**Assignment - Designing FIFA Database**
**ISYS 2014 Database Systems**
**Atharva Dengle (20376888)**
**20/10/2023 Semester 2**

## Introduction

The women's football dataset to build for FIFA World Cup matches was selected for this assignment. The main focus will be on a few matches played in 2023, with another dataset referenced to meet the 250-300 rows criteria. Entities such as Team, Manager, Player, Match, FinalScore, PenaltyScore, Referee and Stadium are created. Furthermore, tables, a relational schema, an Entity-Relationship Diagram (ERD) model, and implemented the database in an SQL environment with queries and complex additions such as procedures triggers. Finally, the database was connected to another programming language (Python).

## Entity, Relationship and Data Description Explanation

**For the main entities, I used:**
**Team -** to highlight each team participating in the matches represented by the teamName, which is a unique identity for each team as they are all from different countries.
**Manager -** to indicate the individual who manages a team and is a separate entity from the team
**Player -** the team's composition is through multiple players (with a player being the captain) with unique playerIDs for each person.
**Match -** separates all matches by a matchID and indicates the two teams involved. The match also requires a stadiumID for the match to take place.
**Final Score -** The ending scores of both teams are required to be shown to identify a match winner.
**Penalty Score -** The penalty scores are required to check the match's outcome if the final score is a tie.
**Referee -** The main dictator of the match, who decides valid and invalid plays, can give out red and yellow cards to players who violate rules (however, for simplicity in this database, only the amount of red and yellow cards given out by the referee will be considered)
**Stadium -** The destination of where the match is held and contains area data and maximum seating capacity.

**For the relationships between entities:**
**Manages** - a single manager "manages" a team, and each team can only have one manager.
**Part of** - between players and teams, multiple players can be part of one team, indicating that the player entity and the "part of" relationship are weak.
**Leads** - despite not being reflected in the tables, a captain will lead the rest of the "athletes" or other players within a team.
**Participate in** - one team can participate in several matches, which is a one-to-many relationship.
**Played in** - a match is played in a stadium; several matches can also be played in a singular stadium, which is later seen in the data.
**Has** - a match has a final score for both the home and away teams, and each team can only have a singular score for a match.

**Has** - again, the penalty scores are singular for each team within matches. There are no penalty scores for several matches and they are kept at 0-0 as the final score is considered before the penalty scores.

**Data types considered and used:**
The majority of the data types used are the **VARCHAR** variables, and this is due to the fluctuating data for names (of team, player, manager and referee), which can be different in length for each entry. **INT** is used for the scoring system, recording the team's establishment year, the stadium's seating capacity and unique identifiers for all entities. Despite places where **CHAR** can be used, **VARCHAR** is more versatile in its ability to accept different lengths of inputs and, hence, was mainly implemented in the database.

| **Entity Sets** | **Keys** | **Other attributes** |
|---|---|---|
| Team | teamName | establishYear |
| Manager | managerID, teamName (FK) | mfirstName, mlastName |
| Player | playerID, teamName (FK) | pfirstName, plastName |
| Match | matchID, stadiumID (FK) | homeTeam, awayTeam |
| Final Score | matchID (FK) | homeFinal, awayFinal |
| Penalty Score | matchID (FK) | homePenalty, awayPenalty |
| Referee | refereeID, matchID (FK) | firstName, lastName, redCards, yellowCards |
| Stadium | stadiumID | stadiumName, city, country, seatingCapacity |

| **Relationship Sets** | **Entity Sets** | **Attributes of Relationships** |
|---|---|---|
| Manages | Manager, Team | |
| Part of | Player, Team | |
| Participate in | Team, Match | |
| Played in | Match, Stadium | |
| Officiated | Referee, Match | |
| Has | Match, Final Score | |
| Has | Match, Penalty Score | |

