FOOTBALL LEAGUE

Database Implementation

Marta Moreira
Martamoreira@Rockborne.com

Table of Contents

Table of figures

Introduction

This report showcases the steps of the development and implementation of a Database using PostgreSQL regarding a Football League. This project started with the design of the structure of the database system (Task 1) and was followed by the construction and implementation of the designed database according to the requirements delineated in the project brief (Task 2).

Task 2 encompasses the updates of the database design according to the feedback received and the implementation of this using PostgreSQL. The implementation started with the definition of tables with the corresponding relationships and domains, followed by the population of the tables with the data given in Task 1. After creating the tables and inserting the data, tests were conducted to validate the implementation of the database, which led to further updates on the database structure in order to optimise its design.

Methodology

The Entity - Relationship Model (ER Model) was designed base on the requirements for the database described in the project brief as well as a logical and applicable to the real world conditions.

The methodology describes the design made taking into account the appropriate data types, constraints and relationships between tables. This will be followed by creating and populating the database using PostgreSQL to simulate real/world scenarios and validate the functionality of the database.

Finally, some operations were made to validate the logic of the database design as well as the performance of the creation and population of the tables.

ER Diagram

Figure 1 represent the first draft of the ER diagram representing the database design.

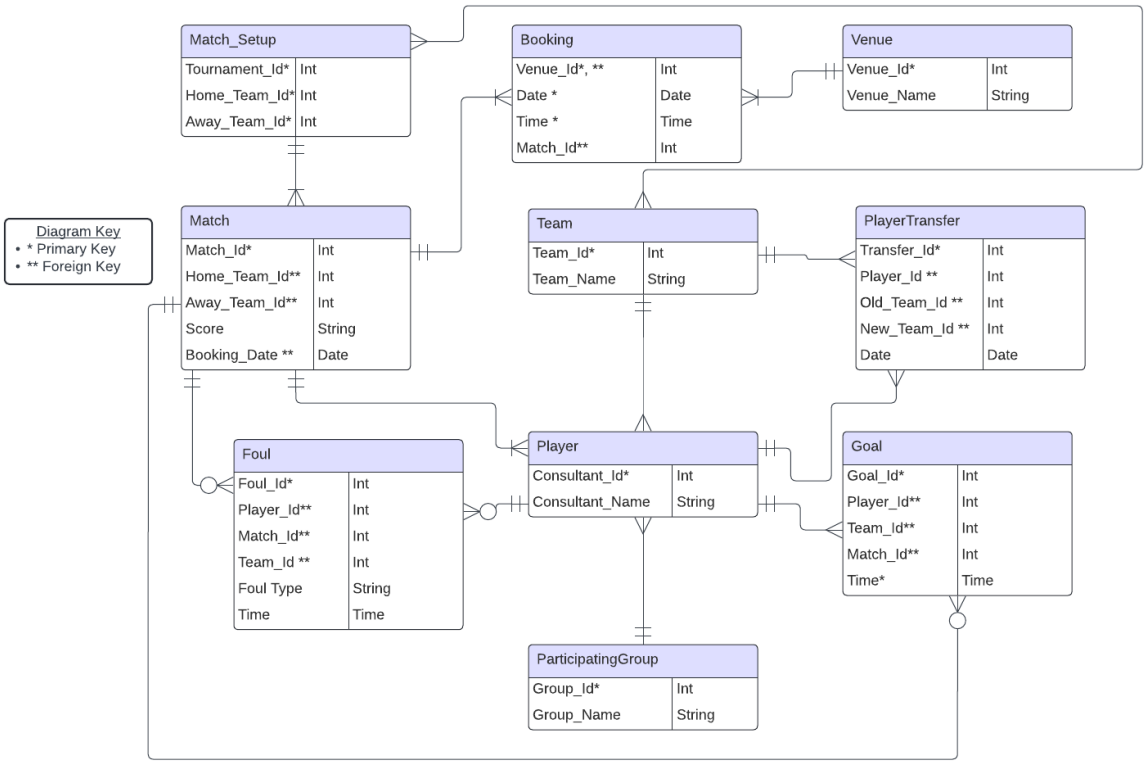

Figure 1 - First Draft of the ER Diagram

Figure 2 represent the updated version of the ER diagram considering the feedback received and data testing and validation through PostgreSQL.

