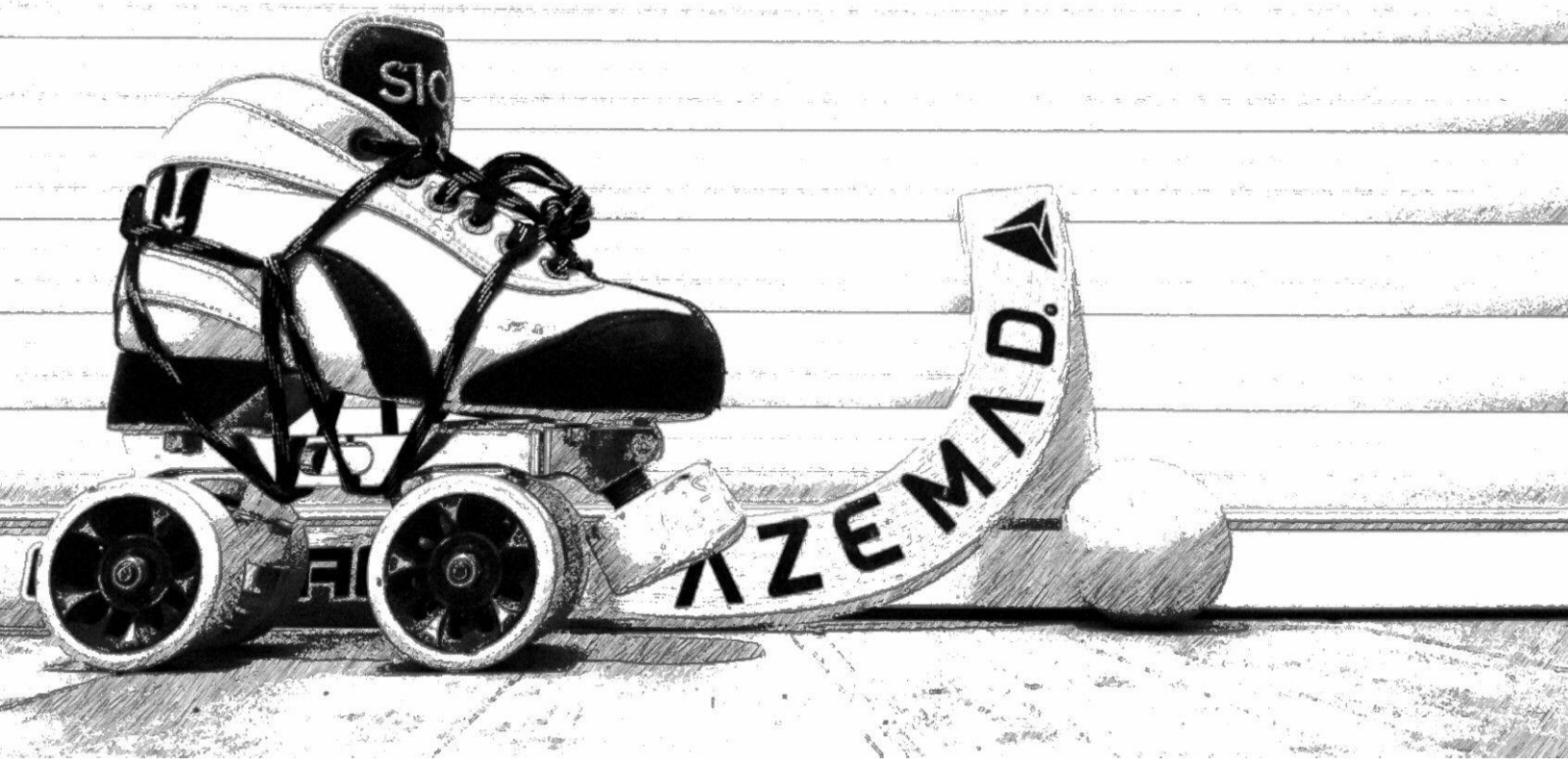
Torneio de Hoquei

universidade de aveiro

P4 - G7

Bases de Dados 2023

Hugo Correia nMec:108215
Joaquim Rosa nMec:109089



Índice

Conteúdo

Introdução

O objetivo deste projeto é criar uma interface aplicando uma base de dados que armazena informações relacionadas à primeira liga de hóquei em patins em Portugal. Dentro da informação da mesma está os clubes, jogadores, treinadores, árbitros, estádios, jogos, classificação dos clubes, estatísticas de jogadores (distinguido por campista e guarda-redes)

A intenção é ter uma plataforma onde administradores e clientes possam acessar e interagir com essas informações de forma eficiente e intuitiva.

A ideia deste projeto provém da dificuldade dos stakeholders em encontrarem uma plataforma que esteja sempre atualizada com as informações anteriormente referidas.

Na ótica dos clientes, o mesmo terão um acesso a todas as mesmas, podendo filtrar os dados por clubes, jogadores,etc, tudo de forma intuitiva.

Na ótica do admin, o mesmo poderá gerenciar a informação proveniente da base de dados atualizando os respectivos dados à medida da decorrência do torneio.

Na realização deste projeto os elementos integrantes do mesmo usaram o GitHub como plataforma de organização: BD_HoqueiDB

Estruturação do Projeto

Dentro do repositório BD_HoqueiDB e do ficheiro .zip estão contidos todos os ficheiros necessários para a execução correta do sistema.

Cada um das estruturas tem o seu próprio ficheiro por questões de organização. Dentro de cada ficheiro .sql estão presentes comentários indicando a funcionalidade da querie específica.

- **SQL Queries:**
  - **StoredProcedures.sql –** Script SQL contendo as Stored Procedures;
  - **UDFs.sql –** Script SQL contendo as Views;
  - **Indexes.sql –** Script SQL contendo os Indexes;
  - **Triggers.sql –** Script SQL contendo os Triggers;
  - **HoqueiSchema.sql –** Script SQL de criação dos esquema suporte do sistema;
  - **Insertions.sql –** Documento SQL contendo os dados a ser inseridos nas tabelas;
  - **SQL DDL.sql –** Script SQL de criação das tabelas necessárias ao sistema contendo todas as primary/foreign keys. (A cada execução refaz as tabelas);
  - **ClearSP.sql –** Script SQL com o intuíto de dar clear a todas as stored procedures existentes na base de dados.
  - **ClearUDFs.sql –** Script SQL com o intuíto de dar clear a todas as udfs existentes na base de dados.

- **Projeto_final**:
  - Pasta contendo todos os ficheiros de suporte à criação da interface;
  - Interface gráfica baseada em formulários gráficos básicos utilizando Windows Forms em Visual Studio (C#)
  - No contexto do problema tem como principal objetivo a manipulação da base de dados criada.

