DBF Implementation  
for Action Soccer Manager

# Procedures and processes involved

In the process of balancing teams, the goal is to make the resulting teams that play against each other have a more competitive game, rather than one team overpowering all the other teams. This helps the game be more suspenseful and enjoyable.

In this process, the match organizer will do one of the following to the players:

* Place in a new team
* Transfer to a different team
* Leave them, allowing the player to remain in their current team, or no team – if they were not assigned to one

In a team, there has to be a captain to act as a liaison between the match officials and team players during a game. Since a player can be transferred to a new team – including team captains – a new captain would need to be appointed on the team’s next game.

A team may be disbanded, but records of its performance should still be kept. This can result in a player not being a part of any team for a moment in time, like a ‘free agent’.

User details are not removed from the database upon ‘leaving’ this system– unless they are spectators or have made no contribution to any games – as their historical data is tied to all. Their login details will be removed upon deleting their account.

Only under special circumstances may a game’s result be nullified. Affected stats involving the players and teams need to be removed.

The following business rules further explain the procedure for helping the match officials make better decisions to improve in team balancing.

## Business rules

A USER must have LOGIN credentials. The LOGIN credentials include the user’s unique username (LOGIN\_UNAME) and is verified by their password (LOGIN\_PASS). All users are uniquely identified by their user ID (USER\_ID).

A USER is either a SPECTATOR or is further classified as either a PLAYER or MATCH\_OFFICIAL. This is indicated by their USER\_TYPE. A USER is further characterized by their first name (USER\_FNAME), last name (USER\_LNAME), their date of birth (USER\_DOB) and their age (USER\_AGE). Their age is continuously calculated using their date of birth to remain up to date. They may also upload a profile picture (USER\_PICTURE).  
A USER has only one set of LOGIN credentials, and a set of LOGIN credentials can be associated with only one user. The relationship between the USER and LOGIN entities is 1:1 (mandatory one-to-one).

A PLAYER is a specific type of USER who participates in the football matches. They are uniquely identified by their user ID (USER\_ID), and are further characterized by their dominant foot they use to play with (PLAYER\_FOOT), the total number of games they have played (PLAYER\_GAMES\_PCOUNT), the total number of games they have won, drawn and lost (PLAYER\_TWINS, PLAYER\_TDRAWS and PLAYER\_TLOSSES), their overall average rating (PLAYER\_AVG\_RATING), and the team they are currently playing for (TEAM\_ID). All these attributes (excluding their USER\_ID, PLAYER\_FOOT and TEAM\_ID) need to be updated after each match they play to remain current.  
A PLAYER can be the captain of no TEAM or at most one TEAM (the one they are currently playing for). A TEAM can have none or one PLAYER as its team captain (in a current moment in time). The relationship between the PLAYER and TEAM entities is 1:1 (optional one-to-one).  
Since a PLAYER is a specific type of USER, and a USER is a generalized PLAYER, the relationship between the USER and PLAYER entities is 1:1 (mandatory one-to-one).

A MATCH\_OFFICIAL is the only type of USER who may officiate the teams and GAMEs played between them. They are also uniquely identified by their user ID (USER\_ID). They can either be a referee, match organizer, or both. Only referees (MOF\_REFEREE) may record the results from a GAME they umpired in. Only match organizers (MOF\_ORGANIZER) may create a TEAM and pair them with another TEAM to play against each other in a GAME.  
Since a MATCH\_OFFICIAL is a specific type of USER, and a USER generalizes a MATCH\_OFFICIAL, the relationship between MATCH\_OFFICIAL and USER entities is 1:1 (mandatory one-to-one).  
A MATCH\_OFFICIAL can referee none or multiple GAMEs – if they are a referee . A GAME is umpired by only one MATCH\_OFFICIAL – that is a referee. The relationship between the MATCH\_OFFICIAL and GAME entities is 1:M (optional one-to-many).  
A MATCH\_OFFICIAL can create none, or multiple TEAMs­ – if they are an organizer. A TEAM is created by only one MATCH\_OFFICIAL – who is an organizer. The relationship between the MATCH\_OFFICIAL and TEAM entities is 1:N (optional one-to-many).

A TEAM is uniquely identified by its team ID (TEAM\_ID). It is described by its team name (TEAM\_NAME), the date it was founded (TEAM\_FOUNDED), the total number of games it has played (TEAM\_PCOUNT), the number of wins (TEAM\_WINS), losses (TEAM\_LOSSES) and draws (TEAM\_DRAWS) it has had. It is further described by its total goal difference (TEAM\_G\_DIFF), the total amount of goals for (TEAM\_G\_FOR) and against (TEAM\_G\_DIFF) the team. A TEAM can have a team captain, identified by their user ID (USER\_ID). A TEAM can also be detailed by its team logo (TEAM\_LOGO).  
Two TEAMs may play no GAME or multiple GAMES against each other. In turn, a GAME involves only two teams. The relationship between the TEAM and GAME entities is 2:P (optional many-to-many).   
A TEAM consists of none or up to 5 PLAYERs and a PLAYER plays for either none, or only one TEAM. The relationship between the TEAM and PLAYER entities is 1:5 (optional one-to-many).