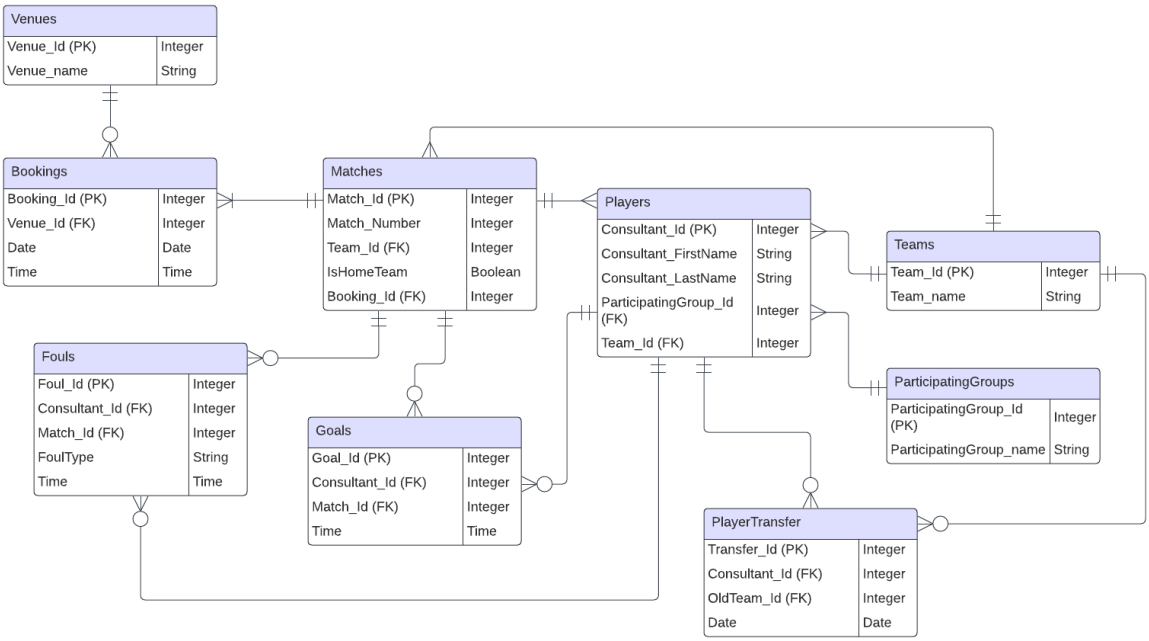

Figure 2 - Final draft of the ER Model

After designing the first draft for the database and receiving feedback on appropriate changes to make, the design was updated and the labels changed to the most common format.

The main updates to the ER model were:

- Matches table
    - Added a Boolean variable to know if it's the home team or away team ("IsHomeTeam")
    - removed the score because it can be calculated through the goals table
    - the booking date was changed to the Booking_id which is connected to the date and venues.
- Bookings table
    - "match_Id" was removed as it would be redundant because the "booking_Id" is already associated with a match_Id.
- Goals and fouls tables
    - "team_ID" was removed as it is already associated with a singular "match_ID" which is associated with a single" Team_ID".
- Transfers table
    - "new_team_Id" was removed from the "Transfers" table because this information is updated in the "Player" table and since the date in already there, it is possible to backtrack which team the player was before.
- Players table

- Information was added on the current team and the participating group each player is in as well as separating first and last name to help with data validation and to more easily sort and research players.

Data Dictionary

A data dictionary represents a guideline for all the information regarding a certain database or computer system. In other words, its composed of descriptive information of each piece of data such as the tables, data types and constraints, primary and foreign keys, relationships between tables and validation rules to ensure accuracy.[1]

According to the Football Database designed:

Team

Description: Contains information about different teams (unique identifiable ID and corresponding name).
Fields:

- Team_Id (Primary Key): Unique identifier for each team. Integer.
- Team_Name: Name of the team. Varchar (50).

ParticipatingGroup

Description: Stores details of participating groups or cohorts.
Fields:

- ParticipatingGroup_Id (Primary Key): Unique identifier for each participating group. Integer.
- Cohort_Name: Name of the cohort. Varchar (50).

Player

Description: Represents individual players associated with teams and participating groups.
Fields:

- Consultant_ID (Primary Key): Unique identifier for each player. Integer.
- Last_Name: Last name of the player. Varchar (50).
- First_Name: First name of the player. Varchar (50).
- Team_ID (Foreign Key): References the Team table to indicate the player's team affiliation. Integer
- Group_ID (Foreign Key): References the ParticipatingGroup table to indicate the player's group affiliation. Integer

Relationships:

- Many-to-One relationship with Team table (each player belongs to one team).
- Many-to-One relationship with ParticipatingGroup table (each player belongs to one group).

By separating the First and Last name, we can more easily validate the data, sort and research players.

Venues

Description: Contains information about venues where matches are held.
Fields:

- Venues_Id (Primary Key): Unique identifier for each venue. Integer.
- Venues_Name: Name of the venue. Varchar (50).

Bookings

Description: Stores booking details for matches at different venues.
Fields:

[1] Data Dictionary in DBMS (tutorialspoint.com)

- Booking_Id (Primary Key): Unique identifier for each booking. Integer.
- Venue_ID (Foreign Key): References the Venues table to indicate the booked venue. Integer.
- Date: Date of the booking. Date.
- Time: Time of the booking. Time.

Relationships:

- Many-to-One relationship with Venues table (each booking corresponds to one venue).

Introducing date and time as a variable ensures that the same venue does not have multiple matches booked simultaneously.

Matches

Description: Represents matches between teams.
Fields:

- Match_Id (Primary Key): Unique identifier for each match. Integer.
- Match_Number: Number assigned to each match. Integer.
- Team_ID (Foreign Key): References the Team table to indicate the participating team. Integer.
- isHomeTeam: Boolean indicating whether the team is the home team. Boolean.
- Booking_ID (Foreign Key): References the Bookings table to indicate the booked venue and time for the match. Integer.

Relationships:

- Many-to-One relationship with Team table (each match involves one team).
- Many-to-One relationship with Bookings table (each match corresponds to one booking).

Each match involves 2 teams and, to ensure a relation of 1 to many between the list of teams and specific match, a boolean variable was introduced to validate if the team in the match is playing at home or away. In other words, the match ID is unique for every match_Number, Team_Id combination. Every Match_Number is in 2 Match_Id's, for the home team and the away team. Additionally, advantages of using a boolean variable include the reduced memory storage, simplicity and readability.
It's important to also take into account that every home_team-Away_team match is unique in a single tournament.

