Basketball Database Schema

Gaurav Ladhar

November 25, 2024

Overview

This project involves designing and implementing a comprehensive basketball Relational Database Management System (RDBMS) using MySQL and phpMyAdmin. This database models a professional basketball environment, capturing key entities such as players, teams, games, player stats, arenas, and positions while allowing for easy future modifiability.

The basketball RDBMS consists of interconnected tables with primary keys, foreign key constraints, lookup tables, and a variety of data types, ensuring data integrity and consistency. The database includes features like cascading updates and deletions, custom default values, and index optimization. Advanced SQL techniques such as triggers, stored procedures, functions, events, and views are incorporated to enhance functionality and usability. The database is implemented in Boyce-Codd normal form (BCNF).

The goal of the project is to design and implement a robust basketball database that enables users to efficiently view and analyze player and team performance. By leveraging phpMyAdmin, the database was structured to handle complex relationships between players, teams, games, and statistics, ensuring data integrity and organization. The system provides insightful views and queries, allowing users to extract detailed metrics and make informed decisions regarding team and player management.

Finally, I created an Enhanced Entity-Relationship (EER) diagram to visually represent the database structure, including tables, relationships, and constraints, ensuring clarity in design and facilitating effective database implementation.

Tools Used

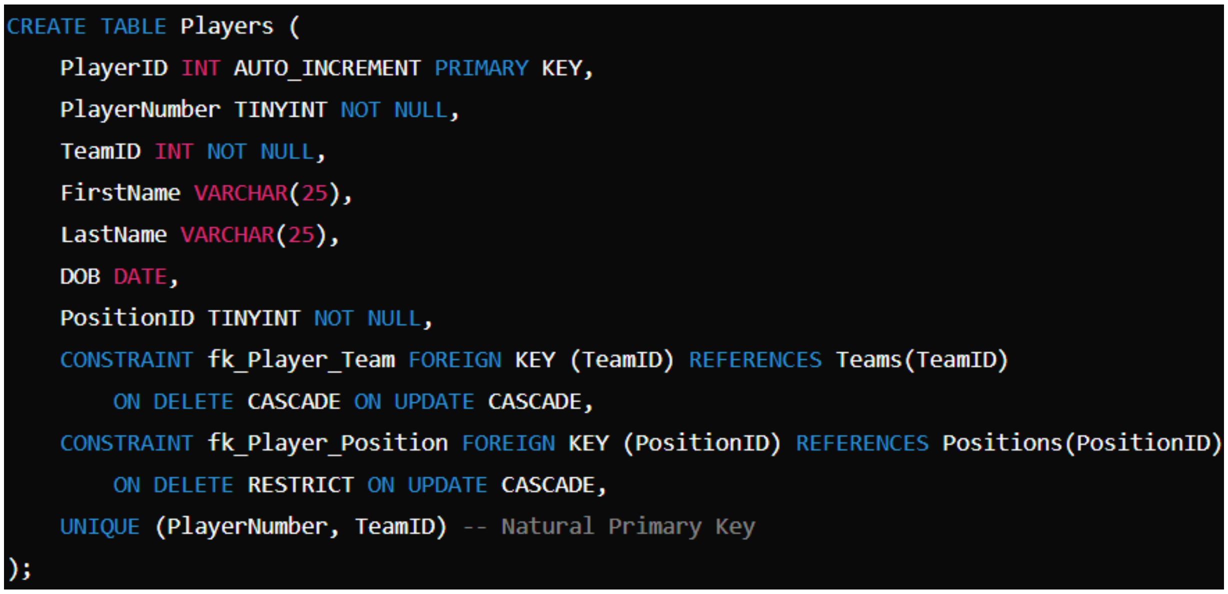
1. MySQL
2. MySQL Workbench
3. phpMyAdmin

Tables & Relationships

1. **Players** - Includes a natural primary key (e.g., PlayerNumber + TeamID) and fields such as name, date of birth, position (foreign key)
2. **Teams** - Includes an auto-incremented integer primary key (e.g., TeamID) and fields like name, city, and coach
3. **Games** - Contains game-specific data such as date, location (ArenaID foreign key), and participating teams
4. **PlayerStats** - Links players and games, showing stats like points, rebounds, assists, steals (uses PlayerID and GameID as a composite primary key)
5. **Arenas** - Lookup table with ArenaID, name, city, and capacity
6. **Positions** - Lookup table with PositionID (e.g., point guard, shooting guard, small forward, power forward, center)

