

BİL 372

Sports Tournament Management System Database creation, population and Performance report

<i>Utku CEYLAN</i>	<i>221101067</i>
<i>Kıvanç TAŞ</i>	<i>221401022</i>
<i>Ali Berk KARAARSLAN</i>	<i>221401010</i>
<i>Mustafa TUFAN</i>	<i>221401029</i>

Table of Contents

Introduction.....	1
Database Choices, Challenges, Lessons Learned	1
Database Choices	1
Challenges.....	2
Lessons Learned.....	3
Database Initialization Script.....	3
Database Population.....	6
Common Queries	7
Performance	17
Tuning tables, queries and Choice of Indices:	17
Impact of changes and how the tests are concluded:	18

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

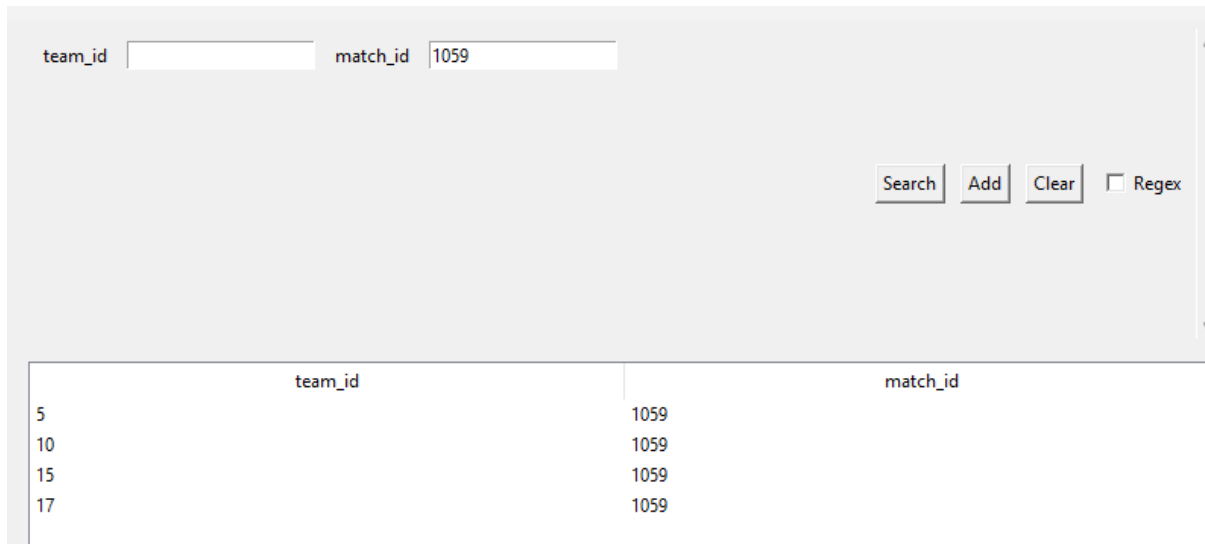
Database Choices

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



team_id	match_id
5	1059
10	1059
15	1059
17	1059

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 others over time; through the team_coached table this can be seen.

coach_id

1

team_id

coaching_begin_date

coaching_end_date

Search

Add

Clear

☐ Regex

coach_id	team_id	coaching_begin_date	coaching_end_date
1	1	Sun, 10 Oct 1954 00:00:00 GMT	Tue, 26 Aug 1958 00:00:00 GMT
1	32	Sat, 15 May 1909 00:00:00 GMT	Sun, 15 Oct 1911 00:00:00 GMT
1	49	Mon, 16 Mar 1964 00:00:00 GMT	Fri, 03 Jan 1969 00:00:00 GMT
1	92	Fri, 05 Mar 1915 00:00:00 GMT	Wed, 02 May 1917 00:00:00 GMT

A coach may instruct different teams over time.

coach_id

team_id

45

coaching_begin_date

coaching_end_date

Edit

Remove

coach_id	team_id	coaching_begin_date	coaching_end_date
32	45	Mon, 06 Feb 1956 00:00:00 GMT	Sat, 17 Oct 1959 00:00:00 GMT
45	45	Sat, 14 Nov 1959 00:00:00 GMT	Thu, 06 Jun 1963 00:00:00 GMT

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 crucial to come up with a good design in the 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 with a 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 are randomized in order to realistically model, a team (say a football team) join more than 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 to better create the realism. For example, online PUBG 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 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 players exist.