| **Relationship Set** | **Cardinality** | **Participation/Constraints** |
|---|---|---|
| Manages | one-to-one | Manager and Team both have **total** participation, as a manager cannot exist without a team, and a team cannot exist without a manager |
| Part of | many-to-one | Player - **total**, a player cannot exist without being associated with a team<br>Team - **partial**, a team can have several players |
| Participate in | one-to-many | Match - **total**, without two teams, a match cannot be held<br>Team - **partial**, team can exist without participating in a match |
| Officiated | many-to-many | Referee - **partial**, a referee can exist without a match<br>Match - **total**, a match cannot proceed without a referee |
| Played in | many-to-one | Match - **total**, a match cannot be played if a stadium does not exist<br>Stadium - **partial**, a stadium can exist without having a match |
| Has | one-to-one | Match - **total**, a match requires a final score<br>Final Score - **total**, a final score is determined at the end of a match |
| Has | one-to-one | Match - **total**, a match requires a penalty score<br>Penalty Score - **total**, a penalty score is determined after the penalties are made |

**Relational Schema:**

Manager(managerID, mfirstName, mlastName, teamName)
    FK teamName REF Team(teamName)
Team(teamName, establishYear)
Player(playerID, pfirstName, plastName, captain_ID, teamName)
    FK teamName REF Team(teamName)
Match(matchID, thome, taway, stadiumID)
    FK thome REF Team(teamName)
    FK taway REF Team(teamName)
    FK stadiumID REF Stadium(stadiumID)
PenaltyScore(matchID, homePenalty, awayPenalty)
    FK matchID REF Match(matchID)
FinalScore(matchID, awayFinal, homeFinal)
    FK matchID REF Match(matchID)
Stadium(stadiumID, stadiumName city, country, seatingCapacity)
Referee(refereeID, firstName, lastName, redCards, yellowCards, matchID)
    FK matchID REF Match(matchID)

**Data Dictionary:**

| Manager | | | | | |
|---|---|---|---|---|---|
| Description: | Information regarding the managers of the team. | | | | |
| **ATTRIBUTE** | **TYPE** | **SIZE** | **PRIMARY KEY** | **NULL** | **DESCRIPTION** |
| managerID | INT | 5 | Y | N | Unique identifiers for each manager |
| mfirstName | VARCHAR | 15 | N | N | Managers first name |
| mlastName | VARCHAR | 15 | N | N | Managers last name |
| teamName | VARCHAR | 15 | N | N | Unique identifiers for each team participating |

| Team | | | | | |
|---|---|---|---|---|---|
| Description: | Information regarding the teams playing in the world cup | | | | |
| **ATTRIBUTE** | **TYPE** | **SIZE** | **PRIMARY KEY** | **NULL** | **DESCRIPTION** |
| teamName | VARCHAR | 15 | Y | N | Unique identifiers for each team participating |
| establishYear | INT | 4 | N | N | Year the team was established |

| Player | | | | | |
|---|---|---|---|---|---|
| Description: | Information regarding the players competing | | | | |
| **ATTRIBUTE** | **TYPE** | **SIZE** | **PRIMARY KEY** | **NULL** | **DESCRIPTION** |
| playerID | INT | 4 | Y | N | Unique identifiers for each player |
| teamName | VARCHAR | 15 | N | N | Unique identifiers for each team participating |
| pfirstName | VARCHAR | 15 | N | N | First name of player |
| plastName | VARCHAR | 15 | N | N | Last name of player |
| captainID | INT | 2 | N | Y | Identifier of a captain |

| Match | | | | | |
|---|---|---|---|---|---|
| Description: | Information regarding the matches that will be played | | | | |
| **ATTRIBUTE** | **TYPE** | **SIZE** | **PRIMARY KEY** | **NULL** | **DESCRIPTION** |
| matchID | INT | 4 | Y | N | Unique identifiers for the matches |
| homeTeam | VARCHAR | 15 | N | N | Name of the home team |
| awayTeam | VARCHAR | 15 | N | N | Name of the away team |
| stadiumID | INT | 6 | N | N | The stadium where the match is held |

