FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Music Streaming Service

Bases de Dados

Docente Lázaro Costa

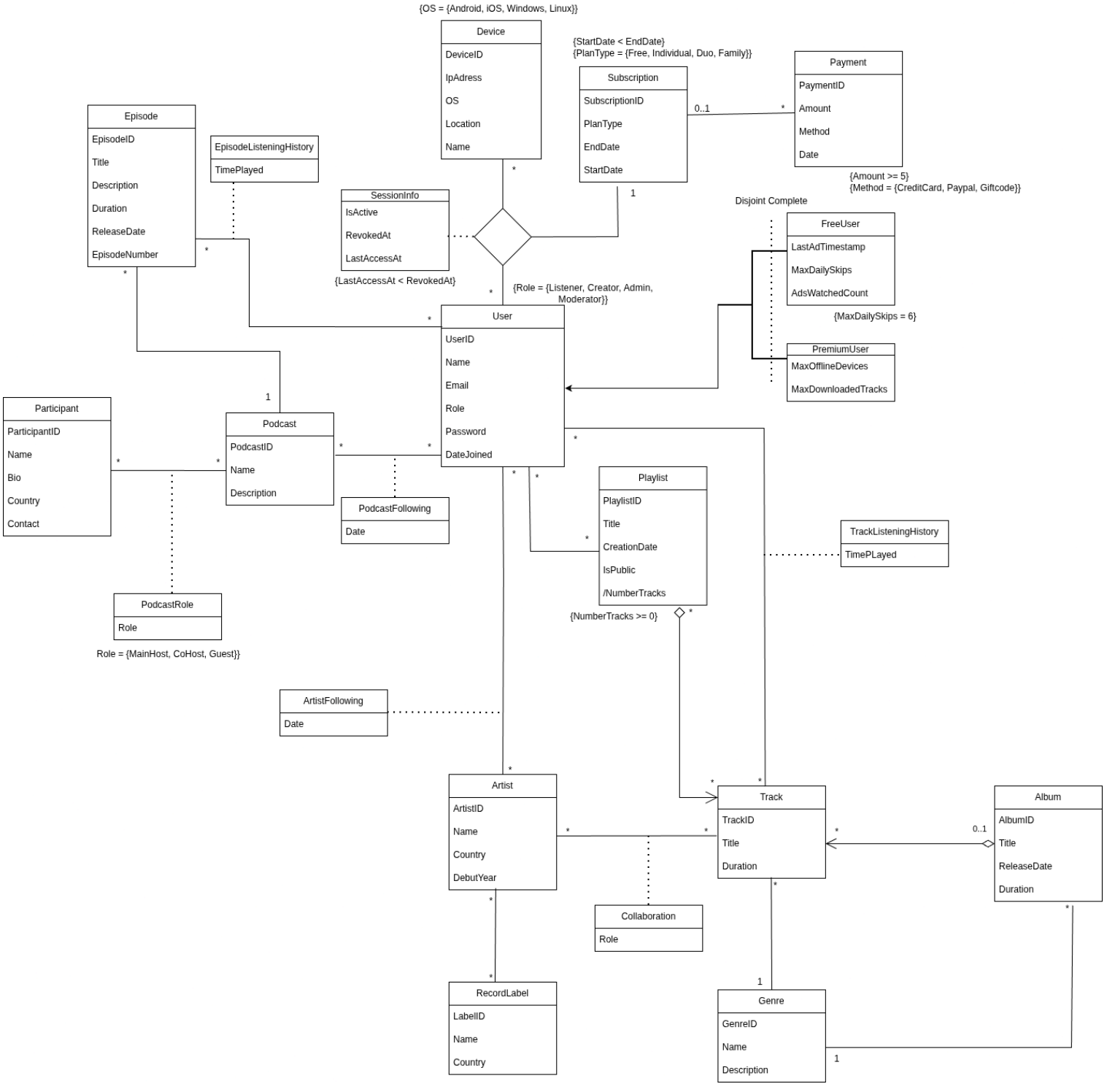
Ricardo Silva, 202004990
Leandro Moreira, 202405528
Ana Ferreira, 202403904

Novembro, 2025

Índice

Índice **2**
1. Modelo Conceptual Refinado **3**
2. Esquema Relacional **4**
2.1. Proposta Inicial 4
2.2. Correção do Modelo com AI Generativa 5
2.2.1. Principais mudanças implementadas 6
2.3. Esquema Relaciona Final (AI) 8
3. Dependências funcionais e Análise das Formas Normais **9**
3.1. Análise das Dependências Funcionais e Formas Normais 9
3.2. Análise das Dependências Funcionais e das Formas Normais com AI Generativa. 11
3.2.1. Principais mudanças implementadas 12
4. Criação da Base de Dados com SQLite **13**
4.1. Proposta Inicial 13
4.2. Análise da Base de Dados SQLite com AI Generativa 13
4.2.1. Principais mudanças implementadas 14
5. Inserção de Dados **16**
5.1. Proposta Inicial 16
5.2. Análise das Inserções na Base de Dados com AI Generativa 16
5.2.1. Principais mudanças implementadas 17
6. Análise Crítica Sobre a Integração de AI **18**
Pontos Fortes 18
Pontos Fracos 18
7. Contribuição dos Elementos do grupo **19**
Primeira Submissão 19
Segunda Submissão 19

