**Introduction**

**Sports Tournament Management System (STMS) is a database application, dedicated to simplifying and optimizing the organization and management of sports tournaments. In this report, we focused on implementing our database design, populating the database and optimizing the performance.  We also added a script to create the database from scratch and commonly used queries.**

**DATABASE CHOICES, CHALLENGES, LESSONS LEARNED**

**DATABASE CHOICES**

For the implementation of STMS, we selected MySQL as the Database Management System. The reasons for this decision are listed as below.

1. **Open Source:** The Community Edition of MySQL is free to use for anybody. This makes MySQL a great choice for teams that do not want to pay money for their project. The source code of MySQL is avaible to view and modify. This allows companies to modify the source code according to their needs.
2. **Easy to Use:** MySQL is known to be an easy language, which helps beginners to learn and experienced people to adapt. MySQL offers user friendly tools and it simplifies the database implementation process
3. **Secure:** MySQL offers features for data encryption to ensure data safety. It has role based access control to ensure safety among teams. It also offers secure connection through SSL which helps to defend against outsider attacks.
4. **High Performance:** MySQL is a great choice for performance. MySQL offers a powerful query optimizer, indexing support and built-in caching.
5. **Industry Standard:** MySQL is widely used among most tech companies. Which allows MySQL to have a large community behind it.

**CHALLENGES**

Must provide a database management system that can keep track of different sports, with different referee counts and different participating team count in a match.

metin, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

More than 2 teams may participate in a single match, depending on the sport type. For example, for football 2 teams can participate in a match but for cycling or pubg team sizes may differ.

A team’s coach may change over time and a coach may instruct other over time, through the team\_coached table this can be seen.

metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

A coach may instruct different teams over time.

metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

A team may be instructed by other coaches over time.

**LESSONS LEARNED**

* Using technologies like Flask, PyMySQL, aiohttp, tkinter.
* The relationship between API and Client.
* Designing a backend for a database system.
* Handling synchronized API requests.
* Generating data for database population purposes.
* It is crutial to come up with a good design at first place, updating a design on the go is very costly.

**DATABASE INITIALIZATION SCRIPT**

DROP DATABASE IF EXISTS STMS;

CREATE DATABASE STMS;

USE STMS;

*-- Sports Table*

CREATE TABLE Sports (

    sport\_id INT AUTO\_INCREMENT PRIMARY KEY,

    name VARCHAR(255) NOT NULL,

    description TEXT,

    rules JSON

);

*-- Tournaments Table*

CREATE TABLE Tournaments (

    tournament\_id INT AUTO\_INCREMENT PRIMARY KEY,

    name VARCHAR(255) NOT NULL,

    sport\_id INT,

    start\_date DATE,

    end\_date DATE,

    location VARCHAR(255),

    FOREIGN KEY (sport\_id) REFERENCES Sports(sport\_id) ON DELETE CASCADE

);

*-- Coaches Table*

CREATE TABLE Coaches (

    coach\_id INT AUTO\_INCREMENT PRIMARY KEY,

    first\_name VARCHAR(255) NOT NULL,

    last\_name VARCHAR(255) NOT NULL,

    experience\_years INT

);

*-- Teams Table*

CREATE TABLE Teams (

    team\_id INT AUTO\_INCREMENT PRIMARY KEY,

    coach INT,

    name VARCHAR(255) NOT NULL,

    founded\_year INT,

    FOREIGN KEY (coach) REFERENCES Coaches(coach\_id) ON DELETE CASCADE

);

