UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA DE BANCO DE DADOS I — PROFESSOR RONALDO CORREA

VOXLOG

GUSTAVO BECELLI DO NACIMENTO (201250241)
DANIEL HENRIQUE SEREZANE PEREIRA (201250047)

PRESIDENTE PRUDENTE – SP
08 DE AGOSTO DE 2022

Índice

Especificação do problema

O Last.fm [1] é uma rede social/serviço que permite aos usuários registrarem cada música que ouvem em seus dispositivos, a partir de uma sincronização do aplicativo que usam para ouvir música com o serviço Last.fm. Este serviço então exibe as informações extraídas das músicas de maneira organizada, focando em apresentar a atividade recente e explicitar as preferências do usuário. Em um perfil, é apresentado as últimas músicas ouvidas pelo usuário, seus gêneros musicais e artistas mais ouvidos em um período (desde sempre, no último mês, na última semana, etc.), quantas horas o usuário passou ouvindo música em um período do dia, dentre outras informações que permitem ao usuário compreender e compartilhar sua relação com a música, comparando suas preferências com a de outros usuários. Ademais, o usuário pode explorar os metadados – dados sobre as músicas ouvidas, e seus respectivos álbuns e artistas – da plataforma, descobrindo novos artistas e álbuns, ou até mesmo obras ainda desconhecidas de um artista que o usuário já gosta.

Outra funcionalidade desta plataforma é o registro de eventos: um usuário informa que algum evento (como um concerto) ocorrerá em algum local e data, e outros usuários podem informar que também comparecerão ao evento, sendo então possível criar encontros e/ou organizar subeventos.

Contudo, o projeto Last.fm encontra-se praticamente abandonado, não recebendo uma atualização considerável desde 2015. Sua API, que permite aos usuários extrair dados relevantes da plataforma, encontra-se igualmente defasada, e possui grande *downtime* (frequentemente está indisponível).

Os metadados coletados pela plataforma – dados sobre as músicas ouvidas, e seus respectivos álbuns e artistas – possuem alta inconsistência, pois não há filtragem alguma realizada sobre eles. Por exemplo, o Last.fm não permite a existência de dois artistas com o mesmo nome — caso isso ocorra, os álbuns e músicas ficam misturados, causando grande confusão ao usuário que está visualizando os metadados.

Por fim, o Last.fm é uma empresa, exigindo pagamento por muitos de seus serviços — por exemplo, não é possível visualizar um resumo da atividade do usuário no último mês sem adquirir uma assinatura na plataforma. O código do software por trás do Last.fm é, portanto, fechado.

A proposta do projeto aqui apresentado é, portanto, oferecer uma alternativa com base em software livre ao Last.fm, oferecendo as mesmas funcionalidades — registro e visualização das músicas ouvidas pelo usuário e criação/interação com eventos — porém, solucionando os problemas acima apresentados. O nome associado ao projeto foi **VOXLOG**.

Ao distribuir o software gratuitamente com código aberto e livre, os problemas de defasagem e alto *downtime* da API são automaticamente solucionados, visto que estas questões ficam a cargo da própria comunidade e/ou usuários do software, assim como a questão dos serviços pagos, que deixam de existir. Contudo, a questão da inconsistência dos dados exige uma outra abordagem: para solucioná-la, o VOXLOG sincronizará periodicamente seus dados com o

MusicBrainz [2], um extenso banco de dados, construído via software livre, que coleta dados sobre obras musicais e seus respectivos autores, e que possui uma ampla comunidade que verifica e corrige (remove duplicatas, padroniza escrita de nomes, etc.) os dados. Contudo, a ideia é que os dados do VOXLOG não se misturem com o do MusicBrainz, para evitar inconsistências. Afinal, o MusicBrainz é atualizado manualmente e, mesmo sendo frequentemente incrementando, pode ocorrer algo como um usuário ouvir uma música que ainda não está no MusicBrainz. Deste modo, o que o VOXLOG propõe é uma tentativa de associação: para cada nova música, álbum e artista introduzido ao VOXLOG, ele tentará encontrar um equivalente no MusicBrainz; caso o faça com sucesso, ele importará os dados desejados (por exemplo, para um álbum, o título do álbum) do MusicBrainz, e guardará o identificador daquela entidade no banco MusicBrainz, pois, caso o usuário deseje saber mais informações a respeito daquela entidade (por exemplo, visualizar todas as músicas de um álbum), estas serão extraídas do MusicBrainz, e não do banco de dados do VOXLOG. O VOXLOG ainda guardará, contudo, dados mínimos para exibir ao usuário caso uma sincronização com o MusicBrainz não seja possível. O VOXLOG ainda propõe que alguns segmentos do MusicBrainz sejam baixados à máquina onde o VOXLOG está sendo executado, para que os dados possam ser acessados mais rapidamente, sem depender da API do MusicBrainz, bastante lenta.

