Projeto Final da Disciplina de Banco de Dados (Bossow)

Antônio Vinicius de Moura - 190084502
Gabriel Mendes Ciriático - 202033202

Dep. Ciência da Computação - Universidade de Brasília (UnB)
Bancos de Dados

1. Introdução

A necessidade de guardar informações sempre esteve presente no cotidiano dos seres humanos, com o avanço da tecnologia surgiram diversos bancos de dados (BD) para armazená-las e conectá-las, no entanto, essa não é uma tarefa fácil e por isso existem diversos sistemas gerenciadores de banco de dados (SGBD) para auxiliar as consultas e manipulações. Cada banco de dados tem o seu propósito e a sua aplicação.

A ideia do projeto aqui apresentado é que seja um banco de dados de uma rede social de jogos, a Bossow. Essa rede social reunirá dados de usuários, jogos e consoles. Usuários possuem informações básicas de acesso (nome, usuário, senha, e-mail), assim como uma biblioteca de jogos adquiridos.

Os jogos adquiridos formam a biblioteca que contém informações sobre quantidade de horas jogadas e porcentagem completada do jogo. Os jogos se relacionam com consoles, onde é possível jogá-los. Usuários também podem publicar notícias no fórum da rede social, além de terem uma área de lembretes próprios, onde podem fazer anotações para não esquecer do que fazer.

Na Bossow, usuários também podem comentar jogos, dando notas, que são utilizadas para um ranking de jogos de acordo com as avaliações. Todas as imagens (fotos de perfil, fotos de notícias, capas de jogos, capturas de telas dos jogos) são armazenadas diretamente no banco de dados da rede social, como dado binário.