A GAME is uniquely identified by a combination of the game’s ID (GAME\_ID) and the MATCH\_OFFICIAL umpiring the GAME (USER\_ID). It is initiated on a certain date (GAME\_DATE) at a certain time (GAME\_TIME). It consists of two teams competing against each other – identified by their TEAM\_IDs – and having a respective home/away score (GAME\_HOME\_SCORE and GAME\_AWAY\_SCORE).  
As described before, a GAME has to consist of only two teams, where two teams may play multiple games against each other. The relationship between GAME and TEAM entities is P:2 (optional many-to-many).  
Also described above, the relationship between the MATCH\_OFFICIAL and GAME entities is 1:M (optional one-to-many).

# Initial ERD

# Recognition of issues

## M:N Relationships

One problematic issue is that the TEAM and GAME entities (which are associated with each other) have a many-to-many relationship with each other. This will cause using and maintaining the database to be problematic. Also the TEAM and PLAYER entities association need to be defined better as there is more than one association between each other.

## NULL values

The following events will result in an attribute being NULL:

* A TEAM has no team captain, when it is disbanded
* A PLAYER is not part of a team, i.e. A player is a ‘free agent’

Since we know the reason for these values being NULL, it would be more appropriate if they were associated with a flag indicating these cases, or even better, redefine their associations.

## Multivalued attributes

We have ensured that there are no multivalued attributes, such as the date and time when a game is played, and whether a match official is a referee, organizer or both, by splitting these attributes into their two respective separate fields.

## Historical data

Since we would like to balance the teams better, we would need to keep track of a PLAYER’s movements in between TEAMs and their detailed performance in a match – such as the position they played in and their rating for that game.  
It would also be wise to keep track of which MATCH\_OFFICIAL created a TEAM, as we can keep track of whose judgment was best.

## Normalization

### 1NF

The database is not yet in the first normal form as there are possible attributes which can be NULL, as stated above.

# Solution and updated ERD

We realized that we also needed to include additional attributes for a PLAYER, as per the assignment requirements. A PLAYER is also characterized by the total goals for (PLAYER\_T\_G\_FOR) and total goals against (PLAYER\_T\_G\_AGAINST) the PLAYER. Upon doing this we realized it would not be unnecessary to also include the total number of goals (PLAYER\_T\_G\_SCORED) scored by a player and the number of times a player was elected to be captain (PLAYER\_CAPTAIN\_COUNT).

## M:N Relationships NULL values and Historical data

To address the many-to-many relationships, NULL valued attributes and historical data, two bridging entities need to be implemented. As a result of these entities, there will be minor changes to the attributes of existing entities in the initial ERD.  
The USER\_ID of the TEAM entity will indicate the MATCH\_OFFICIAL who created the TEAM – this helps keep track of which match organizer made the best decisions.  
The PLAYER entity will no longer have the TEAM\_ID of the team they are playing for, as a PLAYER may be frequently transferred across teams.

### MATCH entity

A MATCH entity will be identified by the GAME\_ID and TEAM\_ID. This will allow a single score to be captured (TEAM\_SCORE) for a TEAM playing in a specific GAME.

A TEAM can play none or multiple MATCHes. A MATCH describes only one TEAM. The relationship between the TEAM and MATCH entities is 1:P (optional one-to-many).

A GAME consists of two TEAMs MATCHed against each other, and only two MATCHed TEAMs can describe one GAME. The relationship between the GAME and MATCH entities is 1:2 (mandatory one-to-many).

### PLAYER\_ARCHIVE entity

A PLAYER\_ARCHIVE will consist details relating to the GAME a PLAYER has participated in. It is uniquely identified by a combination of the GAME\_ID, TEAM\_ID and USER\_ID (of the PLAYER). It consists of the position the PLAYER played in a GAME (PARC\_POSITION), the number of goals scored in that GAME by the PLAYER (PARC\_GOALS\_SCORED) and the player’s rating for that GAME (PARC\_RATING). It will also indicate whether that player was a captain of the TEAM in which they played for in that GAME or not (PARC\_CAPTAIN).

A PLAYER may have no PLAYER\_ARCHIVE, i.e. not played a single game, or have multiple PLAYER\_ARCHIVEs. A PLAYER\_ARCHIVE describes only one PLAYER’s performance in a GAME. The relationship between the PLAYER and PLAYER\_ARCHIVE is 1:S (optional one-to-many).

A GAME is further detailed by only ten PLAYER\_ARCHIVES – since a game consists of ten players. A PLAYER\_ARCHIVE describes the player’s performance in only one GAME. The relationship between the GAME and PLAYER\_ARCHIVE entities is 1:10 (mandatory one-to-many).

A TEAM may have no PLAYER\_ARCHIVE – as it may be newly constructed or no games have been organized with this team as yet – or multiple PLAYER\_ARCHIVEs. A PLAYER\_ARCHIVE describes the player’s performance for only one TEAM. The relationship between the TEAM and PLAYER\_ARCHIVE is 1:R (optional one-to-many).

## Normalization

I believe the database is of the 2nd Normal Form for the following reasons.