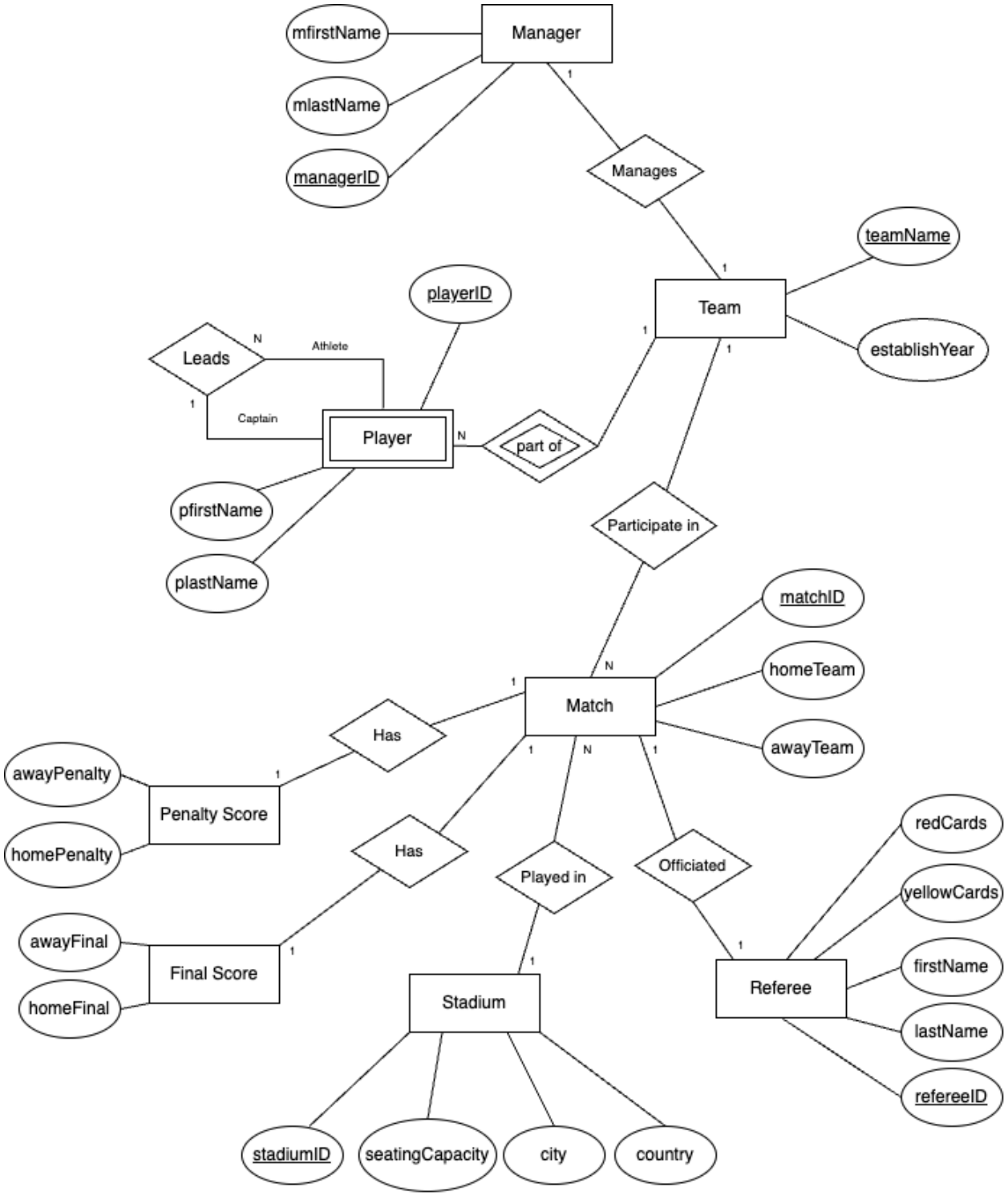
| Penalty Score | | | | | |
|---|---|---|---|---|---|
| Description: | Information regarding the penalty shootout scores | | | | |
| **ATTRIBUTE** | **TYPE** | **SIZE** | **PRIMARY KEY** | **NULL** | **DESCRIPTION** |
| matchID | INT | 4 | Y | N | Unique identifiers for each team participating |
| homePenalty | INT | 2 | N | N | Penalty score of the home team |
| awayPenalty | INT | 2 | N | N | Penalty score of the away team |