1. Modelo Conceptual Refinado



Fonte: https://drive.google.com/file/d/1kdJhaPFMgfobjVmRRJ9_gYwk4n1NjqX8/view?usp=sharing (2ª página)

2. Esquema Relacional

2.1. Proposta Inicial

User (UserID, Name, Email, Password, DateJoined)
Artist (ArtistID, Name, Country, DebutYear)
UserFollowsArtist (UserID->User, ArtistID->Artist, Date)
Podcast (PodcastID, Host, Description)
UserFollowsPodcast (UserID->User, PodcastID->Podcast, Date)
Device (DeviceID, IpAddress, OS, Location)
Subscription (SubscriptionID, PlanType, EndDate, StartDate)
UserDeviceSubscription (UserID->User, DeviceID,->Device, SubscriptionID->Subscription, IsActive, RevokedAt, LastAcessAt)
FreeUser (UserID, LastAdTimestamp, MaxDailySkips, AdsWatchedCount)
PremiumUser (UserID, MaxOfflineDevices, MaxDownloadedTracks)
Track (TrackID, Title, Duration, AlbumID->Album)
UserListenedToTrack (UserID->User, TrackID->Track, TimePlayed)
Playlist (PlaylistID, Title, CreationDate, IsPublic, NumberOfTracks)
TrackInPlaylist (PlaylistID->Playlist, TrackID->Track)
UserCreatePlaylist (UserID->User, PlaylistID->Playlist)
ArtistPublishesTrack (ArtistID->Artist, TrackID->Track, Role)
RecordLabel (LabelID, Name, Country)
ContractWithLabel (ArtistID->Artist, LabelID->RecordLabel)
Genre (GenreID, Name, Description)
TrackOfGenre (TrackID->Track, GenreID->Genre)
Album (AlbumID, Title, ReleaseDate, Duration)
AlbumOfGenre (AlbumID->Album, GenreID->Genre)
TrackAddedToPlaylist (TrackID->Track, PlaylistID->Playlist)
Participant (ParticipantID, Name, Bio, Country, Email)
ParticipatesInPodcast (ParticipantID->Participant, PodcastID->Podcast, PodcastRole)
Episode (EpisodeID, Title, Description, Duration, ReleaseDate, EpisodeNumber, PodcastID->Podcast)
Payment (PaymentID, Amount, Method, Date, SubscriptionID->Subscription)
UserListensEpisode(UserID->User, EpisodeID->Episode, TimePlayed)

2.2. Correção do Modelo com *AI* Generativa

Com vista a otimizar o esquema relacional desenvolvido e minimizar inconsistências na conversão do diagrama *UML* para o modelo relacional, recorreu-se à *AI*. O seguinte *prompt* foi utilizado para análise e validação do modelo:

```text
Pretendo que me ajudes a detetar falhas, pequenos erros ou inconsistências no processo de conversão de um diagrama UML para um esquema relacional.
A análise será feita de forma gradual: irei enviar imagens do UML em partes, e quero que avalies se a conversão está correta.
Sê claro e direto nas respostas, apontando tanto erros relevantes como detalhes menores, como uso incorreto de maiúsculas, nomes inconsistentes ou incoerências de nomenclatura.
Começaremos com o esquema relacional e, em seguida, irei enviando pequenos trechos do diagrama UML para comparação e validação.
```

Fig. 1 - Análise de Erros no Esquema Relacional - *Prompt dado à Perplexity AI*

Ao longo desta etapa do projeto, reparámos que as ferramentas de *AI* apresentavam limitações quando confrontadas com a imagem do diagrama *UML* completo, isto é, com elevada densidade de informação. Verificámos que ao submeter imagens com muitos elementos em simultâneo, a ferramenta tinha dificuldade em processar todos os detalhes com precisão, resultando em erros de interpretação ou na omissão de elementos relevantes.

Por esta razão, optámos por uma abordagem de análise fragmentada e iterativa. Dividimos o diagrama *UML* em segmentos mais pequenos e comparámo-los individualmente com o esquema relacional.

2.2.1. Principais mudanças implementadas

Nomenclatura Errada e Inconsistente

As classes associativas apresentavam nomes completamente diferentes, por exemplo, *Collaboration* no *UML* aparecia como *ArtistPublishesTrack* no esquema relacional, e *SessionInfo* como *UserDeviceSubscription*.

Identificámos também, que as relações *TrackAddedToPlaylist* e *TrackInPlaylist* representavam a mesma relação de muitos para muitos entre *Track* e *Playlist*, causando redundância desnecessária.

Além disso, a *UserCreatesPlaylist* apresenta um problema. O nome sugere que o utilizador cria a *playlist*, mas numa relação onde vários utilizadores estão ligados a várias playlists, nem todos eles necessariamente criaram a mesma. Por este motivo, decidimos renomear a tabela para *UserAccessPlaylist,* um nome mais abrangente que representa o facto de que um utilizador pode ter acesso a uma *playlist*, independentemente de a ter criado ou não.

