identificasse unicamente cada registro.
    - **Vantagens:** A utilização do identificador sintético aumenta a performance de buscas de registros nessas tabelas, uma vez que a chave se trata de um número inteiro. Além disso, reduz-se o espaço em registros de outras tabelas que referenciam registros destas.
    - **Desvantagens:** Há a necessidade de um campo a mais em cada registro da tabela, que não possui significado semântico.
    - **Alternativas:** Uma alternativa seria permanecer com a chave primária inicial para cada tabela, como exposto no Modelo Entidade-Relacionamento.

18. **Atributo derivado 'Idade' de Usuário**

    - **Solução adotada:** O atributo não foi armazenado diretamente na base de dados, uma vez que idade é algo mutável anualmente. Como a idade do usuário não é uma informação essencial para outros relacionamentos, optou-se por essa alternativa.
    - **Vantagens:** O cálculo em tempo real da idade do usuário, quando necessário, reduz a possibilidade de inconsistências da idade com a data de nascimento relativa.
    - **Desvantagens:** O atributo de idade precisa ser calculado sempre que necessário.
    - **Alternativas:** A alternativa seria o armazenamento desse atributo diretamente na base de dados, mas que não foi optado por abrir brechas para inconsistências.

19. **Relacionamentos ‘Remove’ entre Moderador, Tag e Musica – 1:M:P**

    - **Solução adotada:** Houve a criação da tabela “TagRemovida”, de forma que ‘Tag’ e ‘Musica’ formassem uma chave primária composta. A criação da tabela justifica-se considerando que a proporção de tags removidas são minoria comparados com a totalidade. A remoção de uma *tag* por parte de um moderador é local àquela música.
    - **Vantagens:** A cardinalidade do relacionamento é preservada, mesmo em uma tabela separada.
    - **Desvantagens:** Não foram identificadas desvantagens em mapear desta maneira.
    - **Alternativas:** Não foram identificadas alternativas para esse mapeamento.

20. **Atributos derivados ‘NotaMúsica’, ‘NotaÁlbum’ e ‘NotaArtista’**

    - **Solução adotada:** Os atributos não foram armazenados diretamente na base de dados, uma vez que eles dependem das avaliações de determinada música, álbum ou artista naquele instante. Dessa forma, esse atributo é extremamente mutável. Como esses atributos não são necessários para outros relacionamentos, optou-se por essa alternativa.
    - **Vantagens:** O cálculo em tempo real desses atributos, quando necessários, evita que seja exibido um valor desatualizado;
    - **Desvantagens:** O atributo precisa ser recalculado sempre que necessário, exigindo mais poder de processamento.
    - **Alternativas:** A alternativa seria o armazenamento desses atributos diretamente na base de dados, em suas respectivas entidades. Essa alternativa não foi considerada pela brecha para inconsistências que possui.

3.3 Mudanças relacionadas à Segunda Entrega

Para a terceira entrega, alguns pontos levantados pelo monitor da disciplina foram corrigidos. A motivação para essas modificações, além de fixar pequenos erros que passam despercebido, leva em consideração a maior familiaridade do grupo em relação ao de mapeamento atualmente. Assim, segue que:

- **Inserção de identificador sintético**: As tabelas 'Comentário', 'Playlist' e 'Álbum' receberam um identificador sintético, propriamente justificado, como chave primária. Os atributos que estavam sendo utilizados como chave primária anteriormente foram transformados em chave secundária.
- **Participação total em relacionamento 'Segue' removida**: Ficou claro ao grupo que a lógica utilizada para essa participação total foi mal compreendida. Assim, a participação total foi removida do Modelo Entidade-Relacionamento, com as devidas adaptações no Modelo Relacional.
- **Relacionamento 'Remove' entre 'Moderador' e 'Tag' agora é ternário**: Observou-se que seria melhor que a remoção das *tags*, por parte do moderador, fosse restrita apenas àquela música. Assim, o relacionamento 'Remove' tornou-se ternário com cardinalidade 1:M:P entre 'Moderador', 'Música' e 'Tag', respectivamente.
- **Pequenos ajustes**: Erros menores foram identificados no MER e no Modelo Relacional pelos integrantes do grupo na revisão do trabalho, sendo corrigidos. Os erros variam desde chaves compostas que foram inseridas erroneamente, cardinalidades erradas que passaram despercebidas e mudanças de atributos menos importantes.

