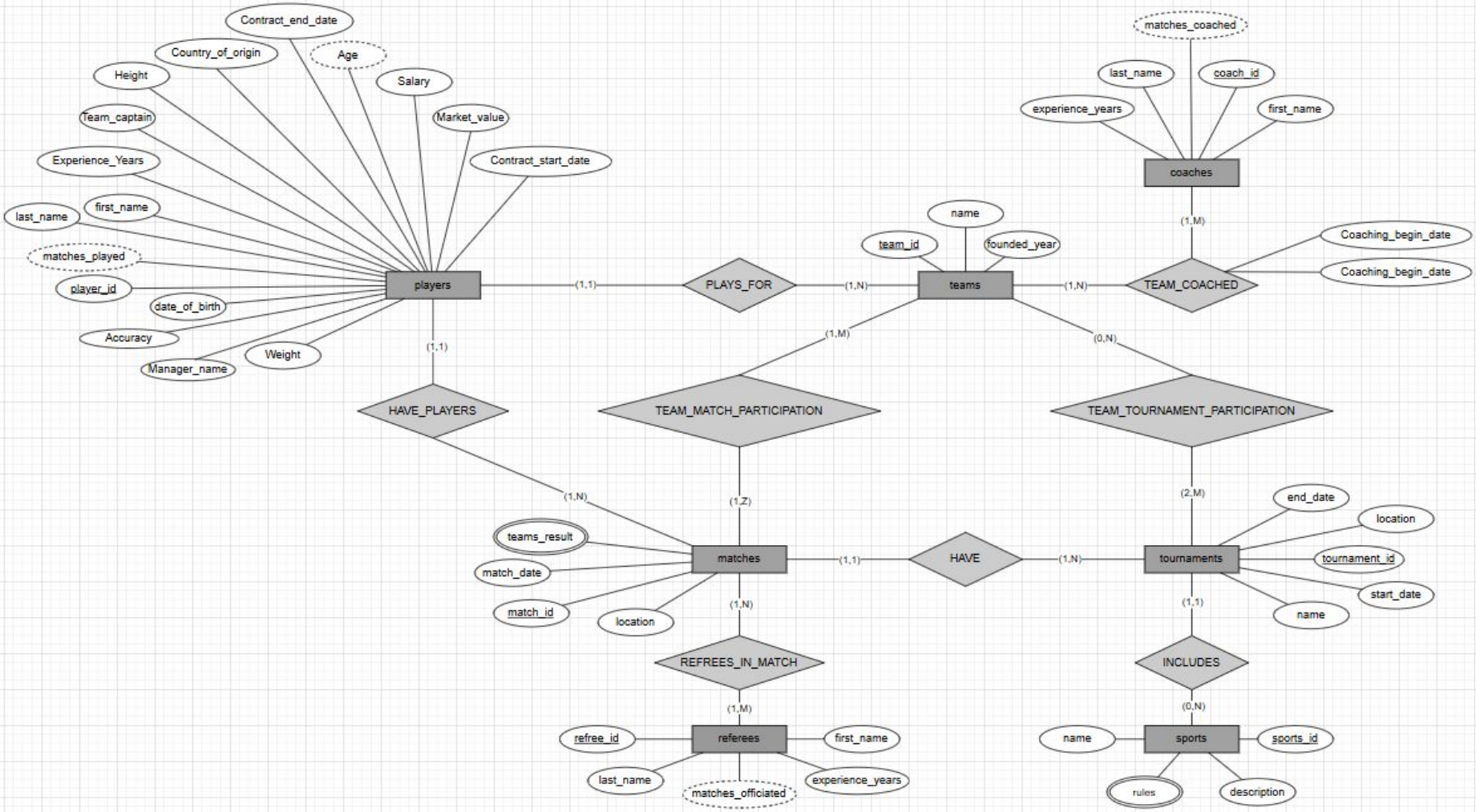
Sports Tournament Management System

Group name: STMS
Github: https://github.com/uUtkuC/STMS
Demo video1: https://www.youtube.com/watch?v=TSFd6GDuPyk&feature=youtu.be
Demo video2:https://www.youtube.com/watch?v=jSTzLS7pyww

Contents

Problem:

- Sport managers require a database to store tournament information.
- Many teams compete.
- Many coaches, referees and players exist.
- Many Sport types for each tournaments exists, From Pubg tournaments, to baseball and Basketball tournaments.