Requisitos Funcionais

Entidade	Funcionalidade
Cliente	● Visualizar informações sobre clubes, jogadores, treinadores, árbitros e pavilhão. ● Visualizar informações sobre jogos e estatísticas. ● Acessar a tabela de classificação atualizada. ● Filtrar e pesquisar informações com base em critérios específicos (por exemplo, jogadores por clube, resultados por equipa). ● Visualizar estatísticas individuais e coletivas dos jogadores e das equipas. ● Acompanhar o desempenho e histórico da temporada de jogadores, treinadores. ● Acompanhar calendário de jogos
Admin	● Gerir informações sobre jogadores, treinadores, árbitros, clubes e pavilhões ● Gerir informações sobre jogos (calendário) e estatísticas. ● Registrar resultados de jogos e atualizar automaticamente a tabela de classificação. ● Alterar a constituição de um clube, i.e remover/adicionar membros do mesmo (jogadores, treinadores, especialistas técnicos)

Entidades

Clube – Representa uma equipa de hóquei em patins participante da liga. Possui informações como ID do clube, nome e ano de fundação. Associado a cada clube estarão também associadas diversos atributos dependentes, sendo o conjunto dos mesmos a estatísticas de classificação desse mesmo clube;

Jogador – Representa um jogador de hóquei em patins. Contém campos como ID, nome do jogador, idade, posição, nacionalidade, número da camisola, ID do clube, advertências, cartões azuis e cartões vermelhos;

Jogador_Campo e Jogador_GuardaRedes: Representa o tipo especifico de jogador, incluindo as duas posições existentes no mundo do hoquei. Conteem informações relevantes para posição do jogador, como, por exemplo, golos marcados ou sofridos;

Especialista_Técnico – Representa um especialista técnico. O mesmo pode ter sua própria especialidade (mecânico, fisioterapeuta ou massagista), estão associados a um clube a partir do seu ID. Para além destas informações possui também o seu próprio ID, nome, idade e nacionalidade;

Treinador – Representa um treinador de hóquei em patins. Incluí informações associadas ao mesmo como ID do clube a que pertence, no seu próprio ID, nome, idade, tipo de treinador e nacionalidade;

Árbitro – Representa um árbitro que supervisiona os jogos da liga. Contém informações como o seu ID, nacionalidade, o tipo de árbitro e o seu nome;

Pavilhão – Representa o local onde os jogos são realizados. Contém o ID do pavilhão, nome, endereço, capacidade e o ID do clube ao qual pertence;

Jogo – Representa um jogo de hóquei em patins disputado entre dois plantéis de duas equipas. Armazena informações como o seu ID, a jornada a que pertence, o resultado final, data e hora, IDs dos clubes participantes, ID do pavilhão e se o jogo já foi ou não realizado;

Plantel – Representa o plantel escalado para x jogo da liga, incluído o seu próprio ID e o ID do jogo em questão;

Plantel_Jogadores e Plantel_Treinadores – Ambas contém num conjunto formam o plantel, tendo respetivamente jogadores e treinadores do mesmo com os seus IDs e com o ID do plantel em questão;

Diagrama Entidade-Relação

Tendo em consideração as entidades definidas anteriormente, foi construído o DER correspondente.

Esta versão não corresponde à apresentada anteriormente tendo sido sujeita a alterações tendo em conta o feedback dado pelo professor e necessidade de adaptarmos a estrutura para responder à análise de requisitos.

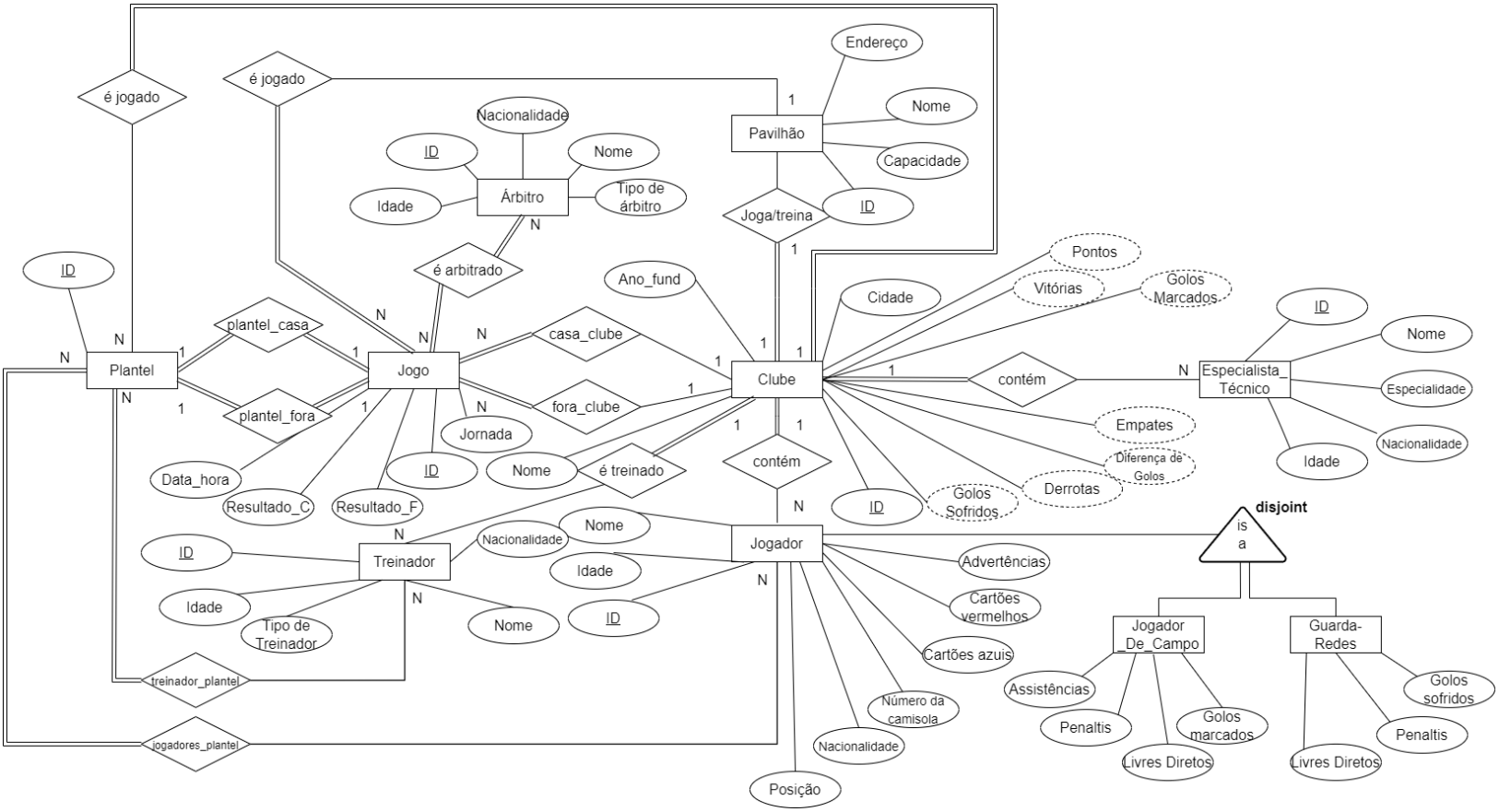


Figura 1 - Diagrama DER

Esquema Relacional

Este esquema representa a tradução do DER reproduzido anteriormente.

