Instituto de Ciências Matemáticas e de Computação

SCC0640 - Bases de Dados

Aplicativo de Organização e Transmissão de Torneios Esportivos Amadores

Alunos:

Leonardo Gueno Risseto	13676482
Lucas Lima Romero	13676325
Luciano Gonçalves Lopes Filho	13676520
Marco Antonio Gaspar Garcia	11833581
Thiago Kashivagui Gonçalves	13676579

Docente:

Profa. Dra. Elaine Parros Machado de Sousa

São Carlos, 2024

Sumário

1 Descrição do problema e dos requisitos de dados **3**

1.1 Motivação . . . 3

1.2 Descrição do Sistema Proposto . . . 3

2 Modelo Entidade-Relacionamento (MER) **5**

2.1 Restrições de Integridade . . . 5

2.1.1 Possível inconsistência entre a data de início do torneio e a data da partida . . . 5

2.1.2 Possível inconsistência no horário das partidas . . . 5

2.1.3 Possível inconsistência no local das partidas . . . 5

2.1.4 Ciclo Esporte-Time-Partida-Torneio . . . 5

2.1.5 Jogadores e Treinadores sem Time . . . 6

2.1.6 Jogos únicos . . . 6

2.1.7 Excentricidade em gênero . . . 6

2.1.8 Locais com mais de uma quadra . . . 6

2.1.9 Chaves . . . 6

2.2 Principais Funcionalidades . . . 6

2.3 Diagrama Entidade-Relacionamento . . . 8

2.3.1 Notes . . . 8

2.4 Mudanças realizadas na segunda entrega do projeto . . . 9

3 Mapeamento Relacional **9**

3.1 Relacional . . . 9

3.2 Notes para o Mapeamento Relacional . . . 11

3.2.1 Mapeamento da entidade Pessoa (N1) . . . 11

3.2.2 Mapeamento do atributo derivado Idade (N2) . . . 11

3.2.3 Relacionamentos N:N com part. total (N3, N8, N11, N12) . . 11

3.2.4 Relacionamentos N:1 com part. total do lado N (N4, N7) . . . 12

3.2.5 Mapeamento da Agregação Assistido (N5) . . . 12

3.2.6 Mapeamento do atributo derivado No. de Visualizações (N6) . 12

3.2.7 Mapeamento de Partida e Local e criação de ID sintético (N9, N15) . . . 13

3.2.8 Mapeamento Relacionamento Exibe (1:1 com part. total) (N10) 13

3.2.9 Mapeamento do atributo É_Moderador (N13) . . . 14

3.2.10 Mapeamento do atributo multivalorado Permissões (N14) . . . 14

3.2.11 Mapeamento geral para relacionamentos N:N . . . 14

3.2.12 Mapeamento da especialização de Usuário: Moderador (N16) . 15

3.3 Mudanças realizadas na terceira entrega do projeto . . . 15

4 Consultas SQL **16**
4.1 Torneios, número de patrocinadores e total investido 16
4.2 Lista de árbitros com número de partidas apitadas e a data da última partida apitada 17
4.3 Média de idade dos jogadores por esporte 18
4.4 Locais que receberam mais partidas no torneio Campeonato Piracicabano de Futebol 18
4.5 Patrocinadores que investiram em todos os torneios de Futebol 19

5 Aplicação **20**
5.1 Ferramentas utilizadas 20
5.2 Utilização da aplicação 21

6 Conclusão **22**

1 Descrição do problema e dos requisitos de dados

1.1 Motivação

Este documento propõe a criação de um banco de dados especializado para um aplicativo de celular destinado à transmissão de jogos e campeonatos esportivos amadores. O objetivo é desenvolver uma estrutura eficiente e robusta, capaz de gerenciar com precisão informações críticas, como dados de usuários, jogadores, equipes, partidas, torneios e transmissões, assegurando uma transmissão de qualidade e facilitando a organização de eventos esportivos voltados ao público amador.

A principal motivação deste projeto é aumentar a visibilidade de atletas amadores, proporcionando uma plataforma acessível que permita a exposição de seus talentos para uma audiência mais ampla. Essa plataforma permitirá que torneios e campeonatos regionais e locais sejam transmitidos ao vivo, ampliando o alcance e o impacto das competições amadoras. Dessa forma, os atletas terão a oportunidade de demonstrar suas habilidades para não apenas familiares e amigos, mas também para olheiros, recrutadores e patrocinadores em potencial, aumentando suas chances de reconhecimento e suporte.

O aplicativo será desenvolvido para suportar a transmissão ao vivo de eventos esportivos regionais e locais de diferentes modalidades esportivas. Para viabilizar essa funcionalidade, o banco de dados desempenhará um papel fundamental, armazenando e organizando informações detalhadas sobre todas as entidades envolvidas. Além disso, o banco de dados garantirá a integridade dos dados e facilitará o acesso rápido e preciso às informações necessárias para as operações do aplicativo, assegurando uma experiência de usuário consistente e eficiente.

1.2 Descrição do Sistema Proposto

O aplicativo foi idealizado para que o usuário consiga assistir transmissões esportivas amadoras, além de conseguir consultar informações relevantes a respeito de jogadores, torneios, entre outras entidades importantes. Sendo assim, o **Usuário** se cadastra com *Nome*, *Credenciais* (*Email* e *Senha*) e *Data de Nascimento* (sua *Idade* é calculada a partir desse dado). Caso ele seja um Usuário **Moderador**, ele conseguirá criar, editar e remover informações sobre seu Time, Jogadores, Torneios, entre outros. Um Moderador tem permissões que um Usuário comum não possui, e estas serão definidas na camada de aplicação do projeto. Essas permissões podem ser múltiplas a depender do caso.

No modelo de dados, a entidade **Pessoa** é definida pelos atributos *Nome*, *CPF*, *Data de Nascimento* (sua *Idade* é calculada a partir desse dado), *Gênero* e *Profissão*, sendo esta última um atributo especializador que determina qual papel a pessoa pode assumir. No caso deste projeto, é necessário que a entidade Pessoa assuma um e apenas um papel. Os possíveis papéis para uma Pessoa são Jogador, Treinador,

Árbitro ou Narrador.

