University Football Database

John Knight

The University of Texas Rio Grande Valley

CYBI-6315 Applied Database Systems

Prof. Jorge Castillo

1 May, 2024

Contents

Introduction

Purpose

The purpose of this database project is to identify a target enterprise, understand the key challenges it faces, and design a database solution tailored to address these issues. This approach will enhance the enterprise's operational efficiency and support its long-term success by providing a scalable and flexible data management system.

Background

In the past 20 years, data-driven decisions have become a crucial part of sports. Perhaps most famously, in baseball, the book and subsequent film *Moneyball*[1] highlighted some of the common misconceptions about game strategy. For instance, teams were overvaluing a player's batting average compared to his on-base percentage, even though the value of a base hit and a walk are very similar.

In basketball, data analytics showed that the expected points from a shot attempt could be maximized by shooting either three-point attempts, or very easy two-point attempts close to the basket. The resulting change in teams' shot locations can be seen in Figure 1.[2]

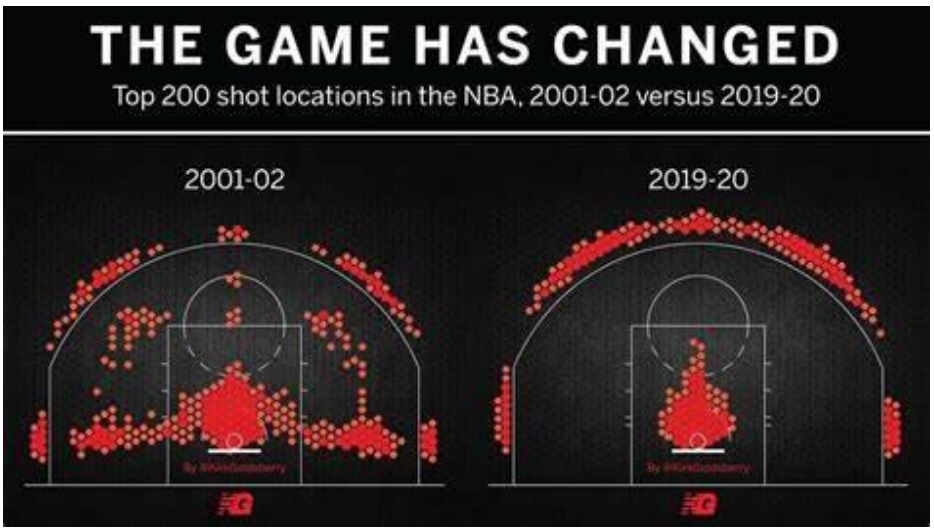

Figure 1. Most common NBA shot locations, 2001-02 compared to 2019-20.

In football, analysts used data and probability to scrutinize decisions made by coaches on fourth down, and generally found them to be too conservative, often opting to punt instead of trying to convert a first down. And even in soccer, one of the hardest sports to apply data due to its lack of discrete states, data has driven some changes to strategy such as a reduction in aerial crosses from the wing.

It is clear that in any modern sports operation, it is essential to have access to well-organized data to help drive decisions relating to recruitment, game strategy, player development, injury recovery, and many other facets of the business.

Motivation

In 2022, the University of Texas Rio Grande Valley announced that it was launching a football program that would begin play in the 2025 season. Coaches and support staff were hired, and in February 2024 the program had added 51 athletes to its football roster, more than a year ahead of its first scheduled game.[3]

This sparked my curiosity about the logistical challenges involved in building a football program from the ground up. It became apparent that a sophisticated database system would be essential for managing such a complex operation. This project presented a perfect opportunity to look into designing a database that can support the dynamic and multi-faceted needs of a collegiate football program.

Goal

The goal of this project is to create a database system for a university football program. The database should contain tables necessary to aid with player recruitment, scouting, tracking statistics, monitoring injuries, and other football-related data.

Methodology

Design approach

MySQL was chosen as the database management system for this project. MySQL is one of the most popular relational database systems in the world, and therefore has a vast community of tools and support resources. MySQL is free and open source, scalable, and integrates well with many programming languages and frameworks.

The three fundamental entities that form the building blocks of a football database are as follows:

- People (players, staff, etc.)
- Teams (could be professional, college, high school, etc.)
- Games (two teams playing against one other)

There are also various entities that describe attributes of the three entities above. For example, a player might sustain an injury; a team has a home stadium; a game is usually played as part of a wider competition.