Nas relações N para M foram criadas 3 novas tabelas (Plantel_Treinadores, Plantel_Jogadores e e_arbitrado)

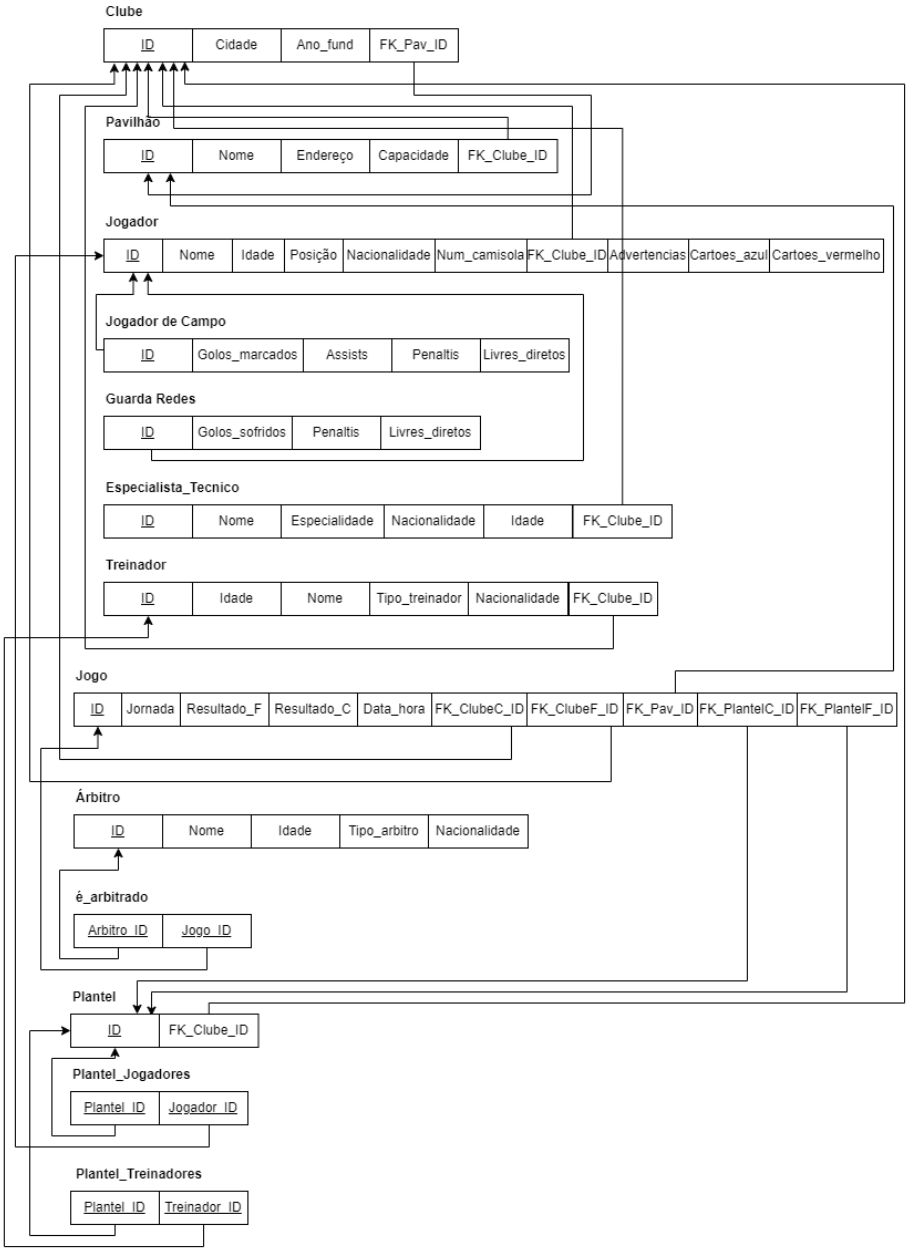


Figura 2 - Diagrama ER

Diagrama DDL da DB

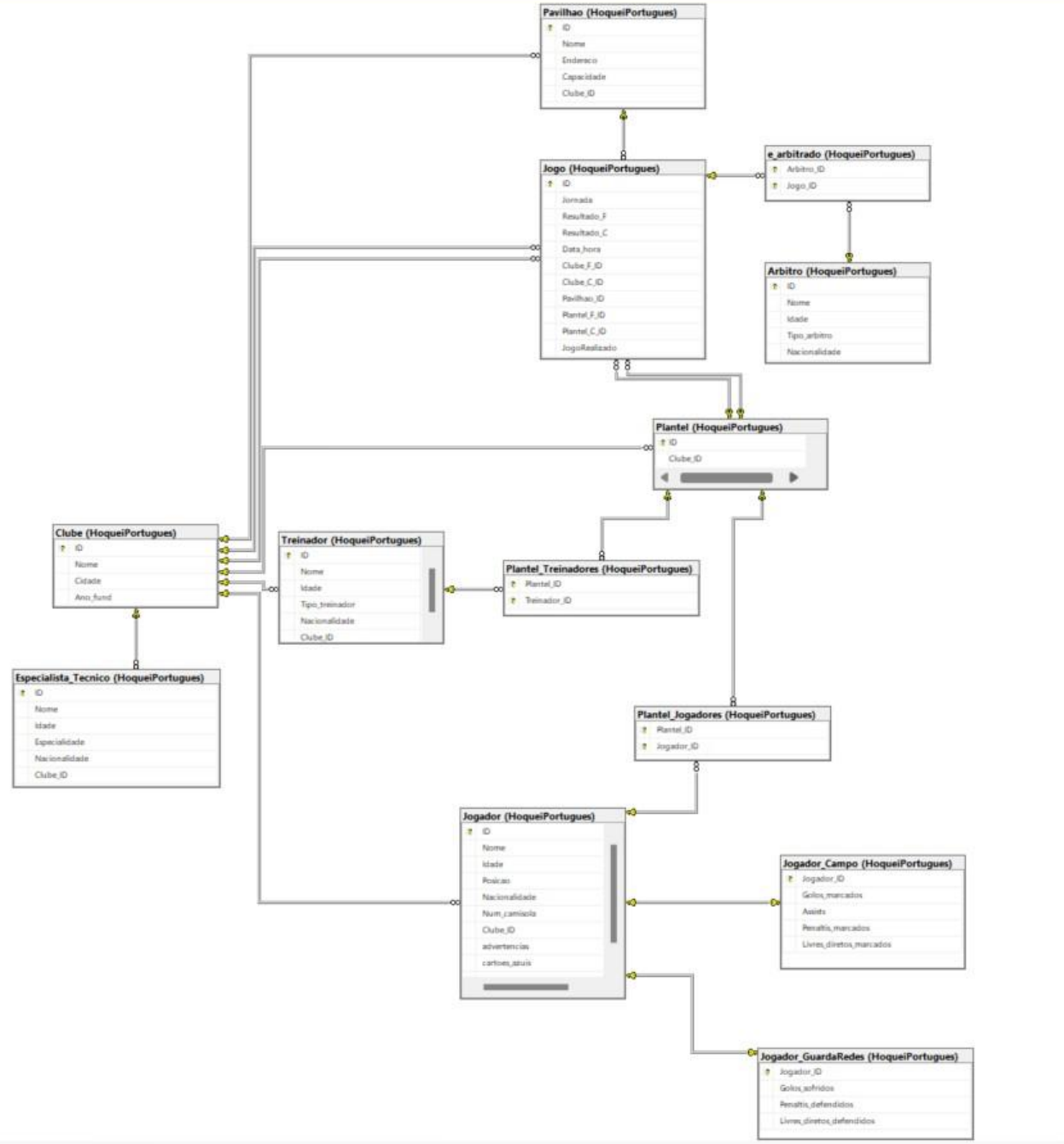


Figura 3 - Diagrama DDL

Interface

Como já foi referido anteriormente recorremos aos Windows Forms e à linguagem C# na criação da interface gráfica da Base de Dados *HoqueiPortugues.* Neste projeto foram criados os seguintes forms:

- **Login_Register –** Interface de registo/login do utilizador. O mesmo servirá de ponto de partida de cada um dos outros dois forms dependendo se for um *Cliente* ou um *Admin*

- **Main_Admin –** Interface onde estará o workflow todo. O admin ao contrário do Cliente poderá, para alem de consultar os dados presentes na manipulação da interface, manipular os mesmos, atualizando-os, eliminando-os ou adicionando-os:
  - **Classificação –** Nesta janela o admin não poderá alterar valores diretamente pois todos os atributos correspondentes aos da tabela estão diretamente dependentes de outros atributos como, por exemplo, o resultado dos jogos.
  - **Resultado –** Nesta janela o admin poderá aceder à diferentes jornada do torneio: dentro das mesmas terá duas hipóteses:
    - Se o jogo ainda contém valores null (i.e, não tem resultados), cria todos os dados necessários à criação do jogo e inicia-o (optámos por simular o jogo graças à grande manipulação de dados que envolveria);
    - Se o jogo já contiver dados: poderá aceder aos mesmos detalhadamente e alterar o resultado (caso se tenha enganado no mesmo, por exemplo)
  - **Equipa –** Dentro da janela Equipa o admin terá acesso aos jogos em que o mesmo está envolvido, acesso à equipa completa dos mesmo (jogadores e treinadores), podendo manipular de diversas maneiras os mesmos (adicionar jogador, remover do clube, etc) e por fim acesso aos especialista_tecnicos, podendo manipular também os mesmos.
  - **Jogadores –** Na janela jogadores, o mesmo poderá pesquisar por jogadores pertences ao campeonato, pesquisar pelos mesmos, filtrando-os de diversas maneiras e ordenando-os à sua vontade.

- **Main_Client –** Igual à do Admin, apenas com as restrições necessárias. Manipulação da base de dados indisponível para o mesmo. Tudo o resto estará disponível

SQL DML