### 1NF

All Primary Keys identified are unique. There are no repeating groups of attributes within an entity.

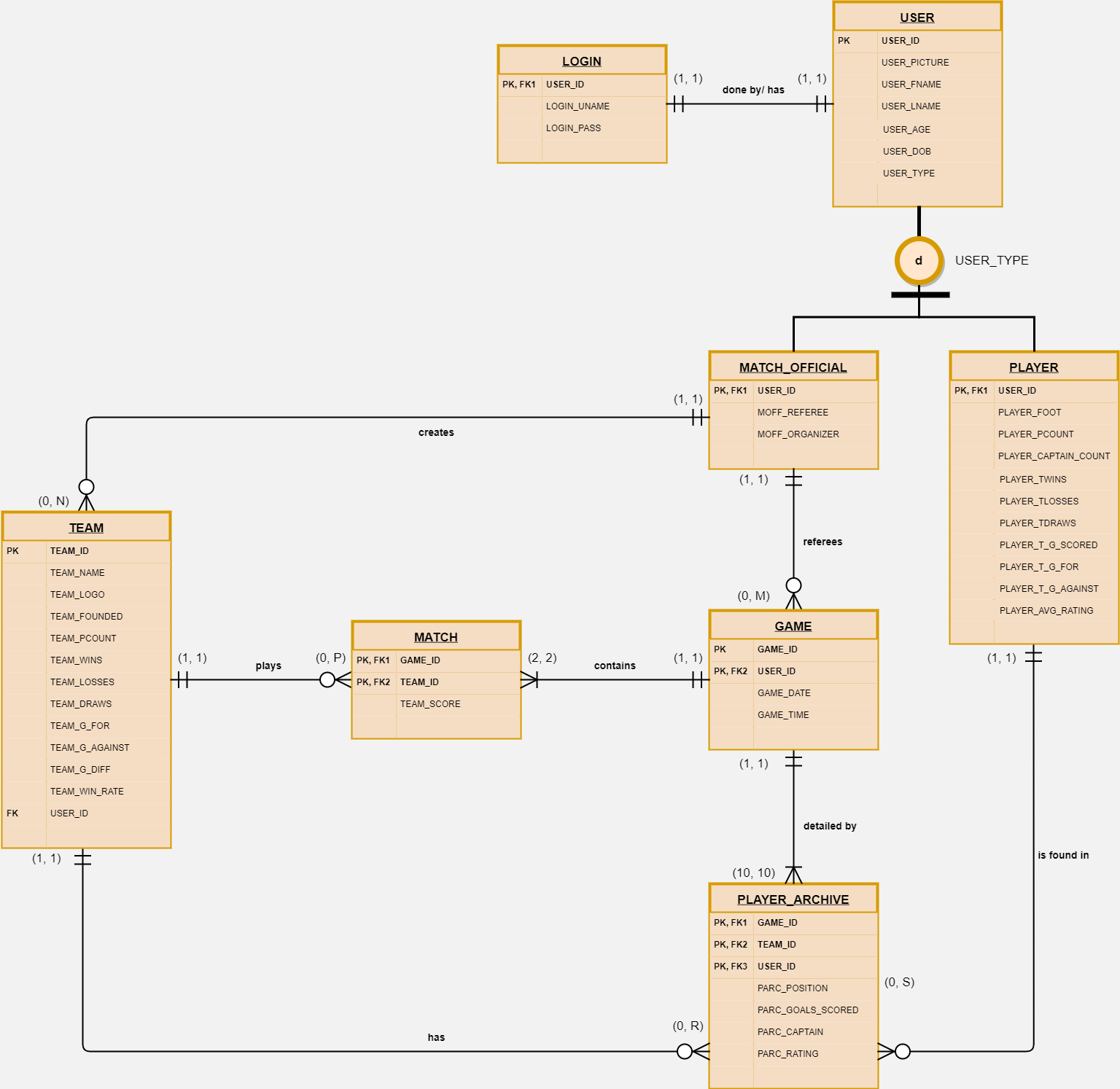
2NF

All non-key attributes are functionally dependent on their Primary Keys.

### 3NF

I do not feel that the database is Normalized to the 3rd Normal Form, as there are some attributes which are calculated form other existing attributes, making them dependent on the non-key attributes of the entity – such as TEAM\_PCOUNT, PLAYER\_PCOUNT, USER\_DOB. Aside from this, if there were no such calculated fields, this database would be easily identifiable to be in the 3rd Normal Form.

