Relatório Base de Dados

P6G5
João Santos - 110555
Henrique Dias - 98153

Índice

Introdução

A aplicação permitirá seguir diferentes temporadas da liga Norte Americana de basquetebol (NBA). A interface foi criada para uso administrativo na gestão dos diversos dados disponíveis com o intuito de ser utilizado por 3 tipos diferentes de utilizadores (utilizador comum, equipa e administrador).

Os dados foram recolhidos através do API oficial da NBA e processados através de pequenos scripts em Python.

Análise de Requisitos

ENTIDADES	FUNÇÕES
Viewer	● Ver as equipas pertencentes à liga e as respectivas informações (treinador sponsor etc) ● Filtrar as equipas por conferência ou por divisão ● Ver a lista de jogos realizados por determinada equipa. ● Ver o plantel de determinada equipa e filtrar por ano ● Ver a estatística média de um jogador.
Team Manager	● Procurar equipa pelo nome ● Adicionar, editar ou remover jogador de uma equipa ● Adicionar, editar ou remover treinador de uma equipa
League Manager	● Procurar equipa pelo nome ● Adicionar/registar jogos de uma equipa ● Ver e editar os pontos da equipa da casa, da equipa visitante e dos jogadores que fizeram parte desse jogo

Entidades

Season - Representa uma temporada da liga. Possui um ID, um ano e um vencedor.

Game - Representa um jogo de uma temporada. Possui um ID, uma data e um vencedor.

Team - Representa uma equipa da liga. Possui um ID, um nome, o nome da cidade onde se localiza e a relação Vitórias-Derrotas da equipa nessa temporada.

Division - Representa uma das divisões da liga, por qual as equipas se distribuem. Possui um ID e nome.

Conference - Representa uma das conferências da liga, onde as divisões, e por consequência, as equipas se distribuem. Possui um ID, Nome e Vencedor.

Squad - Representa o plantel de uma equipa numa dada temporada. Possui um ID e um ano.

Coach - Representa um treinador na liga. Possui um ID, um nome e a idade do treinador.

Player - Representa um jogador na liga. Possui um ID, um nome, a idade, a posição, o peso e a altura do jogador.

Sponsor - Representa um patrocinador na liga. Possui um ID, um nome e um website.

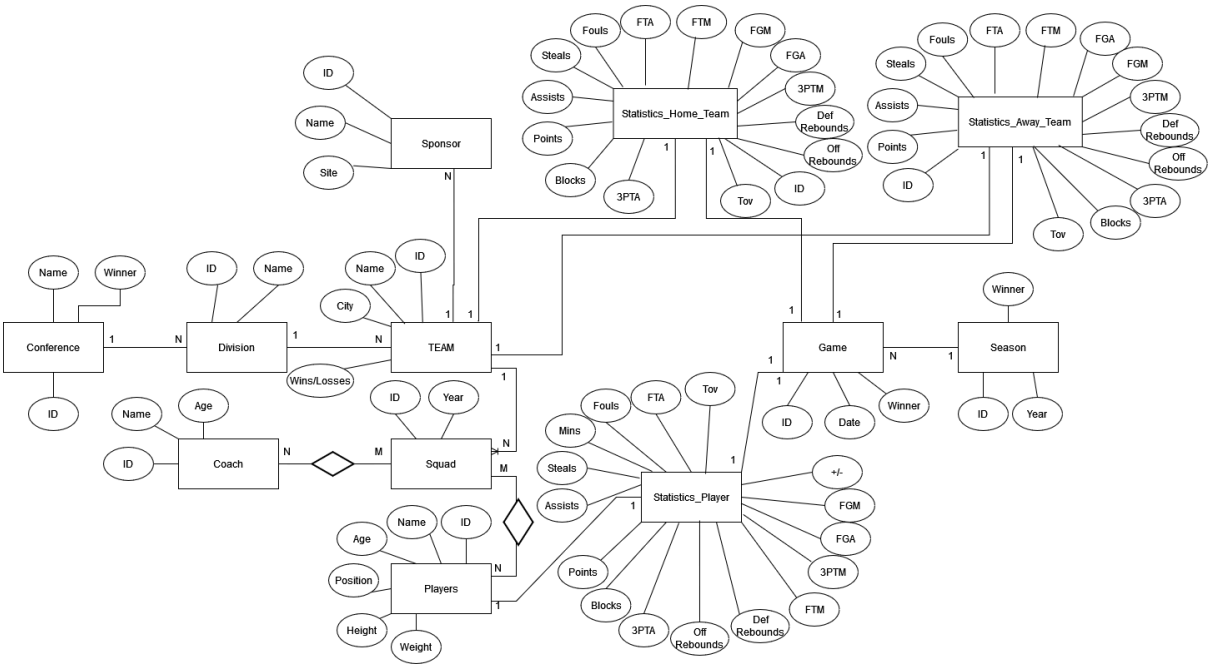
Statistics_Home_Team - Representa as estatísticas da equipa da casa de um jogo. Possui um ID e as estatísticas básicas da equipa nesse jogo (tentativas de lançamento, lançamentos marcados, tentativas de lançamento de 3 pontos, lançamento de 3 pontos marcados, tentativas de lançamento livre, lançamentos livres marcados, ressaltos ofensivos, ressaltos defensivos, assistências, roubos de bola, desarmes de lançamento, faltas e perdas de posse de bola).