| Final Score | | | | | |
|---|---|---|---|---|---|
| Description: | Information regarding the final scores of the match | | | | |
| **ATTRIBUTE** | **TYPE** | **SIZE** | **PRIMARY KEY** | **NULL** | **DESCRIPTION** |
| matchID | INT | 4 | Y | N | Unique identifiers for each team participating |
| homeFinal | INT | 2 | N | N | Final score of the home team |
| awayFinal | INT | 2 | N | N | Final score of the away team |

| Stadium | | | | | |
|---|---|---|---|---|---|
| Description: | Information regarding the stadium where the match is held | | | | |
| **ATTRIBUTE** | **TYPE** | **SIZE** | **PRIMARY KEY** | **NULL** | **DESCRIPTION** |
| stadiumID | INT | 5 | Y | N | Unique identifiers for each stadium |
| stadiumName | VARCHAR | 25 | N | N | Name of the stadium |
| city | VARCHAR | 15 | N | N | Name of the city the stadium is in |
| country | VARCHAR | 15 | N | N | Name of the country the stadium is in |
| seatingCapacity | INT | 6 | N | N | Amount of people the stadium can seat |

| Referee | | | | | |
|---|---|---|---|---|---|
| Description: | Information regarding the referee of each match and their decisions | | | | |
| **ATTRIBUTE** | **TYPE** | **SIZE** | **PRIMARY KEY** | **NULL** | **DESCRIPTION** |
| refereeID | INT | 4 | Y | N | Unique identifiers for each referee |
| rfirstName | VARCHAR | 15 | N | N | Penalty score of the home team |
| rlastName | VARCHAR | 15 | N | N | Penalty score of the away team |
| redCards | INT | 2 | N | N | Number of red cards given out |
| yellowCards | INT | 2 | N | N | Number of yellow cards given out |
| matchID | INT | 4 | N | N | Unique identifier for each team participating in a match |

## Entity-Relationship Diagram (ERD)



## Assumptions of Database Design

The database created is not a comprehensive design of the "true" FIFA World Cup sample data but captures the key aspects of the information that must be stored. Some assumptions that were made include:

- Team managers remain the same (as opposed to changing or having multiple managers over time)
- A team only has a singular captain despite the possibility of a team having more than one.
- Having one referee per match when it is common to have assistant and backup referees.
- Unique identifiers are a requirement for each entity (compared to more use of composite keys).
- It will only capture current rather than historical or previous data that could be significant.
- That a match will go to completion, i.e. it will end with a final score or penalty score, despite the possibility of forfeits/withdrawals.
- Cards given out by the referee are counted separately. Having players have their red/yellow card count in the Player table would be more accurate.
- Functional data is required to be inserted (the majority is NOT NULL)

## Implementation of the Database and Addition of Sample Data

The database was created based on the data dictionary and relational schema made prior. A few variables were selected to create the database by viewing the FIFA World Cup dataset. Each entity was created using the CREATE TABLES functionality, and their respective primary and foreign keys were added along with other attributes. The data type, size and null values were filled before implementing the database, and they were selected by reviewing the example dataset given.

The sample data will be placed individually into each table and written on a separate document before copying into the MySQL environment. Upon insertion, there were initially a few errors due to the creation of some entities before others, especially those that had foreign keys and were referenced in other tables. Hence, it is required to be ordered into the correct sequence that would validate the code. Upon correction, the sample data needed to be sorted into each table and individually inserted into the table using the INSERT INTO() and VALUES() functions. This process was the most repetitive, as many entries needed to be made.

The majority of the data was taken from the provided dataset, but the player data was taken from https://www.kaggle.com/datasets/rossi14/fifa-womens-world-cup-2023-players.

After the sample data was made, there were a few errors initially due to unsuitable values and missing quotations for VARCHAR variables. This was overcome by running each line in the environment individually to see where errors occurred. The createTables and sample data were written into SQL files that could be run with the SOURCE function that references them. When each line of code was successfully run, it was compiled into a single SQL file and then run as a whole. This simplified the process and made it less redundant.

1. **Simple Queries**

**--1**. **Display the teamName of teams that have been established after 1980.**
SELECT teamName FROM Team WHERE establishYear > 1980;

This query selects the teams that were formed after 1980. This information can be used to see which teams are newer and compare their performance to older teams. This query selects a primary key after checking that the year is above a certain threshold (1980).