## Final ERD



# Implementation

## Views

## Tables

### Source of dbSouce.sql

tee dbSource.rtf;

USE d2326254;

SET @USER\_START = 10000;

SET @TEAM\_START = 90000;

SET @GAME\_START = 1;

DROP TABLE IF EXISTS `MATCH`;

DROP TABLE IF EXISTS `PLAYER\_ARCHIVE`;

DROP TABLE IF EXISTS `GAME`;

DROP TABLE IF EXISTS `TEAM`;

DROP TABLE IF EXISTS `PLAYER`;

DROP TABLE IF EXISTS `MATCH\_OFFICIAL`;

DROP TABLE IF EXISTS `LOGIN`;

DROP TABLE IF EXISTS `USER`;

-- `USER` table

CREATE TABLE `USER`(

USER\_ID INTEGER UNSIGNED AUTO\_INCREMENT,

USER\_PICTURE VARCHAR(100) DEFAULT "team.jpg" COMMENT "location of picture, within user\_pics directory",

USER\_FNAME VARCHAR(100) NOT NULL,

USER\_LNAME VARCHAR(100) NOT NULL,

USER\_AGE INTEGER UNSIGNED DEFAULT 0 COMMENT "calculated after inserting a new user and when displaying a SELECT query involving the USER",

USER\_DOB DATE NOT NULL,

USER\_TYPE VARCHAR(14) DEFAULT "spectator" NOT NULL

CHECK (USER\_TYPE = "spectator" || USER\_TYPE = "match\_official" || USER\_TYPE = "player") COMMENT "spectator, match\_official player",

PRIMARY KEY(USER\_ID),

INDEX (USER\_ID)

) AUTO\_INCREMENT = 10000;

-- `LOGIN` table

CREATE TABLE `LOGIN`(

USER\_ID INTEGER UNSIGNED NOT NULL,

LOGIN\_UNAME VARCHAR(100) NOT NULL COMMENT "username",

LOGIN\_PASS TEXT NOT NULL COMMENT "login password",

PRIMARY KEY (USER\_ID),

FOREIGN KEY (USER\_ID) REFERENCES `USER`(USER\_ID) ON DELETE CASCADE,

INDEX (USER\_ID)

);

-- `MATCH\_OFFICIAL` table

CREATE TABLE `MATCH\_OFFICIAL`(

USER\_ID INTEGER UNSIGNED NOT NULL,

MOFF\_REFEREE BOOLEAN DEFAULT true COMMENT "is official a referee?",

MOFF\_ORGANIZER BOOLEAN DEFAULT true COMMENT "is official a team creator/ match organizer?",

PRIMARY KEY (USER\_ID),

FOREIGN KEY (USER\_ID) REFERENCES `USER` (USER\_ID),

INDEX (USER\_ID)

);

-- `PLAYER` table

CREATE TABLE `PLAYER`(

USER\_ID INTEGER UNSIGNED NOT NULL,

PLAYER\_FOOT CHAR(1) DEFAULT "R" COMMENT "player's dominant foot",

PLAYER\_PCOUNT INTEGER UNSIGNED DEFAULT 0 COMMENT "total number of games player has played",

PLAYER\_CAPTAIN\_COUNT INTEGER UNSIGNED DEFAULT 0 COMMENT "number of games player captained",

PLAYER\_TWINS INTEGER UNSIGNED DEFAULT 0 COMMENT "total wins with player",

PLAYER\_TLOSSES INTEGER UNSIGNED DEFAULT 0 COMMENT "total losses with player",

PLAYER\_TDRAWS INTEGER UNSIGNED DEFAULT 0 COMMENT "total draws with player",

PLAYER\_T\_G\_SCORED INTEGER UNSIGNED DEFAULT 0 COMMENT "total goals scored by the player",

PLAYER\_T\_G\_FOR INTEGER UNSIGNED DEFAULT 0 COMMENT "total goals for player",

PLAYER\_T\_G\_AGAINST INTEGER UNSIGNED DEFAULT 0 COMMENT "total goals against player",

PLAYER\_AVG\_RATING DOUBLE(2,1) UNSIGNED DEFAULT 0

CHECK (PLAYER\_AVG\_RATING >= 0 && PLAYER\_AVG\_RATING <= 5) COMMENT "total player average",

PRIMARY KEY (USER\_ID),

FOREIGN KEY (USER\_ID) REFERENCES `USER` (USER\_ID),

INDEX (USER\_ID)

);

-- `TEAM` table

CREATE TABLE `TEAM`(

TEAM\_ID INTEGER UNSIGNED AUTO\_INCREMENT,

TEAM\_NAME VARCHAR(250) NOT NULL,

TEAM\_LOGO VARCHAR(10) DEFAULT "team.jpg" COMMENT "location of team logo, within team\_pics directory",

TEAM\_FOUNDED DATE NOT NULL COMMENT "date team was founded",

TEAM\_PCOUNT INTEGER UNSIGNED DEFAULT 0 COMMENT "total games team has played",

TEAM\_WINS INTEGER UNSIGNED DEFAULT 0,

TEAM\_LOSSES INTEGER UNSIGNED DEFAULT 0,

TEAM\_DRAWS INTEGER UNSIGNED DEFAULT 0,

TEAM\_G\_FOR INTEGER UNSIGNED DEFAULT 0,

TEAM\_G\_AGAINST INTEGER UNSIGNED DEFAULT 0,

TEAM\_G\_DIFF INTEGER DEFAULT 0,

TEAM\_WIN\_RATE DOUBLE(4,2) DEFAULT 0,

USER\_ID INTEGER UNSIGNED NOT NULL COMMENT "organizer",

PRIMARY KEY (TEAM\_ID),

INDEX (TEAM\_ID),

FOREIGN KEY (USER\_ID) REFERENCES `USER` (USER\_ID),

INDEX (USER\_ID)