Statistics_Away_Team - Representa as estatísticas da equipa visitante de um jogo. Possui um ID e as estatísticas básicas da equipa nesse jogo.

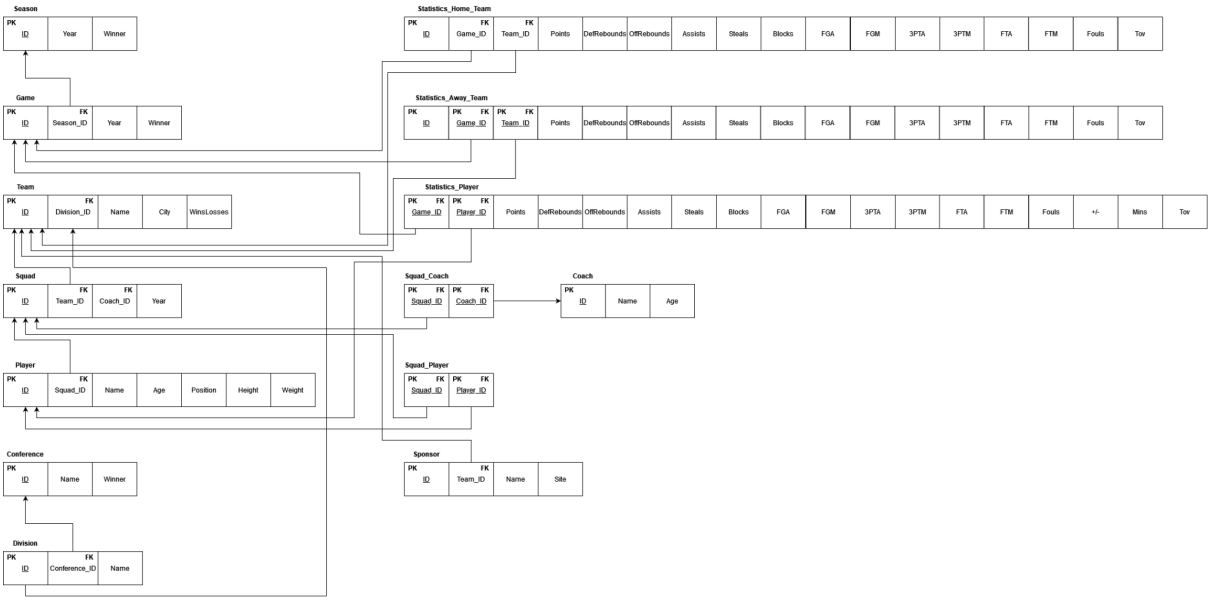
Statistics_Player - Representa as estatísticas individuais de um jogador num jogo. Possui as estatísticas básicas do jogador nesse jogo (minutos, tentativas de lançamento, lançamentos marcados, tentativas de lançamento de 3 pontos, lançamento de 3 pontos marcados, tentativas de lançamento livre, lançamentos livres marcados, ressaltos ofensivos, ressaltos defensivos, assistências, roubos de bola, desarmes de lançamento, faltas, perdas de posse de bola e relação mais/menos).