Quando uma Pessoa assume o papel de **Jogador**, ela está associada diferentes **Times**. Pensando num contexto não-profissional, é comum que um Jogador amador jogue por mais de um Time. Além disso, os jogadores possuem os atributos *Altura* e *Peso*. Já quando uma Pessoa assume o papel de **Treinador**, ela também está associada a um ou mais Times, semelhantemente a um Jogador. Uma Pessoa também pode ser **Árbitro**, responsável por arbitrar inúmeras Partidas, e **Narrador**, cuja função é comentar diversas Transmissões ao vivo no aplicativo.

Um **Time** é caracterizado pelo seu *Nome* e pelo *Nome* de seu **Esporte** associado. Isso se deve pelo caso de um clube amador possuir times de diferentes esportes. Além disso, o Time é composto por muitos Jogadores, e treinado por apenas um Treinador, além de jogarem muitas Partidas. Já um **Esporte** é identificado pelo seu *Nome*, e por sua vez caracteriza um time. Além disso, o Esporte está relacionado a Torneios, de forma que no aplicativo existirão diversos Torneios, cada um com um Esporte definido.

Uma **Transmissão** é caracterizada por sua *URL*, *Data e Hora* e *Número de Visualizações*. A Transmissão é assistida por muitos Usuários, e essa relação é redefinida por uma agregação, denominada **Assistido**. Essa agregação servirá como um histórico de visualizações da Transmissão para cada Usuário, contando com *Data e Hora* e *Tempo Assistido*. Isso torna possível a gravação de diferentes visualizações do mesmo Usuário para a mesma Transmissão. Ademais, Transmissão também é comentada por muitos Narradores, e cada Transmissão é responsável por exibir no aplicativo apenas uma Partida. É importante ressaltar que o atributo *Número de Visualizações* da Transmissão é derivado da quantidade vezes em que um Usuário assiste a mesma Transmissão, ou seja, da quantidade de ocorrências da agregação **Assistido**.

Uma **Partida** é definida por *Data e Hora* e Resultado, além de ser necessário o *Endereço* do **Local** em que ela ocorrerá, *Nome* e *Data de Início* do **Torneio** a qual ela está associada e os **Times** que a disputarão (que como visto, é identificado por *Nome* e *Nome* do **Esporte** associado). Toda Partida é um confronto entre dois ou mais Times, apitada por muitos Árbitros, pertence a um Torneio e ocorre em um Local. Além disso, uma Partida pode ser assistida através de uma Transmissão no aplicativo.

Um **Local** sedia diversas Partidas, sendo caracterizado por seu *Nome*, *Endereço* (Rua, Número, Cidade, Estado e País) e *Capacidade*. Já um **Torneio** tem atributos como *Nome* e *Data* (*Início* e *Fim*). Os Torneios são compostos por muitas Partidas, além de serem patrocinados por muitos Patrocinadores. Além do mais, cada Torneio é referente a apenas um Esporte.

Por fim, os **Patrocinadores** são responsáveis por patrocinar um ou mais Torneios. Estes são caracterizados por *CNPJ*, *Nome* e *Valor Investido*.

2 Modelo Entidade-Relacionamento (MER)

2.1 Restrições de Integridade

2.1.1 Possível inconsistência entre a data de início do torneio e a data da partida

A entidade **Partida** é caracterizada por sua chave fraca *Data e Hora* e pelas chaves de suas entidades **owners**. Sendo assim, Data de início do torneio não é redundante em relação a Data e Hora da partida, pois ambos são necessários para identificar univocamente a entidade Partida. Entretanto, o sistema não garante que Partidas sejam marcadas antes da data de início do Torneio, sendo assim necessário o tratamento desta inconsistência na camada de aplicação do sistema.

2.1.2 Possível inconsistência no horário das partidas

A modelagem atual permite que múltiplas **Partidas** ocorram no mesmo **Local** com sobreposição de horários (uma Partida começa as 16:00 e a outra as 16:01, por exemplo), o que é fisicamente impossível. Essa inconsistência deve ser tratada na camada de aplicação, de forma a fazer com que não seja possível o agendamento de Partidas no mesmo Local antes do término das mesmas.

2.1.3 Possível inconsistência no local das partidas

A modelagem atual permite que uma **Partida** entre os mesmos **Times** seja marcada na mesma *Data e Hora*, mas em **Locais** diferentes. Isso é um caso que deve ser tratado na implementação do aplicativo, verificando se esses Times já não estarão disputando Partidas nesse horário antes de permitir o agendamento de uma nova Partida.

2.1.4 Ciclo Esporte-Time-Partida-Torneio

Existe um ciclo formado pelas entidades Esporte-Time-Partida-Torneio. O modelo permite que um **Torneio** tenha múltiplas **Partidas** de diferentes **Esportes**, apesar da restrição de que cada **Torneio** deve ser de apenas um único **Esporte**. Isso se deve ao fato de que Partida não é relacionada com Esporte diretamente, mas com os Times, que por sua vez tem um Esporte definido, podendo resultar em dados conflitantes caso não haja um tratamento adequado. Assim, esse problema deve ser devidamente antecipado durante a implementação das funcionalidades do aplicativo, de forma a permitir que apenas Times do mesmo Esporte do Torneio participem das Partidas deste Torneio.

2.1.5 Jogadores e Treinadores sem Time

Em um contexto amador, é comum que **Jogadores** e **Treinadores** fiquem algum tempo sem atuar por algum **Time**. Por esse motivo, o banco de dados foi modelado para permitir que a participação de Jogador e Treinador em Time não seja total, assim mantendo-os disponíveis para consulta no aplicativo mesmo sem estarem atuando. Logo, isso se classifica como uma característica do sistema, e não como uma inconsistência.

2.1.6 Jogos únicos

Caso deseje-se criar um jogo único, como um amistoso, por exemplo, deve-se necessariamente criar um **Torneio** no aplicativo, dessa forma caracterizando um Torneio de apenas uma **Partida**. Esta não é uma vulnerabilidade ou inconsistência, mas apenas uma característica da implementação do sistema, portanto, deve-se levar isso em conta ao marcar amistosos.

2.1.7 Excentricidade em gênero

O banco de dados não diferencia gêneros, de modo a permitir **Torneios** com gêneros mistos, uma vez que o público-alvo são amadores. Dessa forma, é permitido e incentivado pela aplicação a mescla de gêneros, não se classificando, portanto, como uma inconsistência, mas sim como uma característica da implementação.