*-- Players Table*

CREATE TABLE Players (

    player\_id INT AUTO\_INCREMENT PRIMARY KEY,

    first\_name VARCHAR(255) NOT NULL,

    last\_name VARCHAR(255) NOT NULL,

    date\_of\_birth DATE,

    country\_of\_origin VARCHAR(255),

    age INT,

    market\_value DECIMAL(15, 2),

    salary DECIMAL(15, 2),

    contract\_start\_date DATE,

    contract\_end\_date DATE,

    accuracy DECIMAL(5,2),

    height DECIMAL(5, 2),

    weight DECIMAL(5, 2),

    team\_captain BOOLEAN,

    experience\_years INT,

    manager\_name VARCHAR(255),

    total\_minutes\_played INT,

    matches\_played INT,

    team\_id INT,

    FOREIGN KEY (team\_id) REFERENCES Teams(team\_id) ON DELETE CASCADE

);

*-- Matches Table*

CREATE TABLE Matches (

    match\_id INT AUTO\_INCREMENT PRIMARY KEY,

    tournament\_id INT,

    match\_date DATE,

    location VARCHAR(255),

    teams\_result JSON,

    FOREIGN KEY (tournament\_id) REFERENCES Tournaments(tournament\_id) ON DELETE CASCADE

);

*-- Referees Table*

CREATE TABLE Referees (

    referee\_id INT AUTO\_INCREMENT PRIMARY KEY,

    first\_name VARCHAR(255) NOT NULL,

    last\_name VARCHAR(255) NOT NULL,

    experience\_years INT

);

*-- Team\_Coached Table*

CREATE TABLE Team\_Coached (

    coach\_id INT,

    team\_id INT,

    coaching\_begin\_date DATE,

    coaching\_end\_date DATE,

    PRIMARY KEY (coach\_id, team\_id),

    FOREIGN KEY (coach\_id) REFERENCES Coaches(coach\_id) ON DELETE CASCADE,

    FOREIGN KEY (team\_id) REFERENCES Teams(team\_id) ON DELETE CASCADE

);

*-- Team\_Tournament\_Participation Table*

CREATE TABLE Team\_Tournament\_Participation (

    team\_id INT,

    tournament\_id INT,

    PRIMARY KEY (team\_id, tournament\_id),

    FOREIGN KEY (team\_id) REFERENCES Teams(team\_id) ON DELETE CASCADE,

    FOREIGN KEY (tournament\_id) REFERENCES Tournaments(tournament\_id) ON DELETE CASCADE

);

*-- Team\_Match\_Participation Table*

CREATE TABLE Team\_Match\_Participation (

    team\_id INT,

    match\_id INT,

    PRIMARY KEY (team\_id, match\_id),

    FOREIGN KEY (team\_id) REFERENCES Teams(team\_id) ON DELETE CASCADE,

    FOREIGN KEY (match\_id) REFERENCES Matches(match\_id) ON DELETE CASCADE

);

*-- Referees\_in\_Match Table*

CREATE TABLE Referees\_in\_Match (

    referee\_id INT,

    match\_id INT,

    PRIMARY KEY (referee\_id, match\_id),

    FOREIGN KEY (referee\_id) REFERENCES Referees(referee\_id) ON DELETE CASCADE,

    FOREIGN KEY (match\_id) REFERENCES Matches(match\_id) ON DELETE CASCADE

);

CREATE INDEX idx\_team\_id on Team\_Tournament\_Participation (team\_id);

CREATE INDEX idx\_team\_match\_id on Team\_Match\_Participation (team\_id);

*-- Sports Table*

CREATE INDEX idx\_sports\_name ON Sports(name);

*-- Tournaments Table*

CREATE INDEX idx\_tournaments\_sport\_id ON Tournaments(sport\_id);

CREATE INDEX idx\_tournaments\_location ON Tournaments(location);

*-- Coaches Table*

CREATE INDEX idx\_coaches\_name ON Coaches(first\_name, last\_name);

*-- Teams Table*

CREATE INDEX idx\_teams\_coach ON Teams(coach);

*-- Players Table*

CREATE INDEX idx\_players\_team\_id ON Players(team\_id);

CREATE INDEX idx\_players\_dob ON Players(date\_of\_birth);

*-- Matches Table*

CREATE INDEX idx\_matches\_tournament\_id ON Matches(tournament\_id);