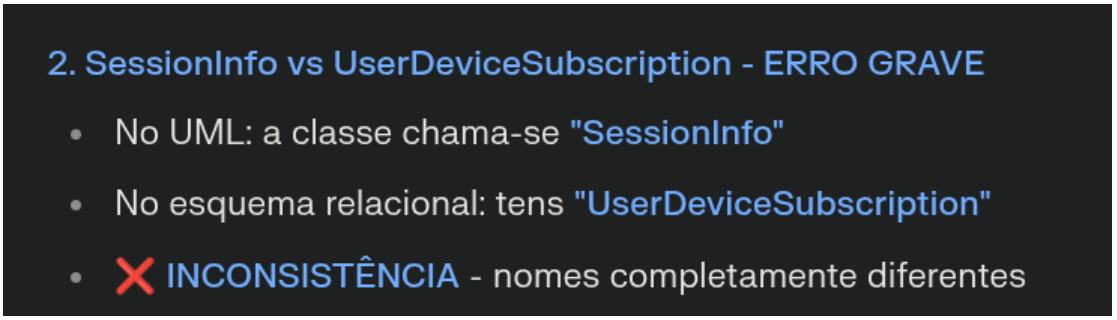


Fig. 2 - Inconsistência - Resposta da *Perplexity IA*

Atributos em Falta/Incorretos

Vários atributos presentes no *UML* não foram transpostos para o esquema, por exemplo: *User* - faltava o atributo *"Role"*; *Podcast* - faltava o *"Name"*. Outros eram diferentes: *Podcast* tinha *"Host"* no esquema mas deveria ter *"Name"*; *Participant* tinha *"Email"* em vez de ter *"Contact".* Adicionalmente, a *AI* sugeriu a remoção dos atributos derivados do esquema relacional, no entanto, após consultar o docente, optámos por mantê-los.

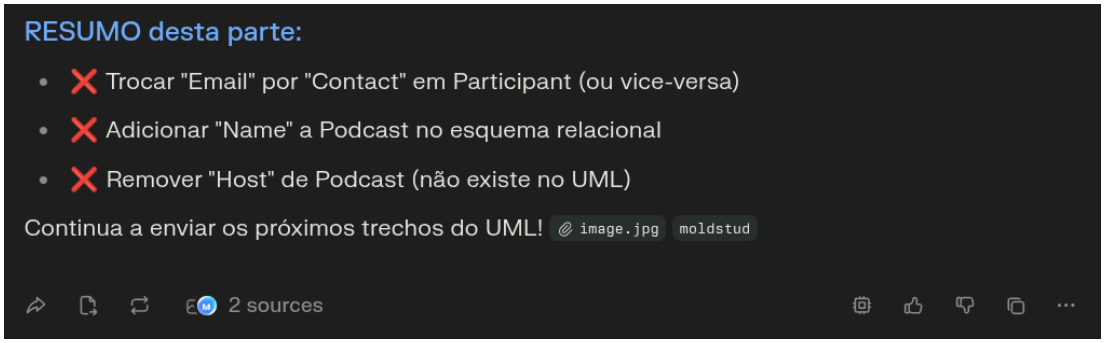


Fig. 3 - Atributos errados - Resposta da *Perplexity IA*

Erros de Escrita

Encontrámos erros como vírgulas a mais e palavras mal escritas, como *"LastAcessAt"* deveria ser *"LastAccessAt"*).

O registo completo da interação com a *Perplexity AI* encontra-se disponível no seguinte link:

https://www.perplexity.ai/search/relational-model

2.3. Esquema Relaciona Final (*AI*)

User (UserID, Name, Email, Role, Password, DateJoined)
Artist (ArtistID, Name, Country, DebutYear)
ArtistFollowing (UserID->User, ArtistID->Artist, Date)
Podcast (PodcastID, Name, Description)
PodcastFollowing (UserID->User, PodcastID->Podcast, Date)
Device (DeviceID, IpAddress, OS, Location)
Subscription (SubscriptionID, PlanType, EndDate, StartDate)
SessionInfo (UserID->User, DeviceID->Device, SubscriptionID->Subscription, IsActive, RevokedAt, LastAccessAt)
FreeUser (UserID->User, LastAdTimestamp, MaxDailySkips, AdsWatchedCount)
PremiumUser (UserID->User, MaxOfflineDevices, MaxDownloadedTracks)
Track (TrackID, Title, Duration, AlbumID->Album)
TrackListeningHistory (UserID->User, TrackID->Track, TimePlayed)
Playlist (PlaylistID, Title, CreationDate, IsPublic, NumberOfTracks)
TrackInPlaylist (PlaylistID->Playlist, TrackID->Track)
UserAccessPlaylist (UserID->User, PlaylistID->Playlist)
Collaboration(ArtistID->Artist, TrackID->Track, Role)
RecordLabel (LabelID, Name, Country)
ContractWithLabel (ArtistID->Artist, LabelID->RecordLabel)
Genre (GenreID, Name, Description)
TrackOfGenre (TrackID->Track, GenreID->Genre)
Album (AlbumID, Title, ReleaseDate, Duration)
AlbumOfGenre (AlbumID->Album, GenreID->Genre)
Participant (ParticipantID, Name, Bio, Country, Contact)
PodcastRole (ParticipantID->Participant, PodcastID->Podcast, Role)
Episode (EpisodeID, Title, Description, Duration, ReleaseDate, EpisodeNumber, PodcastID->Podcast)
Payment (PaymentID, Amount, Method, Date, SubscriptionID->Subscription)
EpisodeListeningHistory (UserID->User, EpisodeID->Episode, TimePlayed)