Goals
Description: The list of goals scored in the whole tournament.
Fields:

- Goal_ID: (Primary Key): A unique identifier for each goal. Integer.
- Player_ID (Foreign Key): Identifies the player who scored the goal. Integer.
- Match_ID (Foreign Key: Indicates the match in which the goal was scored. Integer.
- Time: Time at which the goal was scored. Time.

Relationships:

- Many-to-One relationship with player and match (each goal is associated with one player and one match).
- One-to-Many relationship with matches (one match can have many goals).

Fouls
Description: The list of fouls made in the whole tournament.
Fields:

- Foul ID: (Primary Key): A unique identifier for each foul. Integer.
- Player ID: (Foreign Key) Identifies the player who committed the foul. Integer.
- Foul Type: : Describes the type of foul committed. Varchar (50).
- Match ID(Foreign Key) Indicates the match in which the foul occurred. Integer.

- Time: Time at which the foul occurred, time. Time.

Relationships:

- Many-to-One relationship with player and match (each foul is associated with one player and one match).
- One-to-Many relationship with matches (one match can have many fouls).

For the goals and fouls data, time was introduced to ensure uniqueness. For example, the same player can score multiple goals in the same match, time will be the only attribute to differentiate these. The same logic applied for the fouls.

Transfers:

Description: The list of transfers of players between teams.

Fields:

- Transfer_Id(Primary Key): Each transfer gets a unique identifier for tracking purposes. Integer.
- Player_Id: (Foreign Key) Identifies the player involved in the transfer. Integer.
- New_Team_Id(Foreign Key): Indicates the team to which the player has been transferred. Integer.
- Date: Records the date when the transfer took place. Date.

Relationships:

- Each transfer is associated with one player and one new team.
- Players may have multiple transfers to different teams.

The "transfers" table logs information about player transfers between teams.

Taking into account that it was necessary to differentiate between matches for the tournament and friendly matches, an update on the current model would be the addition of a “FriendlyMatches” table with the following characteristics.

FriendlyMatches:

Description: The list all friendly matches.

Fields:

- FriendlyMacth_Id(Primary Key): Each Friendly match gets a unique identifier for tracking purposes. Integer.
- Match_Id: (Foreign Key) Identifies the match that was friendly. Integer.

Relationships:

- One-to-Many relationship with matches (one match can have many friendly matches).

Data Definition Language

DDL (Data Definition Language) and DML (Data Manipulation Language) are two categories of SQL (Structured Query Language) commands used to interact with databases.

DDL is used to define, modify, and delete database objects such as tables, indexes, and constraints. In this case the following code was used to create the tables.[2]

To define and create a table in SQL the code structure used is as follows:

```
CREATE TABLE table_name (
    attribute1_name data_type [constraint],
    attribute2_name data_type [constraint],
    ...
    [table_constraint]
)
```

Figure 3 – Code structure used to define and create a new table.

The implementation of the previous code structure can be shown in figures 4 to 13.

```
CREATE TABLE team (
    Team_Id int PRIMARY KEY,
    Team_Name varchar(50)
);
```

Figure 4 - Create "Team" table.

```
CREATE TABLE participatingGroup (
    ParticipatingGroup_Id int PRIMARY KEY,
    Cohort_Name varchar(50)
);
```

Figure 5 - Create "participatingGroup" table.

[2] PostgreSQL: Documentation: 16: Chapter 5. Data Definition

```
CREATE TABLE player(
    Consultant_ID int PRIMARY KEY,
    Last_Name varchar(50),
    First_Name varchar(50),
    Team_ID int,
    Group_ID int,
    FOREIGN KEY (Team_ID) REFERENCES team (Team_ID),
    FOREIGN KEY (Group_ID) REFERENCES participatingGroup (participatingGroup_ID)
);
```

Figure 6 - Create "Player" table.

```
CREATE TABLE venues (
    Venues_Id int PRIMARY KEY,
    Venues_Name varchar(50)
);
```

Figure 7 - Create "Venues" table.

```
CREATE TABLE bookings (
    Booking_Id int PRIMARY KEY,
    Venue_ID int,
    Date date,
    Time time,
    FOREIGN KEY (Venue_ID) REFERENCES venues (Venues_Id)
);
```

Figure 8 - Create "Bookings" table.

```
CREATE TABLE matches (
    Match_Id int PRIMARY KEY,
    Match_Number int,
    Team_ID int,
    isHomeTeam boolean,
    Booking_ID int,
    FOREIGN KEY (Team_ID) REFERENCES team (Team_Id),
    FOREIGN KEY (Booking_ID) REFERENCES bookings (Booking_ID)
);
```

Figure 9 - Create "Matches" table.

```sql
CREATE TABLE goals (
    Goal_Id int PRIMARY KEY,
    Player_Id int,
    Match_ID int,
    Time time,
    FOREIGN KEY (Match_ID) REFERENCES matches (Match_ID)
);
```

Figure 10 - Create "Goals" table.

```sql
CREATE TABLE fouls (
    Foul_Id int PRIMARY KEY,
    Player_Id int,
    Foul_Type varchar(50),
    Match_ID int,
    Time time,
    FOREIGN KEY (Player_Id) REFERENCES player (Consultant_Id),
    FOREIGN KEY (Match_ID) REFERENCES matches (Match_ID)
);
```

Figure 11 - Create "Fouls" table.

```sql
CREATE TABLE transfers (
    Transfer_Id serial PRIMARY KEY,
    Player_Id int,
    Old_Team_Id int,
    Date date,
    FOREIGN KEY (Player_Id) REFERENCES player (Consultant_Id),
    FOREIGN KEY (New_Team_Id) REFERENCES team (Team_Id)
);
```

Figure 12 - Create " Transfers" table.