) AUTO\_INCREMENT = 90000;

-- `GAME` table

CREATE TABLE `GAME`(

GAME\_ID INTEGER UNSIGNED AUTO\_INCREMENT,

USER\_ID INTEGER UNSIGNED NOT NULL COMMENT "referee",

GAME\_DATE DATE NOT NULL,

GAME\_TIME TIME NOT NULL,

PRIMARY KEY (GAME\_ID, USER\_ID),

INDEX (GAME\_ID),

FOREIGN KEY(USER\_ID) REFERENCES `MATCH\_OFFICIAL` (USER\_ID),

INDEX (USER\_ID)

) AUTO\_INCREMENT = 1;

-- `MATCH` table

-- When a `GAME` is nullified, relating `MATCH` records need to be removed through a CASADE

CREATE TABLE `MATCH`(

GAME\_ID INTEGER UNSIGNED NOT NULL,

TEAM\_ID INTEGER UNSIGNED NOT NULL,

TEAM\_SCORE INTEGER NOT NULL COMMENT "Can't be unsigned because of the calculations done with it",

PRIMARY KEY (GAME\_ID, TEAM\_ID),

FOREIGN KEY(GAME\_ID) REFERENCES `GAME` (GAME\_ID) ON DELETE CASCADE,

INDEX (GAME\_ID),

FOREIGN KEY(TEAM\_ID) REFERENCES `TEAM` (TEAM\_ID),

INDEX (TEAM\_ID)

);

-- `PLAYER\_ARCHIVE` table

-- When a `GAME` is nullified, relating `PLAYER\_ARCHIVE` records need to be removed through a CASADE

CREATE TABLE `PLAYER\_ARCHIVE`(

GAME\_ID INTEGER UNSIGNED NOT NULL,

TEAM\_ID INTEGER UNSIGNED NOT NULL,

USER\_ID INTEGER UNSIGNED NOT NULL COMMENT "player",

PARC\_POSITION VARCHAR(5) NOT NULL COMMENT "def att mid, and their variations",

PARC\_GOALS\_SCORED INTEGER UNSIGNED DEFAULT 0 NOT NULL,

PARC\_CAPTAIN BOOLEAN DEFAULT false,

PARC\_RATING DOUBLE(2,1) UNSIGNED DEFAULT 0

CHECK (PARC\_RATING >= 0 && PARC\_RATING <= 5) NOT NULL,

PRIMARY KEY (GAME\_ID, TEAM\_ID, USER\_ID),

FOREIGN KEY(GAME\_ID) REFERENCES `GAME` (GAME\_ID) ON DELETE CASCADE,

INDEX (GAME\_ID),

FOREIGN KEY(TEAM\_ID) REFERENCES `TEAM` (TEAM\_ID),

INDEX (TEAM\_ID),

FOREIGN KEY(USER\_ID) REFERENCES `PLAYER` (USER\_ID),

INDEX (USER\_ID)

);

DESCRIBE `USER`;

DESCRIBE `MATCH\_OFFICIAL`;

DESCRIBE `PLAYER`;

DESCRIBE `PLAYER\_ARCHIVE`;

DESCRIBE `TEAM`;

DESCRIBE `MATCH`;

DESCRIBE `GAME`;

notee;

## Functions/Procedures

### Source of dbFunctions.sql

tee dbFunctions.rtf;

DROP PROCEDURE IF EXISTS UPDATE\_USER\_AGE;

DROP PROCEDURE IF EXISTS CREATE\_GAME\_VIEWS;

DROP PROCEDURE IF EXISTS CREATE\_PLAYER\_VIEWS;

DROP PROCEDURE IF EXISTS UPDATE\_TEAM\_STATS;

DROP PROCEDURE IF EXISTS UPDATE\_PLAYER\_STATS;

-- Start of procedure/function declaration

DELIMITER //

-- Update the users age

-- Done when starting up database after being inactive for a long time (days...)

CREATE PROCEDURE UPDATE\_USER\_AGE()

BEGIN

UPDATE `USER`

SET USER\_AGE = 0;

END//

-- (Re)Create views which assist in updating player and team stats

-- when the `MATCH` or `PLAYER\_ARCHIVE` table is modified

-- These views are really helpful...

CREATE PROCEDURE CREATE\_GAME\_VIEWS()

BEGIN

DROP VIEW IF EXISTS `TEAM\_WINNER\_LOSER`;

DROP VIEW IF EXISTS `TEAM\_DRAWN\_GAMES`;

CREATE VIEW `TEAM\_WINNER\_LOSER`

AS

SELECT DISTINCT W.GAME\_ID AS GAME\_ID, W.TEAM\_ID AS WINNER, W.TEAM\_SCORE AS WINNER\_SCORE, L.TEAM\_ID AS LOSER, L.TEAM\_SCORE AS LOSER\_SCORE

FROM `MATCH` AS W INNER JOIN `MATCH` AS L

ON (W.GAME\_ID = L.GAME\_ID && W.TEAM\_ID != L.TEAM\_ID && W.TEAM\_SCORE > L.TEAM\_SCORE);