Ressalta-se que este projeto foi desenvolvido exclusivamente para a disciplina de Banco de Dados I, portanto, muitos detalhes da implementação do software ainda não foram discutidos, logo não serão aqui apresentados.

Tratando-se, então, da modelagem dos dados, torna-se claro que basta modelar as entidades que interagem e são armazenadas pelo software: o usuário; as músicas; os álbuns (que contém as músicas); os artistas (que criam os álbuns); e os eventos, nos quais artistas participam e pessoas comparecem. Também são relevantes informações sobre quando o usuário ouviu alguma música, para que uma linha do tempo possa ser criada. O ato de ouvir uma música é chamado pelo Last.fm de “scrobble”, e este será o nome aqui adotado.

De cada uma dessas entidades, deseja-se armazenar:

- Usuário: nome de usuário, e-mail, data de nascimento, data de criação da conta, resumo biográfico (opcional) e nome real (opcional).
- Scrobble: data e horário onde a música terminou de ser ouvida.
- Música: título e duração.
- Álbum: título.
- Artista: nome.
- Evento: nome, descrição, URL (para mais informações sobre o evento e/ou compra de ingressos), data e horário de início, data e horário de término e o Plus Code [3] do local do evento.

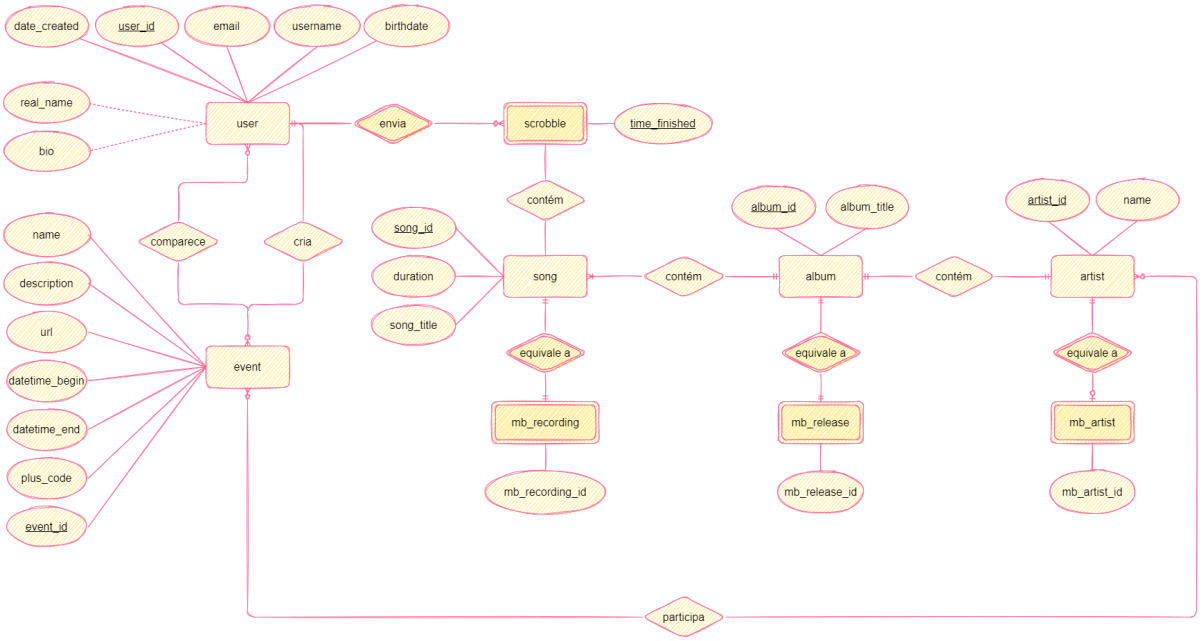
A relação entre as entidades é melhor apresentada no modelo conceitual, em seguida.

Tratando-se das principais consultas a serem respondidas, estas seriam justamente relativas às funcionalidades acima apresentadas, respondendo, por exemplo: quais as músicas que um

usuário ouviu na última semana, qual o artista mais ouvido da plataforma, quais artistas estarão em um evento, quem vai em um evento, dentre outras questões.