Entity-Relationship Diagram

To achieve sufficient normalization, 10 entities were created: player, team, game, staff_member, note, injury, stadium, city, competition, and stat_line. Most of these entities are self-explanatory. A *stat_line* is a set of statistics pertaining to one player in one game. A *note* is a note about a player added by any member of staff (for example, it could be some observations written by a scout who has watched the player). An *injury* contains details of a player injury and includes a description of the injury as well as an estimated return date, which can be updated if the player's recovery is behind or ahead of schedule.

The *city* entity was originally an attribute within *stadium*. However, since *city* includes a compound value such as "Dallas, TX" it was given its own entity, else the database would not be considered normalized. In

an expanded database, more information could be added to *city* such as the country, the population, and the city's longitudinal and latitudinal coordinates.

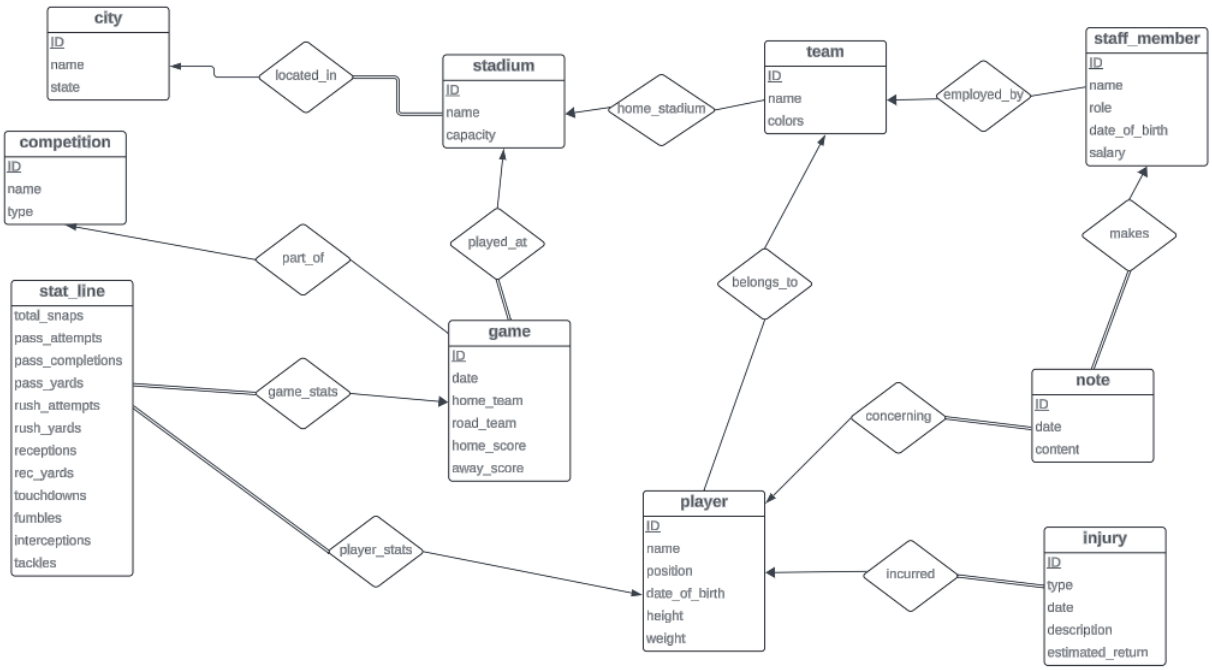

Figure 2. Entity-relationship diagram for university football database.

Relational Schema

In the relational schema, a solid underline denotes the primary key. A dashed underline represents a foreign key. Note that in the *stat_line* table, both game_ID and player_ID together form the primary key, since there can only be one row that represents a single player in a single game. Game_ID and player_ID are both also foreign keys, from the *game* and *player* tables respectively.

```
player (player_ID, first_name, last_name, date_of_birth, height, weight, team_ID)
staff_member (staff_ID, first_name, last_name, date_of_birth, role, salary, team_ID)
team (team_ID, city_name, nickname, color_primary, color_secondary, home_stadium)
game (game_ID, date, team_1, team_2, team_1_score, team_2_score, stadium)
stadium (stadium_ID, name, city, capacity)
competition (competition_ID, name, type)
city (city_ID, name, state)
note (note_ID, added_by, player_ID, date, content)
injury (injury_ID, type, date, description, estimated_return_date, player_ID)
stat_line (game_ID, player_ID, total_snaps, pass_attempts, pass_completions,
pass_yards, rush_attempts, rush_yards, receptions, rec_yards, touchdowns, fumbles,
interceptions, tackles)
```

Access Roles

It is important that each user of the database has a level of access that reflects their role within the organization. Some data may be sensitive, and access should be restricted only to appropriate parties. Similarly, some database users may have limited technical proficiency and their ability to update or delete data should be restricted to prevent causing irreparable harm to the data.