3. Dependências funcionais e Análise das Formas Normais

3.1. Análise das Dependências Funcionais e Formas Normais

Começamos por analisar cada uma das relações e fizemos as dependências funcionais para cada uma delas:

Artist (ArtistID, Name, Country, DebutYear)
ArtistID -> Name, Country, DebutYear

ArtistFollowing (UserID->User, ArtistID->Artist, Date)
UserID, ArtistID -> Date

Podcast (PodcastID, Name, Description)
PodcastID -> Name, Description

PodcastFollowing (UserID->User, PodcastID->Podcast, Date)
UserID, PodcastID -> Date

Subscription (SubscriptionID, PlanType, EndDate, StartDate)
SubscriptionID -> PlanType, EndDate, StartDate

FreeUser (UserID->User, LastAdTimestamp, MaxDailySkips, AdsWatchedCount)
UserID -> LastAdTimestamp, MaxDailySkips, AdsWatchedCount

PremiumUser (UserID->User, MaxOfflineDevices, MaxDownloadedTracks)
UserID -> MaxOfflineDevices, MaxDownloadedTracks

Track (TrackID, Title, Duration, AlbumID->Album)
TrackID -> Title, Duration, AlbumID

TrackListeningHistory (UserID->User, TrackID->Track, TimePlayed)
UserID, TrackID -> TimePlayed

Playlist (PlaylistID, Title, CreationDate, IsPublic, NumberOfTracks)
PlaylistID -> Title, CreationDate, IsPublic, NumberOfTracks

TrackInPlaylist (PlaylistID->Playlist, TrackID->Track)
PlaylistID, TrackID -> PlaylistID, TrackID

UserAccessPlaylist (UserID->User, PlaylistID->Playlist)
UserID, PlaylistID -> UserID, PlaylistID

Collaboration(ArtistID->Artist, TrackID->Track, Role)
ArtistID, TrackID -> ArtistID, TrackID, Role

RecordLabel (LabelID, Name, Country)
LabelID -> Name, Country

ContractWithLabel (ArtistID->Artist, LabelID->RecordLabel)
ArtistID, LabelID -> ArtistID, LabelID

TrackOfGenre (TrackID->Track, GenreID->Genre)
TrackID -> GenreID

Album (AlbumID, Title, ReleaseDate, Duration)
AlbumID -> Title, ReleaseDate, Duration

AlbumOfGenre (AlbumID->Album, GenreID->Genre)
AlbumID -> GenreID

PodCastRole (ParticipantID->Participant, PodcastID->Podcast, Role)
ParticipantID, PodcastID -> Role

Episode (EpisodeID, Title, Description, Duration, ReleaseDate, EpisodeNumber, PodcastID->Podcast)
EpisodeID -> Title, Description, Duration, ReleaseDate,EpisodeNumber, PodcastID

Payment (PaymentID, Amount, Method, Date, SubscriptionID->Subscription)
PaymentID -> Amount, Method, Date, SubscriptionID

EpisodeListeningHistory (UserID->User, EpisodeID->Episode, TimePlayed)
UserID, EpisodeID -> TimePlayed

Depois de analisar as dependências funcionais do esquema relacional, podemos concluir que este já está de acordo com a 3ª Forma Normal e com a Forma Normal de *Boyce Codd*, visto que, na parte esquerda de cada uma das dependências encontramos uma *superkey*. Os únicos aspetos a realçar nesta etapa do projeto, foi a descoberta de relações que contêm mais chaves candidatas para além daquelas que propusemos, nomeadamente, as seguintes relações:

User (UserID, Name, Email, Role, Password, DateJoined)
UserID -> Name, Email, Role, Password, DateJoined
Email -> UserID, Name, Role, Password, DateJoined

Device (DeviceID, IpAddress, OS, Location)
DeviceID -> IpAddress, OS, Location
IpAdress -> OS, Location, DeviceID

SessionInfo (UserID->User, DeviceID->Device, SubscriptionID->Subscription, IsActive, RevokedAt, LastAccessAt)
UserID, DeviceID -> SubscriptionID, IsActive, RevokedAt, LastAcessAt

Genre (GenreID, Name, Description)
GenreID -> Name, Description
Name -> Description, GenreID
Description -> GenreID, Name

Participant (ParticipantID, Name, Bio, Country, Contact)
ParticipantID -> Name, Bio, Country, Contact
Contact -> ParticipantID, Name, Bio, Country