**--2**. **Display the referee ID, firstName, lastName and matchID regarding the referees who have given more than 2 yellow cards.**
SELECT refereeID, rfirstName, rlastName, matchID FROM Referee WHERE yellowCards > 2;

This query selects the referees that give out more than two yellow cards. This information can be viewed to see which referees are more strict in enforcing the World Cup rules. This query was created to show multiple attribute selections with the yellow card attribute being higher than a certain amount (2).

**--3**. **Display the playerID, firstName, lastName and captainID of the players who are captains.**
SELECT playerID, pfirstName, plastName, captainID FROM Player WHERE captainID IS NOT NULL;

It allows the captains of a team to be identified from all teams. The query can be useful when seeing the structure of the team formation, and despite not being included in the database, the scoring information of each player can be identified to see which players make the best calls and should be assigned the title of captain in the future. This query showed how multiple attributes could be selected for players with captainID as NULL.

**--4**. **Display the matchID, and homeFinal scores where the homeFinal scores are greater than the awayFinal scores.**
SELECT matchID, homeFinal FROM FinalScore WHERE homeFinal > awayFinal;

This indicates which teams perform better in their home games (this can differ for each country) but can be analysed to show which teams perform better when playing at a more familiar stadium. The game's stadium could be a favourable consideration for league match organisers in a larger dataset. This query was made to show how multiple attributes could be selected after applying a condition.

**--5**. **Find the establishment year of the team with the earliest establishment year.**
SELECT MIN(establishYEAR) AS earliestYear FROM Team;

The usefulness of this query is limited, but it highlights the oldest team in the matches. This could be used to measure the team's performance over the years. This query was mainly made to display the use of the MIN() function in a query.

## 2. Advanced Queries

**--1**. **List all the matchID and final scores for home and away teams with their teamNames.**
SELECT FinalScore.matchID, `Match`.homeTeam, FinalScore.homeFinal, `Match`.awayTeam, FinalScore.awayFinal FROM `Match`, FinalScore WHERE `Match`.matchID=FinalScore.matchID;

This query shows the final scores of the home and away teams and joins the tables of `Match` and FinalScore, where the matchID is the foreign key in FinalScore and the primary key of `Match`. Other than showing the winner of the match, this information can also be used to predict future teams' wins based on their skill level or players. This query shows comma joins and joins the tables on the unique matchID.

**--2**. **Using a suitable join query, all the matches and their respective referees (should be given as fullname and refereeID).**
SELECT `Match`.matchID, CONCAT(Referee.rfirstName, ' ', Referee.rlastName) AS fullName, Referee.refereeID FROM `Match` JOIN Referee ON `Match`.matchID=Referee.matchID;

This query joins the matches and the referee who officiates the match. It can show how many matches a referee officiates. In a more complex database, the analysis of the referee's performance can be displayed to see if any behave in a biased manner. This query shows the use of the equi join and the CONCAT() feature that combines two columns into one.

**--3**.**Find the matchID with the highest combined away and home scores, and list the teamNames and the respective scores for that match (with appropriate column headings).**
SELECT `Match`.matchID, `Match`.homeTeam, FinalScore.homeFinal, `Match`.awayTeam, FinalScore.awayFinal, (FinalScore.homeFinal + FinalScore.awayFinal) AS CombinedScore FROM `Match` JOIN FinalScore ON FinalScore.matchID=`Match`.matchID WHERE (homeFinal + awayFinal) = (SELECT MAX(homeFinal + awayFinal) FROM FinalScore);

This query joins the `Match` and FinalScore tables and selects the highest combined score match. It shows the match and other information of the highest-scoring match of the sample data given. This would be beneficial to analyse which combination of teams plays the best offensively and defensively, allowing for prediction of future match outcomes. This query showcases different column relations and applies a WHERE clause after the search condition. It also contains a subquery that selects the MAX() of the sum of the home and away final scores.