CREATE VIEW `TEAM\_DRAWN\_GAMES`

AS

SELECT T1.GAME\_ID AS GAME\_ID, T1.TEAM\_ID AS TEAM\_ID, T1.TEAM\_SCORE AS TEAM\_SCORE

FROM `MATCH` AS T1 INNER JOIN `MATCH` AS T2

ON (T1.GAME\_ID = T2.GAME\_ID && T1.TEAM\_ID != T2.TEAM\_ID && T1.TEAM\_SCORE = T2.TEAM\_SCORE);

END//

-- List of games and the teams the player played for, which a player has participated in

CREATE PROCEDURE CREATE\_PLAYER\_VIEWS(IN playerID INTEGER UNSIGNED)

BEGIN

CALL CREATE\_GAME\_VIEWS();

DROP VIEW IF EXISTS `PLAYER\_VIEW`;

DROP VIEW IF EXISTS `PLAYER\_WINNER\_LOSER`;

DROP VIEW IF EXISTS `PLAYER\_DRAWN\_GAMES`;

SET @query = CONCAT("CREATE VIEW `PLAYER\_VIEW` AS ",

"SELECT GAME\_ID, TEAM\_ID, PARC\_GOALS\_SCORED AS GOALS\_SCORED, PARC\_CAPTAIN AS WAS\_CAPTAIN, PARC\_RATING AS PLAYER\_RATING ",

"FROM `PLAYER\_ARCHIVE` ",

"WHERE USER\_ID = ", playerID);

PREPARE createView FROM @query;

EXECUTE createView;

DEALLOCATE PREPARE createView;

CREATE VIEW `PLAYER\_WINNER\_LOSER`

AS

SELECT WL.GAME\_ID, IF(WINNER = PV.TEAM\_ID, "W", "L") AS WIN\_LOSS,

IF(WINNER = PV.TEAM\_ID, WINNER\_SCORE, LOSER\_SCORE ) AS GOALS\_FOR,

IF(WINNER != PV.TEAM\_ID, WINNER\_SCORE, LOSER\_SCORE) AS GOALS\_AGAINST,

GOALS\_SCORED,

PLAYER\_RATING

FROM `TEAM\_WINNER\_LOSER` AS WL INNER JOIN PLAYER\_VIEW AS PV

ON WL.GAME\_ID = PV.GAME\_ID && (WL.WINNER = PV.TEAM\_ID || WL.LOSER = PV.TEAM\_ID);

CREATE VIEW `PLAYER\_DRAWN\_GAMES`

AS

SELECT DG.GAME\_ID, DG.TEAM\_ID, TEAM\_SCORE

FROM `TEAM\_DRAWN\_GAMES` AS DG INNER JOIN `PLAYER\_VIEW` AS PV

ON DG.GAME\_ID = PV.GAME\_ID && DG.TEAM\_ID = PV.TEAM\_ID;

END//

-- Process parameterized query

-- Mandatory to execute prepared statements for parameterized queries

-- A procedure will store the results in a user defined variable

CREATE PROCEDURE PREPARED\_QUERY(IN selectColumns TEXT, IN tableName TEXT, IN whereClause TEXT)

BEGIN

SET @query = CONCAT("SELECT ", selectColumns, " INTO @outputResult FROM ", tableName, " WHERE ", whereClause);

PREPARE selectQuery FROM @query;

EXECUTE selectQuery;

DEALLOCATE PREPARE selectQuery;

END//

-- Update the team stats

-- Done when the match table is affected

-- The main reason for using this procedure on triggers on the `MATCH` table instead of

-- manually passing values relating to a match the team was involved in

-- is because the latter becomes more complex as one would need to determine

-- whether we are deleting or inserting/updating, whether it was a

-- positive/negative influence that needs to be reversed (such as a negative goal difference being reversed)...

CREATE PROCEDURE UPDATE\_TEAM\_STATS(IN teamID INTEGER UNSIGNED)

BEGIN

DECLARE playCount, wins, losses, draws, goalsFor, goalsAgainst INTEGER UNSIGNED DEFAULT 0;

DECLARE goalDifference INTEGER DEFAULT 0;

DECLARE winRate DECIMAL(4,2) DEFAULT 0;

set @teamID = teamID;

CALL CREATE\_GAME\_VIEWS();

CALL PREPARED\_QUERY("COUNT(\*)", "`TEAM\_WINNER\_LOSER`", "WINNER = @teamID");

SET wins = @outputResult;

CALL PREPARED\_QUERY("COUNT(\*)", "`TEAM\_WINNER\_LOSER`", "LOSER = @teamID");

SET losses = @outputResult;

CALL PREPARED\_QUERY("COUNT(\*)", "`TEAM\_DRAWN\_GAMES`", "TEAM\_ID = @teamID");

SET draws = @outputResult;

SET playCount = wins + losses + draws;

SET winRate = wins/playCount \* 100;

CALL PREPARED\_QUERY("SUM(WINNER\_SCORE) ", "`TEAM\_WINNER\_LOSER`", "WINNER = @teamID");

SET goalsFor = @outputResult;

CALL PREPARED\_QUERY("SUM(LOSER\_SCORE)", "`TEAM\_WINNER\_LOSER`", "LOSER = @teamID");

