Universidade de São Paulo
ICMC – Instituto de Ciências Matemáticas e de Computação

SSC0240 – Bases de Dados

Prof. Dra. Elaine Parros M. de Sousa
PAE: André Moreira Souza

Projeto – Inclusão Digital

PLATAFORMA DE ENSINO DE FUNDAMENTOS DE COMPUTAÇÃO PARA INCLUSÃO DIGITAL

Bruno Berndt Lima *12542550*
Daniel Henrique Lelis de Almeida *12543822*
Thiago Shimada *12691032*
Vinicius Kazuo Fujikawa Noguti *11803121*

São Carlos
30 de Junho de 2023

Sumário

1 Introdução

Tendo em vista o desafio de prover Inclusão Digital, principalmente para populações que não cresceram tendo acesso aos meios digitais, seja por falta de recursos e oportunidade, seja por questões temporais, propomos o desenvolvimento de uma plataforma educacional voltada para este público. A ideia é prover cursos e guias de simples acesso para guiar o uso dos meios digitais desde os primeiros fundamentos, promovendo instrução de uso, conscientização e uma futura independência digital.

Sendo assim, tomamos como inspiração uma das maiores plataformas educacionais de livre acesso hoje disponíveis, a *Khan Academy*, porém mudando seu propósito para além do mundo acadêmico e tentando enfatizar a acessibilidade àqueles que não estão habituados aos meios digitais.

2 Modelo Entidade-Relacionamento

2.1 Levantamento de Requisitos

Um usuário do sistema poderá utilizar a plataforma de forma anônima (não autenticada) ou de maneira identificável. Para que isso seja possível, um usuário pode se cadastrar de maneira simples ***(usuário autenticado)***, inserindo apenas um **nome de usuário** (3 a 16 caracteres), uma **senha** (que será armazenada como uma hash, estimando 144 caracteres necessários para sua persistência) e, opcionalmente, um **e-mail** (ocupando no máximo 254 caracteres). Além disso, para moderar a plataforma e atualizá-la com mais conteúdos, teremos também ***usuários administrativos***, esses são semelhante aos usuários convencionais, sendo diferenciados por um booleano **admin**, além disso todo administrador deve obrigatoriamente conter um e-mail cadastrado que será utilizado para autenticação em dois fatores.

Um *administrador* pode banir um usuário da plataforma, quando isso ocorre temos a entidade de ***banimento***, nela são indicados: o **responsável** (referência ao administrador) responsável pelo banimento, o **usuário banido** (referência ao usuário banido), o **horário** (data e hora de criação), a **validade** (data e hora em que o banimento expira) e, opcionalmente, a **causa** (texto) do banimento. Administradores não podem ser banidos.

A ideia é que esse seja um projeto dirigido e voltado para a comunidade, por conta disso é bastante importante que usuários consigam prover ***feedbacks***, sugerindo melhorias, apontando problemas encontrados e solicitando novos tópicos. Esses feedbacks, então, terão uma referência de qual é o **autor** (referência ao usuário que criou o feedback), o **horário** (data e hora de criação do feedback) a **mensagem** (texto de tamanho dinâmico, limitado arbitrariamente à 4096 caracteres) e, opcionalmente, a **data de fechamento** (data e hora quando o feedback foi fechado). Além disso, a fim de garantir a visualização dos feedbacks, temos uma entidade de ***visualização de feedback***, ela associa um **leitor** (referência ao usuário administrador) e um **feedback** (referência ao feedback) junto à informação da **data de visualização** (data e hora quando a visualização foi feita).

A fim de organizar o conteúdo, seguiremos uma divisão hierárquica bastante comum, seguindo em: categorias, tópicos, unidades e conteúdos, do maior para o menor, respectivamente. Primeiramente, teremos a ***categoria***, sendo o nível mais alto da hierarquia, ela é bastante simples, tendo apenas um **nome** (texto), uma **descrição** (texto) e um **ícone** (url). Em sequência, temos os ***cursos***, que contém um **nome** (texto), uma **descrição** (texto) e a **categoria** (referência à categoria). Seguindo adiante, teremos as ***unidades***, nosso próximo nível na hierarquia. As unidades seguem uma estrutura similar, contendo um **nome** (texto), uma **descrição** (texto), sua **ordem** (inteiro), que é usada para controlar a ordem em que as unidades são apresentadas, e o **curso** (referência à curso) a qual pertence. Por último temos os ***tópicos***, eles são o nível mais baixo na hierarquia organizacional, abrigando os conteúdos em si. Os tópicos seguem praticamente a mesma estrutura das unidades, ou seja, possuem um **nome** (texto), uma **descrição** (texto), uma **ordem** (inteiro) e, por fim, a **unidade** (referência à unidade) a qual pertencem.