2. Diagrama de Entidade Relacionamento

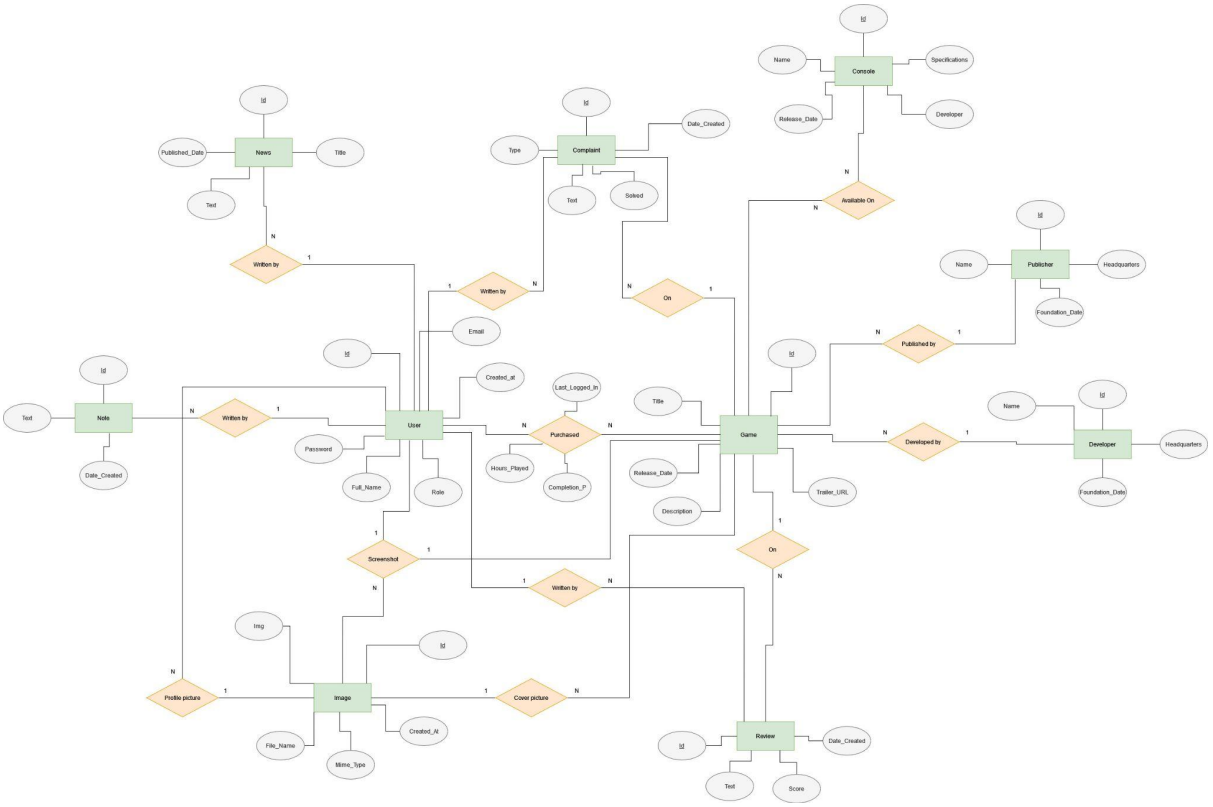

Figura 1. Modelo Entidade Relacionamento do Bossow, com 10 entidades. A imagem detalhada pode ser vista nos anexos do relatório.