Diagramas

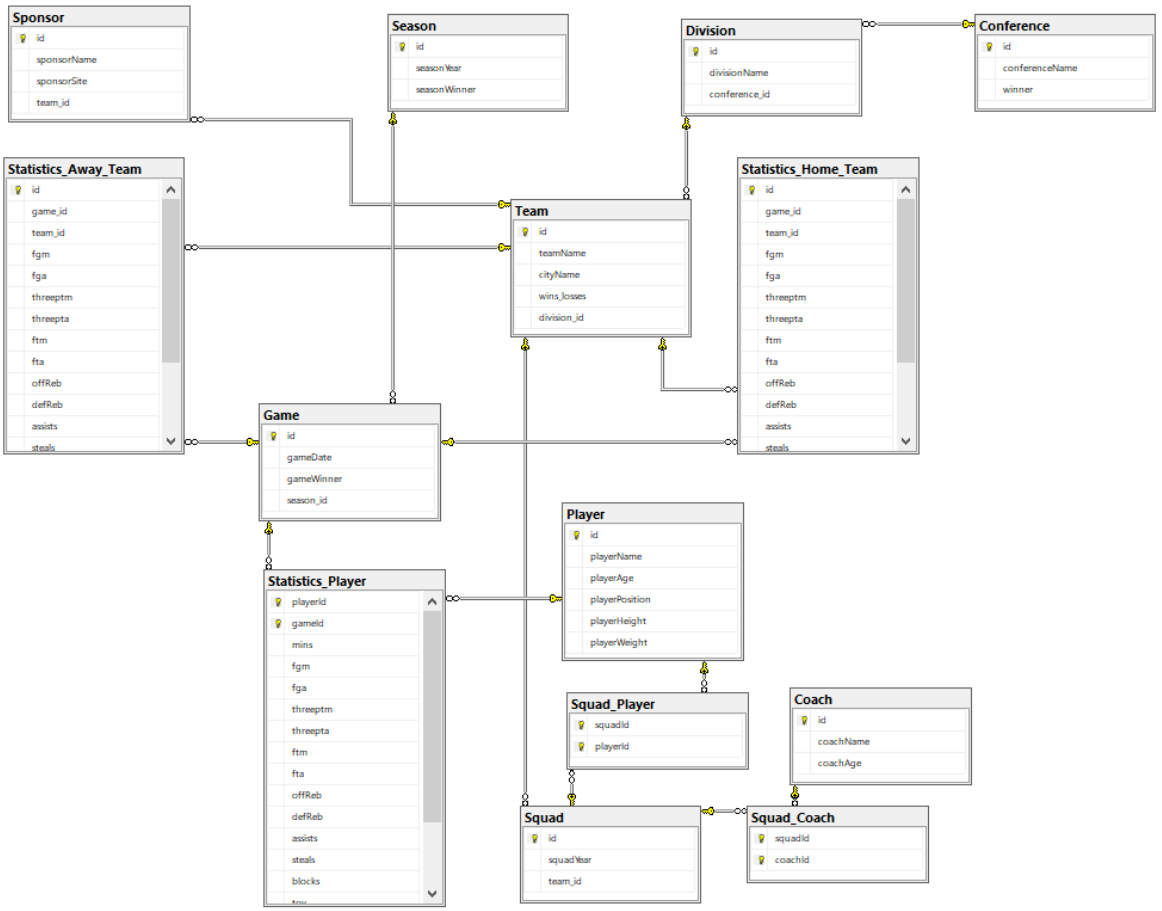
Diagrama Entidade-Relacionamento



Esquema Relacional



Esquema Relacional (SGBD)



Queries SQL

Data Definition Language (DDL)

DDL é uma sub-linguagem do SQL (Structured Query Language) utilizada para definir e gerir a estrutura de uma base de dados.