Um dos principais elementos da plataforma, que seria a linha final desta hierarquia, são os

conteúdos, todos os conteúdos terão alguns campos em comum: um **título** (texto), um **subtítulo** (texto), uma **duração estimada** (inteiro indicando o número de minutos) e o **tópico** (referência ao tópico) a qual pertence. Além disso, inicialmente, teremos três tipos de conteúdo: ***artigos***, que seriam conteúdos primariamente textuais, contendo, além dos comuns entre os conteúdos, o **corpo** do artigo (que será um texto formatado em MD). Teremos também, os ***vídeos***, que contém a **url** do vídeo (texto) e uma **descrição** (texto) do vídeo. Por fim, teremos os ***exercícios***, eles são compostos de um **enunciado (corpo)** (texto formato em MD) e o **limite de seleções de alternativas** (inteiro indicando quantas alternativas um usuário pode selecionar). Além disso, os *exercícios* possuem um número variável de alternativas associadas, cada ***alternativa*** consiste de um **corpo** (texto formatado em MD), opcionalmente, uma **explicação** (texto formatado em MD), se é uma alternativa **correta** (booleano) e, por fim, o **exercício** (referência ao exercício) que faz parte.

Para *usuários autenticados*, será mantido um histórico do progresso dele na plataforma, isto é: para cada *conteúdo*, será registrado um ***evento de progresso***, indicando o **horário** (data e hora) onde foi feito o progresso, qual o **conteúdo** (referência ao conteúdo) que foi visitado e, caso o conteúdo seja um *exercício*, as **alternativas selecionadas** (referências às alternativas). Importante notar que as alternativas selecionadas precisam fazer parte do conteúdo do tipo exercício referente ao evento. Os eventos de progresso podem ser visualizados pelo usuário e podem existir mais de um evento para o mesmo conteúdo. O usuário pode deletar eventos antigos a fim de reiniciar o seu progresso de acordo com a granularidade desejada.

Além disso, *usuários autenticados*, podem deixar comentários em conteúdos, cada ***comentário*** é composto por um **corpo** (texto em MD), o **autor** (referência ao usuário que o criou), o **conteúdo** (referência ao conteúdo) em que o comentário foi feito, o **horário** (data e hora de criação) em que o horário foi feito, se ele é **visível** (booleano indicando se ele é exibido) para casos de deleção e, por fim, um comentário pode ser uma resposta, nesses casos temos o campo de **comentário pai** (referência opcional a um comentário) que indica qual comentário aquele responde. A presença desse mecanismo de respostas cria uma restrição onde precisamos garantir que, caso o comentário seja uma resposta, o conteúdo associado ao comentário pai seja o mesmo que o conteúdo referenciado presente em si. Além disso, a fim de evitar ciclos de relacionamento, precisamos garantir que, ao ir seguindo a cadeia de comentários pai, não haja a existência de um ciclo no momento da criação.

A fim de garantir a moderação da plataforma, *usuários autenticados* podem reportar comentários, para isso temos a entidade de ***report***. Cada report consiste de um **autor** (referência ao usuário que fez o report), o **comentário reportado** (referência ao comentário sendo reportado), o **horário** (data e hora de criação), o **motivo** (texto) e, por fim, o indicador de **verificado** (booleano), que indica que aquele report já foi avaliado por um administrador.

2.2 Funcionalidades

Analisando por cima dos atores do sistema, conseguimos sintetizar as funcionalidades da plataforma da seguinte forma:

- **Usuário Não Cadastrado**
  - Cadastrar-se
  - Navegar pelas categorias, cursos, tópicos, unidades e conteúdos
  - Consumir os conteúdos
  - Realizar exercícios
  - Visualizar comentários de um conteúdo
- **Usuário Cadastrado**
  - *(Todas de um Usuário Não Cadastrado)*
  - Autenticar-se
  - Alterar senha
  - Atualizar e-mail
  - Registrar (automaticamente) o progresso na plataforma
  - Visualizar o progresso atual
  - Reiniciar o progresso (de maneira granular)
  - Criar comentários em um conteúdo (caso não esteja banido)
  - Responder comentários existentes (caso não esteja banido)
  - Reportar um comentário (caso não esteja banido)
  - Deletar um próprio comentário
  - Enviar feedbacks (caso não esteja banido)
- **Usuário Administrador**
  - *(Todas de um Usuário Cadastrado)*
  - Inserir, remover e atualizar: categorias, cursos, tópicos, unidades, artigos, aulas, exercícios e alternativas
  - Deletar e atualizar comentários (de outros usuários)
  - Banir e perdoar usuários cadastrados (que não sejam administradores)
  - Visualizar os reports e comentários associados
  - Marcar reports de um comentário como resolvido (todos até o momento atual)
  - Visualizar feedbacks (histórico de visualização persistido)
  - Marcar (ou desmarcar) feedback como resolvido (mantendo a data de resolução)