Contract_end_date
Country_of_origin
Age
Height
Salary
Team_captain
Market_value
Experience_Years
Contract_start_date
last_name
first_name
matches_played
player_id
date_of_birth
Accuracy
Weight
Manager_name
players
(1,1)
PLAYS_FOR
(1,N)
teams
name
team_id
founded_year
(1,N)
TEAM_COACHED
Coaching_begin_date
Coaching_begin_date
(1,M)
coaches
matches_coached
last_name
coach_id
experience_years
first_name
(1,1)
HAVE_PLAYERS
(1,M)
TEAM_MATCH_PARTICIPATION
(0,N)
TEAM_TOURNAMENT_PARTICIPATION
(1,N)
(1,Z)
(2,M)
teams_result
matches
(1,1)
HAVE
(1,N)
tournaments
end_date
location
tournament_id
start_date
name
match_date
match_id
location
(1,N)
(1,1)
REFREES_IN_MATCH
INCLUDES
(1,M)
(0,N)
refree_id
referees
first_name
last_name
matches_officiated
experience_years
name
sports
sports_id
rules
description

DB POPULATION

The challenge:

Must ensure that teams of a certain sport type take part in a tournament of that certain type.

Must schedule matches in a tournament for all teams ensuring that they all face together.

Must create teams of varying sizes, because each sport type requires different amount of players.

Must ensure each player is generated randomly and is joined into teams. It is also totally possible for teams to change players from each tournament to the other hence that possibility is taken into consideration too.

```python
# Populate players table
players_rows = []
players_in_teams = []
player_count = 1
for tourn in tournament_objs:
    for t in tourn.teams:
        # Get the team-specific player count based on the sport type
        if t.stype == "pubg E-tournament":
            num_players = 1  # PUBG can have 1 player per team
        elif t.stype == "cycling":
            num_players = 1  # Cycling can have 1 player per team
        else:
            # For traditional sports, teams will have 11-20 players
            num_players = rand.randint(11, 20)

        players_temp = []
        #add the choice of captains index here
        # her takımın ilk oyuncusu kaptan olacak
        temp = 0
        for i in range(num_players):

            player_count += 1
            dob = datetime(rand.randint(1980, 2005), rand.randint(1, 12), rand.randint(1, 28))
            start_date_player = random_date(start_date, end_date)
```

Leverages randomization to create unique and diverse profiles while ensuring logical consistency with the structure of matches, tournaments, and team dynamics.

DB POPULATION

```
referee_count_dict = {
    "baseball": 4,            # 4 referees (umpires) in a typical baseball game (plate umpire + 3 base umpires)
    "tennis": 1,              # 1 referee (chair umpire), though there may be line judges
    "basketball": 3,          # 3 referees in a standard basketball game (1 lead, 2 trail)
    "soccer": 4,              # 1 center referee, 2 assistant referees, 1 fourth official
    "pubg E-tournament": 1,   # 1 referee
    "rugby": 3,               # 1 referee and 2 touch judges (assistants)
    "cricket": 3,             # 2 on-field umpires, 1 third umpire (for technology assistance)
    "hockey": 2,              # 2 referees in field hockey (1 umpire per half)
    "golf": 1,                # 1 referee (often an official who oversees the game, particularly for disputes)
    "volleyball": 2,          # 1 first referee, 1 second referee (sometimes with line judges)
    "cycling": 1              # 1 referee (official race director or commissaire)
}
```

Varying number of referees for each match is selected.