3. Modelo Relacional

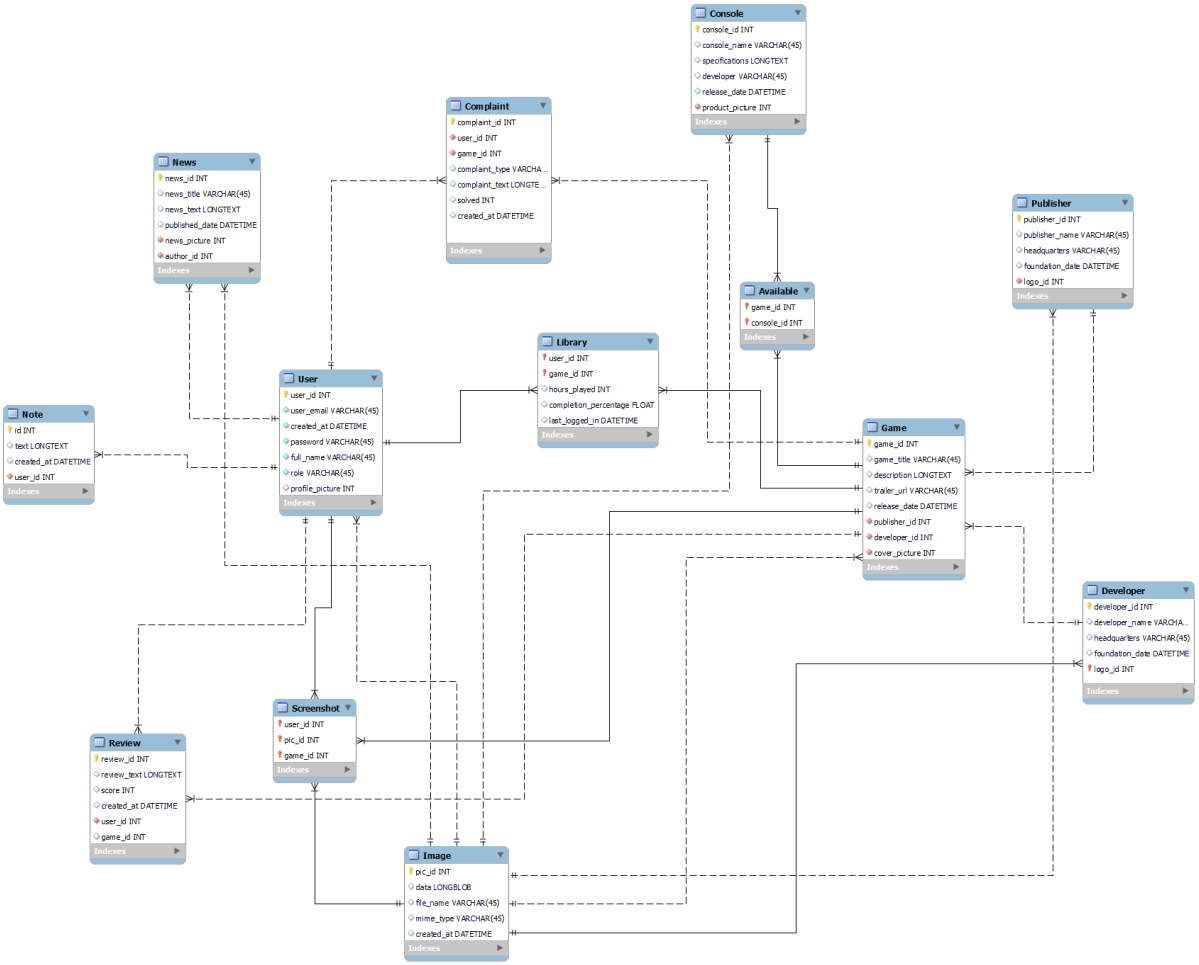

Figura 2. Modelo Relacional do projeto, feito a partir do Modelo Entidade Relacionamento, com as 10 entidades em tabelas, além das tabelas de relações entre entidades N:N.

4. Álgebra relacional

1) $\Pi_{\text{user\_id, user\_email, full\_name, game\_id, title, hours\_played}} (\sigma_{\text{(User.user\_id = Library.user\_id) AND (Game.game\_id = Library.game\_id)}} \text{(User x Game x Library)})$

Primeiro, é feita uma operação de produto cartesiano entre as tabelas *User*, *Game* e *Library*, retornando todas as combinações possíveis entre as linhas e colunas dessas tabelas. Depois, é feita uma operação de seleção, filtrando os dados para pegar as linhas em que *user_id* da tabela *User* é igual a *user_id* da tabela *Library* e *game_id* da tabela Game é igual a *game_id* da tabela *Library*. Isso retorna apenas as linhas onde há relação entre usuário e jogo com base no relacionamento da biblioteca de jogos. Depois disso, é feita uma operação de projeção, pegando-se apenas as colunas *user_id*, *user_email*, *full_name*, *game_id*, *title* e *hours_played*.

2) $\Pi_{\text{game\_title, Developer.developer\_name, Publisher.publisher\_name}} (\sigma_{\text{(Game.developer\_id = Developer.developer\_id) AND (Game.publisher\_id = Publisher.publisher\_id)}} (\text{Game x Developer x Publisher}))$

Primeiro, é feito o produto cartesiano entre as tabelas *Game*, *Developer* e *Publisher*, resultando em todas as combinações possíveis entre as células dessas tabelas. Depois, é feita operação de seleção, pegando-se apenas as linhas onde o *developer_id* em *Game* é igual ao *developer_id* em *Developer* e o *publisher_id* em *Game* é igual ao *publisher_id* em *Publisher*. Isso seleciona apenas as linhas dos jogos com seus respectivos desenvolvedores e publicadores. Por fim, é feita operação de projeção, pegando-se apenas o game_*title* do *Game*, o *developer_name* de *Developer* e o *publisher_name* de *Publisher*.

3) $\Pi_{\text{game\_id, title, user\_email, full\_name, text}} (\sigma_{\text{(User.user\_email = Complaint.user\_email) AND (Game.game\_id = Complaint.game\_id)}} (\text{User x Game x Complaint}))$

Primeiro, é feito o produto cartesiano entre as tabelas *User*, *Game* e *Complaint*, obtendo-se todas as combinações possíveis entre as linhas dessas tabelas. Depois, é feita uma seleção que filtra *user_email* de *User* igual a *user_email* de *Complaint* e *game_id* de *Game* igual a *game_id* de *Complaint*. Com isso, tem-se as linhas apenas das reclamações feitas, de fato, por usuários a jogos, de acordo com a relação estabelecida em *Complaint*. Por fim, é aplicada a operação de projeção, pegando-se as colunas *game_id*, *title*, *user_email*, *full_name* e *text*.