2.3 Diagrama do Modelo Entidade-Relacionamento

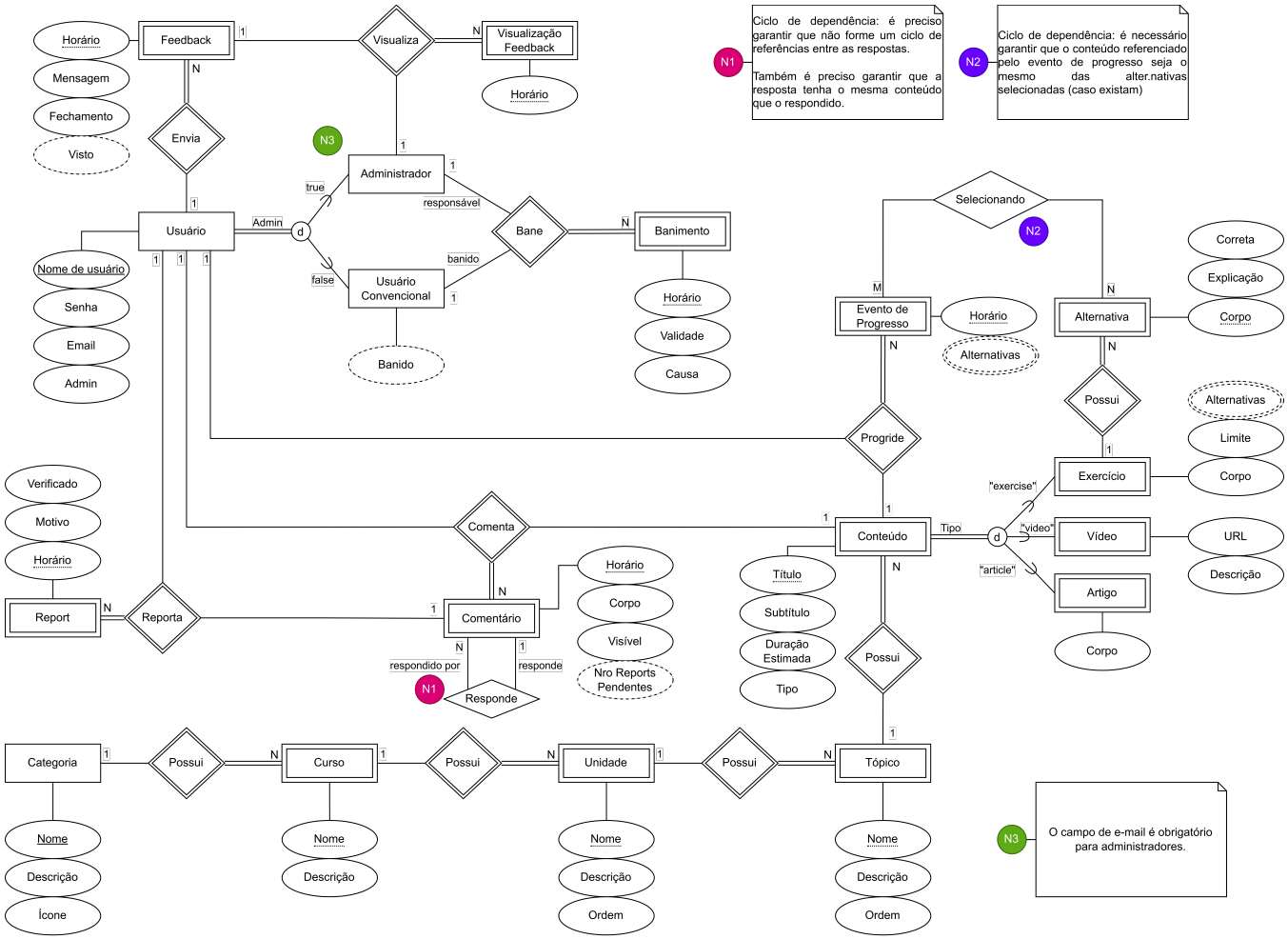

Figura 1 – Diagrama do Modelo Entidade-Relacionamento

2.4 Ciclos e Restrições de Integridade

A modelo proposto apresenta alguns ciclos naturais da solução e restrições de integridade necessárias, trataremos-as individualmente a seguir:

2.4.1 Ciclo: Comentário (Resposta) $\rightarrow$ Comentário (Respondido)

Este é um dos ciclos mais complicados do diagrama, ele se trata de um ciclo de dependência natural da modelagem, para solucioná-lo precisaremos garantir que duas restrições sejam satisfeitas:

1. Ao criar uma resposta, precisamos garantir que a referência de conteúdo do comentário respondido seja a mesma da resposta sendo criada.
2. Ao criar uma resposta, precisamos garantir que, ao seguir a cadeia de comentários pais, não haja repetição de comentários (evitar ciclos, ex: $C_1 \rightarrow\rightarrow C_2 \rightarrow C_3 \rightarrow C1$). É possível garantir isso ao impedir que o campo de comentário respondido seja editável.

2.4.2 Ciclo: Usuário $\rightarrow$ Comentário $\rightarrow$ Report

Este é um ciclo natural da modelagem, não é necessário nenhum tratamento especial, já que apenas indica a possibilidade (irônica) de um usuário reportar o próprio comentário.

2.4.3 Ciclo: Administrador $\rightarrow$ Feedback $\rightarrow$ Visualização Feedback

Mais um ciclo natural do diagrama, no sistema optamos por não restringir o administrador de fazer a maior parte das interações que um usuário convencional faria, sendo assim, caso um administrador crie um feedback, existe a possibilidade de que ele visualize o próprio feedback. Não é necessário nenhum tratamento adicional neste caso.