Esquema Conceitual

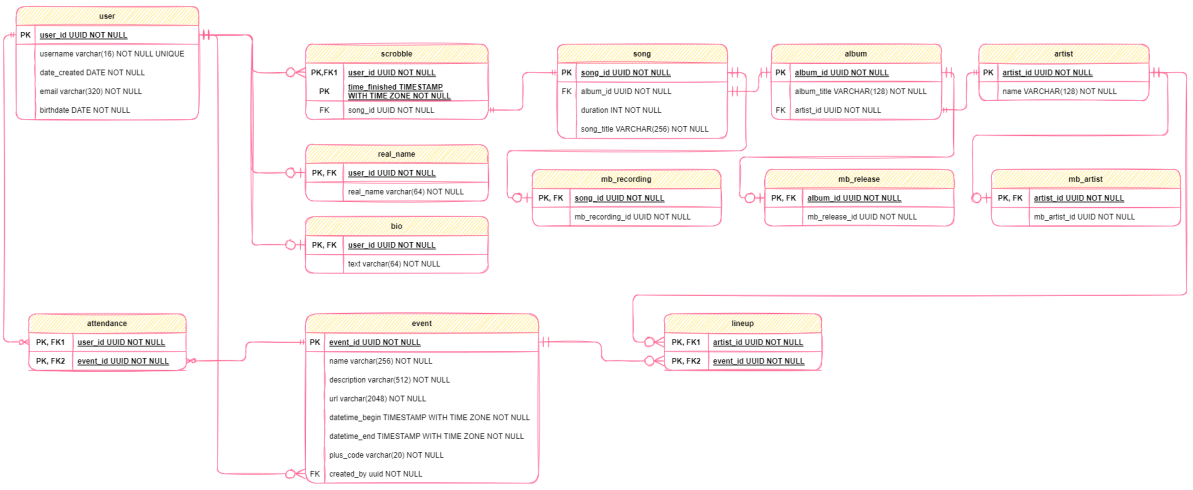
Tanto no esquema conceitual, quanto no relacional, foi utilizada a notação de cardinalidade "pé de galinha", aprendida em sala de aula.



Para uma melhor visualização do esquema conceitual, utilize o PDF, enviado conjuntamente a este documento.

Esquema Relacional

Os tipos de dados aqui apresentados referem-se ao SGBDR PostgreSQL, utilizado na implementação da base de dados.



Para uma melhor visualização do esquema conceitual, utilize o PDF, enviado conjuntamente a este documento.

Normalização

(i) As relações já estão na 1FN, pois:
- Todos os atributos de todas as tabelas já são monovalorados.
- Todos os atributos são atômicos. O nome real de um usuário (real_name), por escolha de implementação, será uma *string* única.

(ii) As relações já estão na 2FN, pois:
- Estão na 1FN, como estabelecido acima.
- Não há dependência funcional total em relação alguma.

(iii) As relações já estão na 3FN, pois:
- Estão na 2FN, como estabelecido acima.
- Não há dependência funcional transitiva em relação alguma.

Portanto, conclui-se que a modelagem do problema foi bem realizada, visto que já resultou em um esquema relacional normalizado.

Especificação de Consultas em Álgebra Relacional e SQL

Aqui estão especificadas cinco consultas úteis ao conhecimento dos dados do banco, que seriam corriqueiras em uma execução real do VOXLOG. Nos exemplos onde são necessários UUIDs [4], são usados valores genéricos 'xxxx'. As consultas também estão disponíveis em scripts/queries.sql em [5]. Foi necessário a utilização de algumas extensões da álgebra relacional para representar funções agregadas – seguindo as sugestões de [6] e [7], estabeleceu-se: γ para GROUP BY, ℑ para SUM e τ para ORDER BY DESC LIMIT 1 (único tipo de ORDER BY utilizado).

Consulta 1 – Calcular quantos segundos de música um usuário ouviu, no total

Álgebra Relacional

```
π ℑ(song.duration) (γ song.duration (scrobble ⊠ (scrobble.song_id = song song_id ^ scrobble.user_id = "xxxx") song))
```

SQL

```sql
SELECT SUM(song.duration) FROM scrobble INNER JOIN song ON (scrobble.song_id = song.song_id AND scrobble.user_id = 'xxxx') GROUP BY song.duration;
```

Consulta 2 - Obter o UUID, nome e tempo de música ouvida do artista mais ouvido da plataforma (em tempo de música ouvida).