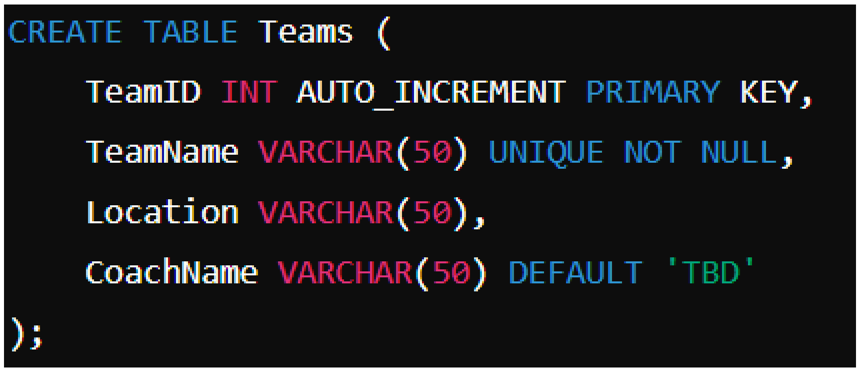
SQL Database Schema Implementation

Players Table



```
CREATE TABLE Players (
    PlayerID INT AUTO_INCREMENT PRIMARY KEY,
    PlayerNumber TINYINT NOT NULL,
    TeamID INT NOT NULL,
    FirstName VARCHAR(25),
    LastName VARCHAR(25),
    DOB DATE,
    PositionID TINYINT NOT NULL,
    CONSTRAINT fk_Player_Team FOREIGN KEY (TeamID) REFERENCES Teams(TeamID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_Player_Position FOREIGN KEY (PositionID) REFERENCES Positions(PositionID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    UNIQUE (PlayerNumber, TeamID) -- Natural Primary Key
);
```

Teams Table



```
CREATE TABLE Teams (
    TeamID INT AUTO_INCREMENT PRIMARY KEY,
    TeamName VARCHAR(50) UNIQUE NOT NULL,
    Location VARCHAR(50),
    CoachName VARCHAR(50) DEFAULT 'TBD'
);
```

Games Table

```
CREATE TABLE Games (
    GameID INT AUTO_INCREMENT PRIMARY KEY,
    GameDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    ArenaID INT NOT NULL,
    Team1ID INT NOT NULL,
    Team2ID INT NOT NULL,
    CONSTRAINT fk_Game_Arena FOREIGN KEY (ArenaID) REFERENCES Arenas(ArenaID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Game_Team1 FOREIGN KEY (Team1ID) REFERENCES Teams(TeamID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_Game_Team2 FOREIGN KEY (Team2ID) REFERENCES Teams(TeamID)
        ON DELETE RESTRICT ON UPDATE CASCADE
);
```

PlayerStats Table

```
CREATE TABLE PlayerStats (
    PlayerID INT NOT NULL,
    GameID INT NOT NULL,
    Points INT DEFAULT 0,
    Rebounds INT DEFAULT 0,
    Assists INT DEFAULT 0,
    PRIMARY KEY (PlayerID, GameID),
    CONSTRAINT fk_Stats_Player FOREIGN KEY (PlayerID) REFERENCES Players(PlayerID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_Stats_Game FOREIGN KEY (GameID) REFERENCES Games(GameID)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

Arenas Table

```
CREATE TABLE Arenas (
    ArenaID INT AUTO_INCREMENT PRIMARY KEY,
    ArenaName VARCHAR(50) UNIQUE NOT NULL,
    Capacity INT DEFAULT 10000
);
```

Positions Table

```
CREATE TABLE Positions (
    PositionID TINYINT AUTO_INCREMENT PRIMARY KEY,
    PositionName VARCHAR(20) UNIQUE NOT NULL
);
```

Creating a View

This SQL view shows player details (PlayerID, Name), the corresponding team, and the games they participated in.

```
CREATE VIEW PlayerGameStats AS
SELECT
    p.PlayerID,
    CONCAT(p.FirstName, ' ', p.LastName) AS PlayerName,
    t.TeamName,
    g.GameDate,
    ps.Points,
    ps.Rebounds,
    ps.Assists
    ps.Steals
FROM
    Players p
JOIN
    Teams t ON p.TeamID = t.TeamID
JOIN
    PlayerStats ps ON p.PlayerID = ps.PlayerID
JOIN
    Games g ON ps.GameID = g.GameID;

```

Creating a Procedure

This SQL procedure inserts a new player into the database and automatically generates a PlayerID.

```sql
DELIMITER $$

CREATE PROCEDURE AddPlayer (
    IN playerNumber INT,
    IN teamID INT,
    IN firstName VARCHAR(50),
    IN lastName VARCHAR(50),
    IN dob DATE,
    IN positionID INT
)
BEGIN
    INSERT INTO Players (PlayerNumber, TeamID, FirstName, LastName, DOB, PositionID)
    VALUES (playerNumber, teamID, firstName, lastName, dob, positionID
END $$