In the original design, seven roles were created:

- Database Administrator
- Director (the person in charge of the football program)
- Analyst
- HR (Human Resources personnel)
- Coach
- Scout
- Player

Upon review, it was decided that these seven roles were perhaps overly specific, and could be consolidated into five. The role of Director was absorbed into Analyst, and the role of Scout was combined with Coach.

The SQL code generating the five access roles can be seen in Figures 3 and 4. The *DatabaseAdministrator* role is granted all privileges. The *Analyst* is allowed to select and insert, as well as being able to update data as it would be time-consuming to contact the administrator every time a small error needed to be corrected.

The *HR* role is purely for personnel in charge of employees, and only concerns the *staff_member* and *player* tables. People with *Coach* access are given the ability to select and insert data, but not to update existing data. Finally, the *Player* role is restricted to looking up information related to games, stats, or injuries (note that players are not allowed to look at notes made by staff members about themselves or other players).

```
-- Access Roles
CREATE ROLE 'DatabaseAdministrator';
GRANT ALL PRIVILEGES ON football_data.* TO 'DatabaseAdministrator';

CREATE ROLE 'Analyst'; -- Also the Director gets this role
GRANT SELECT, INSERT, UPDATE ON football_data.* TO 'Analyst';

CREATE ROLE 'HR';
GRANT SELECT, INSERT, UPDATE ON football_data.staff_member TO 'HR';
GRANT SELECT, INSERT, UPDATE ON football_data.player TO 'HR';
```

Figure 3. Access roles for DatabaseAdministrator, Analyst, and HR.

```
CREATE ROLE 'Coach'; -- Also Scout gets this role
GRANT SELECT, INSERT ON football_data.player TO 'Coach';
GRANT SELECT, INSERT ON football_data.game TO 'Coach';
GRANT SELECT, INSERT ON football_data.stat_line TO 'Coach';
GRANT SELECT, INSERT ON football_data.note TO 'Coach';
GRANT SELECT, INSERT, UPDATE ON football_data.injury TO 'Coach';
GRANT SELECT ON football_data.stadium TO 'Coach';
GRANT SELECT ON football_data.competition TO 'Coach';
GRANT SELECT ON football_data.city TO 'Coach';
GRANT SELECT ON football_data.team TO 'Coach';

CREATE ROLE 'Player';
GRANT SELECT ON football_data.player TO 'Player';
GRANT SELECT ON football_data.game TO 'Player';
GRANT SELECT ON football_data.stat_line TO 'Player';
GRANT SELECT ON football_data.injury TO 'Player';
```

Figure 4. Access roles for Coach and Player.

In order to populate the database with some initial dummy data for testing, the large language model (LLM) ChatGPT was asked to generate appropriate names and attributes to fill the various tables. An example can be seen in Figure 5, where ChatGPT has generated the names of 20 fictitious competitions and their competition type.

```
-- Competitions
INSERT INTO competition (competition_ID, name, type) VALUES (1, 'The Southern Conference', 'College Division I');
INSERT INTO competition (competition_ID, name, type) VALUES (2, 'The Northern League', 'Professional');
INSERT INTO competition (competition_ID, name, type) VALUES (3, 'Eastern Champions Cup', 'College Division II');
INSERT INTO competition (competition_ID, name, type) VALUES (4, 'Western Division Clash', 'College Division II');
INSERT INTO competition (competition_ID, name, type) VALUES (5, 'Central Tournament Series', 'Amateur');
INSERT INTO competition (competition_ID, name, type) VALUES (6, 'National Football League', 'Amateur');
INSERT INTO competition (competition_ID, name, type) VALUES (7, 'Pacific Conference', 'College Division I');
INSERT INTO competition (competition_ID, name, type) VALUES (8, 'Atlantic Bowl', 'Professional');
INSERT INTO competition (competition_ID, name, type) VALUES (9, 'Mountain Ridge Tournament', 'College Division I');
INSERT INTO competition (competition_ID, name, type) VALUES (10, 'Great Lakes League', 'High School');
INSERT INTO competition (competition_ID, name, type) VALUES (11, 'Southwest Football Series', 'Amateur');
INSERT INTO competition (competition_ID, name, type) VALUES (12, 'Northeast Championship', 'Professional');
INSERT INTO competition (competition_ID, name, type) VALUES (13, 'Midwest Football Classic', 'Professional');
INSERT INTO competition (competition_ID, name, type) VALUES (14, 'Sunbelt Bowl', 'College Division I');
INSERT INTO competition (competition_ID, name, type) VALUES (15, 'Iron Gate Series', 'Amateur');
INSERT INTO competition (competition_ID, name, type) VALUES (16, 'Golden State Football Clash', 'Professional');
INSERT INTO competition (competition_ID, name, type) VALUES (17, 'Liberty Football League', 'College Division I');
INSERT INTO competition (competition_ID, name, type) VALUES (18, 'Pioneer Bowl', 'High School');
INSERT INTO competition (competition_ID, name, type) VALUES (19, 'Capital Division Series', 'Professional');
INSERT INTO competition (competition_ID, name, type) VALUES (20, 'Star Valley Championship', 'Professional');
```

Figure 5. Synthetic competition data created with help of LLM.