Álgebra Relacional

```
π artist_id, name, total_time (π ρ
artist_id(artist.artist_id), ρ name (artist.name), ρ
total_time (ℑ (duration)) (τ (γ artist_id, duration (scrobble
* song * album * artist))))
```

SQL

```
SELECT s.artist_id, s.name, s.total_time FROM
(SELECT artist.artist_id, artist.name AS name,
SUM(song.duration) AS total_time FROM scrobble
NATURAL JOIN song
NATURAL JOIN album
NATURAL JOIN artist
GROUP BY artist.artist_id, song.duration) AS s
ORDER BY total_time DESC LIMIT 1;
```

Nota: o PostgreSQL exige o estabelecimento de apelidos quando se faz o SELECT de um SELECT, por isso o "as s" no SQL, que não aparece no query em álgebra relacional.

Consulta 3 - Obter dados completos (todos os dados padrão, nome real e resumo biográfico) de um usuário.

Álgebra Relacional

```
user * real_name * bio
```

SQL

```
SELECT * FROM "user" NATURAL JOIN real_name NATURAL JOIN bio;
```

Consulta 4 - Obter todos os artistas que estarão em um evento.

Álgebra Relacional

```
π artist.artist_id, artist.name (artist ⊠ ((artist.artist_id
= lineup.artist_id) ^ (lineup.event_id = "xxxx")) lineup)
```

SQL

```
SELECT artist.artist_id, artist."name" FROM artist INNER JOIN
lineup ON (artist.artist_id = lineup.artist_id AND
lineup.event_id = 'xxxx');
```

Consulta 5 - Obter todas as pessoas que estarão em um evento.

Álgebra Relacional
`π user.user_id, user.username (user ⊠ ((user.user_id = attendance.user_id) ^ (attendance.event_id = "xxxx")) attendance)`
SQL
`SELECT "user".user_id, "user".username FROM "user" INNER JOIN attendance ON ("user".user_id = attendance.user_id AND attendance.event_id = 'xxxx');`

Implementação da base de dados

A base de dados foi implementada em **PostgreSQL 14**.

O *script* para criação da base de dados em scripts/create.sql, em [5]. Nele são criadas as tabelas estabelecidas no esquema relacional, com suas devidas restrições.

O *script* para criação da base de dados em scripts/feed.sql, em [5]. Nele, aproveita-se das funcionalidades do PostgreSQL, como a criação de funções, laços de repetição, variáveis e *arrays*, para inserir dados fictícios preestabelecidos na base de dados.

Para consultar a base de dados, pode-se utilizar qualquer um dos *queries* estabelecidos acima (também disponíveis em scripts/queries.sql em [5]), contudo, ressalta-se que, para as operações que requerem um UUID, será necessário obtê-lo antes, pois a geração dos mesmos estabelece para eles valores aleatórios. Para obter os UUIDs pode-se fazer outra consulta, ou utilizar uma interface gráfica (recomenda-se o DBeaver).

Protótipo (interfaces)

Uma interface gráfica foi disponibilizada para melhor compreensão da aplicação e está disponível na web [8]. Na vigência da escrita deste documento, esta página da web possui o esboço de um perfil de usuário e é servida estaticamente, alimentada com dados fictícios com a finalidade de demonstração do design adotado e ilustração da experiência do usuário ao utilizar nossos serviços. A implementação da interface gráfica está disponível em client em [5].