2.1.8 Locais com mais de uma quadra

No caso do **Local** ter mais de uma possibilidade de jogo, como por exemplo, um ginásio com muitas quadras, deve-se especificar em seu *Nome*. Um exemplo de Nome do Local dessa forma é: Ginásio Municipal de São Carlos - Quadra 2.

2.1.9 Chaves

Todos os atributos-chaves das entidades, ou seja, aqueles atributos que as identificam univocamente, por definição devem ser únicos e não-nulos. No caso de chaves compostas, a n-upla deve ser única e nenhum dos atributos deve ser nulo.

2.2 Principais Funcionalidades

O aplicativo baseia-se na transmissão de campeonatos esportivos amadores e na consulta de dados relevantes sobre os atletas, treinadores, partidas e torneios. Sendo assim, o **Usuário** do aplicativo é o foco principal de todas as funcionalidades, entre elas:

- Assistir às transmissões ao vivo dos jogos amadores e obter informações como narradores e número total de visualizações.

- Consultar informações relevantes sobre jogadores amadores como altura, peso e time pelo qual jogam;
- Consultar informações relevante sobre treinadores amadores, como o time que treinam.
- Consultar informações relevantes sobre partidas, como data e hora, times envolvidos, local e arbitragem.
- Consultar informações relevantes sobre torneios amadores, como partidas e patrocinadores.

Note que o Usuário comum pode apenas consultar informações, e não inserir, alterar ou remover dados. No entanto, no sistema descrito, um Usuário pode assumir o papel de **Moderador** e obter permissões específicas para gerenciar uma entidade particular à qual está associado. As permissões de um Moderador são variáveis e dependem do contexto da entidade que ele está moderando. Assim, o conceito de moderação é contextual e personalizado, com o Moderador possuindo permissões específicas sobre a entidade que ele modera. Suas funcionalidades podem incluir:

- Todas as funcionalidades do Usuário comum.
- Editar informações associadas a entidade moderada por ele.

As permissões do Moderador são concedidas pelo **Administrador**, e a forma com que isso é feito deverá ser implementada na camada de aplicação do sistema. O Administrador gerencia absolutamente tudo no sistema, portanto, suas funcionalidades incluem:

- Concessão de permissões para usuários, tornando-os Moderadores.
- Remoção de permissões de Moderadores.
- Controle total sobre as entidades do sistema, incluindo a edição, inserção e remoção de registros.

2.3 Diagrama Entidade-Relacionamento

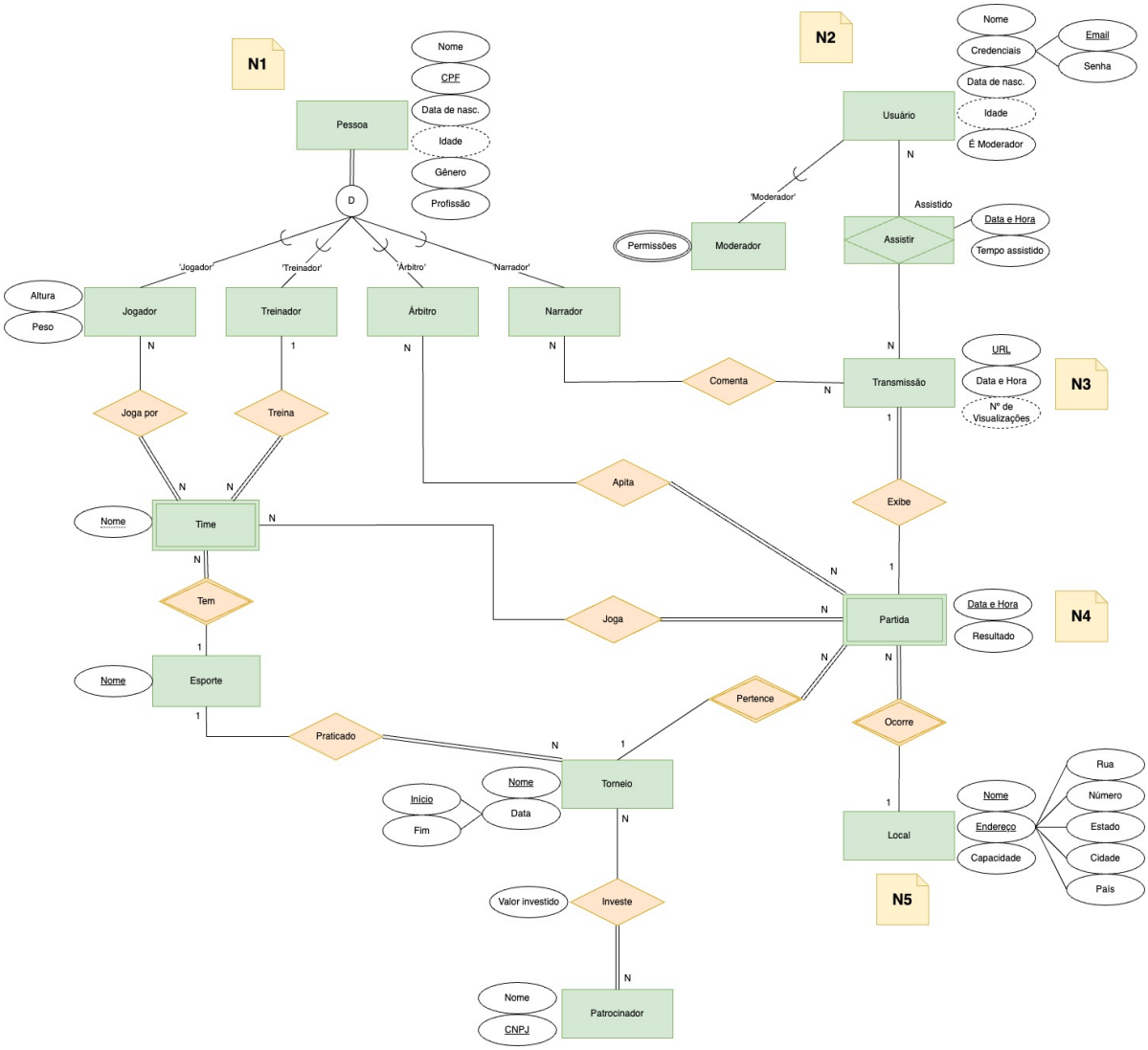


Figura 1: Diagrama Entidade-Relacionamento.