Results & Discussion

Evaluation using test queries

Three test queries were written to replicate some common database usage and ensure the results were as expected.

The first query looks up the rushing stats for all players who had at least one rushing attempt in a particular game. This query was chosen to replicate the type of query a coach or analyst might make when analyzing player performance. Three tables are joined in this query (stat_line, player, and team).

```
-- Query 1
-- View players who had at least one rush attempt in game 1
SELECT P.player_ID, first_name, last_name, CONCAT(T.primary_name, ' ', T.nickname) AS team_name,
position, rush_attempts, rush_yards, touchdowns FROM stat_line S
JOIN player P ON P.player_ID = S.player_ID
JOIN team T ON T.team_ID = P.team_ID
WHERE S.game_id = 1 AND S.rush_attempts > 0
ORDER BY S.rush_yards DESC;
```

Figure 6. Query to select players who had at least one rush attempt in game_id 1.

The results correctly return the six players who had at least one rushing attempt in game_id 1. For the team name, the CONCAT command is used to concatenate the primary_name and nickname attributes into a single attribute called team_name. The results have been sorted by rush_yards in descending order.

player_ID	first_name	last_name	team_name	position	rush_attempts	rush_yards	touchdowns
1	Brandon	White	Franklin Fire Breathers	Running Back	25	110	2
8	Brian	Robinson	Franklin Fire Breathers	Running Back	11	81	0
3	John	Doe	Franklin Fire Breathers	Halfback	8	28	0
17	Kimberly	Juarez	Franklin Fire Breathers	Quarterback	3	26	1
6	Kimberly	Jackson	Franklin Fire Breathers	Fullback	3	8	0
19	Emily	Harris	Franklin Fire Breathers	Fullback	1	3	0

Figure 7. Results of query 1.

The second query looks up the notes for a particular player, then adds a new note, and finally shows the new results. This query was chosen to demonstrate how the *note* table works. Four tables are joined in this query (note, player, team, and staff_member).

```
-- Query 2
-- Add a player note
SELECT P.player_ID, P.first_name, P.last_name, CONCAT(T.primary_name, ' ', T.nickname) AS team_name,
note_date, CONCAT(S.first_name, ' ', S.last_name) AS added_by, content
FROM note N
JOIN player P ON P.player_ID = N.player_ID
JOIN team T ON P.team_ID = T.team_ID
JOIN staff_member S ON S.staff_ID = N.added_by
WHERE P.player_ID = 13;
```

Figure 8. Query to view all notes for player_ID = 13.

player_ID	first_name	last_name	team_name	note_date	added_by	content
13	Jason	Allen	Franklin Fire Breathers	2024-01-09	Sarah Mata	Demonstrates good team spirit and cooperation with teammates.

Figure 9. Results of the previous query showing one note that fits the criteria.

```
INSERT INTO note (note_ID, added_by, player_ID, note_date, content)
VALUES (21, 8, 13, '2024-03-12', 'Player was arrested for DUI.');
```

Figure 10. Query adding a new note for player 13.

player_ID	first_name	last_name	team_name	note_date	added_by	content
13	Jason	Allen	Franklin Fire Breathers	2024-01-09	Sarah Mata	Demonstrates good team spirit and cooperation with teammates.
13	Jason	Allen	Franklin Fire Breathers	2024-03-12	James Woods	Player was arrested for DUI.

Figure 11. The results of the original query now return two rows, including the new note.

After the initial query (Figure 8) there is only a single note in the results, which reads “Demonstrates good team spirit and cooperation with teammates.” When the subsequent note is added using the INSERT INTO command, and the initial query is run again, there are now two results (Figure 11), with the second note reading “Player was arrested for DUI.”

The final test query first shows all the stadiums in Florida, then gives the count and average capacity for those stadiums. This query was chosen to demonstrate a slightly more advanced aggregation query.