A implementação deste projeto de base de dados de hóquei em patins foi estrategicamente estruturada de modo a que todos os comandos de manipulação de dados (DML) fossem executados através de Stored Procedures, User-Defined Function (UDFs) e Triggers. Esta abordagem foi adotada para criar um nível sofisticado de abstração entre a interface do utilizador e a Base de Dados. Essa camada de abstração não só aumenta a segurança do sistema, como também fortalece a integridade dos dados. Isto é conseguido através da implementação de verificações sistemáticas que previnem problemas comuns como as injeções de SQL. Além disso, essa estruturação permite o encapsulamento eficaz da lógica de negócio, garantindo que a manipulação e gestão dos dados sejam realizadas de forma coesa e consistente, alinhada com as melhores práticas de programação de bases de dados.

Normalização

Com base no que a plataforma precisa, nós desenhamos as relações e as entidades que são essenciais para que tudo funcione direito. Ao longo do trabalho, não deixamos de seguir as regras que nos permitiram chegar à Terceira Forma Normal (3FN), e sempre respeitando as formas normais anteriores. Deste modo, conseguimos uma estrutura bem organizada para a nossa base de dados.

Índices

Ao analisar as consultas necessárias para o funcionamento da aplicação, observou-se a frequência com que certos campos são usados para pesquisar e filtrar as mesmas. Esses campos incluem o *Nome* e *Nacionalidade* dos jogadores tanto como o *nome do clube* do próprio. Embora esses campos não sejam chaves primárias, são regularmente utilizados em consultas, justificando a necessidade de índices para melhorar a eficiência dessas operações.

Para otimizar a performance dessas consultas, foram criados índices *Non-clustered* nas respetivas colunas. O mesmo não foi repetido na ordenação das tabelas pois apesar de tornar a pesquisa um processo mais rápido de leitura, ao manipularmos dados com muitos índices, estes serão muito mais lentos a acrescentar o facto de um uso excessivo de índices neste caso ocuparia muito espaço de armazenamento.

```
/*
Indices para a pesquisa de jogadores por nome e nacionalidade
*/
CREATE INDEX idx_nome ON HoqueiPortugues.Jogador (Nome);
CREATE INDEX idx_nacionalidade ON HoqueiPortugues.Jogador (Nacionalidade);
CREATE INDEX idx_nome_clube ON HoqueiPortugues.Clube (Nome);
```

Figura 4 - Exemplos de Índices

Triggers

Para garantir a consistência dos dados e a integridade da base de dados foram criados Triggers.