2.4.4 Ciclo: Conteúdo $\rightarrow$ Evento de Progresso $\rightarrow$ Alternativa

Este é um ciclo natural do diagrama, ele até poderia ser evitado por meio de uma especialização do Evento de Progresso, mas acreditamos que isso criaria uma complexidade desnecessária na modelagem sem benefícios práticos. Consistindo de um ciclo de dependência onde um Evento de Progresso, pode, opcionalmente, ter referências a alternativas selecionadas. Quando isso acontece, é necessário garantir que o conteúdo que a alternativa referencia (do tipo Exercício) seja o mesmo que é referenciado pelo Evento de Progresso, originando, então, uma restrição de integridade para que no momento de cadastro das alternativas associadas ao evento de progresso, seja verificada essa igualdade dos conteúdos referenciados.

2.4.5 Restrição adicional: Administradores precisam de um E-mail cadastrado

Para que um usuário se torne um administrador (campo admin igual a true), é necessário que seu campo de email não seja nulo, criando assim a necessidade de uma restrição de integridade adicional.

2.4.6 Restrição adicional: Administradores não podem ser banidos

Usuários administradores não podem ser banidos, por conta disso é necessário uma restrição de integridade que garanta que usuários banidos não se tornem administradores e que banimentos não sejam criados para administradores. É possível evitar isso também ao impedir que usuários existentes não se tornem administradores e nem administradores tornem-se usuários convencionais, mas não pretendemos seguir por este caminho.

2.5 Alterações para a segunda entrega

Durante a segunda entrega do projeto, foram feitas pequenas correções no MER apresentado na primeira entrega, sendo elas: correção na cardinalidade faltante no relacionamento “Progride”, correção na cardinalidade faltante no relacionamento “Bane” e adição da nota “N3” sobre a exigência de um e-mail não nulo para administradores.

3 Modelo Relacional

3.1 Mapeamento Modelo E-R para Modelo Relacional

A partir do MER apresentado no capítulo anterior, já com as devidas correções aplicadas, seguimos para o mapeamento do modelo E-R para o modelo relacional. Nele, tivemos que tomar algumas decisões específicas do contexto da aplicação, fazendo um julgamento entre as vantagens e desvantagens de cada uma das opções, principalmente no que se trata nos impactos de integridade, complexidade e performance. Seguiremos nas seções a seguir detalhando o diagrama final obtido e explicando as decisões tomadas.

