Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Khant	Student ID:	20618166
Other name(s):	Freddy		
Unit name:	Database Systems	Unit ID:	ISYS2014
Lecturer / unit coordinator:	Nimalika Fernando	Tutor:	Jordan Richards
Date of submission:	15/10/2023	Which assignment?	Final (Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: Freddy Date of signature: 15/10/2023

(By submitting this form, you indicate that you agree with all the above text.)

Document version: 1.1 (2015-11-15)

Introduction

In this project, I was tasked of creating a relational database for the FIFA Women's World Cup, one of the most renowned football events globally. The intention was to create a robust and efficient database that captures essential details about teams, players, games, and stadiums. This report encapsulates our approach to designing, implementing, and querying the database.

Design of the Database

i. Explanation of Selections:

Entities were chosen based on the core components of the FIFA Women's World Cup. The primary entities identified were:

- Player: Representing individual participants in the tournament.
- Team: Representing national teams competing in the tournament.
- Game: Representing individual matches played during the tournament.
- Stadium: Representing venues where matches are held.

The selection of these entities ensures comprehensive coverage of all aspects of the tournament.

Relationships and Data Types:

Relationships between entities were established based on logical associations. For instance:

- A Team has multiple Players.
- A Game involves two Teams and is played in one Stadium.

Data types for attributes were chosen based on the nature of the data they represent. For example:

- Textual data such as names and positions were stored as VARCHAR.
- Numeric data like ages and scores were stored as INT.
- Date-related data, like match dates, were stored as DATE.

This meticulous selection of data types ensures data accuracy and optimizes storage.

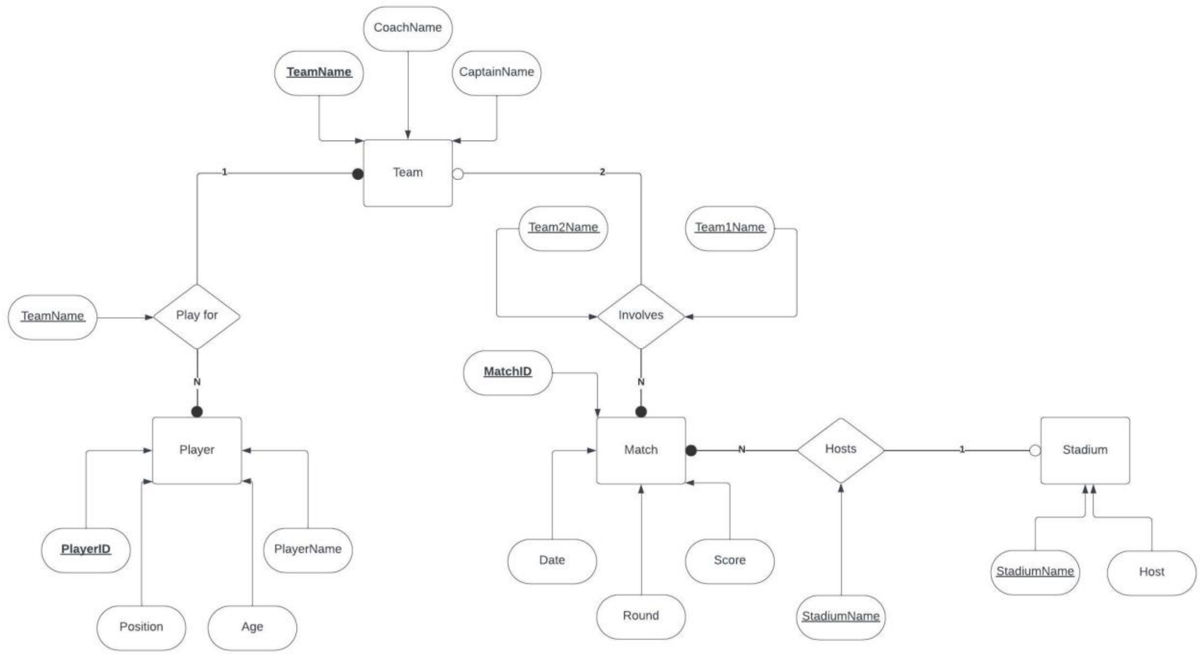
Entity Sets	Keys	Other Attributes
Team	TeamName (Primary)	ManagerName, CaptainName
Player	PlayerID (Primary), TeamName	PlayerName,

Entity Sets	Keys	Other Attributes
	(Foreign)	Position, Age
Game	GameID (Primary), Team1Name (Foreign), Team2Name (Foreign), StadiumName (Foreign)	Round, Date, Score
Stadium	StadiumName (Primary)	Host

Relationship Sets	Between which entity Sets	Attributes
Play for	Player-Team	
Involves	Game-Team	
Hosts	Game-Stadium	

Relationship Sets	Cardinality Constraints	Participation/other constraints
Play for	Many-one (A player plays for one team, a team has many players)	Player - total, Team - total (A player must play in a team, and a team must have players)
Involves	Many-two (A Game involves two teams, teams play in many games)	Game - total, Team - partial (A Game must be played by teams, but a team might not play in a Game)
Hosts	Many-one (Many games can be hosted in one stadium, but a Game is hosted in only one stadium)	Game - total, Stadium - partial (A Game must be played in a stadium, but a stadium might not host any games)

ER Diagram, Relational Schema, Data Description:



The ER Diagram visually represents the entities, their attributes, and the relationships between them. It serves as a blueprint for the database's structure.