DDL foi utilizado na definição das tabelas na base de dados como se pode ver na figura seguinte.

```sql
CREATE TABLE Conference (
    id INT PRIMARY KEY,
    conferenceName VARCHAR(255) NOT NULL,
    winner INT
);

CREATE TABLE Division (
    id INT PRIMARY KEY,
    divisionName VARCHAR(255) NOT NULL,
    conference_id INT,
    FOREIGN KEY (conference_id) REFERENCES Conference(id)
);

CREATE TABLE Team (
    id INT PRIMARY KEY,
    teamName VARCHAR(255) NOT NULL,
    cityName VARCHAR(255) NOT NULL,
    wins_losses VARCHAR(255) NOT NULL,
    conference_id INT,
    FOREIGN KEY (conference_id) REFERENCES Conference(id)
);

CREATE TABLE Sponsor (
    id INT PRIMARY KEY,
    sponsorName VARCHAR(255) NOT NULL,
    sponsorSite VARCHAR(255) NOT NULL,
    team_id INT,
    FOREIGN KEY (team_id) REFERENCES Team(id)
);

CREATE TABLE Squad (
    id INT PRIMARY KEY,
    squadYear INT NOT NULL,
    team_id INT,
    FOREIGN KEY (team_id) REFERENCES Team(id)
);

CREATE TABLE Coach (
    id INT PRIMARY KEY,
    coachName VARCHAR(255) NOT NULL,
    coachAge INT NOT NULL
);
```

```sql
CREATE TABLE Player (
    id INT PRIMARY KEY,
    playerName VARCHAR(255) NOT NULL,
    playerAge INT NOT NULL,
    playerPosition VARCHAR(255) NOT NULL,
    playerHeight INT NOT NULL,
    playerWeight INT NOT NULL
);

CREATE TABLE Squad_Coach (
    squadId INT,
    coachId INT,
    PRIMARY KEY (squadId, coachId),
    FOREIGN KEY (squadId) REFERENCES Squad(id),
    FOREIGN KEY (coachId) REFERENCES Coach(id)
);

CREATE TABLE Squad_Player (
    squadId INT,
    playerId INT,
    PRIMARY KEY (squadId, playerId),
    FOREIGN KEY (squadId) REFERENCES Squad(id),
    FOREIGN KEY (playerId) REFERENCES Player(id)
);

CREATE TABLE Season (
    id INT PRIMARY KEY,
    seasonYear INT NOT NULL,
    seasonWinner INT
);

CREATE TABLE Game (
    id INT PRIMARY KEY,
    gameDate DATE NOT NULL,
    gameWinner INT,
    season_id INT,
    FOREIGN KEY (season_id) REFERENCES Season(id)
);
```

```sql
CREATE TABLE Statistics_Player (
    playerId INT,
    gameId INT,
    assists INT NOT NULL,
    steals INT NOT NULL,
    mins INT NOT NULL,
    fouls INT NOT NULL,
    plus_minus INT NOT NULL,
    fgm INT NOT NULL,
    fga INT NOT NULL,
    three_ptm INT NOT NULL,
    defReb INT NOT NULL,
    offReb INT NOT NULL,
    three_pta INT NOT NULL,
    blocks INT NOT NULL,
    points INT NOT NULL,
    PRIMARY KEY (playerId, gameId),
    FOREIGN KEY (playerId) REFERENCES Player(id),
    FOREIGN KEY (gameId) REFERENCES Game(id)
);

CREATE TABLE Statistics_Away_Team (
    game_id INT,
    team_id INT,
    fgm INT NOT NULL,
    fga INT NOT NULL,
    threeptm INT NOT NULL,
    threepta INT NOT NULL,
    ftm INT NOT NULL,
    fta INT NOT NULL,
    offReb INT NOT NULL,
    defReb INT NOT NULL,
    assists INT NOT NULL,
    steals INT NOT NULL,
    blocks INT NOT NULL,
    tov INT NOT NULL,
    fouls INT NOT NULL,
    points INT NOT NULL,
    PRIMARY KEY (game_id, team_id),
    FOREIGN KEY (game_id) REFERENCES Game(id),
    FOREIGN KEY (team_id) REFERENCES Team(id)
)
```

DML

DML é uma sub-linguagem do SQL (Structured Query Language) utilizada para manipular e gerir os dados dentro das estruturas definidas na base de dados.

DML foi utilizado para a inserção de alguns dos dados da base de dados, os outros foram introduzidos utilizando o GUI do SQL Server Management.