While creating the database, a few changes we made along the way. To be able to delete the incorrect tables the following code was used.

```sql
DROP TABLE table_name;
```

[3]

Figure 13 - Structure to delete a table.

[3] SQL DROP TABLE Statement (w3schools.com)

Data Manipulation Language

DML is used to manipulate the data stored in the database. In this case, the following code demonstrates the structure used to insert the information provided on the tables previously created.

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

[4]

Figure 14 - Code structure to insert information in a table previously created.

The implementation of the previous code structure can be shown in figures 15 to 22.

```
INSERT INTO team (Team_Id, Team_Name) VALUES
(1, 'Data Masters'),
(2, 'Vis Wizards'),
(3, 'BI Gods'),
(4, 'Data Cleaners');
```

Figure 15 - Insert information into "Team" table.

```
INSERT INTO participatingGroup (ParticipatingGroup_Id, Cohort_Name) VALUES
(1, 'Cohort 4'),
(2, 'Cohort 5'),
(3, 'Cohort 6'),
(4, 'Cohort 7'),
(5, 'Bench'),
(6, 'Training Team'),
(7, 'HR'),
(8, 'Consultants');
```

Figure 16 - Insert information into "participatingGroup" table.

[4] PostgreSQL: Documentation: 16: 6.1. Inserting Data

```
INSERT INTO player (Consultant_ID, Last_Name, First_Name, Team_ID, Group_ID) VALUES
(200, 'Petty', 'Aedan', 1, 1),
(201, 'Santos', 'Aliza', 1, 3),
(202, 'Vaughan', 'Kaylynn', 1, 6),
(203, 'Bauer', 'Arjun', 1, 4),
(204, 'Huber', 'Lilian', 1, 2),
(205, 'Roberts', 'Lizeth', 1, 6),
(206, 'Mcdowell', 'Nathan', 1, 7),
(207, 'Ali', 'Alvin', 1, 8),
(208, 'Christensen', 'Jordin', 1, 7),
(209, 'Blevins', 'Saul', 1, 7),
(210, 'Meza', 'Carina', 3, 2),
(211, 'Campos', 'Isabelle', 3, 5),
(212, 'Phelps', 'Kyleigh', 3, 1),
(213, 'Wong', 'Angela', 3, 7),
(214, 'Rojas', 'Kole', 3, 7),
(215, 'Potts', 'Martha', 3, 4),
(216, 'Powell', 'Tomas', 3, 6),
(217, 'Clarke', 'Paxton', 3, 5),
(218, 'Dodson', 'Jamya', 3, 8),
(219, 'Clements', 'Georgia', 3, 8),
(220, 'Crawford', 'Edwin', 2, 2),
(221, 'Osborn', 'Malachi', 2, 8),
(222, 'Kent', 'Zion', 2, 2),
(223, 'Reyes', 'Anahi', 2, 5),
(224, 'Cabrera', 'Maddox', 2, 8),
(225, 'Gutierrez', 'Brody', 2, 4),
(226, 'Stevenson', 'Hayley', 2, 3),
(227, 'Sanchez', 'Kamora', 2, 8),
(228, 'Holmes', 'Livia', 2, 6),
(229, 'Jenkins', 'Tanner', 2, 8),
(230, 'Meadows', 'Madelyn', 4, 5),
(231, 'Wilkerson', 'Paola', 4, 3),
(232, 'Patton', 'Jared', 4, 6),
(233, 'Washington', 'Pierre', 4, 8),
(234, 'Cochran', 'Dominik', 4, 4),
(235, 'Skinner', 'Miya', 4, 4),
(236, 'Barnett', 'Mara', 4, 5),
(237, 'Dodson', 'Cornelius', 4, 2),
(238, 'Kaiser', 'Ashleigh', 4, 6),
(239, 'Meza', 'Weston', 4, 6);
```

Figure 17 - Insert information into " Player" table.

```
INSERT INTO venues (Venues_Id, Venues_Name) VALUES
(1, 'Wimbledon 1'),
(2, 'Wimbledon 2'),
(3, 'Wimbledon 3');
```

Figure 18 - Insert information into "Venues" table.

```sql
INSERT INTO bookings (Booking_Id, Venue_ID, Date, Time)
VALUES
    (1, 1, '2022-10-01', '16:00'),
    (2, 2, '2022-10-01', '16:00'),
    (3, 2, '2022-10-08', '16:00'),
    (4, 3, '2022-10-08', '16:00'),
    (5, 3, '2022-10-22', '16:00'),
    (6, 2, '2022-10-22', '16:00'),
    (7, 1, '2022-10-29', '16:00'),
    (8, 3, '2022-10-29', '16:00'),
    (9, 3, '2022-11-05', '16:00'),
    (10, 1, '2022-11-05', '16:00'),
    (11, 2, '2022-11-12', '16:00'),
    (12, 3, '2022-11-12', '16:00');
```