4) $\Pi_{\text{user\_id, full\_name, note\_text,created\_at, data}} ((\text{User} \bowtie \text{Note}) \bowtie \text{Image})$

Primeiro, é feito um join natural entre as tabelas *User* e *Note*, que se conectam através da coluna *user_id*. Depois, é feita a junção dessa relação resultante com a tabela Image, o que se dá através de *profile_picture* da junção entre *User* e *Note* e *pic_id* de *Image*. Por fim, é realizada uma operação de projeção, pegando-se as colunas *user_email*, *full_name*, *text*, *created_at* e *data*. Com isso, tem-se a visão geral de informações de lembretes por usuário, junto com o arquivo binário da sua foto de perfil.

5) $\Pi_{\text{user\_id, full\_name, game\_id, game\_title, review\_text}} ((\text{User} \bowtie \text{Review}) \bowtie \text{Game})$

Primeiro, é feito um join natural entre as tabelas *User* e *Review*, que se conectam através de *user_id*. Com essa relação, é feito um outro join natural com *Game*, o que é possível através da coluna *game_id*. Por fim, é feita uma operação de projeção, pegando-se as colunas *user_id*, *full_name*, *game_id*, *game_title* e *review_text*. Com isso, temos linhas com informações das análises, dos usuários que publicaram-nas e dos jogos comentados.

5. Avaliação das formas normais

A normalização de um banco de dados é um processo que aplica diferentes fórmulas nas tabelas visando evitar redundância - o que deixa o banco mais rápido de ser acessado, bem como mais leve para ser armazenado.

As formas normais são sequenciais, com as próximas dependendo das anteriores para existirem. Embora haja uma quantidade praticamente infinita de formas normais, neste projeto será feita a avaliação de apenas 3 formas normais, que podem ser resumidas como abaixo:

- **Primeira Forma Normal (1FN):** O valor de uma coluna de uma tabela é indivisível. Isto é, uma célula só pode ter um valor;
- **Segunda Forma Normal (2FN):** Tabela precisa estar 1FN e todo atributo do complemento de uma chave candidata é totalmente funcionalmente dependente daquela chave;
    - As chaves candidatas definem unicidade na tabela - é um conceito próximo das chaves primárias. Atributo complemento da chave são todas as outras chaves da tabela.
- **Terceira Forma Normal (3FN):** Tabela precisa estar 2FN e todos os atributos não-chave devem ser dependentes não-transitivos da chave primária.
    - Isto é, não pode haver uma chave sendo definida por outra que não a primária, já que isso implica transitividade.

Passemos para a análise de 5 tabelas do modelo relacional feito: User, Library, Complaint, Image e Screenshot.

5.1. User

- **1FN:** Está satisfeita, já que é uma tabela onde cada célula comporta apenas um valor, com atributos atômicos;
- **2FN:** A tabela já está 1FN. Agora, tomando *user_email* como chave candidata, é possível ver que todo atributo do complemento é totalmente funcionalmente dependente da candidata. Assim, *created_at*, *password*, *full_name*, *role* e *profile_picture* são totalmente funcionalmente dependentes de *user_email* ;
- **3FN:** A tabela já está 2FN. Agora, tomando *user_email* como chave primária, é possível observar que todas as outras chaves são dependentes não-transitivos dela, sendo independentes entre si (*created_at*, *password*, *full_name*, *role* e *profile_picture* são independentes entre si e dependentes de *user_email*).