Na figura 5 está representado um exemplo da utilização de DML na base de dados, o resto está situado no ficheiro *DML.sql*

```
insert into Statistics_Player values
(1627752,22300061,'00:29:53',6,8,4,6,2,2,1,2,1,0,1,1,0,18,-14.0),
(2544,22300061,'00:29:01',10,16,1,4,0,1,1,7,5,1,0,0,1,21,7.0),
(203076,22300061,'00:34:09',6,17,1,2,4,4,1,7,4,0,2,2,3,17,-17.0),
(1630559,22300061,'00:31:20',4,11,1,2,5,7,4,4,4,2,0,2,2,14,-14.0),
(1626156,22300061,'00:36:11',4,12,2,5,1,2,0,4,7,1,0,3,3,11,1.0),
(1629060,22300061,'00:14:38',3,10,0,3,0,0,2,1,0,0,0,0,2,6,-8.0),
(1629216,22300061,'00:22:18',3,8,0,4,0,0,1,0,2,1,0,2,3,6,-17.0),
(1629637,22300061,'00:06:54',0,0,0,0,0,0,0,1,0,0,0,0,1,0,-7.0),
(1629629,22300061,'00:17:38',2,4,1,2,2,2,2,2,0,0,1,0,2,7,7.0),
(1626174,22300061,'00:15:28',3,4,0,1,1,2,1,3,0,0,0,1,1,7,2.0),
(1631108,22300061,'00:01:15',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.0),
(1641721,22300061,'00:01:15',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.0),
(1629008,22300061,'00:30:07',5,13,2,9,0,0,2,10,2,2,0,0,1,12,12.0),
(203932,22300061,'00:34:58',7,11,1,2,0,0,2,5,5,2,1,0,0,15,6.0),
(203999,22300061,'00:36:15',12,22,3,5,2,4,3,10,11,1,1,2,2,29,15.0),
(203484,22300061,'00:36:14',8,12,2,3,2,2,1,1,1,3,1,3,5,20,10.0),
(1627750,22300061,'00:34:14',8,13,3,5,2,2,0,2,6,0,1,1,3,21,3.0),
(202704,22300061,'00:24:04',3,8,2,5,0,0,0,3,1,1,0,2,0,8,11.0),
(1631212,22300061,'00:10:51',1,3,1,3,0,0,0,0,0,0,1,1,1,3,1.0),
(1630192,22300061,'00:11:45',1,3,0,1,2,2,0,0,1,0,0,1,2,4,-3.0),
(1631128,22300061,'00:19:20',2,5,0,1,1,2,1,2,2,0,1,1,1,5,5.0),
(1631221,22300061,'00:00:44',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.0),
(1630296,22300061,'00:00:44',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.0),
(1629618,22300061,'00:00:44',1,1,0,0,0,0,0,0,0,0,0,0,2,0.0),
(1629006,22300062,'00:31:38',7,9,1,1,2,2,4,1,1,1,0,2,2,17,2.0),
```

Views

Views permitem ao utilizador criar uma apresentação virtual dos dados de uma ou mais tabelas. As views são como consultas armazenadas que podem ser tratadas como tabelas, mas não armazenam dados por si mesmas.

Na figura estão os Views que foram utilizados na base de dados.

```
CREATE VIEW SquadPlayersView
AS
SELECT P.id AS PlayerId, P.playerName, P.playerAge, P.playerPosition, P.playerHeight, P.playerWeight,
       S.team_id AS TeamId, S.squadYear
FROM Player P
INNER JOIN Squad_Player SP ON P.id = SP.playerId
INNER JOIN Squad S ON SP.squadId = S.id;

CREATE VIEW TeamStatisticsView
AS
SELECT SH.game_id AS GameId, SH.team_id AS TeamId, 'Home' AS Location, SH.points AS Points
FROM Statistics_Home_Team SH
UNION ALL
SELECT SA.game_id AS GameId, SA.team_id AS TeamId, 'Away' AS Location, SA.points AS Points
FROM Statistics_Away_Team SA;
```

Indexes

Indexes servem para melhorar a velocidade das operações de consulta. Os índices permitem acesso mais rápido e eficiente aos dados de uma tabela.

```
CREATE INDEX IX_Team_TeamName ON Team (teamName);
```

User Defined Functions

UDF são funções criadas pelo utilizador numa base de dados para realizar operações que não estão disponíveis nas funções integradas do SGBD (Sistema de Gestão de Bases de Dados). As UDFs permitem estender as funcionalidades do SQL, dando a possibilidade de simplificar consultas e melhorar a reutilização de código.