Figure 19 - Insert information into "Bookings" table.

```sql
INSERT INTO matches (Match_Id, Match_Number, Team_ID, isHomeTeam, Booking_ID) VALUES
(1, 1, 1, true, 1),   -- Data Masters (Home Team), Match 1
(2, 1, 3, false, 1),  -- BI Gods (Away Team), Match 1
(3, 2, 2, true, 2),   -- Vis Wizards (Home Team), Match 2
(4, 2, 4, false, 2),  -- Data Cleaners (Away Team), Match 2
(5, 3, 1, true, 3),   -- Data Masters (Home Team), Match 3
(6, 3, 2, false, 3),  -- Vis Wizards (Away Team), Match 3
(7, 4, 3, true, 4),   -- BI Gods (Home Team), Match 4
(8, 4, 4, false, 4),  -- Data Cleaners (Away Team), Match 4
(9, 5, 1, true, 5),   -- Data Masters (Home Team), Match 5
(10, 5, 4, false, 5),  -- Data Cleaners (Away Team), Match 5
(11, 6, 3, true, 6),   -- BI Gods (Home Team), Match 6
(12, 6, 2, false, 6),  -- Vis Wizards (Away Team), Match 6
(13, 7, 3, true, 7),   -- BI Gods (Home Team), Match 7
(14, 7, 1, false, 7),  -- Data Masters (Away Team), Match 7
(15, 8, 4, true, 8),   -- Data Cleaners (Home Team), Match 8
(16, 8, 2, false, 8),  -- Vis Wizards (Away Team), Match 8
(17, 9, 2, true, 9),   -- Vis Wizards (Home Team), Match 9
(18, 9, 1, false, 9),  -- Data Masters (Away Team), Match 9
(19, 10, 4, true, 10), -- Data Cleaners (Home Team), Match 10
(20, 10, 3, false, 10),-- BI Gods (Away Team), Match 10
(21, 11, 4, true, 11), -- Data Cleaners (Home Team), Match 11
(22, 11, 1, false, 11),-- Data Masters (Away Team), Match 11
(23, 12, 2, true, 12), -- Vis Wizards (Home Team), Match 12
(24, 12, 3, false, 12);-- BI Gods (Away Team), Match 12
```

Figure 20 - Insert information into "Matches" table.

```sql
INSERT INTO goals (Goal_Id, Player_Id, Match_ID, Time) VALUES
(1, 207, 1, NULL),   -- Data Masters
(2, 206, 1, NULL),   -- Data Masters
(3, 234, 2, NULL),   -- Data Cleaner
(4, 226, 2, NULL),   -- Vis Wizards
(5, 207, 3, NULL),   -- Data Masters
(6, 226, 3, NULL),   -- Vis Wizards
(7, 218, 4, NULL),   -- BI Gods
(8, 216, 4, NULL),   -- BI Gods
(9, 234, 4, NULL),   -- Data Cleaner
(10, 218, 6, NULL),   -- BI Gods
(11, 227, 6, NULL),   -- Vis Wizards
(12, 226, 6, NULL),   -- Vis Wizards
(13, 215, 7, NULL),   -- BI Gods
(14, 218, 7, NULL),   -- BI Gods
(15, 207, 7, NULL),   -- Data Masters
(16, 202, 7, NULL),   -- Data Masters
(17, 234, 8, NULL),   -- Data Cleaner
(18, 224, 8, NULL),   -- Vis Wizards
(19, 228, 9, NULL),   -- Vis Wizards
(20, 225, 9, NULL),   -- Vis Wizards
(21, 206, 9, NULL),   -- Data Masters
(22, 207, 9, NULL),   -- Data Masters
(23, 218, 10, NULL),   -- BI Gods
(24, 234, 11, NULL),   -- Data Cleaner
(25, 236, 11, NULL),   -- Data Cleaner
(26, 225, 12, NULL),   -- Vis Wizards
(27, 224, 12, NULL),   -- Vis Wizards
(28, 218, 12, NULL),   -- BI Gods
(29, 217, 12, NULL);   -- BI Gods
```

Figure 21 - Insert information into " Goals" table.

```sql
INSERT INTO fouls (Foul_Id, Player_Id, Foul_Type, Match_ID, Time) VALUES
(1, 211, 'Yellow card', 1, NULL),
(2, 229, 'Yellow card', 2, NULL),
(3, 200, 'Yellow card', 3, NULL),
(4, 208, 'Yellow card', 3, NULL),
(5, 231, 'Red card', 4, NULL),
(6, 209, 'Yellow card', 5, NULL),
(7, 237, 'Yellow card', 5, NULL),
(8, 202, 'Yellow card', 7, NULL),
(9, 234, 'Yellow card', 8, NULL),
(10, 222, 'Yellow card', 9, NULL),
(11, 231, 'Red card', 10, NULL),
(12, 200, 'Yellow card', 11, NULL),
(13, 216, 'Yellow card', 12, NULL);
```

Figure 22 - Insert information into "Fouls" table.