Os seguintes Triggers foram criados para gerir diferentes manipulações por parte do admin:

```
CREATE TRIGGER HoqueiPortugues.verificarArbitroJornada
ON HoqueiPortugues.e_arbitrado
INSTEAD OF INSERT
AS
BEGIN
 -- Para cada linha na tabela inserida
    BEGIN TRY
        DECLARE @Jogo_ID int, @Arbitro_ID int;
        DECLARE inserted_cursor CURSOR FOR SELECT Jogo_ID, Arbitro_ID FROM inserted;
        OPEN inserted_cursor;

        FETCH NEXT FROM inserted_cursor INTO @Jogo_ID, @Arbitro_ID;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            -- Determina a jornada do jogo
            DECLARE @Jornada int;
            SELECT @Jornada = Jornada FROM HoqueiPortugues.Jogo WHERE ID = @Jogo_ID;

            -- Verifica se o árbitro já arbitrou um jogo nesta jornada
            IF EXISTS (SELECT 1
                       FROM HoqueiPortugues.e_arbitrado ea
                       JOIN HoqueiPortugues.Jogo j ON ea.Jogo_ID = j.ID
                       WHERE ea.Arbitro_ID = @Arbitro_ID AND j.Jornada = @Jornada)
            BEGIN
                RAISERROR('Um dos árbitros já arbitrou um jogo nesta jornada', 16, 1);
                RETURN;
            END

            -- Se o árbitro ainda não arbitraram um jogo nesta jornada, insere o árbitro
            INSERT INTO HoqueiPortugues.e_arbitrado (Jogo_ID, Arbitro_ID) VALUES (@Jogo_ID, @Arbitro_ID);

            FETCH NEXT FROM inserted_cursor INTO @Jogo_ID, @Arbitro_ID;
        END
    END TRY
    BEGIN CATCH
        IF CURSOR_STATUS('local', 'inserted_cursor') >= 0
        BEGIN
            CLOSE inserted_cursor;
            DEALLOCATE inserted_cursor;
        END;
        -- Re-lança o erro para que possa ser tratado por outro bloco CATCH ou retornado ao cliente
        THROW 50000, 'Erro ao inserir árbitro', 1;
    END CATCH
END;
GO
```

Figura 5 - Trigger verificarArbitroJornada

1. **eliminarUltimoPlantel –** Este Trigger é ativo antes da remoção de um plantel. Atualiza a tabela *Jogo* para que nenhum jogo referencie o plantel, sendo removido e removendo todas as referências ao plantel em outras tabelas;
2. **verificarArbitroJornada –** Este Trigger é ativo antes da inserção de um novo árbitro em um jogo. Ele verifica se o árbitro já está atribuído a um jogo na mesma jornada e, se for o caso, impede a inserção;
3. **verificarPlantelJogadores –** Este Trigger é ativo antes da inserção de um novo jogador em um plantel. Ele verifica se o jogador já está atribuído ao mesmo plantel e, se for o caso, impede a inserção;
4. **verificarPlantelTreinadores –** Semelhante ao anterior, este Trigger é ativo antes da inserção de um novo treinador em um plantel. Ele verifica se o treinador já está atribuído ao mesmo plantel e, se for o caso, impede a inserção.

Em destaque está o Trigger **verificarArbitroJornada** para além da preservação da lógica do problema, preserva também a integridade das regras do jogo:

UDFs

No decorrer da realização do projeto o uso de UDFs foi sendo pouco preciso já que em geral o ideal seria sempre criar uma SP em dez de uma UDF. Apesar disso foi priorizado o seu uso em relação às VIEWs devido ao seu maior desempenho e a capacidade de passar parâmetros.

```
IF OBJECT_ID('HoqueiPortugues.ufnJogadoresSemClube') IS NOT NULL
    DROP FUNCTION HoqueiPortugues.ufnJogadoresSemClube;
GO

CREATE FUNCTION HoqueiPortugues.ufnJogadoresSemClube()
RETURNS TABLE
AS
RETURN
(
    SELECT * FROM HoqueiPortugues.Jogador WHERE Clube_ID IS NULL
);
GO
```

Figura 6 - UDF ufnJogadoresSemClube

ufnJogadoresSemClube -- Esta função retorna uma tabela contendo os jogadores que não possuem um clube associado;

ufnTreinadoresSemClube -- Esta função retorna uma tabela contendo os treinadores que não possuem um clube associado;

fnObterTreinadoresPorJogo -- Esta função retorna uma tabela com os treinadores que participaram num jogo específico, incluindo seu ID de clube, nome e tipo de treinador. A função realiza uma junção entre as tabelas Jogo, Plantel, Plantel_Treinadores e Treinador para obter as informações necessárias;

fnObterJogadoresPorJogo -- Esta função retorna uma tabela com os jogadores que participaram num jogo específico, incluindo o ID do clube, nome do jogador e posição. A função realiza uma junção entre as tabelas Jogo, Plantel, Plantel_Jogadores e Jogador para obter as informações desejadas;

fnConsultarJogo – Esta função retorna informações detalhadas de um jogo específico, incluindo o ID do jogo, número da jornada, nome do pavilhão, nomes dos clubes que jogaram, IDs dos clubes, resultado do clube da casa, resultado do clube visitante, nome do árbitro e data/hora do jogo. A função realiza várias junções entre as tabelas Jogo, e_arbitrado, Arbitro, Pavilhao e Clube para obter os dados requeridos;

fnVerCalendarioEquipa -- Esta função retorna uma tabela com o calendário de jogos de uma determinada equipa. Ela exibe a jornada, nomes dos clubes que jogam, resultado do clube da casa, resultado do clube visitante, nome do pavilhão e data/hora do jogo. A função realiza junções entre as tabelas Jogo, Pavilhao e Clube para obter as informações desejadas;

Stored procedures

Grande parte das chamadas à Base de Dados são feitas através de Stored Procedures para criar uma camada de abstração. Segue portanto uma descrição breve de cada uma:

1. **HoqueiPortugues.calcularClassificacao –** Este procedure é usado para calcular a classificação de cada clube em uma liga de hóquei. Cria uma tabela temporária para armazenar informações como ID do Clube, Nome do Clube, Jogos, Pontos, Vitórias, Empates, Derrotas, Gols Marcados, Gols Sofridos e Diferença de Gols. O procedure percorre cada clube e calcula essas estatísticas com base em informações armazenadas na tabela Jogo. No final, retorna essas estatísticas e limpa a tabela temporária. O uso de uma tabela temporária ajuda a garantir a integridade dos dados e a eficiência do procedure;
2. **HoqueiPortugues.contratarJogador –** Este procedure é usado para contratar um novo jogador que não esteja na base de dados. Recebe detalhes como nome, idade, posição, nacionalidade, número da camisola e ID do clube. Verifica a existência do clube e a disponibilidade do número da camisola. O jogador é então inserido nas tabelas Jogador, Jogador_Campo ou Jogador_GuardaRedes, dependendo da sua posição. Isso mantém a integridade dos dados garantindo que cada jogador seja corretamente associado ao seu clube e que dois jogadores não compartilhem o mesmo número de camisola no mesmo clube;
3. **HoqueiPortugues.contratarJogadorClube –** Este procedure é usado para transferir um jogador de um clube para outro. Verifica se o jogador e os clubes existem e se o jogador está atualmente associado ao clube antigo. Em seguida, atualiza o ID do clube do jogador para o novo clube;
4. **HoqueiPortugues.jogadorSemClube –** Este procedure é usado para fazer um jogador sair de um clube, ficando assim sem clube. Verifica a existência do jogador e do clube, bem como se o jogador está associado a esse clube. Verifica também se o clube tem jogadores suficientes restantes. Se todas as condições forem atendidas, o jogador é atualizado para não ter clube;
5. **HoqueiPortugues.contratarJogadorSemClube –** Este procedure é usado para contratar um jogador que não tem clube. Verifica a existência do jogador e do clube, e se o jogador está atualmente sem clube. Se for verdade, atualiza o clube do jogador para o novo clube;