3.2 Diagrama Modelo Relacional

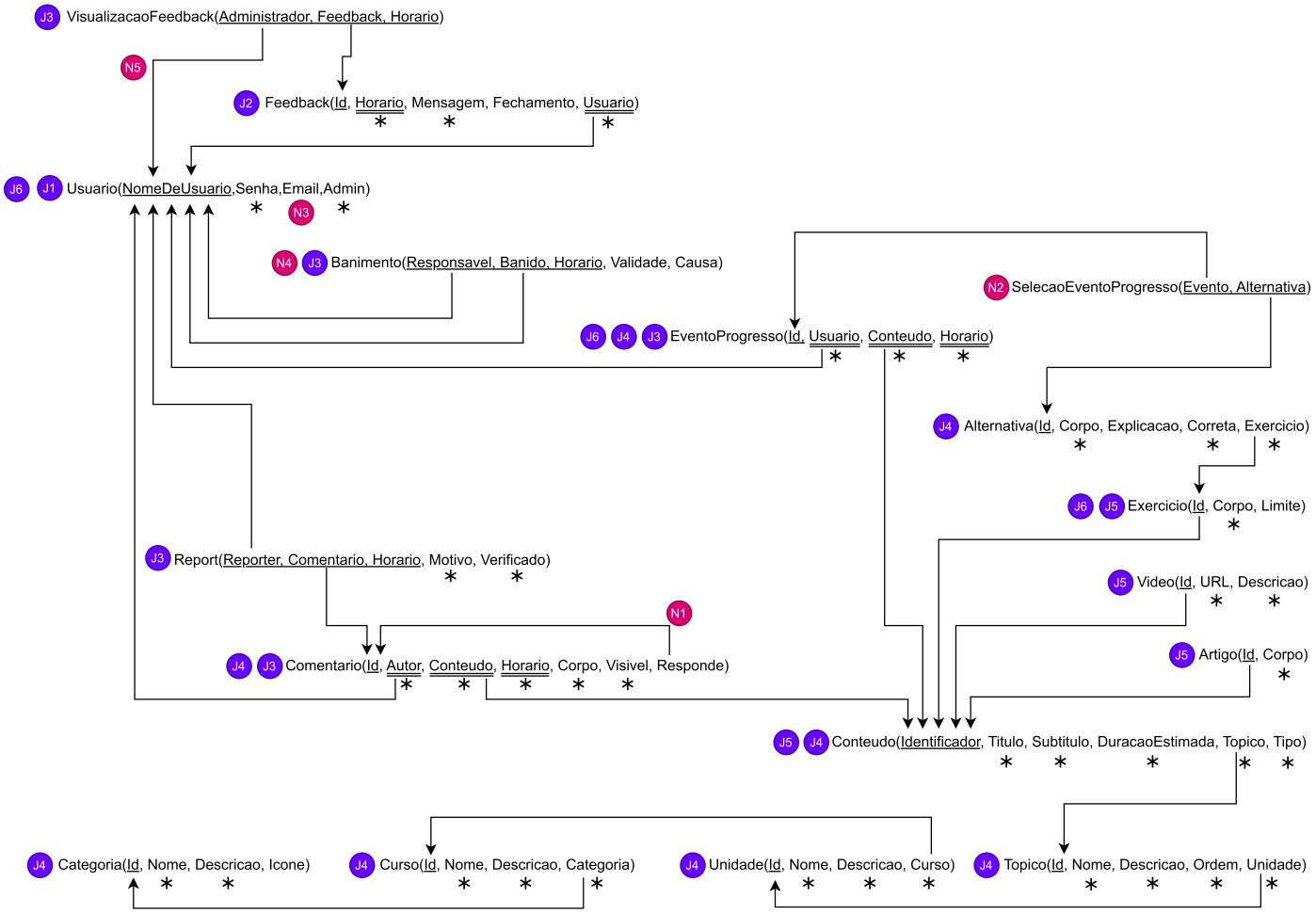

Figura 2 – Diagrama Modelo Relacional

3.3 Notas e Justificativas da Modelagem

3.3.1 **J1:** Especialização de “Usuário”

3.3.1.1 Solução Adotada

Para a especialização de “Usuário”, optamos por mapear as entidades em apenas uma tabela “Usuário”, adicionando um campo “admin” (booleano) para indicar se o usuário é um administrador ou um usuário convencional. Acreditamos que essa solução é a mais adequada para o contexto da aplicação, já que, os campos entre cada tipo de usuário são os mesmos, impactando apenas os relacionamentos específicos de cada tipo de usuário, o que irá levar a necessidade de algumas restrições adicionais em nível de aplicação a fim de garantir a integridade.

3.3.1.2 Vantagens

Redução dos recursos de armazenamento e processamento, já que evita a criação de tabelas adicionais para cada tipo de usuário (que não teriam campos específicos).

3.3.1.3 Desvantagens

A principal desvantagem dessa abordagem é a integridade dos relacionamentos específicos de cada tipo de usuário, levando a necessidade de restrições adicionais em nível de aplicação para garantir integridade.

3.3.1.4 Alternativa

Uma solução alternativa consiste em termos tabelas específicas para cada tipo de especialização de usuário, porém, acreditamos que isso criaria uma complexidade adicional desnecessária, além dos impactos em armazenamento e processamento. Além disso, mesmo se optássemos por essa solução, não conseguiríamos evitar completamente as restrições adicionais em nível de aplicação, já que, mesmo com tabelas específicas para cada tipo de usuário, ainda teríamos que garantir que um usuário não se torne administrador e vice-versa.

3.3.2 **J2:** Adição de identificador sintético em “Feedback”

3.3.2.1 Solução Adotada

A fim de se evitar a replicação de uma chave muito grande nos relacionamentos de feedback, optamos pela adição de um identificador sintético. Além do mais, o uso de horário juntamente à referência do usuário havia sido uma solução mais voltada a favorecer a semântica do MER do que uma real necessidade de garantir a unicidade da combinação deste par de campos, sendo assim não os adicionaremos como uma chave secundária.

3.3.2.2 Vantagens

Redução dos recursos de armazenamento necessários para manter o índice de chave primária e redução da replicação de dados nas chaves estrangeiras que referenciam “Feedback”.

3.3.2.3 Desvantagens

Para recuperar informações de um feedback, no contexto de uma chave estrangeira, será necessário realizar uma busca adicional na tabela “Feedback”. Além disso, o mesmo ocorre para filtros que envolvam os campos “horário” e “usuário”.

3.3.2.4 Alternativa

A alternativa principal consiste em manter a chave primária composta por “horário” e “usuário”, porém acreditamos que este caminho levará a um aumento de complexidade nas entidades identificadas por Feedback sem benefícios suficientes.

3.3.3 **J3:** Mapeamento de relacionamento identificador ternário 1:1:N

3.3.3.1 Solução Adotada

Os relacionamentos identificadores ternários 1:1:N foram mapeados por meio da inclusão dos campos identificadores das entidades identificadas pelo relacionamento, desta forma evitando a criação de tabelas adicionais sem prejuízos a integridade.

3.3.3.2 Vantagens

Evita os custos em termos de processamento e armazenamento de tabelas adicionais sem prejuízos a integridade.

3.3.3.3 Desvantagens

No contexto desta aplicação, não vemos desvantagens para a solução adotada.

3.3.3.4 Alternativa

A alternativa principal consiste na criação de uma tabela para cada um dos relacionamentos 1:1:N apontados, porém, como apontado acima, está solução apenas traria prejuízos dentro do contexto desta aplicação.

3.3.4 **J4:** Adição de identificadores sintéticos

3.3.4.1 Solução Adotada

Além da entidade de “Feedback”, algumas outras entidades do modelo relacional também receberam a adição de identificadores sintéticos a fim de evitar a replicação de chaves primárias muito grandes (geralmente textuais) nos relacionamentos identificadores. São os casos de

“Categoria”, “Curso”, “Unidade”, “Tópico”, “Conteúdo”, “Alternativa”, “Comentário” e “Evento de Progresso”.