CREATE INDEX idx\_matches\_date ON Matches(match\_date);

*-- Referees Table*

CREATE INDEX idx\_referees\_experience\_years ON Referees(experience\_years);

*-- Team\_Tournament\_Participation Table*

CREATE INDEX idx\_tournament\_id ON Team\_Tournament\_Participation(tournament\_id);

*-- Team\_Match\_Participation Table*

CREATE INDEX idx\_match\_id ON Team\_Match\_Participation(match\_id);

**DATABASE POPULATION**

For our database design to be as realistic as possible, we have spent a great amount of effort on the population script. We are using a python script to populate the tables through an object oriented point of view.

Firstly sports are created each which specific set of rules stored in unique JSON files. Then tournaments of certain sport types are created. Then teams of certain sport types are created and are placed in these tournaments. The team creation and addition is randomized in order to realistically model, a team (say a football team) join more the 1 tournaments (may not join a tournament, may join all the football tournaments, may join only some of them).

Then players are generated again using the “randomness” we are creating and placing players in order to preserve the notion that “teams may change players over time”. Then players Each tournament is given a realistic location in order to better create the realism. For example, online pbug tournaments can only be played in the location in “Internet”.

The participation of teams in matches is logged, avoiding duplicates, and referees are assigned to matches based on the sport type and required count.

More than 2 teams may participate in a single match, depending on the sport type. For example, for football 2 teams can participate in a match but for cycling or pubg team sizes may differ. This constraint is handled by the population code.

The entire process is carefully structured to simulate a fully populated sports tournament system, with teams, players, referees, matches, and coaches linked together effectively.

All of these numbers are applicable to changes

num\_tournaments = 30

num\_teams\_total = 200

num\_coaches = num\_teams\_total

num\_referees = 20

num players: differs due to randomization. Currently 8988 palyers exist.

The code allows for any number individual of teams,tournaments,referees,coaches and consequently players to be generated and harmonizes them together to provide maximum efficiency.

We have added many player statistics and attributes in order to better present each player.

metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**COMMON QUERIES**

**A Total of 30 common queries are given below with explanations.**

1. **SELECT \* FROM players;**

**Is used to view info regarding players, includes first and last name, date\_of\_birth and team\_id**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT distinct name FROM Sports;**

**Retrieve all different sport names**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT name FROM Tournaments**

**WHERE sport\_id = (**

**SELECT sport\_id FROM Sports**

**WHERE name = 'basketball');**

**SELECT all tournaments that are of type basketball. Here different sport types may be written to get the desired output ( change name = “X” in the subquery.**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT \* FROM Coaches;**

**Get the details of all coaches**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT \* FROM Sports WHERE name = 'basketball';**

**Get the details of a specific sport (e.g., basketball)**

**metin, elektronik donanım, ekran görüntüsü, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT team\_id, COUNT(\*) AS num\_players FROM Players GROUP BY team\_id;**

**Find number of players in each team**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT sport\_id, COUNT(\*) AS num\_tournaments FROM Tournaments GROUP BY sport\_id;**

**Get the number of tournaments for each sport.**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT first\_name, last\_name FROM Players WHERE date\_of\_birth < '1993-01-01'; Get the players who are older than 30 years**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT first\_name, last\_name FROM Players WHERE team\_id = (SELECT team\_id FROM Teams WHERE team\_id = '3');**

**Get all players for a specific team (e.g., Team A)**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**If the team\_id = ‘2’ is chosen:**

**metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**This is because different sports compose of different sized teams. And this is a unique feature of our dbms that allows for multiple sports to be shown.**

1. **SELECT match\_id FROM Matches as m**

**WHERE m.tournament\_id =**

**(SELECT tournament\_id FROM**

**tournaments as t WHERE t.name = 'Tournament 2');**

**List all teams that have participated in a specific tournament**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT name FROM Tournaments WHERE location = 'Sivas;**