4 Implementação

Nesta seção, introduziremos o protótipo de aplicação e a implementação do banco de dados da plataforma *Rotten Lemons*. O protótipo foi implementado utilizando a linguagem *Python*,

enquanto para o banco de dados foi utilizado o SGBD *PostgreSQL*. O repositório com todo o conteúdo criado para essa disciplina pode ser visto em natan-dot-com/Rotten-Lemons.

Nesta etapa do projeto, foi montado o esquema da base de dados — consistindo em todas as tabelas, com seus devidos tratamentos de consistência — e *scripts* de alimentação dessa base de dados. Além disso, cinco consultas de complexidade média e alta foram formuladas para teste e um protótipo inicial, integrado à base de dados, foi proposto pelo grupo.

4.1 Estrutura de diretórios

```
.
├── README.md
├── Rotten-Lemons-ERM.drawio
├── Rotten-Lemons-ERM.png
├── Rotten-Lemons-RelSchema.drawio
├── Rotten-Lemons-RelSchema.png
├── Application
│   ├── app.py
│   ├── completer.py
│   └── funcionalidades.md
└── SQL
    ├── consultas.sql
    ├── dados.sql
    ├── drop.sql
    ├── esquema.sql
    ├── testes.sql
    └── triggers
        ├── album_circular.sql
        ├── bloqueia_banidos.sql
        ├── classifica_por_tag_removida.sql
        ├── permissao_ban.sql
        ├── permissao_remover.sql
        └── tag_remover_consistente.sql
```

Na raiz do repositório, além dos arquivos utilizados nas entregas anteriores, temos os diretórios `SQL` — que contém os arquivos relativos ao banco de dados — e `Application` — que contém a implementação da aplicação.

Em `SQL`, é possível encontrar *scripts* do esquema, da alimentação com dados, e de teste das operações de checagem e *triggers*. O *script* das cinco consultas, explicadas a seguir, também se encontra nesse diretório. Por fim, o diretório `triggers` armazena as funções de *triggers*, utilizadas no esquema para garantir consistência. Em `Application`, o arquivo `app.py` é o arquivo principal da aplicação.

4.2 *Script* de consultas

C1. **Consultar as *tags* mais populares de cada música. Citar também as músicas sem *tags* classificadas.**

```
01 | SELECT DISTINCT ON (TMP.NOME_MUSICA) AR.NOME_ARTISTA,
02 |     AL.NOME_ALBUM, TMP.NOME_MUSICA, TAG_POPULAR,
03 |     CONTAGEM_UTILIZACOES
04 | FROM ARTISTA AR
05 | JOIN ALBUM AL ON (AR.NOME_ARTISTA = AL.ARTISTA)
06 | RIGHT JOIN (
07 |     SELECT M.NOME_MUSICA, M.ALBUM, COALESCE(CP.TAG, '-') AS
08 |         TAG_POPULAR, COUNT(CP.TAG) AS CONTAGEM_UTILIZACOES
09 |     FROM MUSICA M
10 |     LEFT JOIN CLASSIFICA_POR CP ON (M.ID_MUSICA = CP.ID_MUSICA)
11 |     GROUP BY M.NOME_MUSICA, CP.TAG, M.ALBUM
12 |     ORDER BY M.NOME_MUSICA, COUNT(CP.TAG) DESC
13 | ) TMP ON (TMP.ALBUM = AL.ID_ALBUM)
14 | ORDER BY TMP.NOME_MUSICA;
```

Consulta 1: Tags mais populares de cada música, em SQL

Nessa consulta, inicia-se encontrando a contagem de cada *tag* usada para classificar cada música. Tem-se, então, a lista de ocorrências de cada par (Música, Tag). Essa

tabela pode ser ordenada de maneira decrescente e, utilizando uma seleção por valores distintos de nome da música, obtemos a *tag* com maior ocorrência em cada música. Junções com as tabelas de álbum e artista são feitas para resgatar, respectivamente, o álbum e artista de cada música. O número de vezes que a *tag* foi utilizada naquela música também aparece na resposta da consulta.