A referee may officiate more than 1 match and similarly a match can be officiated by 1 or more referees depending on the sport type

DB POPULATION

Realistic tournament locations are provided for tournaments to take place in.

```
tournament_location_dict = {
    0:"Ankara",
    1:"Dubai",
    2:"New York",
    3:"Sivas",
    4:"Arizona",
    5:"Berlin",
    6:"Tokyo",
    7:"Madrid",
    8:"Kiev",
    9:"Paris",
    10:"Internet"
}
```

DB POPULATION

Some additional points considered:

A team may be coached by multiple coaches over years. This relation is stored in the : Team_Coached table.

JSON files are used to store the rules of each sport and for team results.

All e-sport tournaments must take place in the location “Internet”

Each player has many unique stats that both identify them that range from overall match accuracy stats and market value to height and country of origin details.

For the database population purposes, mysql connector and mydb.cursor() is used

Some Common Queries

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

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

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

Some Common Queries

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

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

Indexing & Performance

```sql
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);
```

API

Role of API in STMS

- The API serves as a communication layer between the user interface and the database.
- API receives requests from GUI, process the requests and interacts with the database
- API can perform 4 actions: Retrieving, Adding, Updating and Deleting data

Technologies Used for API

FLASK

Flask is a web framework used to build the API

PyMySQL

PyMySQL allows us to connect to MySQL database from python applications

aiohttp

aiohttp handles asynchronous HTTP requests

API Functions

- get_tables() **[GET]**
- get_table_schema() **[GET]**
- get_table_data() **[GET]**
- add_data() **[POST]**
- search_data() **[POST]**
- update_data() **[PUT]**
- delete_data() **[DELETE]**

API Functions

get_tables(): Fetches the tables of database

```
SHOW TABLES;
```

get_table_schema(): Fetches the structure of selected table

```
DESCRIBE `{table_name}`;
```

get_table_data(): Fetches the data from selected table

```
SELECT * FROM `{table_name}`;
```

add_data(): Adds data to selected table

```
INSERT INTO `{table_name}` ({columns}) VALUES ({placeholders});
```

search_data(): Searches data from selected table

```
SELECT * FROM `{table_name}` WHERE {where_clause};
```

update_data(): Updates data from selected table

```
UPDATE `{table_name}` SET {set_clause} WHERE {where_clause};
```

delete_data(): Deletes data from selected table

```
DELETE FROM `{table_name}` WHERE {where_clause};
```