5.2. Library

- **1FN:** Está satisfeita, já que é uma tabela onde cada célula comporta apenas um valor, com atributos atômicos;

- **2FN:** A tabela já está 1FN. Agora, tomando *user_email* e *game_id* como chaves candidatas, é possível ver que todo atributo do complemento é totalmente funcionalmente dependente das candidatas. Assim, *hours_played*, *completion_percentage* e *last_logged_in* têm valores que dependem apenas de *user_email* e *game_id* juntos;
- **3FN:** A tabela já está 2FN. Agora, tomando *user_email* e *game_id* como chaves primárias, é possível observar que todas as outras chaves são dependentes não-transitivos dela, sendo independentes entre si (*hours_played*, *completion_percentage* e *last_logged_in* não dependem uma da outra, mas dependem todas de *user_email* e *game_id*).

Complaint

- **1FN:** Está satisfeita, já que é uma tabela onde cada célula comporta apenas um valor, com atributos atômicos;
- **2FN:** A tabela já está 1FN. Agora, tomando *complaint_id* como chave candidata, é possível ver que todo atributo do complemento é totalmente dependente da candidata. Assim, *user_email*, *game_id*, *type*, *text*, *solved* e *date_created* são totalmente funcionalmente dependentes de *complaint_id*;
- **3FN:** A tabela já está 2FN. Agora, tomando *complaint_id* como chave primária, é possível observar que todas as outras chaves dependentes não-transitivos dela, já que são independentes entre si, mas dependentes da primária (*user_email*, *game_id*, *type*, *text*, *solved* e *date_created* não dependem entre si, dependem apenas de *complaint_id*).
    - Observe que um usuário qualquer pode fazer uma denúncia de qualquer jogo, mesmo que ele não esteja na sua biblioteca, o que é uma escolha de negócio feita. Se o usuário só pudesse reclamar de um jogo adquirido, então essa tabela não estaria 3FN, já que *game_id* dependeria de *user_email*.

5.3. Image

- **1FN:** Está satisfeita, já que é uma tabela onde cada célula comporta apenas um valor, com atributos atômicos;
- **2FN:** A tabela já está 1FN. Agora, tomando *pic_id* como chave candidata, é possível ver que nem todo atributo do complemento é totalmente dependente da candidata. No caso, embora *data*, *file_name* e *created_at* sejam totalmente dependetes de *pic_id*, *mime_type* depende de *data*, já que é definido pelo navegador com base no conteúdo dos dados binários do arquivo;
- **3FN:** A tabela não está 2FN, logo não é possível estar 3FN.

5.4. Screenshot

- **1FN:** Está satisfeita, já que é uma tabela onde cada célula comporta apenas um valor, com atributos atômicos;

- **2FN:** A tabela já está 1FN. Agora, tomando *pic_id*, *user_email* e *game_id* como chaves candidatas, não há nenhum outro atributo para não ser totalmente dependente delas. Logo, está 2FN;
- **3FN:** A tabela já está 2FN. Agora, tomando *pic_id*, *user_email* e *game_id* como chaves primárias, não há nenhuma outra chave para não ser dependentes não-transitivos delas. Logo, está 3FN.

6. Camadas para acesso ao banco de dados

O projeto desenvolvido é construído em cima de três camadas: de apresentação, onde há a interface com o usuário; a de negócios, onde o que o usuário insere é validado segundo as regras definidas durante a elaboração do projeto; e a de persistência, que faz o contato direto com o banco de dados, persistindo os dados.

No diagrama abaixo, é possível ver como é feito o mapeamento das camadas para o banco de dados no caso de registro de uma conta na rede social. A tela da interface é montada no HTML *sign_up.html*, que é acessado e modificado pelo usuário. As informações ali inseridas vão para a camada de negócios, onde são checadas informações como tamanho de senha, validade de e-mail inserido etc. Por fim, é criado um usuário através do *Data Access Object* de usuário, conectado diretamente ao banco de dados.