2.3.1 Notes

- **Note 1:** *Profissão* é o atributo especializador de **Pessoa**.
- **Note 2:** *É Moderador* é o atributo especializador de **Usuário**.
- **Note 3:** O atributo *Número de Visualizações* é derivado da quantidade de ocorrências da agregação **Assistido**.
- **Note 4:** O atributo *Resultado* inicia nulo, e após o término da **Partida** é atualizado.
- **Note 5:** No caso de haver duas quadras no mesmo local, o número da quadra deve ser especificada em *Nome*. Exemplo: "Estádio Municipal de São Carlos - Quadra 2".

2.4 Mudanças realizadas na segunda entrega do projeto

Após as correções sugeridas pelo monitor, foram realizadas alterações significativas no diagrama MER. Primeiramente, foram corrigidas as especializações que anteriormente apresentavam-se com uma notação incorreta, tanto em **Pessoa** quanto em **Usuário**. Além disso, foi removido o indicador de disjunção para a entidade *Usuário*, pois, uma vez que há apenas uma especialização possível, não há necessidade de representar a disjunção.

Outra correção importante foi a alteração do relacionamento *Joga*, que deixou de ser considerado um relacionamento fraco. Essa correção ocorreu de forma a evitar o erro conceitual cometido anteriormente, em que havia uma entidade fraca deste relacionamento. Conforme orientado, não é permitido que uma entidade fraca possua um conjunto de relacionamentos fracos em uma relação N:N, o que justificou a revisão dessa parte do modelo.

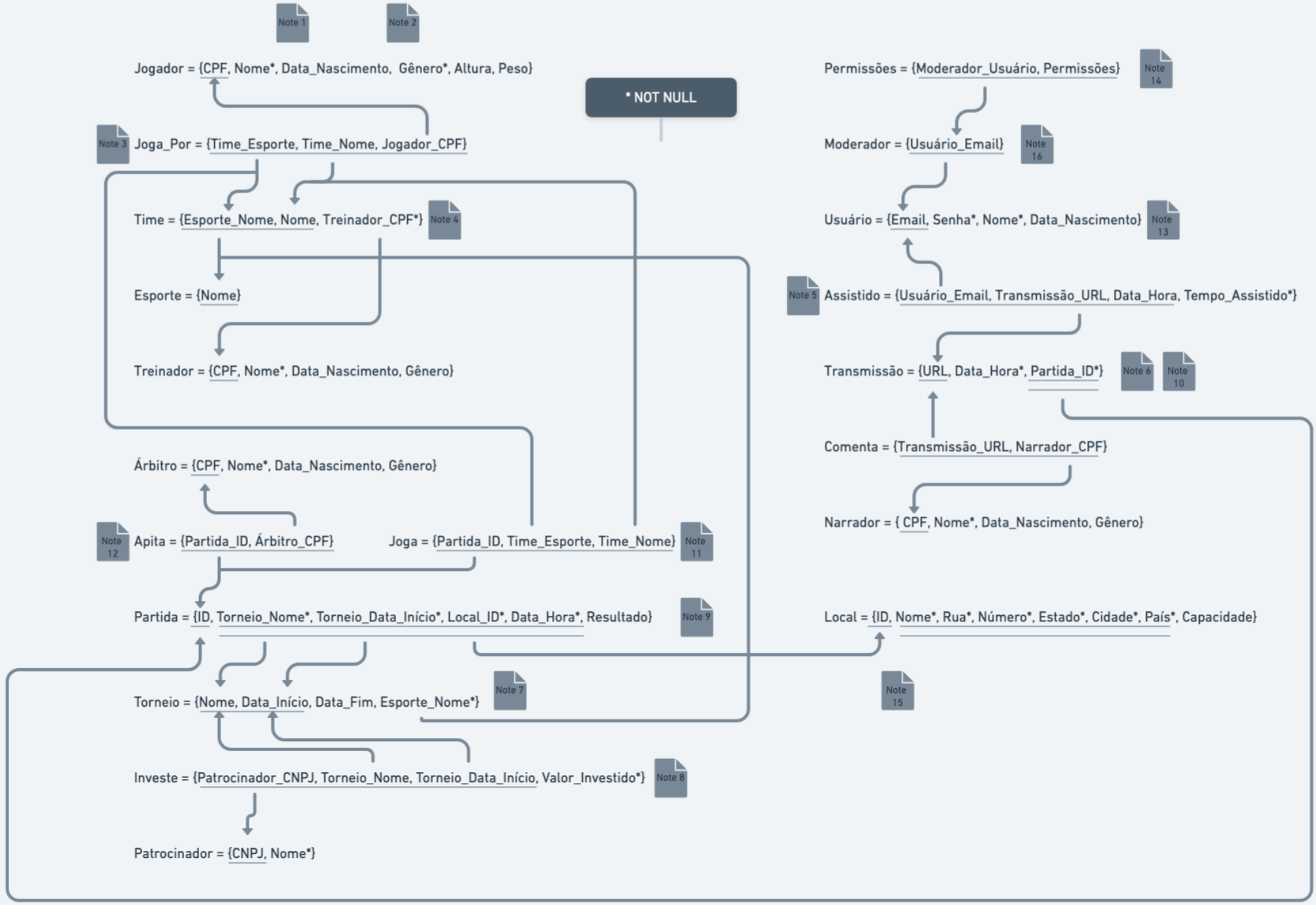
Por fim, a última alteração foi a mudança do atributo *Valor Investido*. Anteriormente, este atributo era pertencente a um **Patrocinador**. Nesta versão do MER, este atributo passou a pertencer ao relacionamento *Investe*, o que semanticamente pareceu ser mais conveniente.

O diagrama MER contido na seção 2.3 deste trabalho já apresenta as modificações comentadas acima.

3 Mapeamento Relacional

3.1 Relacional

A partir do diagrama Entidade-Relacionamento desenvolvido e explicado na seção anterior, foi realizado um Mapeamento Relacional deste diagrama, que pode ser encontrado na página a seguir. Utilizou-se a plataforma Whimsical para a esquematização.