Entity Sets:

1. **Team**
   - **Primary Key**: TeamName
   - **Attributes**: ManagerName, CaptainName
2. **Player**
   - **Primary Key**: PlayerID
   - **Foreign Key**: TeamName
   - **Attributes**: PlayerName, Position, Age
3. **Game (Match)**
   - **Primary Key**: GameID
   - **Foreign Keys**: Team1Name, Team2Name, StadiumName
   - **Attributes**: Round, Date, Score
4. **Stadium**

- **Primary Key**: StadiumName
- **Attributes**: Host

Relationship Sets:

1. **Play for**
   - **Between**: Player and Team
   - **Cardinality**: Many-to-one (A player plays for one team, a team has many players)
   - **Constraints**: Player - total, Team - total (A player must play in a team, and a team must have players)
2. **Involves**
   - **Between**: Match and Team
   - **Cardinality**: Many-to-two (A match involves two teams, teams play in many games)
   - **Constraints**: Game - total, Team - partial (A game must be played by teams, but a team might not play in a game)
3. **Hosts**
   - **Between**: Match and Stadium
   - **Cardinality**: Many-to-one (Many games can be hosted in one stadium, but a game is hosted in only one stadium)
   - **Constraints**: Game - total, Stadium - partial (A game must be played in a stadium, but a stadium might not host any matches)
   -

ii. Assumptions:

A country has only one team representing it

A player can only play for one team in the tournament.

A Game involves two distinct teams.

A player is between the ages of 16 and 40

Relational Schema:

1. **Team:**

- Team(TeamName, ManagerName, CaptainName)

- Primary Key: TeamName

2. **Player:**

- Player(PlayerID, TeamName, PlayerName, Position, Age)
- Primary Key: PlayerID
- Foreign Key:
- TeamName REF Team(TeamName)

3. **Stadium:**

- Stadium(StadiumName, Host)
- Primary Key: StadiumName

4. **Game:**

- Game(GameID, Team1Name, Team2Name, StadiumName, Round, Date, Score)
- Primary Key: GameID
- Foreign Keys:
- Team1Name REF Team(TeamName)
- Team2Name REF Team(TeamName)
- StadiumName REF Stadium(StadiumName)

Implementation of the Database and Adding Sample Data

Using MySQL, tables for Teams, Players, Games, and Stadiums were set up. Each table was designed with specific constraints to ensure data integrity. Sample data, sourced from the provided links on the Blackboard page were used, not all details from the sample data were implemented and used to a full extent. Specifically, I used the sample data I thought fit and were necessary for my application, since I chose my data to only include the statistics of the 2023 FIFA Women's World Cup – Specifically the Quarter to Grand Final Stages.

Team

Attribute Name	**Data Type**	**Description**	**Constraints**
TeamName	VARCHAR(20)	Unique Team Name, Denoted by their country	Primary Key, NOT NULL

Attribute Name	**Data Type**	**Description**	**Constraints**
ManagerName	VARCHAR(50)	Name of the team's coach	NOT NULL
CaptainName	VARCHAR(50)	Name of the team’s captain	NOT NULL

Player

Attribute Name	**Data Type**	**Description**	**Constraints**
PlayerID	INT	Unique identifier for player	Primary Key, NOT NULL
TeamName	VARCHAR(20)	Identifier linking to the player's team	Foreign Key references **Team(TeamName)**, NOT NULL
PlayerName	VARCHAR(50)	Name of the player	NOT NULL
Position	VARCHAR(30)	Playing position of the player (e.g., Forward, Midfielder)	NOT NULL
Age	INT	Age of the player	NOT NULL, CHECK (Age >= 16 AND Age <= 40)

Business Rule: A player's age must be between 16 and 40.

Game

Attribute Name	Data Type	Description	Constraints
GameID	INT	Unique identifier for the Game	Primary Key, NOT NULL
Team1Name	VARCHAR(20)	Identifier linking to the first team	Foreign Key references **Team(TeamName)**, NOT NULL
Team2Name	VARCHAR(20)	Identifier linking to the second team	Foreign Key references **Team(TeamName)**, NOT NULL, **Team1Name ≠ Team2Name**
StadiumName	VARCHAR(20	Identifier linking to the Game's venue	Foreign Key references **Stadium(StadiumName)**, NOT NULL
Round	VARCHAR(20)	Round of the Game	NOT NULL
Date	DATE	Date of the Game	NOT NULL
Score	VARCHAR(5)	Score of the Game	NOT NULL

Stadium

Attribute Name	Data Type	Description	Constraints
StadiumName	VARCHAR(20	Unique identifier for stadium	Primary Key, NOT NULL
Host	VARCHAR(20)	Host country of the stadium	NOT NULL

Business Rule: A Game must be played between two different teams (i.e., a team cannot play against itself).

e. Use of the Database

i. Design and Implement Queries:

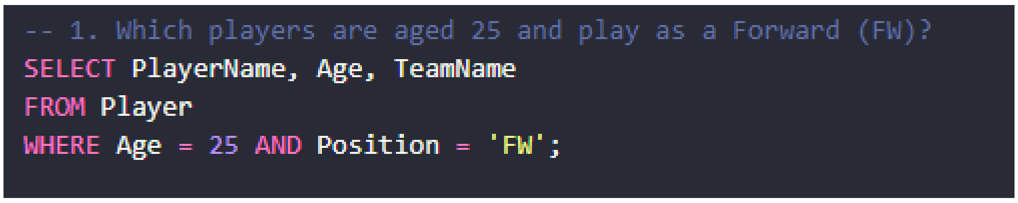
Various queries were designed to extract meaningful insights, such as "Retrieve all players from a particular team" or "Which player played in the most number of Games" The results of these queries provide valuable information for analysts and fans alike.

Evidence:

Design and Implement Queries:

The essence of any database lies in its ability to answer complex questions and provide insights into the stored data. To harness the full potential of the FIFA Women's World Cup Database, a series of SQL queries were designed and implemented. These queries were crafted to address specific questions about the data, providing valuable insights into the tournament. Below is a breakdown of these queries and their significance:

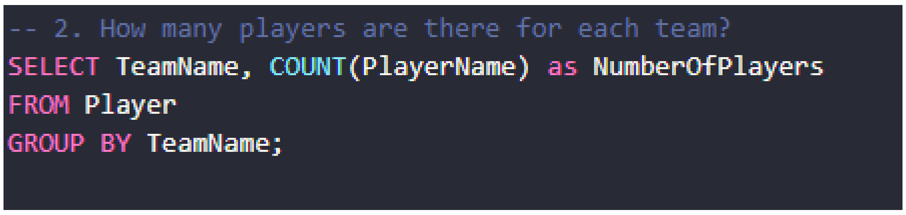
1. **Players aged 25 playing as a Forward (FW)**:



```sql
-- 1. Which players are aged 25 and play as a Forward (FW)?
SELECT PlayerName, Age, TeamName
FROM Player
WHERE Age = 25 AND Position = 'FW';
```

Purpose: This query identifies players who are at the peak of their careers and play in the forward position.

2. **Player Count for Each Team**:



```sql
-- 2. How many players are there for each team?
SELECT TeamName, COUNT(PlayerName) as NumberOfPlayers
FROM Player
GROUP BY TeamName;
```

Purpose: To understand the number of players in each team

3. **Youngest and Oldest Players in the Tournament**:

```sql
-- 3. Who are the youngest and oldest players in the tournament?
(SELECT PlayerName, Age, TeamName
FROM Player
ORDER BY Age ASC LIMIT 1)
UNION
(SELECT PlayerName, Age, TeamName
FROM Player
ORDER BY Age DESC LIMIT 1);
```

Purpose: To identify the age extremes of players participating in the tournament

4. **Most Popular Stadium:**

```sql
-- 4. Which stadium hosted the most games?
SELECT StadiumName, COUNT(GameID) as NumberOfMatches
FROM Game
GROUP BY StadiumName
ORDER BY NumberOfMatches DESC LIMIT 1;
```

Purpose: This query helps in understanding which stadium was preferred or chosen the most

5. **Teams that Played at "Accor Stadium, Sydney"**:

```sql
-- 5. Which teams played at "Accor Stadium, Sydney"?
SELECT DISTINCT Team1Name as Team
FROM Game
WHERE StadiumName = 'Accor Stadium, Sydney'
UNION
SELECT DISTINCT Team2Name as Team
FROM Game
WHERE StadiumName = 'Accor Stadium, Sydney';
```

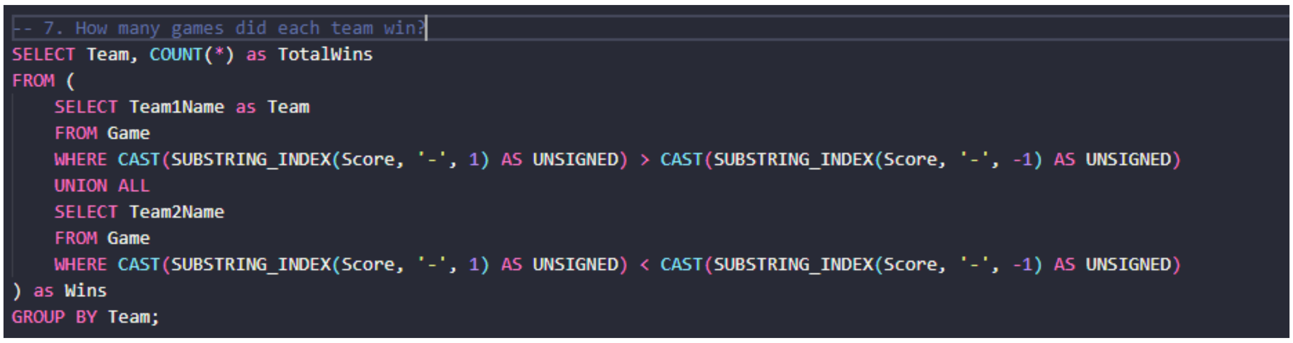
Purpose: To identify, which teams had the opportunity of playing in the famous Accor Stadium in Sydney