SET goalsFor = goalsFor + @outputResult;

CALL PREPARED\_QUERY("SUM(LOSER\_SCORE) ", "`TEAM\_WINNER\_LOSER`", "WINNER = @teamID");

SET goalsAgainst = @outputResult;

CALL PREPARED\_QUERY("SUM(WINNER\_SCORE) ", "`TEAM\_WINNER\_LOSER`", "LOSER = @teamID");

SET goalsAgainst = goalsAgainst + @outputResult;

-- Since games here have been drawn, the score is the same for both teams

CALL PREPARED\_QUERY("SUM(TEAM\_SCORE)", "`TEAM\_DRAWN\_GAMES`", "TEAM\_ID = @teamID");

SET goalsFor = goalsFor + @outputResult;

SET goalsAgainst = goalsAgainst + @outputResult;

CALL PREPARED\_QUERY("SUM(WINNER\_SCORE - LOSER\_SCORE) ", "`TEAM\_WINNER\_LOSER`", "WINNER = @teamID");

SET goalDifference = @outputResult;

CALL PREPARED\_QUERY("SUM(LOSER\_SCORE - WINNER\_SCORE) ", "`TEAM\_WINNER\_LOSER`", "LOSER = @teamID");

SET goalDifference = goalDifference + @outputResult;

UPDATE `TEAM`

SET

TEAM\_PCOUNT = playCount,

TEAM\_WINS = wins,

TEAM\_LOSSES = losses,

TEAM\_DRAWS = draws,

TEAM\_G\_FOR = goalsFor,

TEAM\_G\_AGAINST = goalsAgainst,

TEAM\_G\_DIFF = goalDifference,

TEAM\_WIN\_RATE = winRate

WHERE TEAM\_ID = teamID;

END//

-- Update the player stats

-- Similar to updating team stats, but is much more specific

CREATE PROCEDURE UPDATE\_PLAYER\_STATS(IN playerID INTEGER UNSIGNED)

BEGIN

DECLARE playCount, wins, losses, draws, goalsScored, goalsFor, goalsAgainst, captainCount INTEGER UNSIGNED DEFAULT 0;

DECLARE averageRating DOUBLE DEFAULT 0;

SET @playerID = playerID;

CALL CREATE\_PLAYER\_VIEWS(playerID);

CALL PREPARED\_QUERY("SUM(GOALS\_SCORED)", "`PLAYER\_VIEW`", "true");

SET goalsScored = @outputResult;

CALL PREPARED\_QUERY("COUNT(\*)", "`PLAYER\_VIEW`", "WAS\_CAPTAIN = true");

SET captainCount = @outputResult;

CALL PREPARED\_QUERY("SUM(PLAYER\_RATING)", "`PLAYER\_VIEW`", "true");

SET averageRating = @outputResult;

CALL PREPARED\_QUERY("COUNT(\*)", "`PLAYER\_VIEW`", "true");

SET playCount = @outputResult;

SET averageRating = averageRating / playCount;

CALL PREPARED\_QUERY("SUM(GOALS\_FOR)", "`PLAYER\_WINNER\_LOSER`", "true");

SET goalsFor = @outputResult;

CALL PREPARED\_QUERY("SUM(GOALS\_AGAINST)", "`PLAYER\_WINNER\_LOSER`", "true");

SET goalsAgainst = @outputResult;

CALL PREPARED\_QUERY("SUM(GOALS\_AGAINST)", "`PLAYER\_WINNER\_LOSER`", "true");

-- Since games here have been drawn, the score is the same for both teams

CALL PREPARED\_QUERY("SUM(TEAM\_SCORE)", "`PLAYER\_DRAWN\_GAMES`", "true");

SET goalsFor = goalsFor + @outputResult;

SET goalsAgainst = goalsAgainst + @outputResult;

CALL PREPARED\_QUERY("COUNT(\*)", "`PLAYER\_WINNER\_LOSER`", "WIN\_LOSS = 'W'");

SET wins = @outputResult;

CALL PREPARED\_QUERY("COUNT(\*)", "`PLAYER\_WINNER\_LOSER`", "WIN\_LOSS = 'W'");

SET losses = @outputResult;

CALL PREPARED\_QUERY("COUNT(\*)", "`PLAYER\_DRAWN\_GAMES`", "true");

SET draws = @outputResult;

UPDATE `PLAYER`

SET

PLAYER\_PCOUNT = playCount,

PLAYER\_CAPTAIN\_COUNT = captainCount,

PLAYER\_TWINS = wins,

PLAYER\_TLOSSES = losses,

PLAYER\_TDRAWS = draws,

PLAYER\_T\_G\_SCORED = goalsScored,

PLAYER\_T\_G\_FOR = goalsFor,

PLAYER\_T\_G\_AGAINST = goalsAgainst,

PLAYER\_AVG\_RATING = averageRating

WHERE USER\_ID = playerID;

END//

DELIMITER ;

notee;

## Triggers

### Source of dbTriggers.sql

tee dbTriggers.rtf;