3.2. Análise das Dependências Funcionais e das Formas Normais com *AI* Generativa

Começamos por enviar o seguinte *prompt* à *AI* com o intuito de encontrar possíveis falhas na nossa análise das dependências funcionais e das formas normais:

```text
Identifica todas as dependências funcionais presentes em cada relação,
classificando-as como triviais ou não triviais, e determina as chaves candidatas
de cada relação. Verifica se cada relação está em 3NF analisando a existência de
dependências transitivas e dependências parciais. Para a BCNF, verifica se para
cada dependência funcional não trivial X → Y, o X é uma superchave, e identifica
casos onde uma relação está em 3NF mas não em BCNF.
Aponta possíveis anomalias de inserção, atualização e eliminação, assim como
redundância de dados. Para cada problema identificado, propõe decomposições
concretas que garantam a preservação de dependências funcionais e a propriedade de
junção sem perdas. Apresenta o esquema normalizado final com justificações claras
para cada alteração, usando exemplos específicos do esquema fornecido para
ilustrar os pontos críticos da análise.
```

Fig. 4 - Normalização e Dependências Funcionais - *Prompt dado à Perplexity AI*

3.2.1. Principais mudanças implementadas

No âmbito de encontrar possíveis falhas, a *AI* foi utilizada sobretudo como apoio na validação da análise já realizada, não tendo sido propostas alterações ao esquema relacional no que diz respeito à identificação de dependências funcionais, nem ao nível de normalização (3FN/BCNF) das relações. A *AI* limitou-se a confirmar que, assumindo apenas as dependências induzidas por chaves primárias e estrangeiras, as relações encontram-se adequadamente normalizadas e que não existem casos em que fosse necessária uma decomposição adicional por possíveis violações.

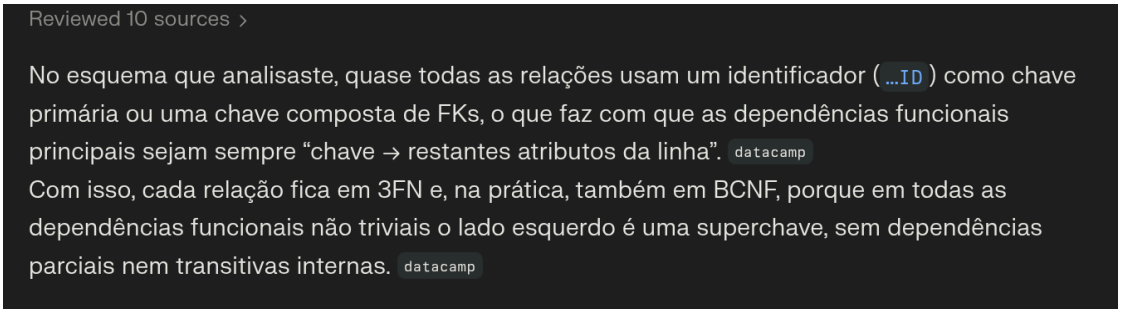


Fig. 5 - Sem Melhorias Aparentes - Resposta da *Perplexity IA*

O registo completo da interação com a *Perplexity AI* encontra-se disponível no seguinte link:

https://www.perplexity.ai/search/dependencies-analysis

4. Criação da Base de Dados com *SQLite*

4.1. Proposta Inicial

O script **create1.sql**, disponível em [github.com/create1.sql](), contém os comandos necessários para criar a base de dados em *SQLite*. Além da conversão do nosso esquema relacional para *SQLite*, foram também adicionadas algumas restrições que não existiam originalmente, de modo a tornar a implementação mais completa.

4.2. Análise da Base de Dados *SQLite* com *AI* Generativa

Começámos por enviar o seguinte *prompt* à *AI* com o objetivo de identificar possíveis falhas na sintaxe do código, problemas de semântica e erros de contexto na nossa *script*. Esta abordagem serve para garantir que o código está correto e alinhado com o modelo relacional definido, para além de ajudar a identificar qualquer inconsistência ou algum erro que não detetamos que possa comprometer o funcionamento da base de dados.

```
text

Atua como um especialista sénior em engenharia de dados e SQLite. A tua tarefa é
analisar e refinar um script de criação SQL, comparando-o rigorosamente com a
descrição formal do esquema relacional (notação textual) fornecida em anexo.

As tuas instruções são:

1. Validação Estrita pela Notação Relacional:
   - O esquema relacional fornecido (em texto/notação) é a fonte absoluta
da verdade. Não é um diagrama visual, mas sim a definição lógica das tabelas
e atributos.
   - Verifica se o script SQL implementa exatamente as relações, chaves primárias
e chaves estrangeiras descritas na notação.
   - Não removas tabelas nem alteres a estrutura relacional
(cardinalidades/ligações) já validada.

2. Melhoria Contextual e Técnica (Semântica):
   - Podes e deves aplicar melhorias baseadas no contexto dos dados inferido pelos
nomes dos atributos na notação.
   - Ajusta os tipos de dados (Data Types) para tirar partido das afinidades do
SQLite (ex: tratar datas, booleanos ou moeda de forma robusta).
   - Adiciona Constraints de Domínio (CHECK) se o contexto o pedir (ex: garantir
valores positivos em preços, formatos de texto, etc.).
   - Define regras de integridade referencial (ON DELETE / ON UPDATE) que façam
sentido para o negócio, sem violar o esquema.

3. Formatação e Entrega:
   - O código final deve manter estritamente a mesma estrutura de linhas e
indentação do script original para permitir uma análise de mudanças via 'diff'.
   - Entrega apenas o código refinado e uma lista objetiva das alterações
contextuais/técnicas que implementaste.
```

Fig. 6 - Criação da Base de Dados em SQLite - *Prompt dado à Perplexity IA*

4.2.1. Principais mudanças implementadas

Para facilitar a análise, recorremos a comandos bastante utilizados em distribuições *Linux*, como é o caso do *diff,* que nos permite visualizar as diferenças entre dois ficheiros. Esta ferramenta foi utilizada ao longo do projeto para comparar as diferentes versões de código produzidas pela *AI*.