**Get all tournaments held in a specific location**

**metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**If location = ‘Internet’:**

**metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**This is a feature that allows not only physical sports but also internet based sports to be accurately stored.**

1. **SELECT name FROM Teams WHERE coach = (SELECT coach\_id FROM Coaches WHERE first\_name = "Suzann" AND last\_name = "Robbins");**

**Find teams that Suzann Robbins coaches.**

**metin, yazı tipi, sayı, numara, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT first\_name, last\_name FROM Referees WHERE experience\_years > 5;**

**Get all the referees who have more than 5 years of experience.**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT \* FROM tournaments, sports WHERE tournaments.sport\_id = sports.sport\_id ORDER BY tournament\_id ASC;**

**View all tournaments and sports, seeing what the sport type of a tournament is.**

**metin, yazı tipi, sayı, numara, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT player\_id, first\_name, last\_name FROM Players WHERE age > 25 AND market\_value > 10000;**

**Retrieve all players older than 25 with a market value greater than 10000.**

1. **SELECT name FROM Sports WHERE sport\_id IN (SELECT sport\_id FROM Tournaments WHERE start\_date BETWEEN '2023-01-01' AND '2023-12-31');**

**Retrieve sports for tournaments that are held in the year 2023.**

1. **SELECT COUNT(\*) AS total\_matches FROM Matches WHERE match\_date < '2024-11-01';**

**Count matches before a certain date.**

**metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT team\_id, name FROM Teams WHERE team\_id NOT IN (SELECT team\_id FROM Team\_Tournament\_Participation WHERE tournament\_id = 5);**

**Retrieve teams that did not participate in a specific tournament (in this case tournament ID = 5).**

1. **SELECT referee\_id, first\_name, last\_name FROM Referees WHERE referee\_id NOT IN (SELECT referee\_id FROM Referees\_in\_Match WHERE match\_id = 3);**

**Retrieve referees who did not officiate in a specific match.**

1. **SELECT DISTINCT location, matches.match\_id FROM Matches WHERE match\_date BETWEEN '2020-11-01' AND '2024-11-30';**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**View locations of matches that are played in a certain date interval.**

1. **SELECT tournament\_id, COUNT(\*) AS num\_matches FROM Matches GROUP BY tournament\_id;**

**Retrieve the number of matches for each tournament.**

1. **SELECT team\_id, SUM(total\_minutes\_played) AS total\_played\_minutes FROM Players GROUP BY team\_id;**

**Retrieve the total minutes each player played in each team.**

1. **SELECT team\_id, name FROM Teams WHERE founded\_year < 2000;**

**Retrieve teams founded before the year 2000.**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT team\_id, name FROM Teams WHERE coach IN (SELECT coach\_id FROM Coaches WHERE experience\_years > 10);**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT team\_id, AVG(salary) AS average\_salary FROM Players GROUP BY team\_id;**

**Retrieve the average salary of players for each team.**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT referee\_id, COUNT(\*) AS total\_matches FROM Referees\_in\_Match GROUP BY referee\_id;**

**Retrieve the total number of matches officiated by each referee.**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT player\_id, first\_name, last\_name FROM Players WHERE team\_captain = TRUE;**

**Retrieve the players who are team captains.**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT COUNT(p.team\_id) FROM players as p, teams as t**

**WHERE p.team\_id = t.team\_id GROUP BY p.team\_id;**

**Count how many players each team has.**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT player\_id, first\_name, last\_name FROM Players WHERE team\_id IN (SELECT team\_id FROM Teams WHERE coach IN (SELECT coach\_id FROM Coaches WHERE first\_name = 'Beckie' AND last\_name = 'Calhoun'));**

**Find all players coached by a specific coach. metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

1. **SELECT team\_id, COUNT(DISTINCT player\_id) AS total\_players FROM Players GROUP BY team\_id HAVING total\_players <5;**

**SELECT teams that have less than 5 players.**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**PERFORMANCE**

The STMS database is read-heavy database, hence indices will generally result in improved performance as long as the indexed columns are used. During this project we have aimed to create balance between optimizing read performance and avoiding excessive overhead on write operations. We have indexed columns that were used the most by the select queries which helped to save time. However these indices have caused slight delays with queries where in addition to the indexed columns, other columns had to be retrieved too.