6. **Highest Scoring Team**:

```sql
-- 6. Which team has accumulated the highest total score across all matches in the tournament?
SELECT Team, SUM(TotalGoals) as OverallGoals
FROM (
    SELECT Team1Name as Team, CAST(SUBSTRING_INDEX(Score, '-', 1) AS UNSIGNED) as TotalGoals
    FROM Game

    UNION ALL

    SELECT Team2Name as Team, CAST(SUBSTRING_INDEX(Score, '-', -1) AS UNSIGNED) as TotalGoals
    FROM Game
) as CombinedScores
GROUP BY Team
ORDER BY OverallGoals DESC
```

Purpose: To identify which team scored the most goals in the world up

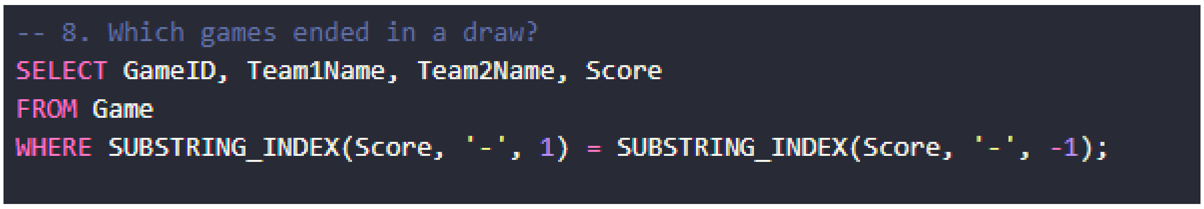
7. **Wins for each Team**



```
-- 7. How many games did each team win?
SELECT Team, COUNT(*) as TotalWins
FROM (
    SELECT Team1Name as Team
    FROM Game
    WHERE CAST(SUBSTRING_INDEX(Score, '-', 1) AS UNSIGNED) > CAST(SUBSTRING_INDEX(Score, '-', -1) AS UNSIGNED)
    UNION ALL
    SELECT Team2Name
    FROM Game
    WHERE CAST(SUBSTRING_INDEX(Score, '-', 1) AS UNSIGNED) < CAST(SUBSTRING_INDEX(Score, '-', -1) AS UNSIGNED)
) as Wins
GROUP BY Team;
```

Purpose: Identify how many wins each team has

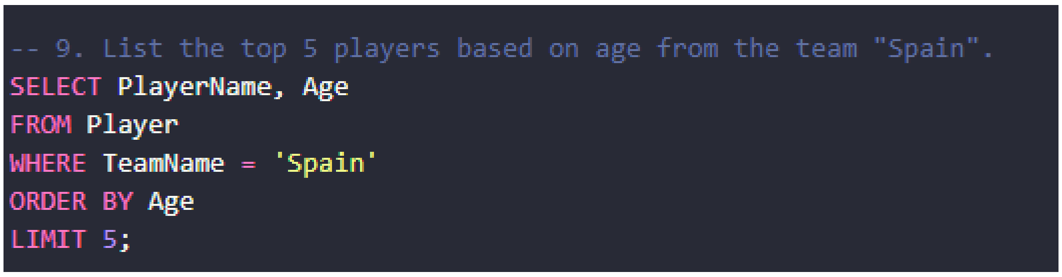
8. **Games That Ended in Draw**



```
-- 8. Which games ended in a draw?
SELECT GameID, Team1Name, Team2Name, Score
FROM Game
WHERE SUBSTRING_INDEX(Score, '-', 1) = SUBSTRING_INDEX(Score, '-', -1);
```

Purpose: Identify how many games ended in a draw

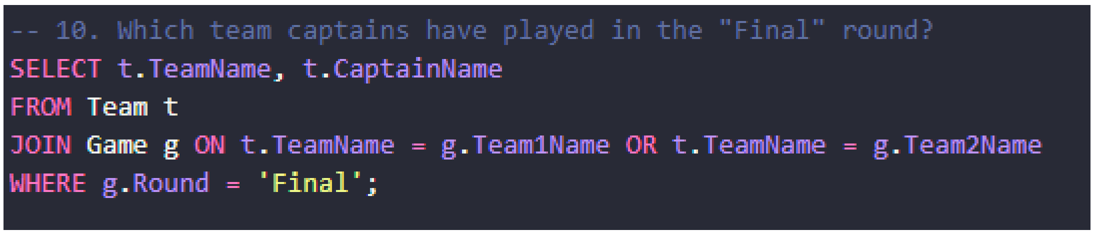
9. **Top 5 Oldest Players in Spain**



```
-- 9. List the top 5 players based on age from the team "Spain".
SELECT PlayerName, Age
FROM Player
WHERE TeamName = 'Spain'
ORDER BY Age
LIMIT 5;
```

Purpose: Identify 5 Oldest Players in Spain

10. **Captains that Played In Grand Final**



```
-- 10. Which team captains have played in the "Final" round?
SELECT t.TeamName, t.CaptainName
FROM Team t
JOIN Game g ON t.TeamName = g.Team1Name OR t.TeamName = g.Team2Name
WHERE g.Round = 'Final';
```