3.3.4.2 Vantagens

Redução dos recursos de armazenamento necessários para manter o índice de chave primária e redução da replicação de dados nas entidades identificadas por relacionamentos com estas entidades.

3.3.4.3 Desvantagens

Assim como apontado em **J2**, para recuperar informações de uma entidade identificada por uma chave estrangeira, será necessário realizar uma busca adicional na tabela da entidade em questão. Além disso, o mesmo ocorre para filtros que envolvam os campos que compunham originalmente a chave primária.

3.3.4.4 Alternativa

A principal alternativa a este caso seria manter as chaves primárias compostas, porém isso levaria a um grande aumento nos custos de armazenamento e processamento, ainda mais quando levamos em consideração a cadeia de chaves estrangeiras que levariam entidades como “Conteúdo” a terem chaves compostas por 5 campos textuais, sendo 4 deles replicação de dados resultantes da chave estrangeira apontando a “Tópico”.

3.3.5 **J5:** Especialização de “Conteúdo”

3.3.5.1 Solução Adotada

Para a especialização de “Conteúdo”, optamos por mapear as entidades por meio da entidade pai “Conteúdo”, adicionando um campo “tipo” (enum) para indicar qual o tipo de conteúdo em questão, mantivemos também nessa tabela os campos em comum entre os conteúdos, que também são os mais relevantes em contextos de busca, poupando recursos de processamento quando comparado a isolar esses campos em cada sub-entidade. Para as especializações, optamos por criar uma entidade para cada uma, já que cada uma delas possui campos específicos que não são compartilhados entre todas, além de relacionamentos estritos a tipos de entidade. Dessa forma, conseguimos um bom balanço em garantir a integridade dos dados e evitar a replicação de dados.

3.3.5.2 Vantagens

A principal vantagem dessa solução é a garantia de integridade dos dados, já que evita a replicação de dados e manterá apenas os dados necessários.

3.3.5.3 Desvantagens

Em questão de desvantagens dessa solução, temos como principal problema a necessidade de processamento adicional para obter os dados específicos da especialização, porém isso não é de grande impacto para aplicação em questão.

Além disso, será necessário garantir uma restrição de integridade em nível de aplicação: garantir que um conteúdo não pode ser de mais de um tipo ao mesmo tempo e deve necessariamente existir um registro na tabela de especialização associada ao seu tipo. Isso porque a solução por si só em nível de modelagem não garante especialização total, nem disjunta.

3.3.5.4 Alternativa

Uma alternativa para este caso seria manter todos os campos de todas as especializações na entidade pai, porém isso levaria a um desperdício de recursos de armazenamento, já que teríamos campos nulos em parte dos registros. Além disso, as restrições de integridade dos relacionamentos teriam que ser garantidas em nível de aplicação.

3.3.6 **J6:** Abstenção de atributos derivados

3.3.6.1 Solução Adotada

A fim de evitar a necessidade de atualização de dados derivados, optamos por abster os mesmos do modelo relacional.

3.3.6.2 Vantagens

Maior garantia de consistência dos dados, já que não há a necessidade de atualização de dados derivados. Além disso, evita o armazenamento de dados redundantes.

3.3.6.3 Desvantagens

Para obter o valor dos atributos será necessário realizar uma busca adicional, o que irá aumentar o custo de processamento. Isso se tornaria principalmente problemático em contextos onde precisamos filtrar os dados por esses atributos, porém não será o caso desta aplicação.

3.3.6.4 Alternativa

A principal alternativa para este caso seria manter os atributos derivados no modelo relacional, porém, nesse caso, as desvantagens são muito mais impactantes do que as vantagens. Além disso, caso eventualmente se torne necessário, é possível contornar os custos de processamento por meio do uso de views, ou até mesmo adicionar os atributos derivados no modelo relacional.

3.3.7 **N1:** Ciclo de dependência da FK ‘Responde’ de “Comentário”

Será necessário, em nível de aplicação, garantir que o conteúdo referenciado na resposta seja o mesmo do comentário respondido. Além disso, é importante que não haja formação de um ciclo de respostas nos comentários.

3.3.8 **N2:** Ciclo de dependência das seleções de alternativas de “Evento de Progresso”

Será necessário, em nível de aplicação, garantir que as alternativas selecionadas em “SelecaoEventoProgreso” façam parte do conteúdo referenciado em “Evento de Progresso”.

3.3.9 **N3:** Administradores precisam de um e-mail cadastrado

Será necessário, em nível de aplicação, garantir que o campo de e-mail de administradores não seja nulo. Já que a entidade é compartilhada entre os dois tipos de usuário, não é possível garantir essa restrição no nível de modelagem relacional.

3.3.10 **N4:** Usuários administradores não podem ser banidos

Será necessário, em nível de aplicação, garantir que usuários administradores não sejam referenciados pelo campo ‘Banido’ de “Banimento”, já que administradores não deveriam poder ser banidos.

3.3.11 **N5:** Visualização de “Feedback” por administradores

Será necessário, em nível de aplicação, garantir que a referência ‘Administrador’ de “VisualizacaoFeedback” aponte apenas para Usuários administradores, já que temos a mesma entidade para usuários convencionais e administradores.