The code allows for any 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.

1 • **SELECT COUNT(*) FROM PLAYERS;**

The screenshot shows a database interface with a query bar containing 'COUNT(*)' and a result grid below it. The result grid has a single cell with the value '8988'.

Result Grid
COUNT(*)
8988

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

The screenshot shows a database interface with a query bar containing 'SELECT * FROM players;' and a result grid below it. The result grid displays a list of players with columns: player_id, first_name, last_name, date_of_birth, country_of_origin, age, market_value, salary, contract_start_date, contract_end_date, height, weight, team_captain, experience_years, and manager_name.

player_id	first_name	last_name	date_of_birth	country_of_origin	age	market_value	salary	contract_start_date	contract_end_date	height	weight	team_captain	experience_years	manager_name
1	Randee	Lowery	2002-03-25	Egypt	35	69356.04	9288.45	2010-12-23	2026-12-25	184.00	76.00	1	21	JonellDuarde
2	Ealasaid	Lowery	2002-05-05	Spain	21	93668.78	1406.43	2013-12-21	2030-09-08	174.00	80.00	0	28	GladSexton
3	Joni	Hull	2000-02-27	Mexico	23	88455.20	4930.82	2034-09-02	2035-03-14	192.00	99.00	0	28	JonellHull
4	Becka	Lin	2005-10-21	Canada	28	59403.05	6136.45	2016-02-13	2026-12-11	215.00	66.00	0	10	JordainRobbins
5	Abagail	Sexton	1999-07-06	Spain	35	83985.45	1535.90	2008-05-27	2024-08-02	170.00	92.00	0	8	BeckiJones
6	Eba	Davis	2000-10-16	South Africa	30	18488.12	8854.93	2014-12-16	2023-03-30	181.00	93.00	0	8	AbagailSmith
7	Merlina	Dennis	1994-05-08	South Africa	33	57271.43	2356.42	2018-01-18	2024-10-01	175.00	90.00	0	23	SusyCummings
8	Quel	Mrlaan	1008-00-26	France	77	83410.81	2578.86	2031-08-21	2035-00-18	174.00	70.00	0	26	JonellSmith

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

Retrieve all different sport names

	name
▶	baseball
	tennis
	basketball
	soccer
	pubg E-tournament
	rugby
	cricket
	golf

```
3. SELECT name FROM Tournaments
WHERE sport_id = (
SELECT sport_id FROM Sports
WHERE name = 'basketball');
```

SELECT all basketball tournaments. Here different sport types may be written to get the desired output (change name = "X" in the subquery.

	name
▶	Tournament of basketball 6
	Tournament of basketball 13
	Tournament of basketball 15
	Tournament of basketball 23
	Tournament of basketball 29

```
4. SELECT * FROM Coaches;
```









Get the details of all coaches

	coach_id	first_name	last_name	experience_years
▶	1	Beckie	Calhoun	29
	2	Eartha	Duarte	8
	3	Becka	Chung	24
	4	Abbe	Lin	25
	5	Becky	Williams	16
	6	Chrystel	Brown	17
	7	Jonis	Sharp	30
	8	Suzann	Brown	7

5. `SELECT * FROM Sports WHERE name = 'basketball';`

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

```
1  SELECT * FROM Sports WHERE name = 'basketball';
```