Purpose: Identify the Team Captains that played in the Grand Final

ii. Design and Implementation of Advanced Features:

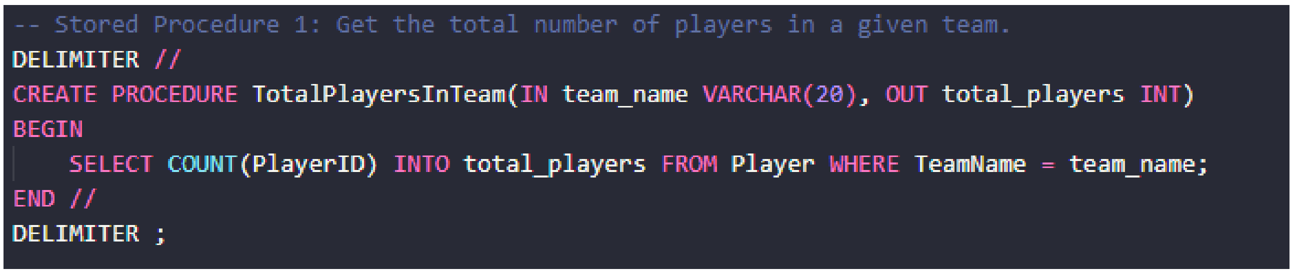
In database management, advanced features play a pivotal role in enhancing the functionality, efficiency, and user experience of the system. For our FIFA Women's World Cup Database, we've integrated several of these advanced features to ensure optimal performance and ease of use.

1. Stored Procedures:

Purpose: Stored procedures allow us to encapsulate a series of SQL statements into a single routine. This not only promotes code reusability but also improves performance as the database can optimize the procedure's execution.

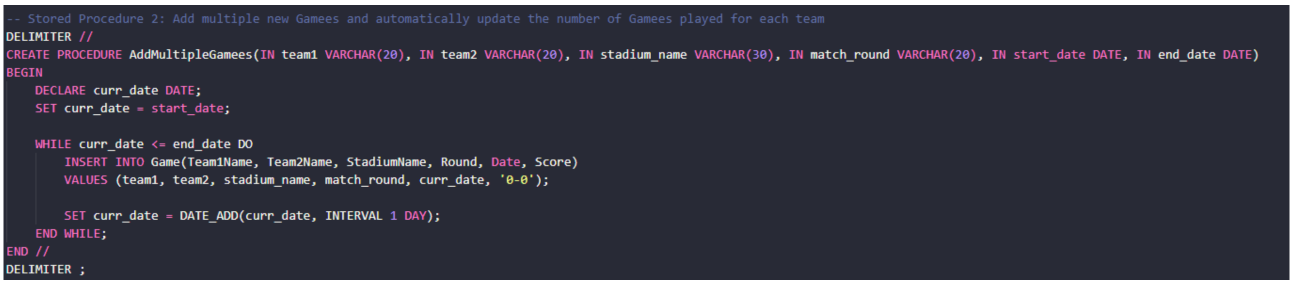
Implementation:

- **TotalPlayersInTeam**: This procedure retrieves the total number of players in a given team. By passing the team's name as an input parameter, the procedure returns the count of players, streamlining the process of gathering team-specific data.



```sql
-- Stored Procedure 1: Get the total number of players in a given team.
DELIMITER //
CREATE PROCEDURE TotalPlayersInTeam(IN team_name VARCHAR(20), OUT total_players INT)
BEGIN
    SELECT COUNT(PlayerID) INTO total_players FROM Player WHERE TeamName = team_name;
END //
DELIMITER ;
```

- **AddMultipleGames**: Designed to facilitate the insertion of multiple games, this procedure takes in parameters like teams, stadium name, match round, and date range. It then iterates over the date range, adding game records for each day, ensuring that data entry is efficient and error-free.



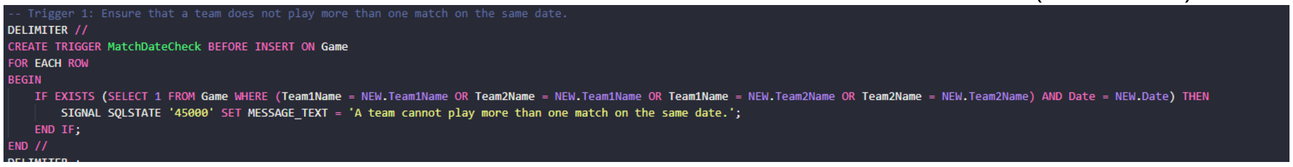
```sql
-- Stored Procedure 2: Add multiple new Gamees and automatically update the number of Gamees played for each team
DELIMITER //
CREATE PROCEDURE AddMultipleGamees(IN team1 VARCHAR(20), IN team2 VARCHAR(20), IN stadium_name VARCHAR(30), IN match_round VARCHAR(20), IN start_date DATE, IN end_date DATE)
BEGIN
    DECLARE curr_date DATE;
    SET curr_date = start_date;

    WHILE curr_date <= end_date DO
        INSERT INTO Game(Team1Name, Team2Name, StadiumName, Round, Date, Score)
        VALUES (team1, team2, stadium_name, match_round, curr_date, '0-0');

        SET curr_date = DATE_ADD(curr_date, INTERVAL 1 DAY);
    END WHILE;
END //
DELIMITER ;
```

2. Triggers:

Purpose: Triggers are automated actions that the database performs in response to specific events. They are crucial for maintaining data integrity and automating tasks that would otherwise be manual and error-prone.