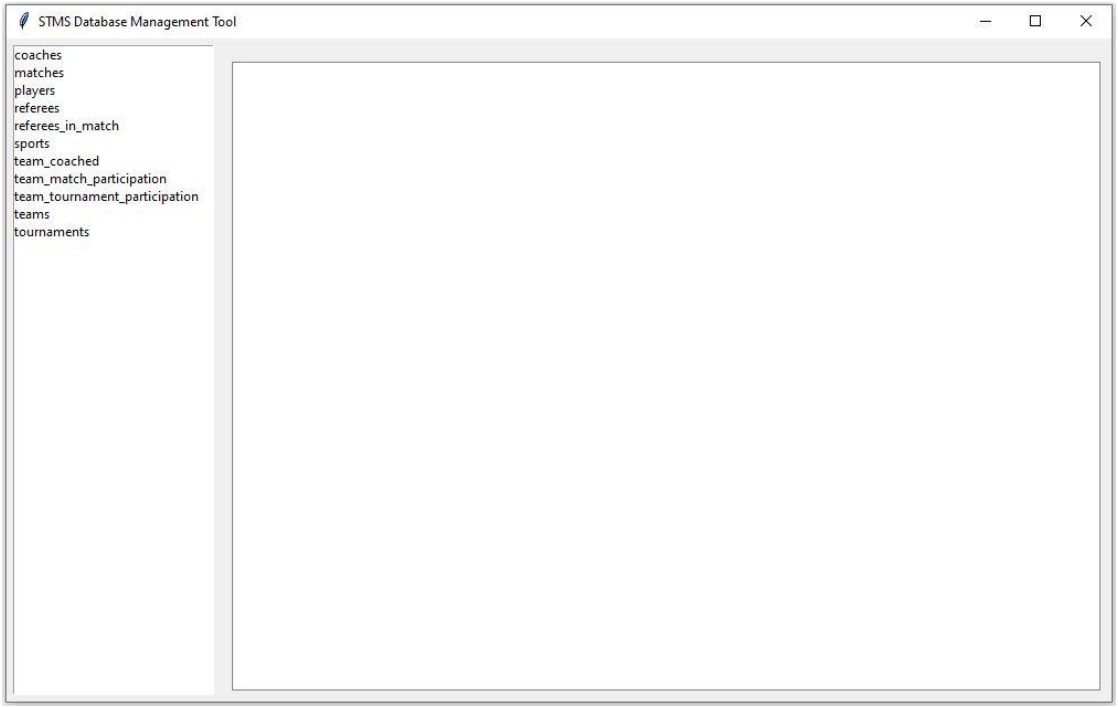
GUI

Overview

- A user-friendly **Graphical User Interface (GUI)** was developed to interact with the underlying database, allowing seamless data management for our system.
- The GUI was built using **Tkinter**, a lightweight yet powerful Python library, ensuring cross-platform compatibility.

GUI

Overview



GUI

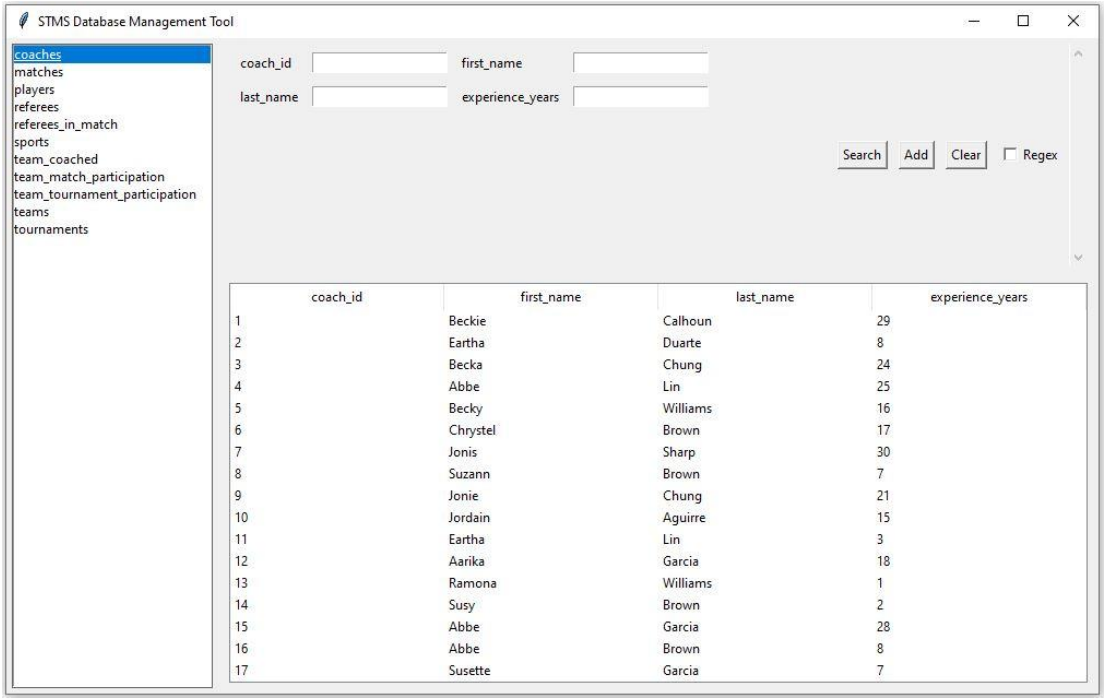
Key Features

Dynamic Table Interaction:

- Automatically fetches and displays tables from the database.
- Dynamically generates input fields for each table based on its schema, ensuring flexibility for future database changes.