**--4**. **Find the refereeID, firstName, lastName (as fullname) and the yellowCards number for the referee who has given out the largest amount of yellow Cards**
SELECT refereeID AS RefereeID, CONCAT(rfirstName, ' ', rlastName) AS FullName, yellowCards AS YellowCards FROM Referee WHERE yellowCards >= ALL(SELECT yellowCards FROM Referee);

This query selects the referee with the highest yellow card amount as their attribute. This would be useful in analysing which countries and matches receive the highest yellow cards, potentially flagging players who play violently. Despite this database having yellow cards as a referee attribute, this could be implemented in a more complex database. This query uses a subquery and the ALL() function and finds the referee who has given the most yellow cards.

**--5**. **List the matchID, stadiumID, stadiumName, homeTeam, awayTeam and seatingCapacity for the matches played where the seatingCapacity exceeds 50000**

```
SELECT `Match`.matchID, Stadium.stadiumID, Stadium.stadiumName, `Match`.homeTeam, `Match`.awayTeam, Stadium.seatingCapacity FROM Stadium JOIN `Match` ON `Match`.stadiumID=Stadium.stadiumID WHERE seatingCapacity >= 50000;
```

This query indicates several attributes and joins the stadium and match tables. Viewing the information for the stadiums with large capacities makes it possible to predict the crowd attracted to matches between countries. This can help better cater to meet the seating requirements and avoid overcrowding. The query uses equi join with the stadiumID being the main condition, and the seating capacity is to exceed 50,000 people.

**--6**. **Find the average establishYear and select the matchID, homeTeam, awayTeam and stadiumName of the matches played by the team.**

```
SELECT `Match`.matchID, `Match`.homeTeam, `Match`.awayTeam, Stadium.stadiumName
FROM `Match`
JOIN Team ON `Match`.homeTeam = Team.teamName OR `Match`.awayTeam = Team.teamName
JOIN Stadium ON `Match`.stadiumID = Stadium.stadiumID
WHERE Team.establishYear = FLOOR((SELECT AVG(establishYear) FROM Team));
```

This query finds several attributes and joins team, match and stadium tables. Seeing which team/s were formed by finding the average year of all teams being viewed can help set a performance benchmark for older or newer teams. This query utilises two equi joins, with the teamName and the stadiumID being the main conditions, and also uses subqueries to find the average of the years using the AVG() function (the FLOOR() function is used in this instance to make the query work upon calculating the team that will be selected using the average establishment year).

**<u>Use of views/procedures/triggers</u>**

**View:**

```
CREATE VIEW MatchDetails AS
SELECT
    `Match`.matchID,
    `Match`.homeTeam,
    `Match`.awayTeam,
    FinalScore.homeFinal,
    FinalScore.awayFinal,
    Stadium.stadiumName,
    Stadium.city,
    Stadium.country
 FROM `Match`
 JOIN FinalScore ON `Match`.matchID = FinalScore.matchID
 JOIN Stadium ON `Match`.stadiumID = Stadium.stadiumID;
```

This view creates an overall view of the dataset, compiling several attributes from different tables to display the data as a whole for each data row. This simple view allows the user to see the overall tournament data for the 2023 FIFA World Cup.

**Procedure 1:**

```
DELIMITER $$
CREATE PROCEDURE GetMatchesForTeam(
    IN teamName VARCHAR(15)
 )
 BEGIN
    SELECT *
    FROM `Match`
    WHERE homeTeam = teamName OR awayTeam = teamName;
 END $$
```

This procedure finds the matchID, homeTeam, awayTeam and stadiumID for matches that are played by the team that is placed in the parameter. This is useful in determining what teams will be playing against each other in the tournament.

**Procedure 2:**

```
DELIMITER $$

CREATE PROCEDURE AddPlayer(
   IN newPlayerID INT,
   IN newTeamName VARCHAR(15),
   IN newFirstName VARCHAR(10),
   IN newLastName VARCHAR(10),
   IN newCaptainID INT
)
BEGIN
   INSERT INTO Player(playerID, teamName, pfirstName, plastName, captainID)
   VALUES(newPlayerID, newTeamName, newFirstName, newLastName, newCaptainID);
   IF newCaptainID IS NOT NULL THEN
      UPDATE Player SET captainID = NULL WHERE teamName = newTeamName AND
captainID IS NOT NULL;
      UPDATE Player SET captainID = newCaptainID WHERE teamName = newTeamName
AND playerID = newPlayerID;
   ELSE
      SET newCaptainID = NULL;
   END IF;
END $$

DELIMITER ;
```

This procedure allows for players to be inserted when placed in the parameters, and if the captainID is given instead of being left as NULL, the new insertion will be set to as the new captain of the team, and will update the old captain to a normal player. This allows for each change to the captains to be recorded effortlessly.