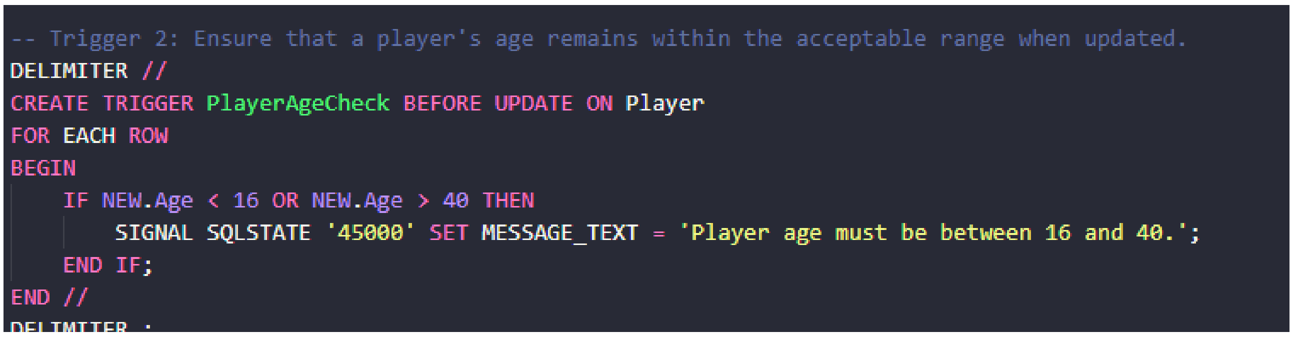
Implementation:

- **MatchDateCheck**: This trigger ensures that a team doesn't play more than one match on the same date. Before inserting a new game record, it checks existing records for date conflicts, preventing data inconsistencies.



```
-- Trigger 1: Ensure that a team does not play more than one match on the same date.
DELIMITER //
CREATE TRIGGER MatchDateCheck BEFORE INSERT ON Game
FOR EACH ROW
BEGIN
    IF EXISTS (SELECT 1 FROM Game WHERE (Team1Name = NEW.Team1Name OR Team2Name = NEW.Team1Name OR Team1Name = NEW.Team2Name OR Team2Name = NEW.Team2Name) AND Date = NEW.Date) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A team cannot play more than one match on the same date.';
    END IF;
END //
```

- **PlayerAgeCheck**: To maintain the realism of our database, this trigger checks a player's age during updates. If the age is outside the acceptable range (16-40), the update is halted, ensuring that player data remains plausible.



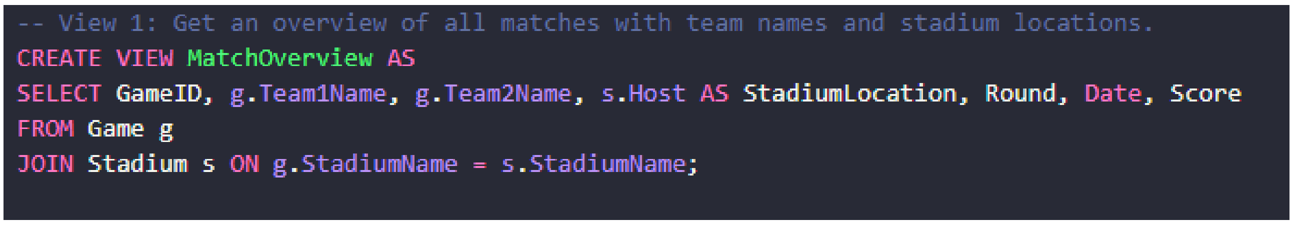
```
-- Trigger 2: Ensure that a player's age remains within the acceptable range when updated.
DELIMITER //
CREATE TRIGGER PlayerAgeCheck BEFORE UPDATE ON Player
FOR EACH ROW
BEGIN
    IF NEW.Age < 16 OR NEW.Age > 40 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Player age must be between 16 and 40.';
    END IF;
END //
```

3. **Views:**

Purpose: Views provide a virtual table based on the result-set of an SQL statement. They allow users to access multiple tables' data in a simplified and secure manner.