Result Grid		 Filter Rows: <input type="text"/>	Edit:   	Export/Import:  	Wrap Cell Content: 
sport_id	name	description	rules		
▶ 2	basketball	Description for basketball	{"general_rules": "Rules for bas..."}		

6. `SELECT team_id, COUNT(*) AS num_players FROM Players GROUP BY team_id;`

Find number of players in each team

team_id	num_players
1	11
2	1
3	15
4	70
5	4
6	4
7	72
8	56

7. `SELECT sport_id, COUNT(*) AS num_tournaments FROM Tournaments GROUP BY sport_id;`

Get the number of tournaments for each sport.

	sport_id	num_tournaments
▶	1	1
	2	4
	3	5
	4	4
	5	2
	6	2
	7	1
	8	2

8. `SELECT first_name, last_name FROM Players WHERE date_of_birth < '1993-01-01';`

Get players who are older than 30 years old

	first_name	last_name
▶	Lillie	Jones
	Abigail	Barber
	Jonis	Dennis
	Aarika	Ochoa
	Giulietta	Aguirre
	Ealasaid	Sloan
	Rana	Gentry
	Becki	Martinez
	Randa	Lowery
	Abigail	Sloan
	Cicily	Davis
	Beckie	Mclean
	Ebba	Hendricks
	Ramona	Hendricks
	Merline	Gentry
	Chryste	Calhoun
	Cicely	Lin
	Becka	Lin
	Randee	Rodriguez
	Lilly	Jones
	Ciel	Martinez

9. `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)

	first_name	last_name
▶	Rana	McLean
	Aarika	Sharp
	Easter	Martinez
	Beckie	Williams
	Suzann	Calhoun
	Ramonda	Robbins
	Merna	Duarte
	Joni	Aguirre

If the team_id = '2' is chosen:

	first_name	last_name
▶	Jonie	Davis

This is because different sports are composed of different sized teams. And this is a unique feature of our DBMS that allows for multiple sports to be shown.

10. `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

	match_id
	15
	16
	17
	18
	19
	20
	21
	22

11. `SELECT name FROM Tournaments WHERE location = 'Sivas';`

Get all tournaments held in a specific location

	name
▶	Tournament of soccer 1
	Tournament of tennis 3
	Tournament of volleyball 4
	Tournament of baseball 11
	Tournament of volleyball 16

If location = 'Internet':

	name
▶	Tournament of Pubg 1
	Tournament of Pubg 2
	Tournament of Pubg 3

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

12. `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.

	name
▶	Team 18
	Team 23
	Team 26

13. `SELECT first_name, last_name FROM Referees WHERE experience_years > 5;`

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

Result Grid				Filter Rows:	
	first_name	last_name			
▶	Susie	Lin			
	Giustina	Garcia			
	Randa	Sexton			
	Abagael	Jones			
	Eba	Rodriguez			
	Gizela	Smith			
	Susette	Sexton			
	Chryste	Gentry			

14. `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.

tournament_id	name	sport_id	start_date	end_date	location	sport_id	name	description	rules
4	Tournament of volleyball	9	1994-09-20	1994-10-04	Sivas	9	volleyball	Description for volleyball	{"general_rules": "Volleyball is played by two teams of six players each"
5	Tournament of soccer	4	2021-12-18	2022-07-21	Madrid	4	soccer	Description for soccer	{"general_rules": "Soccer is played by two teams of eleven players on"
6	Tournament of basketball	3	2017-05-26	2017-05-30	Tokyo	3	basketball	Description for basketball	{"general_rules": "Baseball is played by two teams of nine players each"
7	Tournament of rugby	6	2001-05-05	2001-08-24	Dubai	6	rugby	Description for rugby	{"general_rules": "Rugby is played by two teams of 15 players each or"
8	Tournament of Pubg	6	1985-07-04	1986-01-04	Internet	6	rugby	Description for rugby	{"general_rules": "Rugby is played by two teams of 15 players each or"
9	Tournament of volleyball	9	1996-01-24	1996-09-16	Dubai	9	volleyball	Description for volleyball	{"general_rules": "Volleyball is played by two teams of six players each"
10	Tournament of cycling	10	2004-02-18	2004-07-18	Berlin	10	cycling	Description for cycling	{"general_rules": "Cycling is an individual sport where participants race"
11	Tournament of hocke	1	2016-04-23	2016-05-20	Sivas	1	hocke	Description for hocke	{"general_rules": "Rackethall is played by two teams of five players each"

15. `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.