6. **HoqueiPortugues.contratarTreinadorClube –** Este procedure é utilizado para transferir um treinador de um clube para outro na base de dados HoqueiPortugues. Verifica primeiro a existência do treinador e do clube novo. Além disso, garante que o clube antigo não fique apenas com um treinador principal ou adjunto, mantendo a integridade dos dados. Além disso, valida se o clube novo não excederia o limite de 3 treinadores com a nova contratação. Se todas as condições forem satisfeitas, a associação do treinador ao clube é atualizada no banco de dados, assegurando a precisão e a proteção dos dados. Caso alguma das verificações falhe, um erro é levantado e o processo é interrompido, prevenindo a violação da regra de negócio.
7. **HoqueiPortugues.contratarTreinadorClube –** Este procedure é usado para transferir um treinador de um clube para outro. Verifica se o treinador e os clubes existem e se o treinador está atualmente associado ao clube antigo. Além disso, também verifica se o clube antigo teria pelo menos um treinador principal e um adjunto após a saída do treinador, e se o clube novo teria no máximo 3 treinadores após a adição. Se todas as condições forem atendidas, o clube do treinador é atualizado para o novo clube;
8. **HoqueiPortugues.treinadorSemClube –** Este procedure é usado para fazer um treinador sair de um clube, tornando-se assim sem clube. Verifica a existência do treinador e do clube, bem como se o treinador está associado a esse clube. Além disso, verifica se o clube teria pelo menos um treinador principal e um adjunto após a saída do treinador. Se todas as condições forem atendidas, o clube do treinador é atualizado para NULL;
9. **HoqueiPortugues.contratarTreinadorSemClube –** Este procedure é usado para contratar um treinador que atualmente não tem clube. Verifica a existência do treinador e do clube, e se o treinador está atualmente sem clube. Se for verdade, atualiza o clube do treinador para o novo clube;
10. **HoqueiPortugues.adicionarEspecialistaTecnico –** Este procedure é usado para adicionar um especialista técnico a um clube. Verifica a existência do clube e a validade da especialidade do especialista técnico, que deve ser uma destas três: 'Mecânico', 'Massagista', 'Fisioterapeuta'. Se for verdade, insere o novo especialista técnico na tabela EspecialistaTecnico;
11. **HoqueiPortugues.removerEspecialistaTecnico –** Este procedure é usado para remover um especialista técnico de um clube. Verifica a existência do clube e do especialista técnico, bem como se o especialista técnico está associado a esse clube. Se todas as condições forem atendidas, o especialista técnico é removido da tabela EspecialistaTecnico;

12. **HoqueiPortugues.simularJogo** - Este procedure é usado para simular um jogo de hóquei. Recebe os IDs do jogo e as pontuações finais dos dois planteis (Clubes diferentes) como entrada. Em seguida, faz o seguinte:
    - Procura os IDs do plantel dos dois últimos criados e atualiza a tabela Jogo com as pontuações finais e IDs do plantel;
    - Cria duas tabelas de variáveis para armazenar os IDs dos jogadores de cada plantel;
    - Atualiza as estatísticas dos jogadores de cada equipa de acordo com as regras especificadas (por exemplo, cada jogador tem uma certa probabilidade de receber um aviso, um cartão azul ou um cartão vermelho);
    - Atualiza as estatísticas do guarda-redes e dos jogadores de campo separadamente, distribuindo golos marcados e sofridos entre os jogadores de forma aleatória;
    - Este procedure é uma maneira de manter a integridade dos dados, pois garante que as estatísticas dos jogadores são atualizadas de forma consistente com o resultado do jogo;
13. **PesquisaJogadoresCampo -** Este procedure é usado para pesquisar e classificar os jogadores de campo por diferentes estatísticas. Recebe o nome do jogador, a coluna pela qual ordenar e opcionalmente a nacionalidade do jogador e o nome do clube como entrada. Em seguida, faz o seguinte:
    - Seleciona as estatísticas relevantes para todos os jogadores que correspondem aos critérios de pesquisa e ordena-os de acordo com a coluna especificada.
    - Este procedure ajuda a proteger os dados, pois evita a necessidade de o utilizador final escrever diretamente as consultas SQL para prcurar essas informações. Em vez disso, eles podem simplesmente chamar esse procedure com os parâmetros apropriados;
14. **HoqueiPortugues.PesquisaJogadoresCampo –** Esta stored procedure tem como objetivo pesquisar jogadores de campo com base em vários critérios, como nome, nacionalidade e clube. Também fornece a capacidade de ordenar os resultados com base em diferentes estatísticas de desempenho, como número de golos marcados, assistências, grandes penalidades marcadas, etc;
15. **HoqueiPortugues.PesquisaGuardaRedes –** Similar à primeira, esta stored procedure é usada para procurar guarda-redes baseado em vários critérios e ordenar os resultados com base em diferentes estatísticas de desempenho;

16. **HoqueiPortugues.STeliminarUltimoPlantel**
    HoqueiPortugues.STeliminarUltimosPlanteis – Estas stored procedures são usadas para eliminar o último ou os dois últimos plantéis criados respetivamente. Isso é útil para manter a atualidade dos dados no sistema e permitir que não sejam criados plantéis a mais preservando a cache do banco de dados;
17. **HoqueiPortugues.adicionarArbitros –** Este procedimento é usado para adicionar árbitros a um jogo específico, garantindo que os árbitros não sejam a mesma pessoa;
18. **HoqueiPortugues.eliminarArbitros –** Utilizado para remover os árbitros associados a um jogo específico;
19. **HoqueiPortugues.updateResultado –** Usado para atualizar o resultado de um jogo que já aconteceu. Este procedimento verifica se o jogo já foi realizado antes de permitir a atualização do resultado.