DELIMITER //

--

-- `USER` triggers

--

-- After inserting and updating a `USER`, their age needs to be kept up to date

-- Unfortuantely there is no GOTO statement to point to the repeated procedures for these triggers

-- At the start of the app, a check for updating ages will also be done since it can be a while since the last time the DB was -- active

CREATE TRIGGER on\_insert\_user

BEFORE INSERT

ON `USER`

FOR EACH ROW

BEGIN

DECLARE age INTEGER;

SET age = TIMESTAMPDIFF(YEAR, NEW.USER\_DOB, CURRENT\_DATE());

IF (age != NEW.USER\_AGE) THEN

SET NEW.USER\_AGE = age;

END IF;

END//

CREATE TRIGGER on\_update\_user

BEFORE UPDATE

ON `USER`

FOR EACH ROW

BEGIN

DECLARE age INTEGER;

SET age = TIMESTAMPDIFF(YEAR, NEW.USER\_DOB, CURRENT\_DATE());

IF (age != NEW.USER\_AGE) THEN

SET NEW.USER\_AGE = age;

END IF;

END//

DELIMITER ;

notee;

### Source of dbPopulate.sql (To be continued)

tee dbPopulate.rtf;

USE d2326254;

SET @USER\_START = 10000;

SET @TEAM\_START = 90000;

SET @GAME\_START = 1;

INSERT INTO `USER` (USER\_PICTURE, USER\_FNAME, USER\_LNAME, USER\_DOB, USER\_TYPE)

VALUES

("moose918.jpg", "Musa", "Gumpu", "2002-02-03", "player"),

("mikey-mikey.jpg", "Michael", "le Forestier", "2001-06-04", "match\_official"),

("KTG.jpg", "Katlego", "Kungoane", "2001-09-03", "spectator");

INSERT INTO `LOGIN` (USER\_ID, LOGIN\_UNAME, LOGIN\_PASS)

VALUES

(10000, "moose918", "moose"),

(10001, "mikey-mikey", "forest"),

(10002, "KatTheGee", "KTG");

INSERT INTO `MATCH\_OFFICIAL` (USER\_ID, MOFF\_REFEREE, MOFF\_ORGANIZER)

VALUES

(10001, true, true);

INSERT INTO `PLAYER` (USER\_ID, PLAYER\_FOOT)

VALUES

(10000, "L");

INSERT INTO `TEAM` (TEAM\_NAME, TEAM\_LOGO, TEAM\_FOUNDED, USER\_ID)

VALUES

("Barefoot", "team.jpg", "2000-01-05", 10001 ),

("Ashigaru", "team.jpg", "2005-01-07", 10001),

("Storror", "team.jpg", "2010-10-10", 10001);

-- GAME SITUATION

INSERT INTO `GAME` (USER\_ID, GAME\_DATE, GAME\_TIME)

VALUES

(10001, "2020-06-05", "08:00"),

(10001, "2021-03-10", "10:00"),

(10001, "2021-05-09", "14:00"),

(10001, "1999-10-09", "05:00"),

(10001, "2019-08-04", "12:00"),

(10001, "2012-08-15", "19:00");

INSERT INTO `MATCH` (GAME\_ID, TEAM\_ID, TEAM\_SCORE)

VALUES

(1, 90000, 3),

(1, 90001, 1),

(2, 90001, 2),

(2, 90000, 0),

(3, 90002, 5),

(3, 90000, 1),

(4, 90002, 7),

(4, 90001, 3),

(5, 90000, 0),

(5, 90001, 0),

(6, 90001, 1),

(6, 90000, 1);

INSERT INTO `PLAYER\_ARCHIVE` (GAME\_ID, TEAM\_ID, USER\_ID, PARC\_POSITION, PARC\_GOALS\_SCORED, PARC\_CAPTAIN, PARC\_RATING)

VALUES

(1, 90000, 10000, "MID", 2, false, 4.3),

(2, 90000, 10000, "ATT", 0, true, 3.0),

(6, 90000, 10000, "DEF", 1, true, 4.9);

TABLE `USER`;

TABLE `MATCH\_OFFICIAL`;

TABLE `PLAYER`;

TABLE `PLAYER\_ARCHIVE`;

TABLE `TEAM`;

TABLE `MATCH`;

notee;

MATCH and PLAYER\_ARCHIVES have CASCADE DELETE when a GAME is nullified/deleted.

LOGIN has CASCADE DELETE when a USER is removed

UPDATE USER\_AGEs, done upon connection being made

CREATE GAME and PLAYER VIEWS

UPDATE TEAM stats

UPDATE PLAYER stats

WINNERS\_LOSERS VIEW, DRAWN\_GAMES VIEW can be used for both the PLAYER and TEAM entities

BEFORE INSERTING a USER

Update USER age

DELETING A USER – that is a SPECTATOR

Delete associated LOGIN

DELETING A non-contributing USER – MATCH\_OFFICIAL or PLAYER

Delete Child

AFTER INSERTING/DELETING/UPDATING PLAYER\_ARCHIVE

Update PLAYER stats

AFTER INSERTING/DELETING/UPDATING MATCH

Update TEAM stats

Can’t use as “Explicit or implicit commit is not allowed in stored function or trigger. “ and views are required to process details