16. `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.

17. `SELECT COUNT(*) AS total_matches FROM Matches WHERE match_date < '2024-11-01';`

Count matches before a certain date.

	total_matches
▶	1855

18. `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).

19. `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.

20. `SELECT DISTINCT location, matches.match_id FROM Matches WHERE match_date BETWEEN '2020-11-01' AND '2024-11-30';`

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

	location	match_id
	Madrid	211
	Madrid	212
	Madrid	213
	Ankara	824
	Ankara	825
	Ankara	826
	Ankara	827
	Ankara	828

21. `SELECT tournament_id, COUNT(*) AS num_matches FROM Matches GROUP BY tournament_id;`

Retrieve the number of matches for each tournament.

22. `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.

23. `SELECT team_id, name FROM Teams WHERE founded_year < 2000;`

Retrieve teams founded before the year 2000.

	team_id	name
▶	1	Team 1
	2	Team 2
	3	Team 3
	4	Team 4
	7	Team 7
	8	Team 8
	9	Team 9
	10	Team 10

24. `SELECT team_id, name FROM Teams WHERE coach IN (SELECT coach_id FROM Coaches WHERE experience_years > 10);`

	team_id	name
▶	1	Team 1
	3	Team 3
	4	Team 4
	5	Team 5
	6	Team 6
	7	Team 7
	9	Team 9
	10	Team 10

25. `SELECT team_id, AVG(salary) AS average_salary FROM Players GROUP BY team_id;`

Retrieve the average salary of players for each team.

	team_id	average_salary
▶	1	5008.585455
	2	5839.880000
	3	5214.621333
	4	5014.735143
	5	4660.865000
	6	8265.730000
	7	5292.866389
	8	5786.919286

26. `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.

	referee_id	total_matches
	13	97
	14	79
	15	89
	16	90
	17	87
	18	84
	19	103
	20	97

27. `SELECT player_id, first_name, last_name FROM Players WHERE team_captain = TRUE;`

Retrieve the players who are team captains.

	player_id	first_name	last_name
▶	1	Randee	Lowery
	21	Cicily	Hull
	33	Glad	Ochoa
	47	Ealasaid	Barber
	58	Suzann	Miller
	70	Gladi	Robbins
	85	Eba	Hull
	105	Chrystel	Williams

28. `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.

	COUNT(p.team_id)
	4
	13
	18
	18
	4
	9
	16
	15

29. `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.

	player_id	first_name	last_name
▶	6245	Easter	Martinez
	6246	Lily	Calhoun
	6247	Giustina	Barber
	6248	Becka	Chung
	6249	Jonie	Rodriguez
	6250	Randee	Brown
	6251	Aarika	Lin
	6252	Lilyan	Rodriguez

```
30. 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.

	team_id	total_players
▶	2	1
	5	4
	6	4
	16	4
	17	4
	39	1
	43	1
	48	4

Performance

The STMS database is a 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 a 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 have added 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 indexed and non-indexed format. These results are held in the file "time_improvement_with_indices.txt". In this file, negative percentage increases correlate that the query was indeed ran faster. The 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.

In the github a text file is generated where all the 30 common queries are tested with and without the indices. The link is provided below:

https://github.com/uUtkuC/STMS/blob/main/Database_creation_population_performance/performance/time_improvement_with_indices.txt

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%