3.2 Notes para o Mapeamento Relacional

3.2.1 Mapeamento da entidade Pessoa (N1)

- **Solução:** Não foi mapeada a entidade **Pessoa** diretamente, pois não há relacionamento direto com ela, mas apenas com suas especializações.
- **Vantagem:** Dessa forma, é possível garantir a especialização total, pois é garantido que todas as entradas das tabelas específicas serão pessoas.
- **Desvantagem:** A criação de quatro tabelas implica na repetição de atributos comuns à todas as especializações da entidade Pessoa. Além disso, não é garantido que haverá disjunção, devendo ser tratado na aplicação.
- **Alternativa:** Existe a possibilidade da criação de apenas uma tabela para representar a entidade Pessoa. Entretanto, seriam necessários mais quatro atributos booleanos para a identificação de cada especialização, e como são disjuntos, existiriam muitos valores nulos. Além disso, tanto a especialização total quanto a disjunção não são garantidas nesse modelo, fazendo-se necessária a verificação desses problemas na aplicação.

3.2.2 Mapeamento do atributo derivado Idade (N2)

- **Solução:** O atributo idade está presente no MER, tanto nas especializações de **Pessoa** quanto em **Usuário**. Optou-se por não mapear este atributo no esquema relacional, pois seria de grande custo para o SGBD atualizá-lo diariamente.
- **Vantagem:** O cálculo de idade é relativamente barato de se fazer, e por não ser uma informação que será acessada com frequência, não existe uma grande necessidade de ser mapeada.
- **Desvantagem:** Não possuir um registro direto da idade dos usuários, sendo necessário o cálculo da mesma toda vez que for necessário o acesso a essa informação.
- **Alternativa:** Mapear o atributo, mesmo sendo uma operação cara para o SGBD.

3.2.3 Relacionamentos N:N com part. total (N3, N8, N11, N12)

- **Solução:** Foi criada uma nova tabela para estes relacionamentos *Joga_Por, Investe, Joga, Apita*.
- **Vantagem:** A cardinalidade destes relacionamentos podem ser garantidas.

- **Desvantagem:** Não é possível garantir a participação total, sendo necessário verificar essa restrição na aplicação.
- **Alternativa:** Não foi identificada uma alternativa para este caso.

3.2.4 Relacionamentos N:1 com part. total do lado N (N4, N7)

- **Solução:** Os relacionamentos (*Treina*, *Praticado*) foram mapeados nas tabelas referentes as entidades do lado N (*Time, Torneio*).
- **Vantagem:** Ao evitar a criação de uma nova tabela, também evita-se a ocorrência de junções (*joins*), que são operações caras em bancos de dados. Além disso, é possível garantir a participação total da entidade do lado N, colocando `not null` na chave estrangeira referente a entidade do lado 1 (*Treinador_CPF*, *Esporte_Nome*).
- **Desvantagem:** Esta solução só apresenta desvantagens expressivas se os relacionamentos tiverem pouca participação, e, consequentemente, tiverem muitos valores nulos. No entanto, nos relacionamentos supracitados, supomos participação considerável e balanceada.
- **Alternativa:** Poderiam ser criadas novas tabelas para armazenar estes relacionamentos. Entretanto, a participação total não poderia ser garantida diretamente no modelo.

3.2.5 Mapeamento da Agregação Assistido (N5)

- **Solução:** Mapeou-se somente a agregação **Assistido**, sendo sua chave composta por Usuário_Email, URL_Transmissão, Data_e_Hora. Optou-se por não mapear o relacionamento *Assistir*, pois não existem atributos próprios.
- **Vantagem:** O mapeamento da entidade agregada é obrigatório nesse caso, pois a entidade possui atributos próprios. Esse tipo de mapeamento permite-nos construir uma espécie de histórico de visualizações.
- **Desvantagem:** Este mapeamento não possui desvantagens claras.
- **Alternativa:** É possível mapear o relacionamento e a entidade separadamente, em diferentes tabelas, mas como explicado, não traria nenhuma vantagem ao sistema, tanto em custo quanto em semântica.

3.2.6 Mapeamento do atributo derivado No. de Visualizações (N6)

- **Solução:** Não foi mapeado o atributo derivado Número de Visualizações, pois o acesso a essa informação será feito *on demand* na aplicação.

- **Vantagem:** Esta abordagem é mais barata para o SGBD, pois seriam necessárias atualizações extremamente frequentes para esse dado. Isso deve ser implementado de forma mais eficiente na aplicação.
- **Desvantagem:** O cálculo desse atributo não é barato caso o número de entradas nessa tabela seja muito grande.
- **Alternativa:** Existe a possibilidade de mapeá-lo diretamente no esquema relacional, e para isso, deve-se levar em conta a vantagem e a desvantagem descrita acima. No caso, foi decidido que a melhor solução era não mapeá-lo diretamente.

3.2.7 Mapeamento de Partida e Local e criação de ID sintético (N9, N15)

- **Solução:** Olhando para o MER, as chaves primárias de Partida e Local são compostas por muitos campos. Por isso, optou-se pela criação de um ID artificial em ambos os casos, a fim de reduzir o tamanho de sua chaves primárias. Além disso, é importante salientar que suas antigas chaves primárias compostas se tornaram chaves secundárias, com todos os elementos `not null`, para manter a consistência e a semântica.
- **Vantagem:** A criação do ID artificial como chave primária da tabela faz com que as buscas sejam mais eficientes do que uma chave composta por muitos campos.
- **Desvantagem:** A desvantagem do ID artificial, nesse caso, além de ser uma informação inócua, que só serve para reduzir a chave primária, é o fato de necessitar junções adicionais, para obter informações - a exemplo de Data e Hora da partida -, em outras tabelas que fazem referência à ela.
- **Alternativa:** Mapear os todos os atributos como chave primária.

3.2.8 Mapeamento Relacionamento Exibe (1:1 com part. total) (N10)

- **Solução:** O relacionamento 1:1 com participação total foi mapeado na tabela de **Transmissão**, adicionando a ela uma chave estrangeira de *Partida*.
- **Vantagem:** Reduzimos o custo de junções (não criando outra tabela) e garantimos Participação Total com `not null` e a cardinalidade 1:1, utilizando uma chave secundária.
- **Desvantagem:** Esta solução só apresenta desvantagens expressivas se os relacionamentos tiverem pouca participação, e, consequentemente, tiverem muitos valores nulos. No entanto, nos relacionamentos supracitados, supomos participação considerável e balanceada.