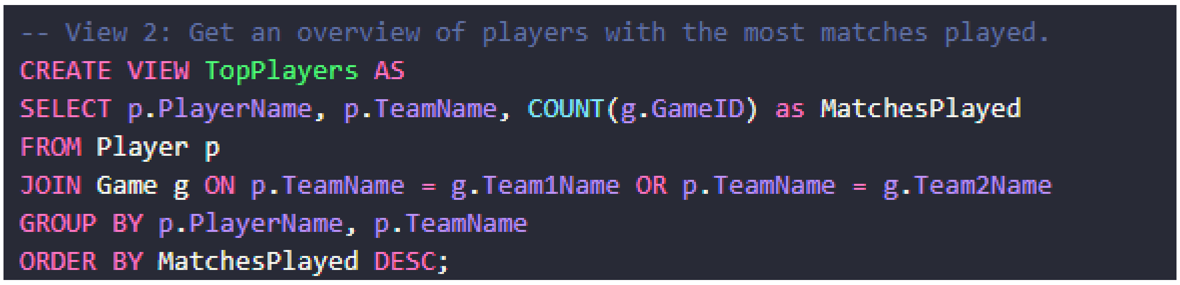
Implementation:

- **MatchOverview**: This view offers a comprehensive look at all matches, amalgamating data from the Game and Stadium tables. It provides details like team names, stadium locations, match rounds, and scores.



```
-- View 1: Get an overview of all matches with team names and stadium locations.
CREATE VIEW MatchOverview AS
SELECT GameID, g.Team1Name, g.Team2Name, s.Host AS StadiumLocation, Round, Date, Score
FROM Game g
JOIN Stadium s ON g.StadiumName = s.StadiumName;
```

- **TopPlayers: Aimed at spotlighting standout players, this view lists players based on the number of matches they've played.** It joins the Player and Game tables, offering insights into each player's contribution to their team.



```
-- View 2: Get an overview of players with the most matches played.
CREATE VIEW TopPlayers AS
SELECT p.PlayerName, p.TeamName, COUNT(g.GameID) as MatchesPlayed
FROM Player p
JOIN Game g ON p.TeamName = g.Team1Name OR p.TeamName = g.Team2Name
GROUP BY p.PlayerName, p.TeamName
ORDER BY MatchesPlayed DESC;
```

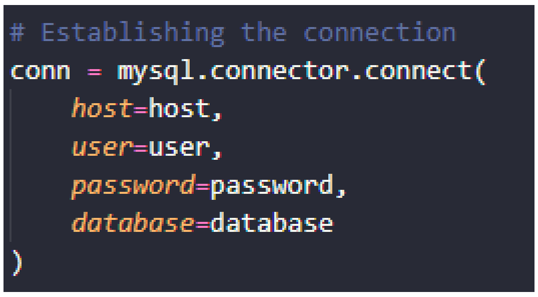
By integrating these advanced features, our database not only ensures data integrity and consistency but also offers a user-friendly interface for complex operations. Whether it's automating data checks with triggers, simplifying data retrieval with stored procedures, or offering consolidated data views, these features elevate the database's overall functionality.

iii. Database Connectivity and Python Implementation:

The FIFA Women's World Cup Database is designed to be interactive and easily accessible. To achieve this, a Python script was developed to connect to the MySQL database and execute various SQL queries. The script leverages the mysql.connector library, which provides a seamless interface between Python and MySQL.

1. **Establishing a Connection:**

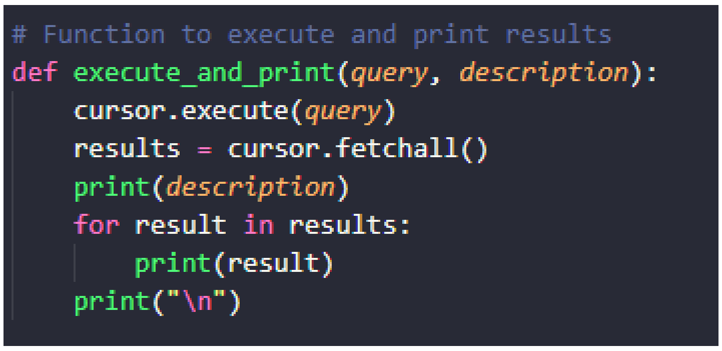
The script initiates a connection to the database using the provided credentials (host, user, password, and database name). This connection serves as the bridge between the Python environment and the MySQL server.



```python
# Establishing the connection
conn = mysql.connector.connect(
    host=host,
    user=user,
    password=password,
    database=database
)
```

2. **Executing Queries**:

A cursor object is created to facilitate the execution of SQL statements. The script contains a function, **execute_and_print**, which not only runs the SQL queries but also prints their results with a description. This function is used to execute a series of predefined queries, such as retrieving players of a certain age and position, counting players for each team, and identifying the youngest and oldest players in the tournament, among others.



```python
# Function to execute and print results
def execute_and_print(query, description):
    cursor.execute(query)
    results = cursor.fetchall()
    print(description)
    for result in results:
        print(result)
    print("\n")
```

Where a query is passed through as an argument:

```
# Query 6
execute_and_print("""
SELECT Team, SUM(TotalGoals) as OverallGoals
FROM (
    SELECT Team1Name as Team, CAST(SUBSTRING_INDEX(Score, '-', 1) AS UNSIGNED) as TotalGoals
    FROM Game
    UNION ALL
    SELECT Team2Name as Team, CAST(SUBSTRING_INDEX(Score, '-', -1) AS UNSIGNED) as TotalGoals
    FROM Game
) as CombinedScores
GROUP BY Team
ORDER BY OverallGoals DESC
LIMIT 1;
""", "Team with the highest total score across all matches:")
```