C2. **Visualizar as notas médias de todos os artistas da base de dados.**

```
01 | SELECT AR.NOME_ARTISTA, ROUND(AVG(NOTA_ALBUM), 3)
02 |     AS NOTA_ARTISTA
03 | FROM ARTISTA AR, (
04 |     SELECT AL.NOME_ALBUM, AL.ARTISTA,
05 |         ROUND(AVG(NOTA_MUSICA), 3) AS NOTA_ALBUM
06 |     FROM ALBUM AL, (
07 |         SELECT M.ID_MUSICA, M.ALBUM,
08 |             ROUND(AVG(A.ESTRELAS), 3) AS NOTA_MUSICA
09 |         FROM MUSICA M, AVALIA A
10 |         WHERE M.ID_MUSICA = A.ID_MUSICA
11 |         GROUP BY (M.ID_MUSICA, M.ALBUM)
12 |     ) NOTAS_M
13 |     WHERE NOTAS_M.ALBUM = AL.ID_ALBUM
14 |     GROUP BY (AL.NOME_ALBUM, AL.ARTISTA)
15 | ) NOTAS_AL
16 | WHERE NOTAS_AL.ARTISTA = AR.NOME_ARTISTA
17 | GROUP BY (AR.NOME_ARTISTA)
18 | ORDER BY (NOTA_ARTISTA) DESC;
```

Consulta 2: Consulta para cálculo das notas médias dos artistas, em SQL

Para essa consulta, foi considerado que a nota de um artista é dada pela média das notas de seus álbuns, na qual é dada pela média das notas de suas músicas e que, por consequência, é dada pela média de todas as avaliações naquela música. Com isso, usou-se três seleções correlacionadas para cálculo de cada nota hierarquicamente utilizando as funções de agregação AVG e ROUND. A saída foi ordenada de forma decrescente.

Como comentado anteriormente, não foram colocados atributos na base de dados para armazenar notas médias de músicas, álbuns e artistas, uma vez que essas mutacionam constantemente. Essa consulta é uma possibilidade de resgate desses valores calculados *on-time*, sem possibilidade de inconsistência na base.

C3. **Selecionar músicas que foram classificadas com todas as *tags* disponíveis na base de dados.**

```
01 | SELECT DISTINCT AR.NOME_ARTISTA, AL.NOME_ALBUM, M.NOME_MUSICA
02 | FROM ARTISTA AR
03 | JOIN ALBUM AL ON (AL.ARTISTA = AR.NOME_ARTISTA)
04 | JOIN MUSICA M ON (M.ALBUM = AL.ID_ALBUM)
05 | WHERE NOT EXISTS (
06 |     (SELECT * FROM TAG)
07 |     EXCEPT
08 |     (SELECT CP.TAG FROM CLASSIFICA_POR CP WHERE
09 |         CP.ID_MUSICA = M.ID_MUSICA)
10 | );
```

Consulta 3: Seleção de músicas classificadas com todas as tags, em SQL

A consulta foi baseada no operador de divisão (%), presente conceitualmente na Álgebra Relacional. A implementação desse operador foi feita utilizando-se o operador de diferença, EXCEPT, entre o conjunto que representa todas as *tags* disponíveis no banco de dados, e o conjunto de todas as *tags* distintas que determinada música foi classificada. Para que uma música tenha sido classificada com todas as *tags* disponíveis, o resultado da operação citada deve ser o conjunto vazio.