Abaixo estão apresentadas algumas das procedures que considerámos mais interessantes e que mais se destacam dentro do projeto. Destacamos as procedures **HoqueiPortugues.STeliminarUltimoPlantel e HoqueiPortugues.STeliminarUltimosPlanteis** pois apesar da sua utilização não ser explícita na interface, têm uma importância acrescida na gestão da DB. (Figura 10)

```
-- Distribuir as advertencias, cartoes azuis e cartoes vermelhos pelos jogadores da equipa de fora
WHILE EXISTS (SELECT * FROM @JogadoresF)
BEGIN
    SELECT TOP 1 @Jogador_Id = Jogador_ID FROM @JogadoresF ORDER BY NEWID();

    UPDATE HoqueiPortugues.Jogador
    SET advertencias = CASE WHEN RAND() < 0.4 THEN advertencias + CAST(ROUND((4 * RAND() + 1), 0) AS INT) ELSE advertencias END,
        cartoes_azuis = CASE WHEN RAND() < 0.15 THEN cartoes_azuis + 1 ELSE cartoes_azuis END,
        cartoes_vermelhos = CASE WHEN RAND() < 0.05 THEN cartoes_vermelhos + 1 ELSE cartoes_vermelhos END
    WHERE ID = @Jogador_Id;

    IF EXISTS (SELECT 1 FROM HoqueiPortugues.Jogador_GuardaRedes WHERE Jogador_ID = @Jogador_Id)
    BEGIN

        SET @NumberOfPlayersF = (SELECT COUNT(*) FROM @JogadoresF);

        IF @CounterF = 0
        BEGIN
            UPDATE HoqueiPortugues.Jogador_GuardaRedes
            SET Golos_sofridos = Golos_sofridos + @Resultado_C,
            Livres_Diretos_Defendidos = Livres_Diretos_Defendidos + CAST(ROUND((4 * RAND() + 1), 0) AS INT),
            Penaltis_Defendidos = Penaltis_Defendidos + CAST(ROUND((4 * RAND() + 1), 0) AS INT)
            WHERE Jogador_ID = @Jogador_Id;
        END


        IF @NumberOfPlayersF = 1
        BEGIN
            DELETE FROM @JogadoresF WHERE Jogador_ID = @Jogador_Id;
        END
        SET @CounterF = @CounterF + 1;
    END
    ELSE
    BEGIN
        -- Distribuir os golos pelos jogadores de campo
        IF @GolosMarcadosF > 0
        BEGIN

            SET @Livres_Diretos_DefendidosC = CASE WHEN RAND() < 0.80 THEN 1 ELSE 0 END;
            SET @Penaltis_DefendidosC = CASE WHEN @Livres_Diretos_DefendidosC = 0 AND RAND() < 0.80 THEN 1 ELSE 0 END;

            UPDATE HoqueiPortugues.Jogador_Campo
            SET Golos_marcados = Golos_marcados + 1,
                Livres_diretos_marcados = CASE WHEN @Livres_Diretos_DefendidosC = 0 AND @Penaltis_DefendidosC = 0 THEN Livres_diretos_marcados + 1 ELSE Livres_diretos_marcados END,
                Penaltis_marcados = CASE WHEN @Penaltis_DefendidosC = 0 AND @Livres_Diretos_DefendidosC = 0 THEN Penaltis_marcados + 1 ELSE Penaltis_marcados END
            WHERE Jogador_ID = @Jogador_Id;

            SET @GolosMarcadosF = @GolosMarcadosF - 1;

            IF @AssistsF < @Resultado_F
            BEGIN
                SELECT TOP 1 @Assistant_Id = Jogador_ID FROM @JogadoresF ORDER BY NEWID();

                UPDATE HoqueiPortugues.Jogador_Campo
                SET Assists = Assists + 1
                WHERE Jogador_ID = @Assistant_Id;
                SET @AssistsF = @AssistsF + 1;
            END
        END
        ELSE
        BEGIN
            DELETE FROM @JogadoresF WHERE Jogador_ID = @Jogador_Id;
        END
    END
END
```

Figura 7 - Um excerto da ST simularJogo

```sql
CREATE PROCEDURE HoqueiPortugues.jogadorSemClube
    @Jogador_ID AS int , @Clube_ID AS int
AS
BEGIN
    --Verifica se o jogador existe
    IF NOT EXISTS (SELECT * FROM HoqueiPortugues.Jogador WHERE ID = @Jogador_ID)
        BEGIN
            RAISERROR('Jogador não existe', 16, 1);
            RETURN;
        END

    --Verifica se o clube existe
    IF NOT EXISTS (SELECT * FROM HoqueiPortugues.Clube WHERE ID = @Clube_ID)
        BEGIN
            RAISERROR('Clube não existe', 16, 1);
            RETURN;
        END

    --Verifica se o clube ficaria sem jogadores suficientes (8)
    IF (SELECT COUNT(*) FROM HoqueiPortugues.Jogador WHERE Clube_ID = @Clube_ID) = 8
        BEGIN
            RAISERROR('O clube não pode ficar só com 8 jogadores', 16, 1);
            RETURN;
        END

    --Verifica se o jogador pertence ao clube
    IF NOT EXISTS (SELECT * FROM HoqueiPortugues.Jogador WHERE ID = @Jogador_ID AND Clube_ID = @Clube_ID)
        BEGIN
            RAISERROR('Jogador não pertence ao clube', 16, 1);
            RETURN;
        END
    ELSE
        BEGIN
            UPDATE HoqueiPortugues.Jogador SET Clube_ID = NULL WHERE ID = @Jogador_ID AND Clube_ID = @Clube_ID;
        END
END;

/*
Contratar um jogador que não tem clube
*/

--Exclui o procedimento se ele já existir
IF OBJECT_ID('HoqueiPortugues.contratarJogadorSemClube', 'P') IS NOT NULL DROP PROCEDURE HoqueiPortugues.contratarJogadorSemClube;
GO

CREATE PROCEDURE HoqueiPortugues.contratarJogadorSemClube
    @Jogador_ID AS int , @Clube_Novo AS int
AS
BEGIN
    --Verifica se o jogador existe
    IF NOT EXISTS (SELECT * FROM HoqueiPortugues.Jogador WHERE ID = @Jogador_ID)
        BEGIN
            RAISERROR('Jogador não existe', 16, 1);
            RETURN;
        END

    --Verifica se o clube existe
    IF NOT EXISTS (SELECT * FROM HoqueiPortugues.Clube WHERE ID = @Clube_Novo)
        BEGIN
            RAISERROR('Clube não existe', 16, 1);
            RETURN;
        END

    --Verifica se o jogador pertence ao clube
    IF NOT EXISTS (SELECT * FROM HoqueiPortugues.Jogador WHERE ID = @Jogador_ID AND Clube_ID IS NULL)
        BEGIN
            RAISERROR('Não existem jogadores sem Clube', 16, 1);
            RETURN;
        END
    ELSE
        BEGIN
            UPDATE HoqueiPortugues.Jogador SET Clube_ID = @Clube_Novo WHERE ID = @Jogador_ID AND Clube_ID IS NULL;
        END
END;
```