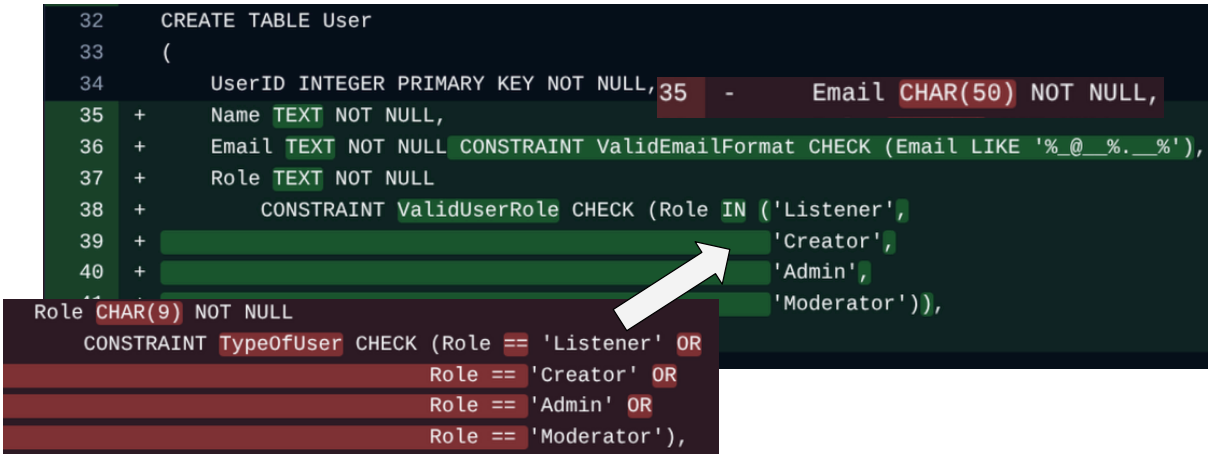


Fig. 7 - create1.sql vs create2.sql

As principais mudanças com a *AI* envolveram principalmente a alteração de múltiplas variáveis do tipo *CHAR para TEXT,* e de *DATETIME* (*TEXT)* para *INTEGER* no contexto do histórico de audições. Estas alterações devem-se à arquitetura do *SQLite*, onde *TEXT* oferece maior flexibilidade e compatibilidade comparado a tipos de comprimento fixo como o *CHAR.* No caso de *DATETIME* para *INTEGER,* deveu-se ao facto da intenção ser guardar onde o utilizador parou de ouvir o conteúdo e não o momento em que o fez. Adicionalmente, corrigimos a sintaxe das comparações, garantindo que as operações relacionais funcionem corretamente.

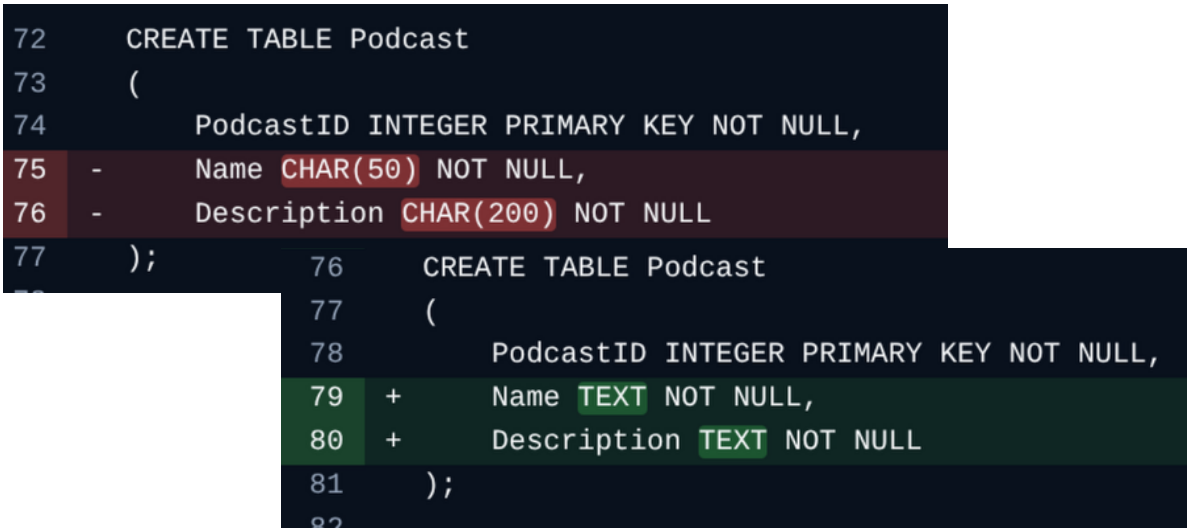


Fig. 8 - create1.sql vs create2.sql

Para além das alterações de tipos de dados, foram adicionadas diversas restrições que não existiam no esquema original. Estas *constraints* incluem validações de domínio através de *CHECK* para campos numéricos e booleanos, assegurando que os valores, como de quantidades, sejam positivos e que campos booleanos aceitem apenas 0 ou 1.