C4. **Recuperar todos os comentários, feitos por críticos, de todas as músicas.**

```
01 | SELECT U.NOME_USUARIO, C.CONTEUDO, M.NOME_MUSICA, A.NOME_ALBUM,
02 |     A.ARTISTA
03 | FROM COMENTARIO_MUSICA CM
04 | NATURAL JOIN COMENTARIO C
05 | NATURAL JOIN USUARIO U
06 | JOIN MUSICA M ON M.ID_MUSICA = CM.ID_MUSICA
07 | JOIN ALBUM A ON A.ID_ALBUM = M.ALBUM
08 | WHERE U.EH_CRITICO = TRUE;
```

Consulta 4: Consulta dos comentários feitos por críticos, em SQL

Nessa consulta, utilizou-se de junções entre as tabelas de Comentário, Usuário, Música e Álbum para resgate das informações relevantes na identificação do comentário e da música comentada. Além disso, utilizou-se um filtro para resgate apenas dos registros cujo usuário autor do comentário está na categoria de crítico.

C5. **Selecionar o nome e artista principal das músicas com o maior número bruto de *tags* classificadas.**

```
01 | SELECT M.NOME_MUSICA, AR.NOME_ARTISTA, COUNT(CP.TAG)
02 |     AS CONTAGEM
03 | FROM MUSICA M
04 | NATURAL JOIN CLASSIFICA_POR CP
05 | JOIN ALBUM AL ON (M.ALBUM = AL.ID_ALBUM)
06 | JOIN ARTISTA AR ON (AL.ARTISTA = AR.NOME_ARTISTA)
07 | GROUP BY M.NOME_MUSICA, AR.NOME_ARTISTA
08 | ORDER BY (CONTAGEM) DESC;
```

Consulta 5: Consulta do número total de *tags* classificadas em cada música, em SQL

Nessa consulta, junções foram feitas entre as tabelas de Música, Álbum e Artista para resgate das informações que identificam cada música. Além disso, foi avaliada a ocorrência de cada música em uma classificação, independentemente da *tag* e do usuário,

utilizando a função `COUNT`. O resultado foi ordenado de maneira decrescente em função da contagem de *tags*.

4.3 Aplicação

4.3.1 Descrição da aplicação

Trata-se de um *shell* interativo, criado na linguagem *Python*, que possui alguns comandos de inserção e consulta de registros na base de dados. A aplicação foi integrada ao banco de dados através da biblioteca *Psycopg2*, que oferece uma interface de comunicação entre o *Python* e o *PostgreSQL*.

4.3.2 Requisitos do sistema

A aplicação foi desenvolvida e testada em ambiente de *Windows 11* (WSL) e *Linux* — mais especificamente, *Debian 11 Bullseye*. Foi utilizada a versão 3.9.2 do *Python* para execução da aplicação, e versão 13.7 do *PostgreSQL* para o banco de dados.

4.3.3 Instruções para utilização

Para quaisquer funcionalidades, o usuário deve ter feito *log-in* no sistema (seja como usuário comum, moderador ou administrador. O usuário pode registrar uma nova conta utilizando do comando próprio para isso.

O nome do banco de dados e o nome do usuário utilizado para acessar o *PostgreSQL* podem ser configurados através das variáveis de ambiente `DBNAME` e `DBUSER`, repectivamente. A aplicação irá automaticamente resgatar e utilizar esses valores internamente. É recomendada a definição dessas variáveis para evitar possíveis erros de permissão no acesso ao *PostgreSQL*.