GUI

Key Features: **Dynamic Table Interaction**



GUI

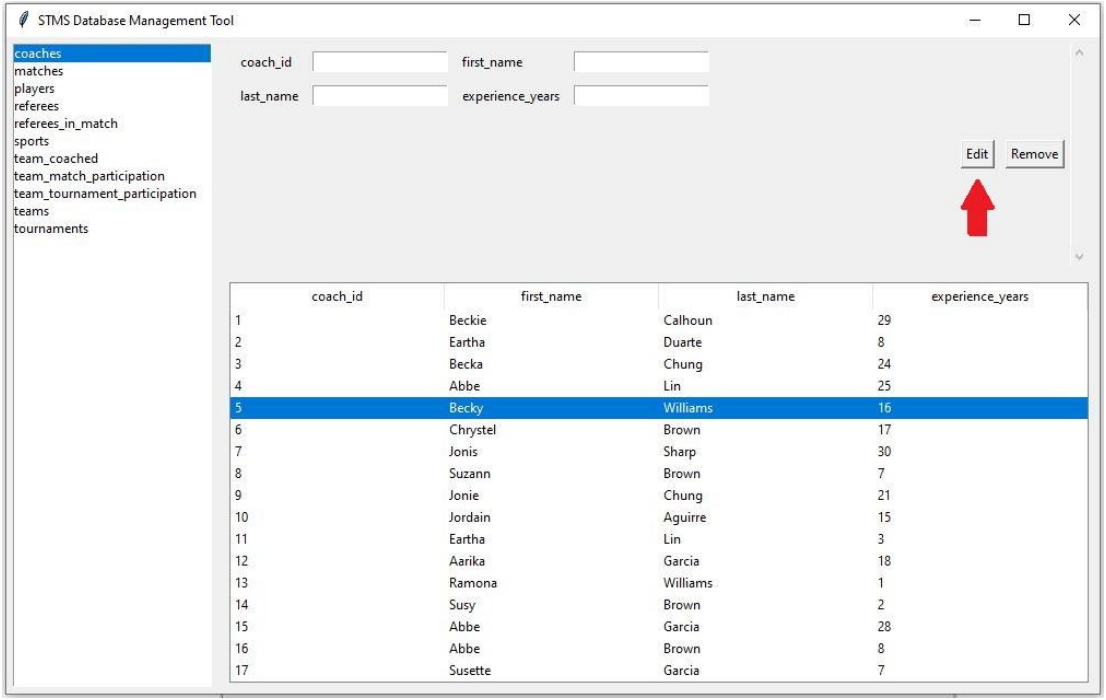
Key Features

Comprehensive Data Operations:

- **Add**: Insert new records directly into the database.
- **Edit**: Modify existing records in an intuitive way.
- **Delete**: Remove records with a single click, updating the database in real-time.