- **Alternativa:** Criar uma nova tabela para esse relacionamento, mas sem a garantia da participação total.

3.2.9 Mapeamento do atributo É_Moderador (N13)

- **Solução:** O atributo não foi mapeado na tabela **Usuário**.
- **Vantagem:** Reduz-se o número de atributos, economizando uma coluna, além de evitar a necessidade de checagem de consistência, de que todo moderador tenha esse atributo ativo.
- **Desvantagem:** Não é possível fazer buscas na tabela Usuário com o filtro de Moderador. No entanto, a premissa foi que essas consultas teriam uma frequência muito baixa.
- **Alternativa:** Mapear o atributo, com o intuito de fazer buscas por Moderadores na tabela Usuário e ter que lidar com a consistência.

3.2.10 Mapeamento do atributo multivalorado Permissões (N14)

- **Solução:** Foi criada uma nova tabela para este atributo, pois o número de permissões por usuário moderador é variável.
- **Vantagem:** Permite que os moderadores tenham diferentes permissões, de maneira flexível.
- **Desvantagem:** Torna-se necessário fazer um join a mais, para recuperar a informação completa dos *Moderadores* com suas Permissões associadas.
- **Alternativa:** A alternativa era mapear o atributo Permissões com uma quantidade fixa de atributos na entidade *Moderador*.

3.2.11 Mapeamento geral para relacionamentos N:N

- **Solução:** Em relacionamentos N:N, com part. total ou não, optou-se por mapear de forma a criar uma nova tabela para armazenar informações sobre esse relacionamento. Em todos os casos, a chave primária dessa nova tabela é composta pelas chaves primárias das entidades envolvidas, referenciadas por meio de chaves estrangeiras. Além disso, caso haja atributos específicos do relacionamento, estes também foram mapeados na nova tabela criada.
- **Vantagem:** Garantia da cardinalidade N:N.
- **Desvantagem:** Além do aumento do custo computacional pela criação de uma nova tabela, aumentando a ocorrência de *joins*, não existem grandes desvantagens para esse tipo mapeamento.
- **Alternativa:** Não foi identificada uma alternativa para este caso.

3.2.12 Mapeamento da especialização de Usuário: Moderador (N16)

- **Solução:** Optou-se por mapear a especialização de Usuário, os Moderadores, pois é interessante a existência de uma tabela separada, somente com esse tipo de usuário especial, afinal, traria um maior controle para o administrador da Base de Dados.

- **Vantagem:** Maior controle e mais facilidade de buscas dos Usuários que são Moderadores.

- **Desvantagem:** Aumento da complexidade da busca de informações, pois para cruzar dados da tabela Usuário e Moderador, serão necessárias junções.

- **Alternativa:** Mapear os Moderadores diretamente na tabela Usuário, através de um atributo booleano *"É_Moderador"*.

3.3 Mudanças realizadas na terceira entrega do projeto

Após as correções sugeridas pelo monitor, foram realizadas alterações significativas no esquema Relacional e adicionamos algumas justificativas. Primeiramente, corrigiu-se um erro de consistência em relação ao MER, da primeira parte, na tabela **Local**. Por motivos de desatenção, mapeou-se erroneamente a tabela em relação ao modelo, de modo que a chave primária no Relacional não correspondia àquela modelada na entidade. Corrigiu-se este problema, de modo que a chave primária dessa tabela passou a ser igual à chave do modelo.

Com essa mudança, um outro problema surgiu, pois a tabela **Partida** possui uma chave estrangeira de **Local**. Como a chave primária agora é composta por vários campos, acabou-se optando pela criação de um ID sintético para ser a nova chave primária de Local, com o fim de facilitar a relação entre Local e Partida por meio da chave estrangeira.

Por fim, algumas decisões tomadas não tiveram suas justificativas devidamente explicitadas na segunda entrega. As justificativas faltantes eram as seguintes:

- Alternativa para o mapeamento da entidade agregada **Assistido**, em que era possível mapear a entidade e o relacionamento separadamente.

- Mapeamento da especialização de **Usuário** em uma tabela separada, no caso, chamada de **Moderador**. Isso ocorreu pois seria interessante para o sistema uma tabela separada somente contendo aqueles usuários que são moderadores.

As justificativas faltantes foram devidamente adicionadas na seção 3.2 deste documento, e o esquema Relacional (seção 3.1) foi devidamente alterado.

4 Consultas SQL

Esta seção apresenta o script SQL com cinco consultas de complexidade média e alta. Escolheu-se variadas consultas de forma a testar a consistência do projeto de banco de dados.

4.1 Torneios, número de patrocinadores e total investido

Esta consulta retorna o nome do torneio, a data de início, o número de patrocinadores, o total investido e os nomes dos patrocinadores para cada torneio:

```
SELECT
    t.nome AS torneio_nome,
    t.data_inicio AS inicio_torneio,
    COUNT(i.patrocinador_cnpj) AS numero_patrocinadores,
    SUM(i.valor_investido) AS total_investido,
    string_agg(DISTINCT p.nome, ', ') AS patrocinadores_nomes
FROM
    Torneios t
LEFT OUTER JOIN Investe i
ON
    t.nome = i.torneio_nome AND t.data_inicio = i.torneio_data_inicio
LEFT OUTER JOIN Patrocinadores p
ON
    i.patrocinador_cnpj = p.cnpj
GROUP BY
    t.nome, t.data_inicio
ORDER BY
    t.nome ASC, t.data_inicio DESC;
```

Esta consulta realiza as seguintes operações:

- Seleciona o nome do torneio (`torneio_nome`), a data de início (`inicio_torneio`), o número de patrocinadores (`numero_patrocinadores`), o total investido (`total_investido`) e os nomes dos patrocinadores (`patrocinadores_nomes`).
- Realiza junções externas à esquerda (`LEFT OUTER JOIN`) entre as tabelas `Torneios`, `Investe` e `Patrocinadores`. Isso garante que todos os torneios sejam retornados, mesmo que não tenham patrocinadores associados.
- A função `COUNT` conta o número de patrocinadores por torneio, e a função `SUM` calcula o valor total investido por patrocinador em cada torneio.