3. **Stored Procedures:**

The script demonstrates the ability to call stored procedures. For instance, the TotalPlayersInTeam procedure is invoked to retrieve the total number of players in a specified team. The result is fetched and printed, showcasing the seamless integration of stored procedures with the Python environment.

```
# Executing Stored Procedure
team_name = 'Spain'
cursor.callproc('TotalPlayersInTeam', [team_name, 0])
for result in cursor.stored_results():
    total_players = result.fetchone()[0]
print(f"Total players in team {team_name}: {total_players}\n")
```

4. **Triggers:**

While triggers are automatically executed in the database upon certain events, the script includes operations that would activate these triggers. For instance, an attempt to update a player's age to an invalid value would activate the PlayerAgeCheck trigger, resulting in an error message. This demonstrates the database's ability to maintain data integrity through triggers.

```
# Demonstrating Trigger
# This will activate the PlayerAgeCheck trigger
try:
    cursor.execute("UPDATE Player SET Age = 15 WHERE PlayerName = 'SomePlayer';")
    conn.commit()
except mysql.connector.Error as err:
    print(f"Error: {err.msg}\n")
```

5. **Views:**

Views, which are virtual tables based on the result-set of an SQL statement, are also accessed through the script. The MatchOverview view, for example, provides an overview of matches with team names and stadium locations. The script queries this view and displays the results, highlighting the utility of views in consolidating and presenting data.

```python
# Executing a query on a view
execute_and_print("SELECT * FROM MatchOverview LIMIT 5;", "Overview of first 5 matches:")
```

6. **CRUD Operations**

Beyond fetching data, the script demonstrates CRUD (Create, Read, Update, Delete) operations. A new player is inserted into the **Player** table, their age is subsequently updated, and finally, the player is deleted. These operations exemplify the script's capability to modify the database content programmatically.

```python
# CRUD Operations

# Inserting a new player
cursor.execute("INSERT INTO Player (PlayerName, Age, TeamName, Position) VALUES ('John Doe', 24, 'TeamX', 'MF');")
print("New player inserted successfully!")

# Updating the player's age
cursor.execute("UPDATE Player SET Age = 26 WHERE PlayerName = 'John Doe';")
print("Player's age updated successfully!")

# Deleting the player
cursor.execute("DELETE FROM Player WHERE PlayerName = 'John Doe';")
print("Player deleted successfully!")
```

f. Discussion

The process of designing, implementing, and interacting with the FIFA Women's World Cup Database has been both enlightening and challenging. This section reflects on the accomplishments, hurdles faced, and potential avenues for further enhancement.

1. Achievements: The database, from its inception, was envisioned to be comprehensive and user-friendly. The successful creation of a robust relational schema, integration of advanced features like stored procedures, triggers, and views, and the seamless connectivity with Python are testament to the project's success. The database not only stores data efficiently but also ensures data integrity and provides meaningful insights through various queries.

2. Challenges: Throughout the development process, several challenges arose. Designing a schema that accurately represents the complexities of a world cup event, while ensuring normalization, was definitely not an easy task. Implementing advanced features, especially triggers, demanded a deep understanding of the underlying data and potential anomalies. Ensuring that the Python script effectively interacts with the database, especially when invoking stored procedures and handling trigger-induced errors, also posed its set of challenges.

3. Limitations: While the database is comprehensive, there are areas where it can be expanded. For instance, player statistics like goals scored, assists, and minutes played could be incorporated to provide deeper insights. Additionally, the database currently focuses on a single tournament edition; future iterations could expand to cover multiple editions, introducing temporal dimensions to the data.

4. Future Enhancements: There are several avenues for improvement:

- **Larger, more comprehensive database:** The database currently only covers simple data and statistics from the FIFA Women's World Cup, and could be further enhanced and improved to include more games from Group Stages to the Round of 16. Also, more comprehensive statistics, such as each player’s goals, assists, cards and etc.

- **Python CSV Reader:** The database python application could be enhanced to allow a csv file to be read and create its own database for the FIFA Women’s World Cup, allowing it to be more robust.

4. **Concluding Thoughts**: The FIFA Women's World Cup Database project has been a valuable learning experience. It has reinforced the importance of a well-designed database system and showcased the power of integrating databases with programming languages like Python. While the current iteration of the database is robust and functional, the potential for growth and enhancement is vast, mirroring the ever-evolving world of data management.

References

1. FIFA Women's World Cup (1991-2023) Dataset. Kaggle. Retrieved from https://www.kaggle.com/datasets/piterfm/football-fifa-womens-world-cup-1991-2023

2. MySQL 8.0 Reference Manual. MySQL. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/

3. Python MySQL - MySQL Connector Python. MySQL-Python. Retrieved from https://pynative.com/python-mysql-database-connection/

4. FIFA Women's World Cup Archive. FIFA. Retrieved from https://www.fifa.com/womensworldcup/archive/

5. Database Systems (ISYS2014) Lectures 3-10. Discipline of Computing, School of Electrical Engineering, Computing and Mathematical Sciences (EECMS).

6. Database Systems (ISYS2014) Tutorials 3-10. Discipline of Computing, School of Electrical Engineering, Computing and Mathematical Sciences (EECMS).