Abaixo temos algumas udfs implementadas.

```sql
CREATE FUNCTION [dbo].[GetSponsorName] (@TeamId INT)
RETURNS NVARCHAR(100)
AS
BEGIN
    DECLARE @SponsorName NVARCHAR(100);

    SELECT @SponsorName = sponsorName
    FROM Sponsor
    WHERE team_id = @TeamId;

    RETURN @SponsorName;
END;
```

```sql
CREATE FUNCTION [dbo].[employeeDeptHighAverage](@departId  INT)
RETURNS @table TABLE (
    pname  VARCHAR(15),
    pnumber INT,
    plocation VARCHAR(15),
    dnum INT,
    budget FLOAT,
    totalbudget FLOAT )
AS
BEGIN
    DECLARE C CURSOR
        FOR
            SELECT Pname, Pnumber, Plocation, Dnum, SUM((Salary*1.0*Hours)/40) AS Budget
            FROM project JOIN works_on
            ON Pnumber=Pno
            JOIN employee
            ON Essn=Ssn
            WHERE Dnum=@departId
            GROUP BY Pnumber, Pname, Plocation, Dnum;

    DECLARE @pname AS VARCHAR(15), @pnumber AS INT, @plocation as VARCHAR(15), @dnum AS INT, @budget AS FLOAT, @totalbudget AS FLOAT;
    SET @totalbudget = 0;
    OPEN C;
    FETCH C INTO @pname, @pnumber, @plocation, @dnum, @budget;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @totalbudget += @budget;
        INSERT INTO @table VALUES (@pname, @pnumber, @plocation, @dnum, @budget, @totalbudget);
        FETCH C INTO @pname, @pnumber, @plocation, @dnum, @budget;
    END
    CLOSE C;
    DEALLOCATE C;
    RETURN;
END
```

Stored Procedures

O recurso que mais utilizamos foram stored procedures pois estas permitem armazenar e executar blocos de dados de código SQL de forma organizada e reutilizável. Neste projeto usamos principalmente para a adição, alteração e exclusão de dados. Quando mais que uma tabela era alterada na stored procedure usamos transactions também. Aqui mostramos 3 exemplos de uma stored procedure, outra que usa uma transação e outra que usa um cursor o restante irá no ficheiro .sql.

```sql
CREATE PROCEDURE [dbo].[AddPlayer]
    @PlayerName NVARCHAR(50),
    @PlayerAge INT,
    @PlayerPosition NVARCHAR(20),
    @PlayerHeight FLOAT,
    @PlayerWeight FLOAT,
    @TeamName NVARCHAR(50)
AS
BEGIN
    DECLARE @TeamId INT;
    DECLARE @NewPlayerId INT;

    -- Obter o TeamId baseado no TeamName
    SELECT @TeamId = id FROM Team WHERE teamName = @TeamName;

    IF @TeamId IS NULL
    BEGIN
        RAISERROR ('Team not found', 16, 1);
        RETURN;
    END

    -- Calcular o próximo ID para o novo jogador
    SELECT @NewPlayerId = ISNULL(MAX(id), 0) + 1 FROM Player;

    -- Inserir o novo jogador na tabela Player
    INSERT INTO Player (id, playerName, playerAge, playerPosition, playerHeight, playerWeight)
    VALUES (@NewPlayerId, @PlayerName, @PlayerAge, @PlayerPosition, @PlayerHeight, @PlayerWeight);

    -- Inserir a relação na tabela squad_player
    INSERT INTO squad_player (squadId, playerId)
    SELECT id, @NewPlayerId
    FROM Squad
    WHERE team_id = @TeamId;
END
```

```
CREATE PROCEDURE [dbo].[DeleteCoachById]
    @CoachId INT,
    @TeamName NVARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        IF NOT EXISTS (SELECT 1 FROM Coach WHERE id = @CoachId)
        BEGIN
            RAISERROR('Coach not found', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        DECLARE @SquadId INT;
        SELECT @SquadId = s.id
        FROM Squad s
        JOIN Team t ON s.team_id = t.id
        WHERE t.teamName = @TeamName;

        IF @SquadId IS NOT NULL
        BEGIN
            DELETE FROM Squad_Coach WHERE squadId = @SquadId AND coachId = @CoachId;
        END

        DELETE FROM Coach WHERE id = @CoachId;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