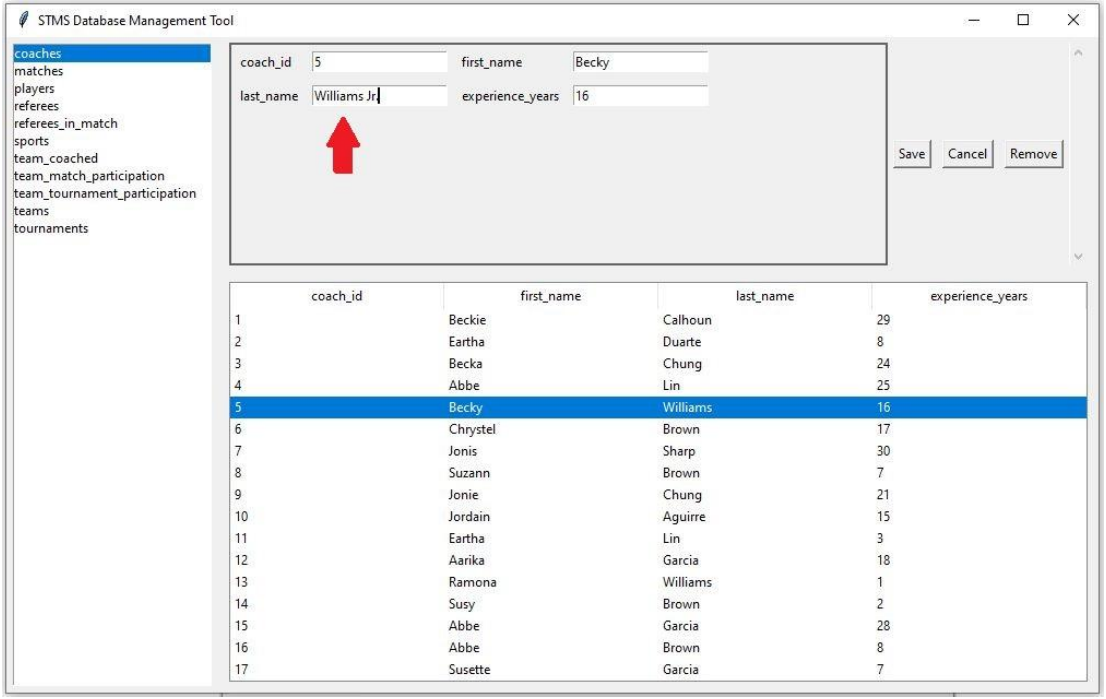
GUI

Key Features: **Comprehensive Data Operations**



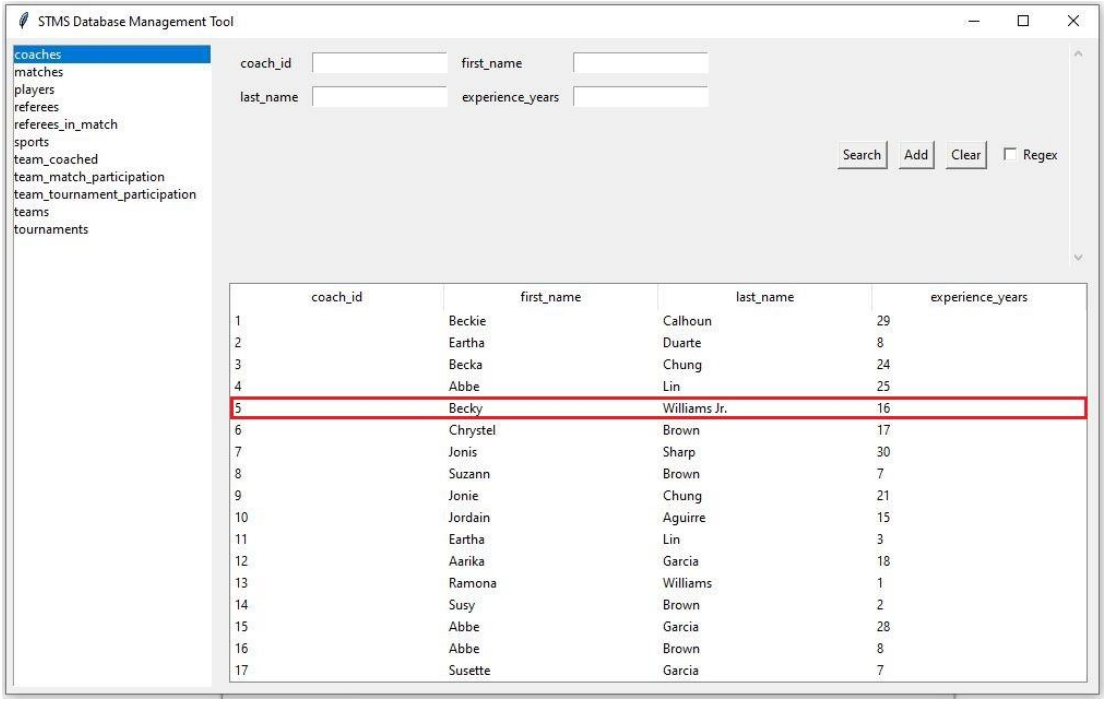
GUI

Key Features: **Comprehensive Data Operations**



GUI

Key Features: **Comprehensive Data Operations**



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 Jr.	16
6	Chrystel	Brown	17
7	Jonis	Sharp	30
8	Suzann	Brown	7
9	Jonie	Chung	21
10	Jordain	Aguirre	15
11	Eartha	Lin	3