3.4 Alterações para a terceira entrega

Para a terceira entrega, adicionamos a justificativa de mapeamento dos atributos derivados do MER. Além disso, foi corrigido a ausência de algumas chaves secundárias que estavam ausentes no modelo relacional. Por fim, algumas justificativas tiveram o texto levemente modificado para melhorar a clareza e incluir detalhes que haviam ficado ausentes.

4 Implementação

4.1 Sobre a implementação

A implementação do projeto foi feita utilizando o SGDB PostgreSQL. Essa decisão foi feita por conta de ser um banco de dados de código aberto, com uma comunidade ativa e um conjunto bem completo de funcionalidades.

Além disso, o código da implementação foi feito em Python 3, sem o uso de nenhum framework. Algumas bibliotecas externas foram utilizadas para facilitar a implementação, são essas o *psycopg* (o 3) para a conexão com o banco de dados e o *termcolor* para a impressão de mensagens coloridas no terminal.

Além disso, é possível executar o projeto usando o *Pipenv* para instalação das dependência e gerenciamento do ambiente virtual. E, para o banco de dados, o *Docker Compose*, que irá criar um container com o banco de dados e executar os scripts SQL de criação do banco de dados e inserção de dados de teste. É importante atentar-se à configuração do ambiente usando o arquivo *.env*. A principal configuração necessária é a URI de conexão com o banco de dados, por meio da variável *DB_URI*.

As credenciais padrão para o banco de dados do *Docker Compose* são: usuário *bee*, senha *bee* e banco de dados *bee*.

4.2 Repositório

O código fonte do projeto está disponível no nosso repositório do GitHub. Os scripts SQL de criação do banco de dados e inserção de dados de teste estão disponíveis na pasta *sql*. O código fonte da implementação está disponível na pasta *app*.

4.3 As consultas

Foram desenvolvidas as seguintes consultas de complexidade média:

4.3.1 Consulta 1

Rendimento de um usuário nos exercícios. Caso não tenha feito nenhum, o rendimento é 0.

Listing 4.1 – Consulta 1

```
SELECT
  ep.usuario,
  COALESCE(
    (
      COUNT(
        CASE
          WHEN a.correta = TRUE THEN 1
        END
```

```
      ) :: NUMERIC / NULLIF(COUNT(a.correta), 0)
    ),
    0
  ) AS rendimento,
  COUNT(a.id) AS exercicios
FROM
  selecaoeventoprogresso sep
  JOIN alternativa a ON sep.alternativa = a.id
  RIGHT JOIN eventoprogresso ep ON ep.id = sep.evento
GROUP BY
  usuario;
```

4.3.2 Consulta 2

Seleciona pessoas cujo tempo gasto em exercícios é maior que a média geral.

Listing 4.2 – Consulta 2

```
SELECT
  ep.usuario,
  SUM(duracao) AS tempo
FROM
  eventoprogresso ep
  JOIN conteudo c ON c.id = ep.conteudo
WHERE
  c.tipo = 'E'
GROUP BY
  ep.usuario
HAVING
  SUM(duracao) >= (
    SELECT
      AVG(s.soma)
    FROM
      (
        SELECT
          SUM(duracao) AS soma
        FROM
          eventoprogresso
          JOIN conteudo cc ON cc.id = conteudo
          AND tipo = 'E'
        GROUP BY
          usuario
```

```
    ) s
);
```

4.3.3 Consulta 3

Conteúdos com mais reports em comentários.

Listing 4.3 – Consulta 3

```
SELECT
  ct.titulo,
  ct.subtitulo,
  ct.tipo,
  count(comentario) AS quantidade
FROM
  report r
  JOIN comentario c ON r.comentario = c.id
  JOIN conteudo ct ON c.conteudo = ct.id
GROUP BY
  ct.titulo,
  ct.subtitulo,
  ct.tipo
ORDER BY
  quantidade DESC;
```

4.3.4 Consulta 4

Interseção dos conteúdos vistos por usuários banidos e por usuários não banidos.

Listing 4.4 – Consulta 4

```
SELECT
  c.titulo,
  c.tipo
FROM
  (
    (
      SELECT
        conteudo
      FROM
        eventoprogresso ep
        JOIN (
          SELECT
            nome
```

```
            FROM
              usuario
            EXCEPT
            SELECT
              banido
            FROM
              banimento
          ) nb ON nb.nome = ep.usuario
      )
      INTERSECT
      (
        SELECT
          conteudo
        FROM
          eventoprogresso ep2
          JOIN (
            SELECT
              nome
            FROM
              usuario u
              JOIN banimento b ON b.banido = u.nome
          ) bn ON bn.nome = ep2.usuario
      )
  ) i
  JOIN conteudo c ON (i.conteudo = c.id)
ORDER BY
  length(c.titulo),
  c.titulo;
```

4.3.5 Consulta 5 - Divisão Relacional

Selecionar todos administradores que visualizaram o feedback de pelo menos os usuários que a admin “carolina456” visualizou.