```
CREATE PROCEDURE CalculateTotalPointsPerPlayer
AS
BEGIN
    DECLARE @playerId INT,
            @prevPlayerId INT = NULL, -- Variável para armazenar o playerId anterior
            @gameId INT,
            @fgm INT,
            @fga INT,
            @threeptm INT,
            @threepta INT,
            @ftm INT,
            @fta INT,
            @offreb INT,
            @defreb INT,
            @assists INT,
            @blocks INT,
            @tov INT,
            @fouls INT,
            @points INT,
            @plus_minus INT,
            @totalPoints INT = 0;


    DECLARE cursor_name CURSOR FOR
    SELECT playerId, gameId, fgm, fga, threeptm, threepta, ftm, fta, offreb, defreb, assists, blocks, tov, fouls, points, plus_minus
    FROM Statistics_player
    ORDER BY playerId;

    OPEN cursor_name;

    FETCH NEXT FROM cursor_name INTO @playerId, @gameId, @fgm, @fga, @threeptm, @threepta, @ftm, @fta, @offreb, @defreb, @assists, @blocks, @tov, @fouls, @points, @plus_minus;
    WHILE @@FETCH_STATUS = 0
    BEGIN

        SET @points = (@fgm * 2) + (@threeptm * 3) + (@ftm);
```

```
        SET @totalPoints = @totalPoints + @points;


        IF @prevPlayerId IS NOT NULL AND @prevPlayerId!= @playerId
        BEGIN

            PRINT 'Player ID: ' + CAST(@prevPlayerId AS VARCHAR) + ', Total Points: ' + CAST(@totalPoints AS VARCHAR);

            SET @totalPoints = 0;
        END

        SET @prevPlayerId = @playerId;
        FETCH NEXT FROM cursor_name INTO @playerId, @gameId, @fgm, @fga, @threeptm, @threepta, @ftm, @fta, @offreb, @defreb, @assists, @blocks, @tov, @fouls, @points, @plus_minus;
    END

    PRINT 'Player ID: ' + CAST(@playerId AS VARCHAR) + ', Total Points: ' + CAST(@totalPoints AS VARCHAR);

    CLOSE cursor_name;
    DEALLOCATE cursor_name;
END
```

Triggers

Usamos triggers para sempre que alguma entidade seja adicionada à bd a informação relativa à quantidade de cada entidade é atualizada. Os triggers permitem que sempre que algum jogador (por exemplo) é adicionado à base de dados, é incrementado um valor na tabela bd_statistics na coluna totalPlayers. Os restantes triggers irão no ficheiro .sql.

```
CREATE TRIGGER [dbo].[trg_decrement_player]
ON [dbo].[Player]
AFTER DELETE
AS
BEGIN
    UPDATE bd_statistics
    SET num_players = num_players - 1
    WHERE id = 1;
END;
```

```
CREATE TRIGGER [dbo].[trg_increment_player]
ON [dbo].[Player]
AFTER INSERT
AS
BEGIN
    UPDATE bd_statistics
    SET num_players = num_players + 1
    WHERE id = 1;
END;
```

Interface

Dispomos de 3 interfaces. A interface View que seria usada por um utilizador comum para consulta de dados como as equipas, jogos, treinador, jogadores, estatísticas, filtragem por conferência ou divisão. A interface Team Manager seria destinada ao responsável de uma equipa em que este encontra a sua equipa pesquisando pelo nome e pode adicionar, editar e remover jogadores ou então remover, editar e adicionar um treinador. Por fim a última interface é destinada ao gestor da liga na qual este consegue adicionar jogos e adicionar informação relativa a esses jogos como os pontos das equipas, as estatísticas das equipas e as estatísticas dos jogadores que participaram naquele jogo. Acabamos por não implementar nenhum sistema de segurança para diferenciar o acesso a estas interfaces.

Desenvolvemos o frontend no visual studio em C# e para alterar a definição da conexão à base de dados basta alterar os valores presentes na linha 45.