**Trigger 1:**

```
DELIMITER $$

CREATE TRIGGER stadiumInsert
BEFORE INSERT
ON Stadium FOR EACH ROW
BEGIN
   DECLARE max_capacity INT;
   SET max_capacity = 114000;
   IF NEW.seatingCapacity > max_capacity THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Seating capacity cannot be larger than 114000 as this is the max
capacity for the largest stadium in the world.';
   END IF;
END$$

DELIMITER ;
```

This trigger ensures that the seating capacity won't exceed 150000 as this is not a feasible amount of seats to have in a stadium (that currently exists). This will ensure any invalid insertions won't be made.

**Trigger 2:**

```
DELIMITER $$

CREATE TRIGGER before_insert_Match
BEFORE INSERT ON `Match`
FOR EACH ROW
BEGIN
   IF NEW.stadiumID IS NULL OR NOT EXISTS (SELECT 1 FROM Stadium WHERE
stadiumID = NEW.stadiumID) THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Error: The specified stadiumID does not exist.';
   END IF;
END$$

DELIMITER ;
```

This trigger ensures any new insertions into `Match` will have a valid stadiumID, if it doesn't exist, a specific error code will be shown. This helps the user identify where he has made the mistake (if the stadiumID provided is incorrect) or it will allow the insertion to go to completion.

## Data Connectivity and Python Implementation

### a. Running previously defined queries using python

```
import mysql.connector
conn = mysql.connector.connect(host='localhost', database='Fifa_WorldCup', user='root',
password='myUserPassword')
database_commands = 'database_commands_copy.sql'
cursor = conn.cursor()
with open(database_commands, 'r') as commands:
    sql_queries = commands.read()
queries = sql_queries.split(';')
query_counter = 0
for query in queries:
    if query.strip():
        query_counter += 1
        print(f"Query {query_counter}")
        print()
        try:
            cursor.execute(query)
            results = cursor.fetchall()
            for row in results:
                print(row)
            print()
            print("Query complete.")
            print()
        except mysql.connector.Error as err:
            print(f"Error: {err}")
conn.commit()
cursor.close()
conn.close()
```

The above code uses a Python to MySQL connection to run the previously mentioned queries. This is all placed into a py (Python) file and can be run to see the queries being run and their corresponding output. This code uses a "with open" function and reads the original file with all the queries. It then strips the data based on the ";" as each separate line to be run. The for loop is used to iterate over each query line, run it, and show the output. The try-except function is an error-handling process that will catch errors if they occur while running the loop. The counter is used to identify the query number and have it as the title for each query output. Finally, the cursor is committed to the database, and the cursor and connection are closed.

### b. Running insert commands in Python

```
import mysql.connector
conn = mysql.connector.connect(host='localhost', database='Fifa_WorldCup', user='root',
password='myUserPassword')
cursor = conn.cursor()
select1 = "SELECT * FROM Player"
cursor.execute(select1)
rows = cursor.fetchall()
print()
```

```
print("Before insertion: ")
print()
print(rows)
print()
insert_multiple = "INSERT INTO Player (playerID, teamName, pfirstName, plastName,
captainID) VALUES (%s, %s, %s, %s, %s)"
data = [(4064, 'Canada', 'Kailen', 'Sheridan', None), (4065, 'Canada', 'Simi', 'Awujo', None),
(4066, 'Canada', 'Nichelle', 'Prince', None),]
cursor.executemany(insert_multiple, data)
conn.commit()
select2 = "SELECT * FROM Player"
cursor.execute(select2)
rows = cursor.fetchall()
print()
print("After insertion: ")
print()
print(rows)
print()
cursor.close()
conn.close()
```

This code uses Python to insert values into the player table. Upon connecting to the SQL server, the initial player table is selected and printed to display the rows before the insertion of the players. Following this, the insert into function and the values in "data" are defined. Using the cursor's executemany function, the players are inserted into the table. Another select-all function is created to show the newly inserted values. Upon showing the added players, the cursor is closed, and the connection is ended.