Firstly, a simple SELECT statement was run so that all Florida stadiums could be viewed. Two tables are joined in this query (stadium and city).

```
-- Query 3
-- Find the average capacity for stadiums in Florida
-- First list the Florida stadiums
SELECT S.name AS stadium_name, C.name AS city_name, C.state, S.capacity
FROM stadium S JOIN city C ON C.city_ID = S.city_ID
WHERE C.state = 'FL';
```

Figure 12. Query to show all stadiums in Florida.

stadium_name	city_name	state	capacity
The Coliseum	Madison	FL	80532
Bulls Pen	Lennyton	FL	91795
Mammoths Icepark	Bilton	FL	57972
Eagles Lair	Fairview	FL	50600

Figure 13. Results of the query showing there are four stadiums in Florida in the database.

Next, the COUNT and AVG commands were used to get the count and mean capacity of stadiums fitting the criteria (state = 'FL'). These are given the aliases florida_stadiums and average_capacity. We can see that the results tally with what would be expected from the initial results in Figure 13.

```
-- Now get the count and average to 2 d.p.
SELECT COUNT(*) AS florida_stadiums, ROUND(AVG(S.capacity), 2) AS average_capacity
FROM stadium S JOIN city C ON C.city_ID = S.city_ID
WHERE C.state = 'FL';
```

Figure 14. This query asks for the count and mean of stadiums in Florida.

florida_stadiums	average_capacity
4	70224.75

Figure 15. The SQL output correctly shows 4 stadiums with an average capacity of 70,224.75.

Technical challenges

One challenge encountered was the order in which tables were created in MySQL. Most tables contain foreign keys, and if a table is created where a foreign key is defined for a table that does not yet exist, an error is returned. Therefore, it is necessary to create the tables in such an order that prevents this from happening.

```sql
CREATE DATABASE football_data;
USE football_data;

CREATE TABLE city (
    city_ID INT PRIMARY KEY,
    name VARCHAR(50),
    state CHAR(2)
);

CREATE TABLE stadium (
    stadium_ID INT PRIMARY KEY,
    name VARCHAR(50),
    city_ID INT,
    capacity INT,
    FOREIGN KEY (city_ID) REFERENCES city(city_ID)
);
```

Figure 16. SQL code to create the database and the first two tables.

Additionally, I originally considered creating the database using SQLITE embedded in Python. However, while coding the database I discovered that SQLITE does not support access roles, and so I decided to switch from SQLITE to MySQL.

Limitations

As described in the Methodology section, a LLM was utilized to create toy examples for testing purposes. This is fine for small amounts of synthetic data, but for real world data (players, teams, games, etc.) a different approach would be required. While there may be existing data available from various sources such as sports websites, it would be extremely time-consuming to manually enter all the required data. Additionally, as new data is created – for example every time a game is played – it would be useful for the organization to have the ability to incorporate the new data into its analysis as soon as possible.

Potential solutions

To address the limitations associated with manual data entry and the need for real-time data integration, automating the data acquisition process through web scraping and establishing a robust data pipeline could be highly effective. By utilizing web scraping techniques, data can be extracted systematically from online sources such as sports websites and APIs that provide real-time statistics on players, teams, and game results. This data can then be cleaned and transformed to fit the schema of the MySQL database. Additionally, setting up a data pipeline using tools like Apache Kafka or AWS Data Pipeline would enable the continuous flow of data into the database. This setup not only minimizes human error but also ensures that the database is always up-to-date, allowing for immediate analysis post-game. Implementing such solutions would significantly enhance the database's capability to handle large volumes of dynamic sports data efficiently.

Conclusion

This project successfully provides a framework for a potentially more complex football database. MySQL provided the ideal choice of database management system, not only for the relatively small database used in this project, but as a framework for a much larger database as the project is scaled to that of a real world football operation.

The three test queries all ran successfully, showcasing some common operations: viewing player stats, viewing and adding player notes, and calculating summary statistics. Of course, for a larger scale database, many more queries of a more complex nature would be required.

One takeaway from this project was the importance of thorough planning before the deployment of a database. It was demonstrated that detailed architectural designs and pre-defined data management strategies are crucial for the successful implementation and scalability of the database. Furthermore, it was learned that anticipatory measures in schema design and infrastructure can significantly mitigate potential issues post-deployment.

Appendix

References

1. Lewis, M. (2003). *Moneyball: The art of winning an unfair game.* W.W. Norton & Company.

2. Goldsberry, K. (2020, January 14). *The game has changed.* Twitter. https://twitter.com/kirkgoldsberry/status/1217109175894831105

3. UTRGV Athletics. (2024, February 7). *Football adds 41 vaqueros for 2024*. https://goutrgv.com/news/2024/2/7/football-adds-41-vaqueros-for-2024.aspx