- A função `string_agg` agrupa os nomes dos patrocinadores por torneio, separados por vírgulas.
- Agrupa os resultados pelas colunas `nome` e `data_inicio` do torneio.
- Ordena os resultados pelo nome do torneio (`ASC`) e pela data de início em ordem decrescente (`DESC`).

4.2 Lista de árbitros com número de partidas apitadas e a data da última partida apitada

Esta consulta retorna o CPF e o nome dos árbitros, o número de partidas apitadas e data da última partida apitada por cada árbitro:

```
SELECT
    ar.cpf AS arbitro_cpf,
    ar.nome AS arbitro_nome,
    COALESCE(COUNT(a.arbitro_cpf), 0) AS num_partidas_apitadas,
    COALESCE(MAX(p.data_hora)::text, 'Sem partidas') AS ultima_partida_apitada
FROM
    Arbitros ar
LEFT OUTER JOIN Apita a
    ON ar.cpf = a.arbitro_cpf
LEFT OUTER JOIN Partidas p
    ON a.partida_id = p.id
GROUP BY
    ar.cpf, ar.nome
ORDER BY
    num_partidas_apitadas DESC, ar.nome ASC;
```

Esta consulta realiza as seguintes operações:

- Seleciona o CPF (`arbitro_cpf`) e o nome (`arbitro_nome`) do árbitro, o número de partidas apitadas (`num_partidas_apitadas`) e a última partida apitada (`ultima_partida_apitada`).
- Realiza junções externas à esquerda (`LEFT OUTER JOIN`) entre as tabelas `Arbitros`, `Apita` e `Partidas`. Isso garante que todos os árbitros sejam incluídos, mesmo que não tenham apitado partidas.
- A função `COALESCE` é usada para garantir que, caso o árbitro não tenha apitado nenhuma partida, o número de partidas seja 0 e a data da última partida seja "Sem partidas".

- A função `COUNT` conta o número de partidas apitadas por cada árbitro, enquanto `MAX` retorna a data e hora da última partida apitada.
- Agrupa os resultados pelas colunas `cpf` e `nome` do árbitro.
- Ordena os resultados pelo número de partidas apitadas em ordem decrescente (`DESC`) e pelo nome do árbitro em ordem crescente (`ASC`).

4.3 Média de idade dos jogadores por esporte

A consulta a seguir calcula a média de idade dos jogadores por esporte:

```
SELECT
    T.nome_esporte,
    AVG(EXTRACT(YEAR FROM AGE(J.Data_Nascimento))) AS Media_Idade
FROM
    Jogadores J
JOIN
    Joga_Por JP ON J.CPF = JP.Jogador_CPF
JOIN
    Times T ON JP.Time_Esporte = T.nome_esporte AND JP.Time_Nome = T.Nome
GROUP BY
    T.nome_esporte
ORDER BY
    Media_Idade;
```

Esta consulta realiza as seguintes operações:

- Calcula a média de idade dos jogadores, extraída da diferença entre a data de nascimento e a data atual.
- Realiza junções (`JOIN`) entre as tabelas `Jogadores`, `Joga_Por` e `Times`.
- Agrupa os resultados por esporte (`Esporte_Nome`).
- Ordena o resultado pela média de idade de forma crescente.

4.4 Locais que receberam mais partidas no torneio Campeonato Piracicabano de Futebol

Esta consulta retorna os locais que receberam mais partidas no torneio denominado "CAMPEONATO PIRACICABANO DE FUTEBOL":

```
SELECT
    L.Nome AS Local,
    L.Cidade,
    L.Estado,
    COUNT(P.ID) AS Total_Partidas
FROM
    Locais L
JOIN
    Partidas P ON L.ID = P.Local_ID
WHERE
    P.Torneio_Nome = 'CAMPEONATO PIRACICABANO DE FUTEBOL'
    AND P.Torneio_Data_Inicio = '2023-09-01'
GROUP BY
    L.ID,
    L.Nome,
    L.Cidade,
    L.Estado
ORDER BY
    Total_Partidas DESC;
```

Esta consulta realiza as seguintes operações:

- Seleciona o nome, cidade e estado dos locais, junto com a contagem do número de partidas realizadas nesses locais.
- Realiza uma junção entre as tabelas `Locais` e `Partidas`.
- Filtra os resultados para incluir apenas as partidas do torneio "CAMPEONATO PIRACICABANO DE FUTEBOL", que começou em 2023-03-01.
- Agrupa os resultados por local.
- Ordena os resultados pelo número de partidas de forma decrescente.

4.5 Patrocinadores que investiram em todos os torneios de Futebol

Esta consulta retorna os patrocinadores que investiram em todos os torneios de "FUTEBOL":

```
SELECT
    P.CNPJ,
    P.Nome
FROM
```

```
    Patrocinadores P
JOIN
    Investe I ON P.CNPJ = I.Patrocinador_CNPJ
JOIN
    Torneios T ON I.Torneio_Nome = T.Nome AND I.Torneio_Data_Inicio = T.Data_Inicio
WHERE
    T.Esporte_Nome = 'FUTEBOL'
GROUP BY
    P.CNPJ,
    P.Nome
HAVING
    COUNT(*) = (
        SELECT COUNT(*)
        FROM Torneios
        WHERE Esporte_Nome = 'FUTEBOL'
    );
```

Esta consulta realiza as seguintes operações:

- Seleciona o CNPJ e o nome do patrocinador (`CNPJ, Nome`) da tabela `Patrocinadores`.
- Realiza junções (`JOIN`) entre as tabelas `Patrocinadores`, `Investe` e `Torneios` para identificar os patrocinadores e os torneios relacionados.
- Filtra os resultados para incluir apenas os torneios do esporte "FUTEBOL"(`T.Esporte_Nome = 'FUTEBOL'`).
- Agrupa os resultados pelos campos `CNPJ` e `Nome` do patrocinador.
- Aplica a cláusula `HAVING` para garantir que o patrocinador tenha investido em todos os torneios de "FUTEBOL", comparando o número de torneios em que o patrocinador investiu com o total de torneios de "FUTEBOL".

5 Aplicação

A aplicação implementada consiste em um sistema capaz de realizar o cadastro de novos jogadores no banco de dados e realizar consultas dos dados de jogadores já existentes no sistema.