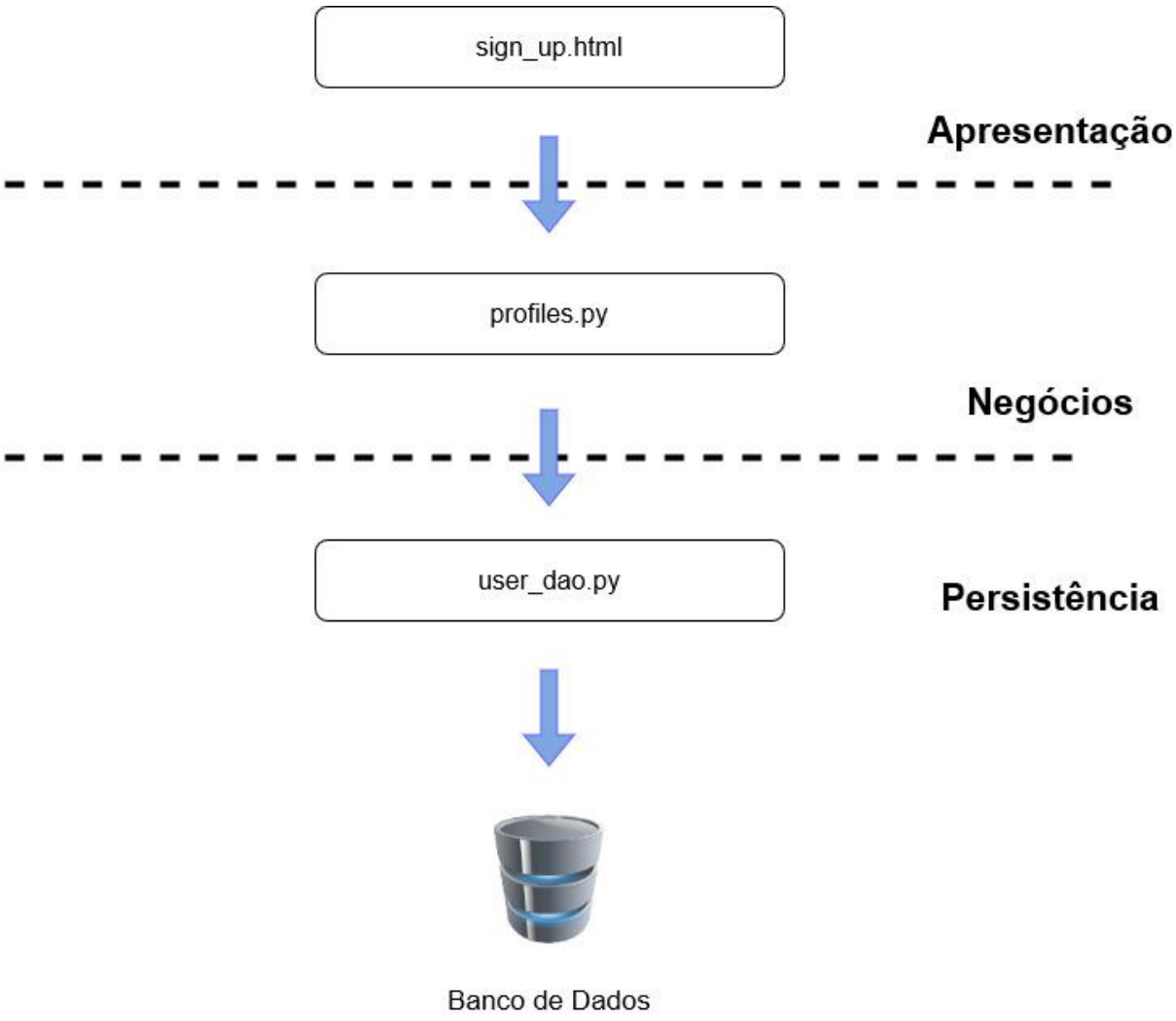

Figura 3. Camadas do projeto: de apresentação, responsável pela interface com o usuário; de negócios, com as regras de negócio; e de persistência, que faz o contato com o banco de dados.

7. Repositório Github

Para acessar todos os códigos utilizados no projeto, assim como o script que gerou o banco de dados, basta clicar no link: https://github.com/ciriatico/bossow_bd_2021_2.