**Tuning tables, queries and Choice of Indices:**

We haveadded indices to columns that are frequently used in filtering, joining, or sorting operations. For example, indexing the team\_id, sport\_id, and date\_of\_birth as these columns are commonly used in queries with WHERE, JOIN, or GROUP BY clauses were an initial decision we have concluded on. We have also indexed coach names and surnames because they were also very widely used.

For example, queries like SELECT team\_id, name FROM Teams WHERE team\_id NOT IN (SELECT team\_id FROM Team\_Tournament\_Participation WHERE tournament\_id = 5); suffered from slight performance degradation, as the index has to be maintained and accessed for each subquery condition and not indexed name attribute from teams was used in addition to the indexed team\_id in teams.

**Impact of changes and how the tests are concluded:**

30 common queries are presented. All of these queries are tested on the database populated with the same data on indexed and not indexed versions. The results reveal that most of the queries are positively affected by the indices. However there were certain occurrences, where the created overhead was seen to be more costly, hence not every query’s runtime increased.

The python file “performance\_test.py”, runs all of the common queries one by one on the database, and records each query’s runtime in a file. This file is run 2 times, once with the database initialization script with no indices and once with the database script with indices. Each individual query result is presented in “query\_times\_no\_indices.txt” and “query\_times\_with\_indices.txt”.

Then another python script “performance\_test\_print.py”, calculates the time improvement between each query in the indiced and non- indiced format. These results are held in the file “time\_imporvement\_with\_indices.txt”. In this file, negative percentage increase’s correlate that the query was indeed ran faster. First 5 queries are also presented below in order to better showcase the logic:

By performing these experiments, we can assess which indices provide the most significant improvements and which types of queries benefit the most. For instance, queries that involve filtering, sorting, or joining on indexed columns often show substantial performance gains.

Query: SELECT \* FROM Players;

Time in query\_times\_no\_indices.txt: 0.076545 seconds

Time in query\_times\_with\_indices.txt: 0.067461 seconds

Time Difference: -0.009084 seconds

Percentage Increase: -11.87%

Query: SELECT DISTINCT name FROM Sports;

Time in query\_times\_no\_indices.txt: 0.001313 seconds

Time in query\_times\_with\_indices.txt: 0.000702 seconds

Time Difference: -0.000611 seconds

Percentage Increase: -46.53%

Query: SELECT name FROM Tournaments WHERE sport\_id = (SELECT sport\_id FROM Sports WHERE name = 'basketball');

Time in query\_times\_no\_indices.txt: 0.000847 seconds

Time in query\_times\_with\_indices.txt: 0.000580 seconds

Time Difference: -0.000267 seconds

Percentage Increase: -31.52%

Query: SELECT \* FROM Coaches;

Time in query\_times\_no\_indices.txt: 0.000872 seconds

Time in query\_times\_with\_indices.txt: 0.000705 seconds

Time Difference: -0.000167 seconds

Percentage Increase: -19.15%

Query: SELECT \* FROM Sports WHERE name = 'basketball';

Time in query\_times\_no\_indices.txt: 0.000317 seconds

Time in query\_times\_with\_indices.txt: 0.000282 seconds

Time Difference: -0.000035 seconds

Percentage Increase: -11.04%

Query: SELECT team\_id, COUNT(\*) AS num\_players FROM Players GROUP BY team\_id;

Time in query\_times\_no\_indices.txt: 0.002228 seconds

Time in query\_times\_with\_indices.txt: 0.001822 seconds

Time Difference: -0.000406 seconds

Percentage Increase: -18.22%