A seguir, estão listados os comandos que foram implementados na aplicação. As permissões de cada comando são gerenciadas a nível de banco de dados.

```
sair - fecha o REPL e encerra a sessão

ajuda '<comando>' - retorna uma mensagem de ajuda sobre o comando

lista comandos - lista todos os comandos disponíveis

registrar - cria um novo usuário

login <email> - faz login com um email
PARAMS:
    <email> - endereço de email para usuário

buscar <tipo> - busca por uma música, álbum, artista ou usuário por nome
PARAMS:
    <tipo> pode ser 'musica', 'album', 'artista' ou 'usuario'

avaliar musica - avalia uma música (por id) numa escala de 5 estrelas

atribuir tag - atribui uma tag a uma música (por id)

remover tag - remove uma tag de uma música (por id)

comentar <tipo> - deixa um comentário
PARAMS:
    <tipo> - pode ser 'artista', 'album' ou 'musica'

remove comentario <id do comentário> - remove um comentário
PARAMS:
    <id do comenário> - id único do comentário

curtir <id da playlist> - curte uma playlist
PARAMS:
    <id da playlist> - id único da playlist
```

```
01 |  remover curtida <id da playlist> - remove a curtida de uma playlist
02 |  PARAMS:
03 |      <id da playlist> - id único da playlist
04 |
05 |  seguir <nome do usuário> - segue um usuário
06 |  PARAMS:
07 |      <nome do usuário> - nome do usuário que gostaria de seguir
08 |
09 |  deixar de seguir <nome do usuário> - deixa de seguir um usuário
10 |  PARAMS:
11 |      <nome do usuário> - nome do usuário que gostaria de deixar de seguir
12 |
13 |  banir <nome do usuário> - bane um usuário (somente para moderadores)
14 |  PARAMS:
15 |      <nome do usuário> - nome do usuário a ser banido
16 |
17 |  perdoar <nome do usuário> - perdoa um usuário (somente para moderadores/
        administradores)
18 |  PARAMS:
19 |      <nome do usuário> - nome do usuário a ser perdoado
20 |
21 |  lista playlists - lista todas as playlists registradas.
22 |
23 |  cria playlist <tag> - cria uma nova playlist (vazia) (somente para
        administradores)
24 |  PARAMS:
25 |      <tag> - nome da tag que será associada à playlist
26 |
27 |  adiciona playlist <id playlist> <id musica> - Adiciona musica à playlist
         (somente para administradores)
28 |  PARAMS:
29 |      <id playlist> - id da playlist que terá um música adicionada.
30 |      <id musica> - id da música a ser adicionada na playlist.
```

```
01 | remove playlist <id playlist> - remove uma playlist (somente para
       administradores).
02 | PARAMS:
03 |     <id playlist> - id da playlist que será removida.
04 |
05 | mostra playlist <id playlist> - mostra as músicas de uma playlist.
06 | PARAMS:
07 |     <id playlist> - id da playlist que será mostrada.
```

5 Conclusão

Como um todo, o grupo classificou o desenvolver da disciplina de Banco de Dados como excelente. O grupo todo teve experiência muito satisfatória com todas as etapas desta — desde as aulas, aos atendimentos de monitoria e ao desenvolvimento do projeto.

Sobre as aulas, foram excelentes. A docente possui uma didática extraordinária e é extremamente empenhada em ensinar e atender seus alunos, além de ter grande domínio do assunto que ensina. A carga de trabalho da disciplina (duas provas e um trabalho prático) e seu método avaliativo estiveram coerentes na visão do grupo, não trazendo grandes dificuldades quanto a isso.

Sobre o trabalho prático, o grupo achou que sua complexidade está dentro do que foi esperado. O trabalho prático abrangeu todos os temas abordados na disciplina de maneira eficiente, permitindo ao grupo visualizar onde estes são aplicados. Além disso, os atendimentos e a relação com o monitor disponível foram ótimos e de grande ajuda no desenvolvimento do projeto.

Sobre as mudanças ocorridas neste ano, o grupo achou ótima a ideia de que as entregas parciais não compunham a nota final, e sim serviam como *feedback* ao grupo. Muitos conceitos no contexto de Bases de Dados são de difícil entendimento quando vistos das primeiras vezes. Isso permitiu com que esses conceitos fossem melhor estabelecidos ao decorrer do projeto — visto que era permitido realizar alterações nas partes anteriores — e evitou que o grupo fosse

prejudicado por estar tendo o contato inicial com o assunto.

Na disciplina, a única dificuldade que o grupo julgou pertinente foi a de lidar com as provas extremamente extensas que foram aplicadas. O conteúdo das provas era totalmente coerente, mas o tempo muitas vezes dificultava a execução dela.

Por fim, o grupo agradece pelo trabalho prestado por parte da docente e do monitor da disciplina, que capacitaram todos do grupo no assunto de Bases de Dados. O grupo, agora, se sente preparado para ser introduzido ao assunto em ambientes corporativos e/ou acadêmicos.

Referências

[ELM16] ELMASRI, R. e NAVATHE, S. B. *Fundamentals of Database Systems*. 2016.