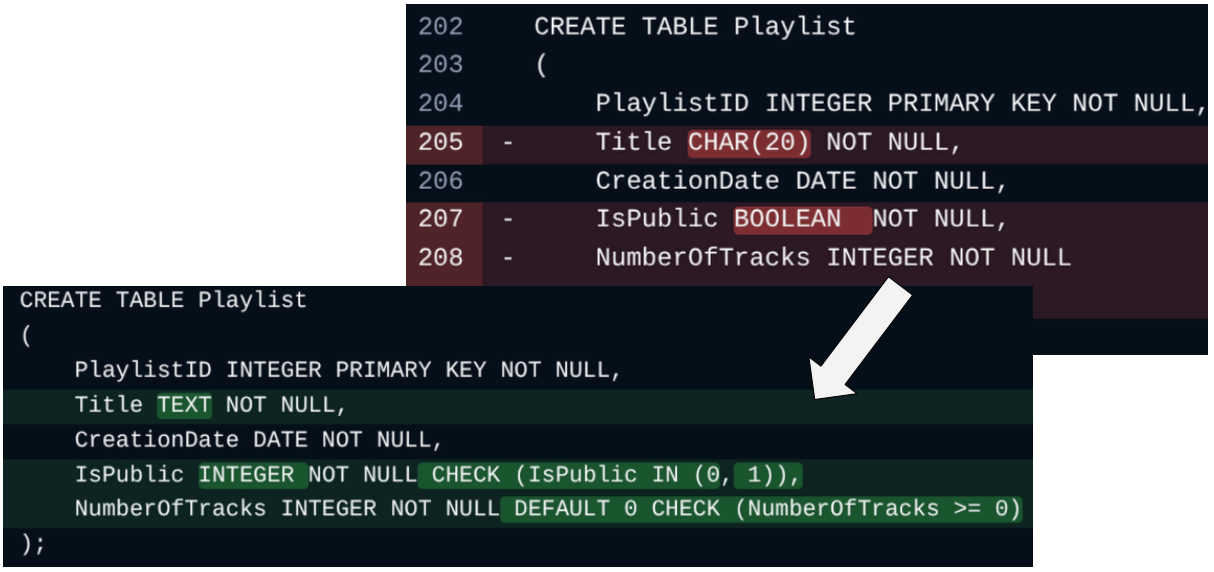


Fig. 9 - create1.sql vs create2.sql

O registo completo da interação com a *Perplexity AI* encontra-se disponível no seguinte link:

https://www.perplexity.ai/search/data-base-creation,

bem como a script refinada com o auxílio da *AI :*

github.com/create2.sql

5. Inserção de Dados

5.1. Proposta Inicial

O ficheiro **populate1.sql**, disponível em github.com/populate1.sql, contém comandos para inserir na base de dados *SQLite* uma quantidade significativa de dados para testes. Os tuplos iniciais foram multiplicados através de ferramentas de *AI*, proporcionando um maior conjunto fictício de utilizadores, artistas, álbuns, *podcasts*, *episódios*, entre outros.

5.2. Análise das Inserções na Base de Dados com *AI* Generativa

Após a criação do esquema da base de dados no ficheiro **create2.sql**, foi necessário analisar e adaptar o **populate1.sql** para garantir a conformidade com o esquema *SQLite* refinado. Recorrendo a ferramentas de *AI*, foi possível procurar por erros de integridade referencial, falhas de sintaxe, inconsistências lógicas e tipos de dados incorretos, resultando numa versão final melhorada denominada **populate2.sql**.

```markdown
Atua como Engenheiro de Bases de Dados Sénior especializado em SQLite. O teu
objetivo é validar e refinar o script de povoamento de uma base de dados. Vou
fornecer-te o esquema da base de dados (DDL) e um ficheiro inicial chamado
populate1.sql. A tua tarefa é analisar este ficheiro em busca de erros de
integridade referencial, falhas de sintaxe, inconsistências lógicas ou
tipos de dados incorretos, e depois gerar uma versão corrigida e melhorada
chamada populate2.sql.

Por favor, começa por apresentar um relatório conciso dos problemas detetados,
focando-te em violações de constraints, chaves estrangeiras inexistentes ou dados
ilógicos. De seguida, fornece o bloco de código SQL completo e pronto a executar
para o populate2.sql. Certifica-te de que usas transações (BEGIN/COMMIT) para
garantir a atomicidade e inclui comentários breves onde aplicaste correções. A
sintaxe deve ser estritamente compatível com SQLite. Seguem abaixo o esquema e o
código atual:
```

Fig. 10 - Inserção de Dados - *Prompt dado à Perplexity AI*

5.2.1. Principais mudanças implementadas

Após refirnarmos a nossa *script* create2.sql, foi necessário mudar alguns detalhes nos comandos de inserção no ficheiro populate1.sql para refletir as alterações de tipos de dados implementadas. As modificações foram relativamente pontuais:

1. Correção de dados do tipo *DATE* para o formato *YYYY-MM-DD,* por vezes apenas ao acrescentar um zero à esquerda para garantir a consistência (ex: "2024-1-5" para "2024-01-05").

```
95  - INSERT INTO Subscription(PlanType, StartDate, EndDate) VALUES ('Individual',
      '2024-07-01', '2024-8-01');
91  + INSERT INTO Subscription(PlanType, StartDate, EndDate) VALUES ('Individual',
      '2024-07-01', '2024-08-01');
```

Fig. 11 - Correção da formatação dos dados

2. Conversão de valores do campo "*TimePlayed*" do tipo *DATETIME* para valores *INTEGER* em segundos, conforme definido no create2.sql.

Estes ajustes, embora necessários, não exigiram grandes intervenções com recurso a *AI.*

```
315 - INSERT INTO EpisodeListeningHistory(UserID, EpisodeID, TimePlayed) VALUES (1, 2,
      '2024-05-02 18:40:55');
316 - INSERT INTO EpisodeListeningHistory(UserID, EpisodeID, TimePlayed) VALUES (2, 3,
      '2024-05-03 09:15:33');
292 + INSERT INTO EpisodeListeningHistory(UserID, EpisodeID, TimePlayed) VALUES (1, 2,
      1500);
293 + INSERT INTO EpisodeListeningHistory(UserID, EpisodeID, TimePlayed) VALUES (2, 3,
      3600);
```

Fig. 12 - Correção do tipo de dados

O registo completo da interação com a *Perplexity AI* encontra-se disponível no seguinte link:

https://www.perplexity.ai/search/populate-sqllite-creation,

bem como a script refinada com o auxílio da *AI:*

github.com/populate2.sql

6. Análise Crítica Sobre a Integração de AI

Ao longo deste projeto, o grupo recorreu à ferramenta *Perplexity AI,* utilizando a funcionalidade "*auto*", o que significa que provavelmente foram utilizados vários modelos de linguagem conforme cada tarefa, escolhidos automaticamente pela própria *AI.*

A experiência revelou que a *AI* foi consideravelmente útil em determinadas fases do projeto, particularmente quando já existia uma base de trabalho consolidada. POr outro lado, é menos eficaz na geração e conteúdo de raiz, raramente correspondendo às expectativas e requisitos do grupo. Por sua vez, a qualidade das respostas obtidas dependeu fortemente na refinação dos próprios *prompts,* utilizando a própria *AI* para o fazer, tornando-se evidente que a elaboração de um bom *prompt* estruturado é muito importante.

Relativamente aos requisitos do projeto, consideramos que a obrigatoriedade de demonstrar a utilização de *AI* em todas ou quase todas as componentes do trabalho deveria ser alterada. Consideramos que uma maior autonomia na decisão de onde aplicar estas tecnologias, fundamentando o porquê, teria permitido um trabalho mais enriquecedor.

Pontos Fortes

- Refinamento do esquema relacional e do código *SQL* - o que faz todo o sentido, já que os modelos de linguagem natural (*LLM's*) são extensivamente treinados para a análise de texto e codificação.

- Identificar pequenos erros que facilmente escaparam à nossa atenção durante revisões manuais - no contexto do código *SQL*, a utilização da ferramenta "*diff*" para comparar scripts/códigos foi uma excelente estratégia, permitindo verificar se a *AI* tinha introduzido alterações inesperadas na versão original que não fossem evidentes. Esta abordagem garantiu um controlo sobre as modificações sugeridas.

Pontos Fracos

- Refinamento do diagrama *UML* - tratando-se de *prompt's* com anexos de imagens e considerando que os *LLM's* são na sua essência orientados para o processamento textual, tornou-se extremamente difícil transmitir de forma adequada a informação necessária para obter melhorias significativas no diagrama.

- Necessidade de iterar múltiplas vezes sobre as *respostas* para obter resultados minimamente satisfatórios - requer um bom refinamento de *prompt's* e uma validação crítica e demorada dos *output's* gerados, o que por vezes pode demorar mais tempo do que propriamente não usar *AI,* e ainda frequentemente no fim, não obter o resultado esperado.

Contribuição dos Elementos do grupo

Primeira Submissão

Na primeira submissão, o Ricardo e o Leandro desenvolveram conjuntamente o trabalho inicial, incluindo o diagrama *UML* e o primeiro relatório.

Segunda Submissão

Na segunda submissão, o Ricardo e o Leandro colaboraram na conversão manual para esquema relacional, na análise de dependências funcionais e formas normais, e na criação dos ficheiros `create1.sql` e `create2.sql`, sendo o Ricardo o principal responsável pela implementação da base de dados em *SQLite*. A Sofia desenvolveu o ficheiro `populate1.sql`, responsabilizando-se pelo carregamento de dados nas tabelas da base de dados. O Leandro, para além da colaboração mencionada, ficou responsável pela integração e melhorias das diferentes fases do trabalho com a *AI* e foi o principal responsável pela elaboração do relatório, com a colaboração do Ricardo.