Code validation

To ensure the tables were created and populated the following code was tested. Figures 23 to 30 represent the code used to visualise the tables in the output and the corresponding output.

```
SELECT *
FROM team;
```

	team_id [PK] integer	team_name character varying (50)
1	1	Data Masters
2	2	Vis Wizards
3	3	BI Gods
4	4	Data Cleaners

Figure 23 - Select statement and outcome for "Team" table.

```
SELECT *
FROM participatingGroup;
```

	participatinggroup_id [PK] integer	cohort_name character varying (50)
1	1	Cohort 4
2	2	Cohort 5
3	3	Cohort 6
4	4	Cohort 7
5	5	Bench
6	6	Training Team
7	7	HR
8	8	Consultants

Figure 24 - - Select statement and outcome for "ParticipatingGroup" table.

```
SELECT *
FROM player;
```

	consultant_id [PK] integer	last_name character varying (50)	first_name character varying (50)	team_id integer	group_id integer
1	200	Petty	Aedan	1	1
2	201	Santos	Aliza	1	3
3	202	Vaughan	Kaylynn	1	6
4	203	Bauer	Arjun	1	4
5	204	Huber	Lilian	1	2
6	205	Roberts	Lizeth	1	6
7	206	Mcdowell	Nathan	1	7
8	207	Ali	Alvin	1	8
9	208	Christensen	Jordin	1	7
10	209	Blevins	Saul	1	7
11	210	Meza	Carina	3	2
12	211	Campos	Isabelle	3	5
13	212	Phelps	Kyleigh	3	1
14	213	Wong	Angela	3	7
15	214	Rojas	Kole	3	7
16	215	Potts	Martha	3	4
17	216	Powell	Tomas	3	6
18	217	Clarke	Paxton	3	5
19	218	Dodson	Jamya	3	8
20	219	Clements	Georgia	3	8
21	220	Crawford	Edwin	2	2
22	221	Osborn	Malachi	2	8
23	222	Kent	Zion	2	2
24	223	Reyes	Anahi	2	5
25	224	Cabrera	Maddox	2	8
26	225	Gutierrez	Brody	2	4

Figure 25 - - Select statement and outcome for "Player" table.

```
SELECT *
FROM venues;
```

	venues_id [PK] integer	venues_name character varying (50)
1	1	Wimbledon 1
2	2	Wimbledon 2
3	3	Wimbledon 3

Figure 26 - - Select statement and outcome for "Venues" table.

```
SELECT *
FROM bookings;
```

	booking_id [PK] integer	venue_id integer	date date	time time without time zone
1	1	1	2022-10-01	16:00:00
2	2	2	2022-10-01	16:00:00
3	3	2	2022-10-08	16:00:00
4	4	3	2022-10-08	16:00:00
5	5	3	2022-10-22	16:00:00
6	6	2	2022-10-22	16:00:00
7	7	1	2022-10-29	16:00:00
8	8	3	2022-10-29	16:00:00
9	9	3	2022-11-05	16:00:00
10	10	1	2022-11-05	16:00:00
11	11	2	2022-11-12	16:00:00
12	12	3	2022-11-12	16:00:00

Figure 27 - Select statement and outcome for "Bookings" table.

```
SELECT *
FROM matches;
```

	match_id [PK] integer	match_number integer	team_id integer	ishometeam boolean	booking_id integer
1	1	1	1	true	1
2	2	1	3	false	1
3	3	2	2	true	2
4	4	2	4	false	2
5	5	3	1	true	3
6	6	3	2	false	3
7	7	4	3	true	4
8	8	4	4	false	4
9	9	5	1	true	5
10	10	5	4	false	5
11	11	6	3	true	6
12	12	6	2	false	6
13	13	7	3	true	7
14	14	7	1	false	7
15	15	8	4	true	8
16	16	8	2	false	8
17	17	9	2	true	9
18	18	9	1	false	9
19	19	10	4	true	10
20	20	10	3	false	10
21	21	11	4	true	11

Figure 28 - Select statement and outcome for "Matches" table.

```
SELECT *
FROM goals;
```

	goal_id [PK] integer	player_id integer	match_id integer	time time without time zone
1	1	207	1	[null]
2	2	206	1	[null]
3	3	234	2	[null]
4	4	226	2	[null]
5	5	207	3	[null]
6	6	226	3	[null]
7	7	218	4	[null]
8	8	216	4	[null]
9	9	234	4	[null]
10	10	218	6	[null]
11	11	227	6	[null]
12	12	226	6	[null]
13	13	215	7	[null]
14	14	218	7	[null]
15	15	207	7	[null]
16	16	202	7	[null]
17	17	234	8	[null]
18	18	224	8	[null]
19	19	228	9	[null]
20	20	225	9	[null]
21	21	206	9	[null]
22	22	207	9	[null]

Figure 29 - Select statement and outcome for "Goals" table.

```
SELECT *
FROM fouls;
```

	foul_id [PK] integer	player_id integer	foul_type character varying (50)	match_id integer	time time without time zone
1	1	211	Yellow card	1	[null]
2	2	229	Yellow card	2	[null]
3	3	200	Yellow card	3	[null]
4	4	208	Yellow card	3	[null]
5	5	231	Red card	4	[null]
6	6	209	Yellow card	5	[null]
7	7	237	Yellow card	5	[null]
8	8	202	Yellow card	7	[null]
9	9	234	Yellow card	8	[null]
10	10	222	Yellow card	9	[null]
11	11	231	Red card	10	[null]
12	12	200	Yellow card	11	[null]
13	13	216	Yellow card	12	[null]

Figure 30 - Select statement and outcome for "Fouls" table.