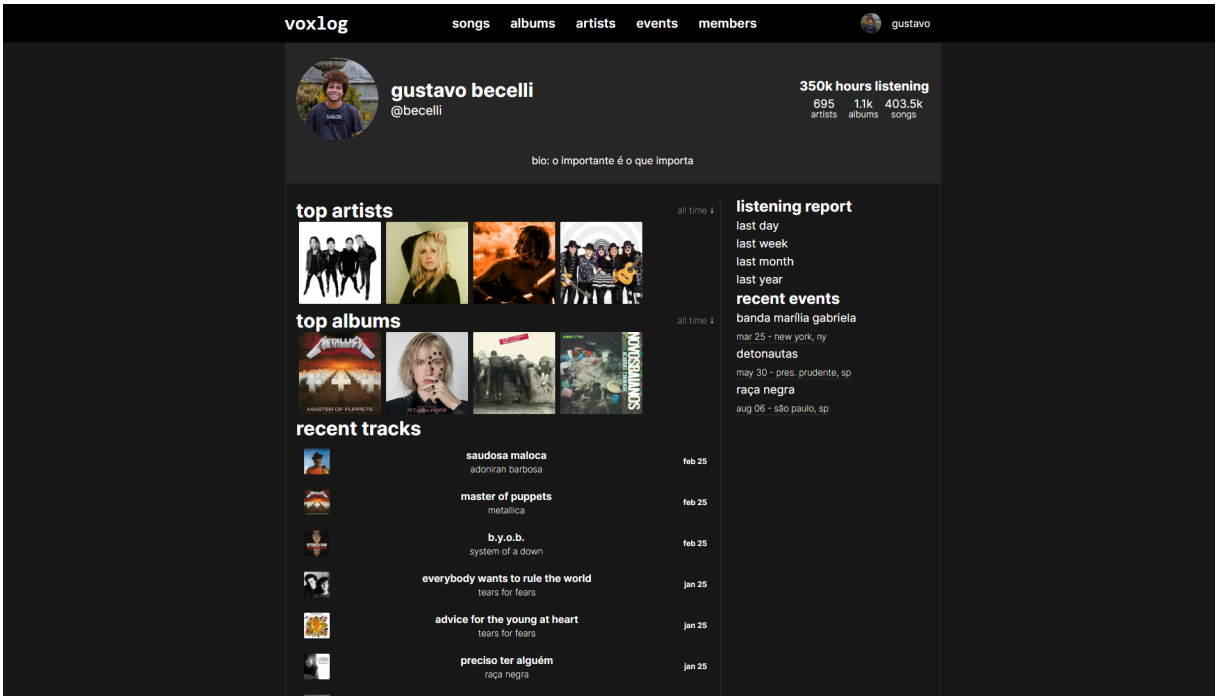


Figura 1. *Captura de tela da aplicação com modo escuro ativado, em um navegador web com a saída de vídeo em proporção 16:9 e resolução 1920 × 1080. Fonte: autores.*

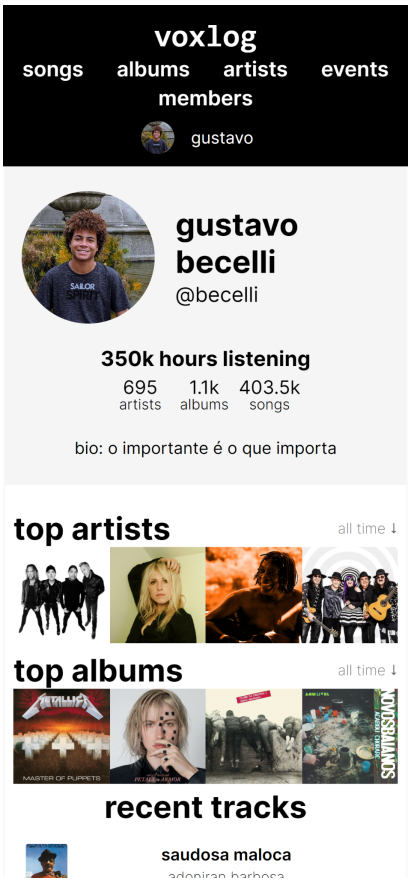


Figura 2. Captura de tela da aplicação, em um navegador web simulando a tela de um iPhone 12 Pro. Fonte: autores.

Referências

[1] Last.fm. Disponível em: https://www.last.fm/. Um perfil de exemplo pode ser visto em: https://www.last.fm/user/Salies.

[2] MusicBrainz. Disponível em: https://musicbrainz.org/.

[3] Google Maps — Plus Codes. Disponível em: https://maps.google.com/pluscodes/.

[4] RFC 4122 – Universally Unique Identifier. Disponível em: https://www.rfc-editor.org/rfc/rfc4122.html.

[5] Voxlog. Disponível em: https://github.com/Salies/voxlog.

[6] Grouping in relational algebra with more than one grouping attribute. Disponível em: https://dba.stackexchange.com/questions/134865/grouping-in-relational-algebra-with-more-than-one-grouping-attribute.

[7] Algebra Relational sql GROUP BY SORT BY ORDER BY. Disponível em: https://stackoverflow.com/questions/28665635/algebra-relational-sql-group-by-sort-by-order-by.

[8] VOXLOG. Disponível em: https://voxlog.vercel.app/profile