5.1 Ferramentas utilizadas

A aplicação foi desenvolvida utilizando uma série de ferramentas atuais e amplamente utilizadas no mercado de trabalho. A linguagem de programação Python, na

versão 3.12.3, foi escolhida pela sua simplicidade e pela vasta gama de bibliotecas disponíveis, facilitando o desenvolvimento e a integração com outras tecnologias. Para a interação com o banco de dados PostgreSQL, foi utilizada a biblioteca Psycopg2, que oferece uma interface eficiente para a execução de comandos SQL e transações, permitindo uma comunicação fluida entre a aplicação Python e o banco de dados.

O sistema de gerenciamento de banco de dados escolhido para a aplicação foi o PostgreSQL, versão 17.2. A escolha do PostgreSQL proporciona uma simples integração com a aplicação em Python.

Para facilitar o desenvolvimento e garantir a consistência entre os diferentes ambientes, a aplicação foi containerizada com Docker. O Docker proporcionou um ambiente isolado para a execução tanto da aplicação Python quanto do banco de dados em PostgreSQL, garantindo que a aplicação funcionasse da mesma maneira em diferentes plataformas. O uso do Docker Compose permitiu orquestrar os containers de forma eficiente, facilitando a definição e execução de múltiplos serviços de forma integrada.

Essas ferramentas foram escolhidas por suas características de robustez, compatibilidade e flexibilidade, proporcionando um desenvolvimento ágil e uma solução final eficiente.

5.2 Utilização da aplicação

Para rodar a aplicação, o primeiro passo é baixar o Docker Desktop, disponível no link: `https://www.docker.com/products/docker-desktop`.

Em seguida, é necessário baixar o diretório da aplicação, que pode ser encontrado no repositório do GitHub: `https://github.com/LeoRissetto/Trabalho-BD`.

Após o download do repositório, entre no diretório `Trabalho-BD` e execute o comando a seguir:

```
docker-compose up --build -d
```

Este comando inicia e executa os serviços definidos no arquivo `docker-compose.yml`, além de iniciar o contêiner em segundo plano (por conta da flag `-d`). A flag `--build` só é necessária na primeira execução.

Com o contêiner rodando, execute o comando abaixo para rodar a aplicação dentro do contêiner, definida pelo serviço `python_app`, no arquivo `docker-compose.yml`:

```
docker compose run --rm python_app
```

O terminal exibirá a seguinte interface:

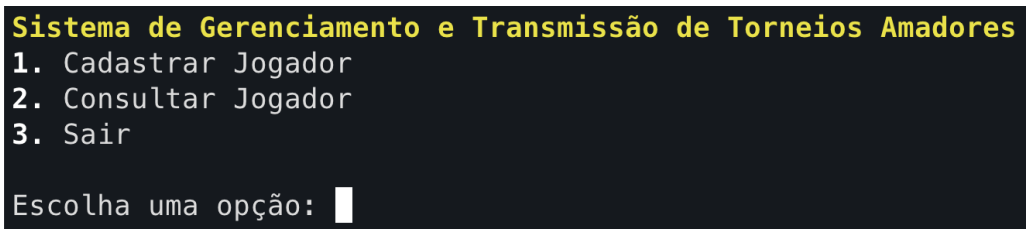


Figura 2: Interface do usuário com o sistema.

O sistema oferece duas funcionalidades principais:

- **Cadastrar Jogador:** Nessa funcionalidade, o usuário preenche os campos *CPF*, *Nome*, *Data de Nascimento*, *Gênero*, *Altura* e *Peso* do jogador, para que o cadastro seja realizado no banco de dados.
- **Consultar Jogador:** Nessa funcionalidade, o sistema solicita o *CPF* ou o *Nome* do jogador e retorna as seguintes informações sobre o jogador: *CPF*, *Nome*, *Data de Nascimento*, *Gênero*, *Altura* e *Peso*.

O banco de dados já inicia com um certo volume de jogadores, disponíveis para a consulta. Uma vez cadastrado o jogador no banco de dados, ele se torna disponível para consulta até que o volume do contêiner seja apagado.

Para parar a execução do contêiner, execute o comando:

```
docker compose down
```

Para apagar os volumes criados, ou seja, para apagar a instância da base de dados, execute:

```
docker compose down -v
```

6 Conclusão

O desenvolvimento deste projeto de banco de dados proporcionou, além de um grande aprendizado, diversos desafios para os integrantes do grupo. Entre os pontos de maior dificuldade, destacam-se a criação do Modelo Entidade-Relacionamento (MER) e a definição dos relacionamentos entre as entidades, o que exigiu um pensamento mais prático, fora do contexto puramente teórico. A criação do MER representou um grande desafio, pois ele foi o começo de tudo, uma parte do trabalho em que tivemos que pensar como seria o nosso banco de dados de forma conceitual. E justamente por ser tão abstrato e livre a diversas possibilidades, o MER foi difícil de ser implementado. Além disso, houve certa dificuldade na parte de implementação da aplicação ao utilizar ferramentas como o Docker, que adicionaram uma camada extra de complexidade ao processo. Por outro lado, a parte de mapeamento

do MER para o modelo relacional não trouxe grandes dificuldades, pois se tratou basicamente de um processo de tradução, em que foi necessário mapear as entidades e seus relacionamentos de forma mais direta.

O aprendizado com o projeto foi significativo, especialmente no que diz respeito à aplicação de conceitos teóricos em um cenário prático. A importância de uma boa modelagem de dados e o impacto de decisões iniciais no funcionamento do sistema foram lições importantes, aprendidas durante o processo de desenvolvimento deste projeto. A implementação da aplicação em Python e a necessidade de integrar com PostgreSQL foram aspectos fulcrais, que também ajudaram a ampliar o entendimento sobre como garantir a integridade e a usabilidade de um sistema.

Em termos de sugestões, o grupo não possui críticas negativas em relação ao projeto, pois ele aborda de forma prática todos os conceitos da disciplina, funcionando como uma excelente oportunidade para reforçar o aprendizado de Banco de Dados. A complexidade do projeto é alta, mas de maneira equilibrada, sem ser excessiva, e sem exigir conteúdos que não foram abordados em aula.

Em geral, o projeto foi uma experiência enriquecedora que contribuiu para o desenvolvimento de habilidades práticas e para a consolidação de conceitos importantes na área de Banco de Dados.