With these figures it is possible to see that the tables were created and properly populated.

Test database implementation

After the implementation phase, tests were conducted to

This involved answering the following questions:

1. Listing all students, who play for a particular department (i.e. Cohort 4 group).
2. Listing all fixtures for a specific date (i.e. 29th of October 2022 and including team names and venues).
3. Listing all the players who have scored more than 2 goals.

Each designed to validate different aspects of the database implementation.

Exercise 1

This code retrieves the full names of the players who belong to a particular group, in this case Cohort 4.

```
SELECT CONCAT(player.First_Name, ' ', player.Last_Name) AS Players
FROM player
JOIN participatingGroup ON player.Group_ID = participatingGroup.ParticipatingGroup_Id
WHERE participatingGroup.Cohort_Name = 'Cohort 4';
```

Figure 31 - Exercise 1 SQL solution

The “SELECT CONCAT” statement selects and binds the first and last name of each player (adding a space in between) in a column entitled “Players”.

The “ FROM” statement retrieves all the data from the “Player” table.

The “JOIN” statement groups the “Player” table to the “ParticipatingGroup” table based on the “group_Id” matching in both tables.

The “WHERE” statement filters the participating group by name, in this case “Cohort 5”.

The output from the code in Figure 31 can be seen in Figure 32.

	players text
1	Aedan Petty
2	Kyleigh Phelps

Figure 32 - Exercise 1 output

Exercise 2

This code retrieves information about matches scheduled for a particular date, in this case (“2022-10-29”), including match number home and away team names and venue name.

```sql
SELECT
    m.Match_Number,
    home_team.Team_Name AS Home_Team,
    away_team.Team_Name AS Away_Team,
    v.Venues_Name
FROM matches m
INNER JOIN team home_team ON m.Team_ID = home_team.Team_Id AND m.isHomeTeam = true
INNER JOIN matches m_away ON m.Match_Number = m_away.Match_Number AND m.Match_Id <> m_away.Match_Id
INNER JOIN team away_team ON m_away.Team_ID = away_team.Team_Id AND m_away.isHomeTeam = false
INNER JOIN bookings b ON m.Booking_ID = b.Booking_Id
INNER JOIN venues v ON b.Venue_ID = v.Venues_Id
WHERE b.Date = '2022-10-29';
```

Figure 33 - Exercise 2 SQL solution

The “SELECT” statement selects all the information we want to have in our output such as the match number (“m.Match_Number”), home team name (“home_team.Team_Name”), away team name (“away_team.Team_Name”) and the name of the venue the match was played on (“v.Venues_Name”).

The “FROM” statement retrieves all the data from the “Matches” table and aliases it with “m” to improve readability.

The “INNER JOIN team home_team ON m.Team_ID = home_team.Team_Id AND m.isHomeTeam = true” statement joins the “Matches” and “Team” table by “Team_Id” and ensure that it is the home team(“isHomeTeam = true”) as well as aliases it with “home_team”.

The “INNER JOIN matches m_away ON m.Match_Number = m_away.Match_Number AND m.Match_Id <> m_away.Match_Id” joins the “Matches” table with itself to get the information about the away team. Since the “Match_Number” is the same for the away and home team in the same match, this is the condition used to match the original match to the new one. To ensure the information on the away team is retrieved, the fact that the Match_Id is different is also a condition.

The “INNER JOIN team away_team ON m_away.Team_ID = away_team.Team_Id AND m_away.isHomeTeam = false” statement going the “Team” table to be able to retrieve the away teams name through its Id. The “m_away.isHomeTeam=false” ensures its the away team.

The “INNER JOIN bookings b ON m.Booking_ID = b.Booking_Id” statement retrieves the information about the venue the match was played on by combining the “Matches” table with the “Bookings” table by its corresponding “booking_Id”.

The “INNER JOIN venues v ON b.Venue_ID = v.Venues_Id” statement, as a continuation from the previous one, retrieves the Venues name by joining the “Venues” table with the “Bookings” table through the corresponding “venues_Id”.

The “WHERE b.Date = ‘2022-10-29’" statement filters the results that only match the date written.

The output from the code in Figure 33 can be seen in Figure 34.

	match_number integer	home_team character varying (50)	away_team character varying (50)	venues_name character varying (50)
1	7	BI Gods	Data Masters	Wimbledon 1
2	8	Data Cleaners	Vis Wizards	Wimbledon 3

Figure 34 - Exercise 2 output

Exercise 3

This code counts the number of goal each player scored and retrieves information on the players who scored more than 2 goals.

```
SELECT
    CONCAT(player.First_Name, ' ', player.Last_Name) AS Players,
    COUNT(goals.Goal_Id) AS Goals_Scored
FROM goals
INNER JOIN player ON goals.Player_Id = player.Consultant_ID
GROUP BY player.Consultant_ID
HAVING COUNT(goals.Goal_Id) > 2;
```

Figure 35 - Exercise 3 SQL solution

The “SELECT” statement is retrieving information of the players full name (selects and binds the first and last name of each player (adding a space in between) in a column entitled “Players”) and the total amount of goals each one scored (“COUNT(goals.Goal_Id)”).

The “FROM” statement retrieves all the data from the “Goals” table.