**c. Running delete commands in Python**

```
import mysql.connector
conn = mysql.connector.connect(host='localhost', database='Fifa_WorldCup', user='root',
password='myUserPassword')
cursor = conn.cursor()
select1 = "SELECT * FROM Manager"
cursor.execute(select1)
rows = cursor.fetchall()
print()
print("Before deletion: ")
print()
print(rows)
print()
delete_multiple = "DELETE FROM Manager WHERE managerID = %s"
data = [(20193,), (23343,), (23084,)]
cursor.executemany(delete_multiple, data)
conn.commit()
select2 = "SELECT * FROM Manager"
cursor.execute(select2)
rows = cursor.fetchall()
print()
print("After deletion: ")
```

```
print()
print(rows)
print()
cursor.close()
conn.close()
```

The structure of this code is similar to inserting Players, but this time, Managers are being deleted. This is possible due to the managers' absence of main primary keys with relations elsewhere (only have teamName as a foreign key, and the manager ID is used nowhere else). This code initially shows the rows of data from the Manager's table, then, using executemany, deletes selected managers, and finally shows the changed table.

**c. Running update commands in Python**

```
import mysql.connector
conn = mysql.connector.connect(host='localhost', database='Fifa_WorldCup', user='root',
password='myUserPassword')
cursor = conn.cursor()
select1 = "SELECT * FROM Stadium"
cursor.execute(select1)
rows = cursor.fetchall()
print()
print("Before update: ")
print()
print(rows)
print()
update_multiple = "UPDATE Stadium SET seatingCapacity = %s WHERE stadiumID = %s"
data = [(40000, 648395), (40000, 845734)]
cursor.executemany(update_multiple, data)
conn.commit()
select2 = "SELECT * FROM Stadium"
cursor.execute(select2)
rows = cursor.fetchall()
print()
print("After update: ")
print()
print(rows)
print()
cursor.close()
conn.close()
```

The above Python code is used to update inserts that already exist. Upon starting the connection, the Stadium table is selected and printed before the update. Using a conditioned query, it is possible to update the Stadiums based on their ID, and in this case, two particular stadiumIDs were chosen. They were to update their seatingCapacity to 40000 from what they were initially inserted. After committing the change, the resulting Stadium table is shown with the updated values.

**Discussion**:

Creating the database, tables and other elements with relevant SQL queries was successful. The addition of data was seamless using the dataset, the insertions and useful simple/advanced queries were generated to view data that can be important to users of this database, such as league organisers and other parties. Python implementation was also achieved by establishing a MySQL connection and running the Python code to manipulate the database, showing an extension of using another programming language in SQL.

The challenges in producing the database would be initially implementing procedures and triggers. This was particularly difficult as some errors were unknown, and its layout and overall structure seemed correct. However, this was overcome and accomplished by checking online and with the PowerPoints given in the unit materials. Another challenge was creating several insert statements, a lengthy and repetitive process to meet the requirements of 250-300 data points. It took work to overcome as there were fewer distinct data points when focusing on the 2023 matches, so another dataset was also referenced.

Not using time data was a limitation of the database. The dates of the matches and other information, such as the player's birthdate, were excluded to keep the database simple, but this could easily be added in future work. Another limitation is the potential redundancy of the Final and Penalty scores. They could have been included in the `Match` table as they must exist for a match. However, both were created as separate entities as they differ in how they are scored. Final scores are scored during the match, but penalty scores are made during the penalty shootouts. The tables are kept separate to indicate how the scores were made. Including "Scores" in the `Match` table in future work is possible as fewer tables will be created, which will avoid redundancy. Some features that should have been considered include having the score that each player makes in the Player table individually. This was not implemented as the overall score is usually referenced when comparing matches, but it is another aspect that could be included in future work. More needy FIFA World Cup data users may require more extensive information such as player position, assists, saves and fouls. However, this was not included in the database as this information was not provided in either dataset used.