Listing 4.5 – Consulta 5

```
SELECT
  DISTINCT administrador
FROM
  visualizacaofeedback vf
WHERE
  NOT EXISTS(
```

```
    (
      SELECT
        usuario
      FROM
        visualizacaofeedback vf2
        JOIN feedback f ON f.id = vf2.feedback
      WHERE
        vf2.administrador = 'carolina456'
    )
    EXCEPT
      (
        SELECT
          usuario
        FROM
          visualizacaofeedback vf3
          JOIN feedback f ON f.id = vf3.feedback
        WHERE
          vf.administrador = vf3.administrador
      )
)
```

4.4 Consultas da aplicação

4.4.1 Inserção de usuário

O código apresentado pode ser encontrado em *app/src/model/usuario.py* na linha 113.

Listing 4.6 – Inserção de usuário

```
def insert_usuario(usuario: Usuario) -> Usuario:
    """Insert a usuario into the database.

    Args:
        usuario (Usuario): The usuario to insert.

    Returns:
        Usuario: The inserted usuario.
    """

    if PASSWORD_RE.match(usuario.senha) is None:
        raise Exception('Senha inválida')

    usuario.senha = hash_password(usuario.senha)
```

```
    with get_conn_pool().connection() as conn:
        with conn.cursor() as cursor:
            try:
                cursor.execute('INSERT INTO usuario VALUES (%s, %s, %s, %s);',
                conn.commit()
                return usuario
            except IntegrityError as e:
                conn.rollback()
                for constraint, message in KNOWN_CONSTRAINTS.items():
                    if constraint in str(e):
                        raise Exception(message)
                raise Exception('Um erro inesperado ocorreu: ' + str(e))
            except Exception as e:
                print(repr(e), file=sys.stderr)
                conn.rollback()
                raise e
```

4.4.2 Obtenção de usuário por nome

O código apresentado pode ser encontrado em *app/src/model/usuario.py* na linha 146.

Listing 4.7 – Obtenção de usuário

```
def get_usuario(nome: str) -> Optional[Usuario]:
    """Get a usuario from the database by nome.

    Args:
        nome (str): The nome of the usuario to get.

    Returns:
        Optional[Usuario]: The usuario if it exists, None otherwise.
    """

    with get_conn_pool().connection() as conn:
        with conn.cursor() as cursor:
            cursor.execute('SELECT * FROM usuario WHERE nome = %s;', (nome,))
            row = cursor.fetchone()
            if row is None:
                return None
            return Usuario.from_tuple(row)
```

4.4.3 Listagem de todos os usuários

O código apresentado pode ser encontrado em *app/src/model/usuario.py* na linha 100.

Listing 4.8 – Listagem de usuários

```
def list_usuarios() -> List[Usuario]:
    """List all usuarios in the database.

    Returns:
        List[Usuario]: A list of all usuarios in the database.
    """

    with get_conn_pool().connection() as conn:
        with conn.cursor() as cursor:
            cursor.execute('SELECT * FROM usuario;')
            return [Usuario.from_tuple(row) for row in cursor.fetchall()]
```

4.4.4 Busca simples de usuários (por nome e/ou email)

O código apresentado pode ser encontrado em *app/src/model/usuario.py* na linha 84.

Listing 4.9 – Busca simples de usuários

```
def search_usuarios(query: str) -> List[Usuario]:
    """Search for usuarios in the database by a given query. The
    query will be tried to match against the textual fields "nome" and "email".

    Args:
        query (str): The query to search for.

    Returns:
        List[Usuario]: A list of usuarios that match the query.
    """

    with get_conn_pool().connection() as conn:
        with conn.cursor() as cursor:
            cursor.execute('SELECT * FROM usuario WHERE nome ILIKE %s OR email
            return [Usuario.from_tuple(row) for row in cursor.fetchall()]
```

5 Conclusão

Este projeto nos proporcionou uma valiosa oportunidade de aprofundar nosso conhecimento sobre o intrincado funcionamento de um banco de dados relacional. Pudemos praticar e aperfeiçoar nossas habilidades na modelagem de um banco de dados e na elaboração de consultas SQL. O percurso completo, desde a captação minuciosa dos requisitos até a implementação de uma aplicação integralmente funcional que utiliza o banco de dados, foi uma experiência altamente enriquecedora e repleta de aprendizado.

Contudo, a execução deste projeto não foi sem obstáculos. O principal desafio foi gerenciar esse empreendimento complexo enquanto navegávamos por uma miríade de outras atividades acadêmicas. Tarefas de outras disciplinas, avaliações e o fato de estarmos em um dos períodos mais exigentes do curso adicionaram uma camada extra de dificuldade à nossa gestão do tempo.

Mesmo assim, reconhecemos a importância vital deste trabalho para nossa formação. Talvez, se tivermos a liberdade de escolher um tema mais abrangente no futuro, possamos encontrar maneiras de integrá-lo com outras disciplinas. Essa abordagem interdisciplinar, embora não oficial, poderia aumentar a relevância e o interesse pelo projeto, tornando a experiência ainda mais significativa.

Referências

ELMASRI, R.; NAVATHE, S. B. *Fundamentals of Database Systems*. 7. ed. [S.l.]: Pearson, 2021.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Database Systems Concepts*. 7. ed. [S.l.]: McGraw-Hill Education, 2020.