The "INNER JOIN player ON goals.Player_Id = player.Consultant_ID” statement joins the “Goals” table with the “Player” table through the “Player_ID“ / “Consultant_ID”.

The “GROUP BY player.Consultant_ID” statement groups the result by the consultant Id resulting in a “COUNT” statement which will be applied to each consultant Id.

The “HAVING COUNT(goals.Goal_Id) > 2” filters the results for the players who have scored more than 2 goals.

The output from the code in Figure 35 can be seen in Figure 36.

	players text	goals_scored bigint
1	Hayley Stevenson	3
2	Alvin Ali	4
3	Jamya Dodson	5
4	Dominik Cochran	4

Figure 36 - Exercise 3 output

Despite the successful completion of these 3 exercises, this database needs further testing and improvement to be optimal for use.

Exercise 1 was turned into views to be easily accessed in different query tools through:

SELECT players

FROM public."StudentsCohort4";

Conclusions

Throughout the project, we learned various aspects of database design and implementation. With this learning the basics of SQL syntax, data definition, manipulation, and optimization. This hands-on experience provided us with valuable insights into database design principles and best practices.

Despite the overall success of the project, there were several challenges during the design and implementation phases as well as time constraints for the delivery of the project. These challenges included managing complex relationships between entities.

Moving forward, there are numerous opportunities for further exploration, including advanced query optimization and the exploration of technologies like machine learning. These advance to continue to evolve and innovate our database management practices will better serve the needs of Rockborne and stakeholders.

Appendix

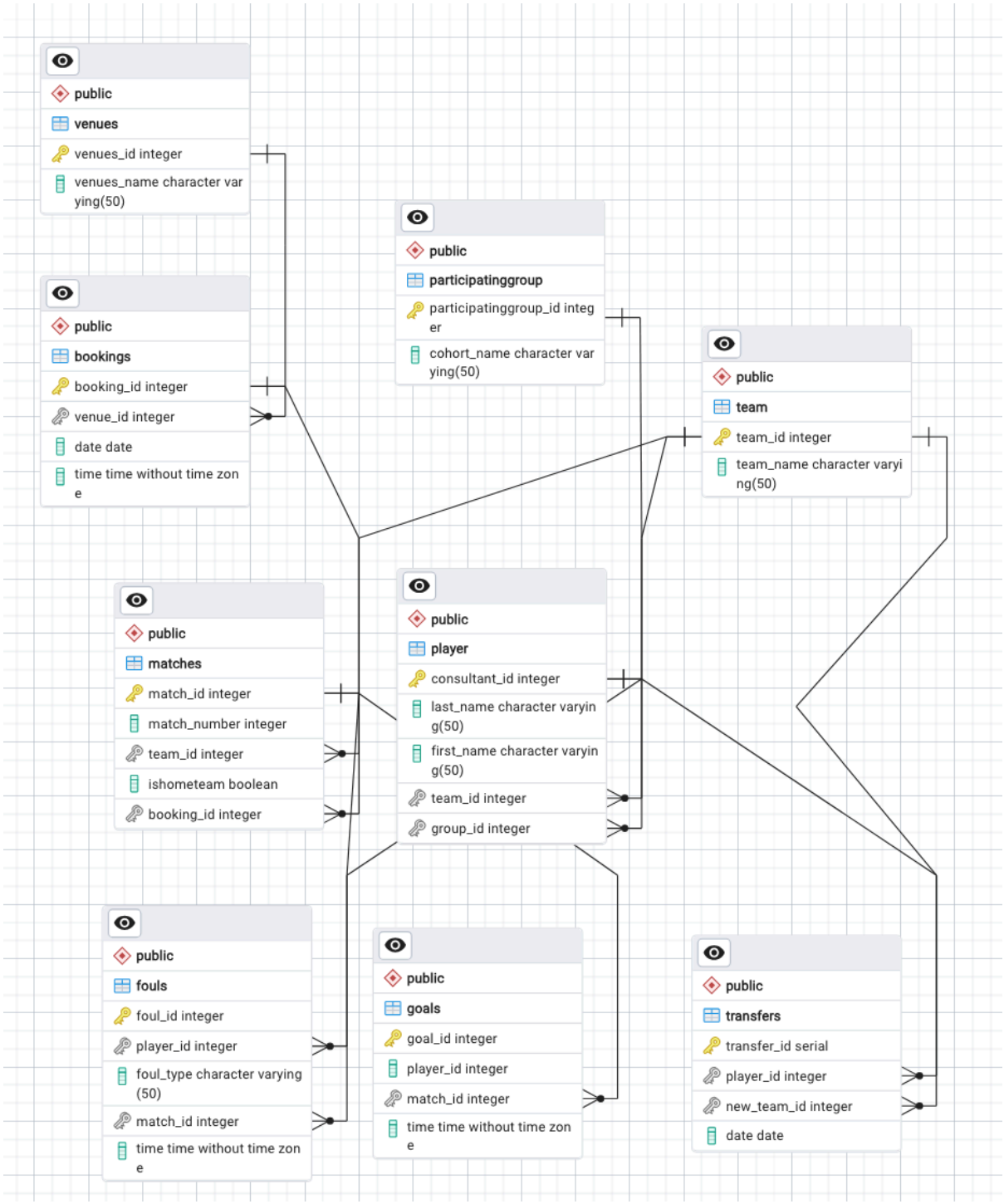

Figure 37 - ER Diagram through PGAdmin