DELIMITER ;
```

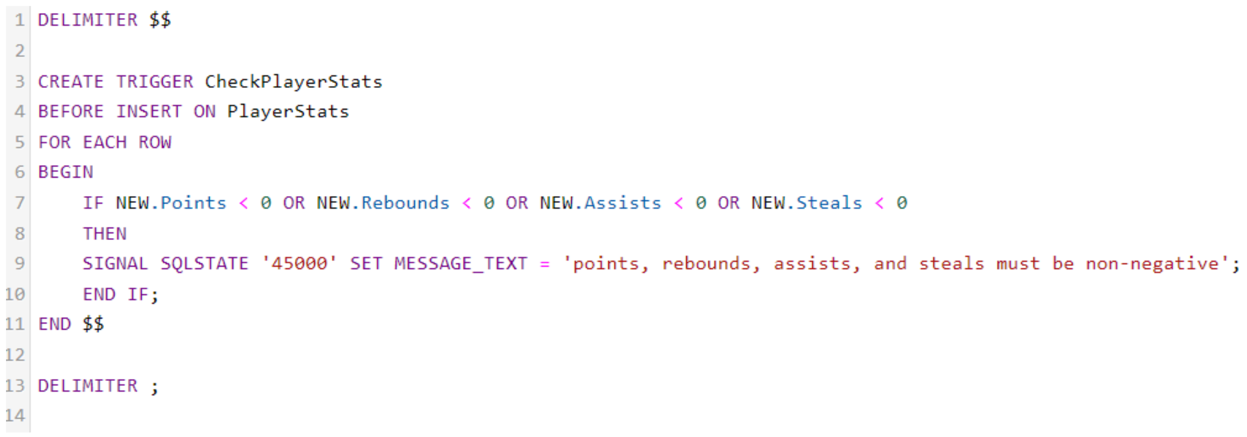
Creating a Function

This SQL function calculates the average points (PPG) of a player.

```sql
BEGIN
    DECLARE avgPoints DECIMAL(5,2);

    SELECT AVG(Points)
    INTO avgPoints
    FROM PlayerStats
    WHERE PlayerStats.PlayerID = playerID;

    RETURN avgPoints;
END
```

Creating a Trigger

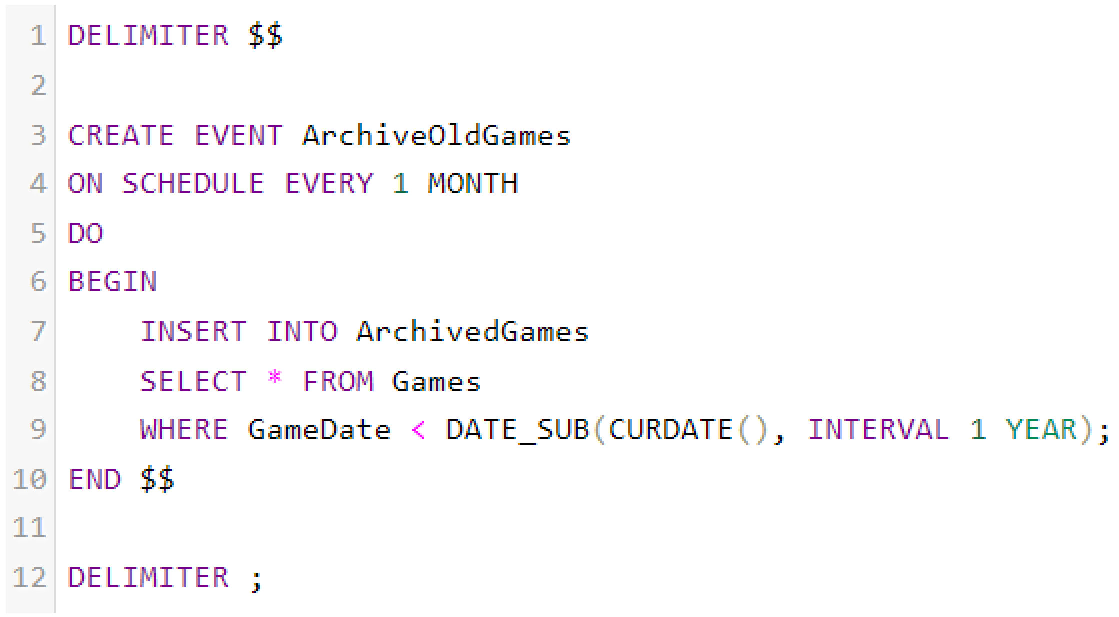
This SQL trigger ensures that points, rebounds, assists, and steals are non-negative when inserting data into PlayerStats.



```sql
DELIMITER $$

CREATE TRIGGER CheckPlayerStats
BEFORE INSERT ON PlayerStats
FOR EACH ROW
BEGIN
    IF NEW.Points < 0 OR NEW.Rebounds < 0 OR NEW.Assists < 0 OR NEW.Steals < 0
    THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'points, rebounds, assists, and steals must be non-negative';
    END IF;
END $$

DELIMITER ;

```

Creating an Event

This SQL event archives games older than one year into an ArchivedGames table.



```sql
DELIMITER $$

CREATE EVENT ArchiveOldGames
ON SCHEDULE EVERY 1 MONTH
DO
BEGIN
    INSERT INTO ArchivedGames
    SELECT * FROM Games
    WHERE GameDate < DATE_SUB(CURDATE(), INTERVAL 1 YEAR);
END $$

DELIMITER ;
```