Figura 8 - STs jogadorSemClube e contratarJogadorSemClube

```sql
--Exclui o procedimento se ele já existir
IF OBJECT_ID('HoqueiPortugues.PesquisaJogadoresCampo', 'P') IS NOT NULL DROP PROCEDURE HoqueiPortugues.PesquisaJogadoresCampo;
GO

CREATE PROCEDURE HoqueiPortugues.PesquisaJogadoresCampo
    @nome varchar(50),
    @orderby varchar(50),
    @nacionalidade varchar(50) = NULL,
    @nome_clube varchar(50) = NULL
AS
BEGIN
    SELECT J.ID, J.Nome, C.ID as Clube_ID, C.Nome as Clube_Nome, JC.Golos_marcados, JC.Assists, J.advertencias, J.cartoes_azuis, J.cartoes_vermelhos, JC.Penaltis_marcados, JC.Livres_diretos_marcados
    FROM HoqueiPortugues.Jogador J
    INNER JOIN HoqueiPortugues.Jogador_Campo JC ON J.ID = JC.Jogador_ID
    INNER JOIN HoqueiPortugues.Clube C ON J.Clube_ID = C.ID
    WHERE J.Nome LIKE '%' + @Nome + '%' AND
        (@nacionalidade IS NULL OR J.Nacionalidade = @nacionalidade) AND
        (@nome_clube IS NULL OR C.Nome = '%' + @nome_clube + '%')
    ORDER BY
        CASE
            WHEN @orderby = 'Golos_Marcados' THEN JC.Golos_marcados
            WHEN @orderby = 'Assists' THEN JC.Assists
            WHEN @orderby = 'Penaltis_Marcados' THEN JC.Penaltis_marcados
            WHEN @orderby = 'Livres_Diretos_Marcados' THEN JC.Livres_diretos_marcados
            WHEN @orderby = 'Advertencias' THEN J.advertencias
            WHEN @orderby = 'Cartoes_Azuis' THEN J.cartoes_azuis
            WHEN @orderby = 'Cartoes_Vermelhos' THEN J.cartoes_vermelhos
        ELSE J.ID
    END
END;
GO

EXEC HoqueiPortugues.PesquisaJogadoresCampo @nome = 'Ma', @orderby = 'Golos_Marcados';

/*
Pesquisar e ordenar guarda redes por diferentes estatisticas
*/

--Exclui o procedimento se ele já existir
IF OBJECT_ID('HoqueiPortugues.PesquisaGuardaRedes', 'P') IS NOT NULL DROP PROCEDURE HoqueiPortugues.PesquisaGuardaRedes;
GO

CREATE PROCEDURE HoqueiPortugues.PesquisaGuardaRedes
    @nome varchar(50),
    @orderby varchar(50)
AS
BEGIN
    SELECT J.ID, J.Nome, C.ID as Clube_ID, C.Nome as Clube_Nome, JG.Golos_sofridos, J.advertencias ,J.cartoes_azuis, J.cartoes_vermelhos, JG.Penaltis_defendidos, JG.Livres_diretos_defendidos
    FROM HoqueiPortugues.Jogador J
    INNER JOIN HoqueiPortugues.Jogador_GuardaRedes JG ON J.ID = JG.Jogador_ID
    INNER JOIN HoqueiPortugues.Clube C ON J.Clube_ID = C.ID
    WHERE J.Nome LIKE '%' + @Nome + '%' AND
        (@nacionalidade IS NULL OR J.Nacionalidade = @nacionalidade) AND
        (@nome_clube IS NULL OR C.Nome = '%' + @nome_clube + '%')
    ORDER BY
        CASE
            WHEN @orderby = 'Golos_Sofridos' THEN JG.Golos_sofridos
            WHEN @orderby = 'Penaltis_Defendidos' THEN JG.Penaltis_defendidos
            WHEN @orderby = 'Livres_Diretos_Defendidos' THEN JG.Livres_diretos_defendidos
            WHEN @orderby = 'Advertencias' THEN J.advertencias
            WHEN @orderby = 'Cartoes_Azuis' THEN J.cartoes_azuis
            WHEN @orderby = 'Cartoes_Vermelhos' THEN J.cartoes_vermelhos
        ELSE J.ID
    END;
END
GO
```

Figura 9 - STs PesquisarJogadorCampo e PesquisarGuardaRedes

```sql
/*
Eliminar o ultimo plantel criado
*/


--Exclui o procedimento se ele já existir
IF OBJECT_ID('HoqueiPortugues.STeliminarUltimoPlantel', 'P') IS NOT NULL DROP PROCEDURE HoqueiPortugues.STeliminarUltimoPlantel;
GO

CREATE PROCEDURE HoqueiPortugues.STeliminarUltimoPlantel
AS
BEGIN
    DELETE TOP (1) FROM HoqueiPortugues.Plantel WHERE ID = (SELECT MAX(ID) FROM HoqueiPortugues.Plantel);
END
GO

EXEC HoqueiPortugues.STeliminarUltimoPlantel;

/*
Eliminar os dois ultimos planteis criados
*/

--Exclui o procedimento se ele já existir
IF OBJECT_ID('HoqueiPortugues.STeliminarUltimosPlanteis', 'P') IS NOT NULL DROP PROCEDURE HoqueiPortugues.STeliminarUltimosPlanteis;
GO

CREATE PROCEDURE HoqueiPortugues.STeliminarUltimosPlanteis
AS
BEGIN
    DELETE TOP (2) FROM HoqueiPortugues.Plantel WHERE ID IN (SELECT TOP (2) ID FROM HoqueiPortugues.Plantel ORDER BY ID DESC);
END
```

Figura 10 - STs STeliminarUltimoPlantel e STeliminarUltimosPlanteis

Conclusão

Com este projeto podemos ver na prática o funcionamento e a utilidade se um Sistema de Gestão de Base de Dados, bem como a importância de ter um local centralizado da informação para garantir a consistência e segurança dos dados. Apesar de não ser a parte central do trabalho, o facto de ter uma interface intuitiva permitiu que a visualização dos dados e a interação com estes fosse mais fluída.

Concluindo, os objetivos propostos no início do projeto foram na sua grande maioria alcançados com o acrescento e a remoção de algumas features ao longo da sua realização. Este conjunto de fatores permitiu o aprofundamento do conhecimento relacionado à matéria lecionada na cadeira de Base de Dados.