12	Aarika	Garcia	18
13	Ramona	Williams	1
14	Susy	Brown	2
15	Abbe	Garcia	28
16	Abbe	Brown	8
17	Susette	Garcia	7

GUI

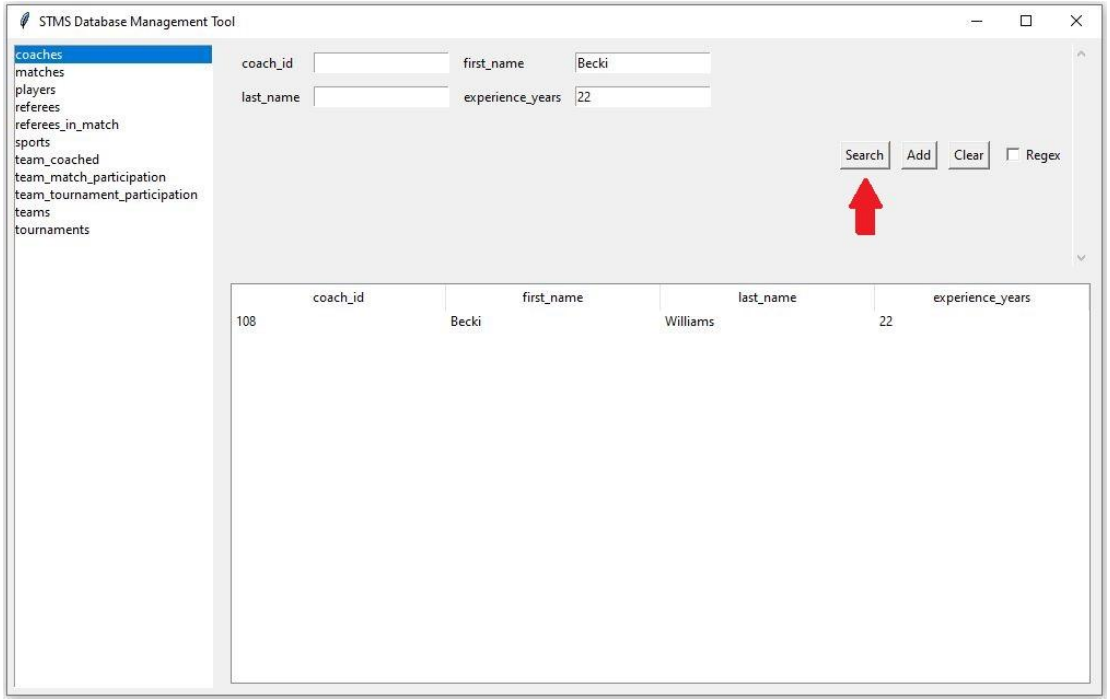
Key Features: **Real-Time Search**

Dynamic Filtering:

- Users can search through the database dynamically based on multiple attributes at once.
- The search considers both exact matches and patterns using regex.

GUI

Key Features: **Real-Time Search**



GUI

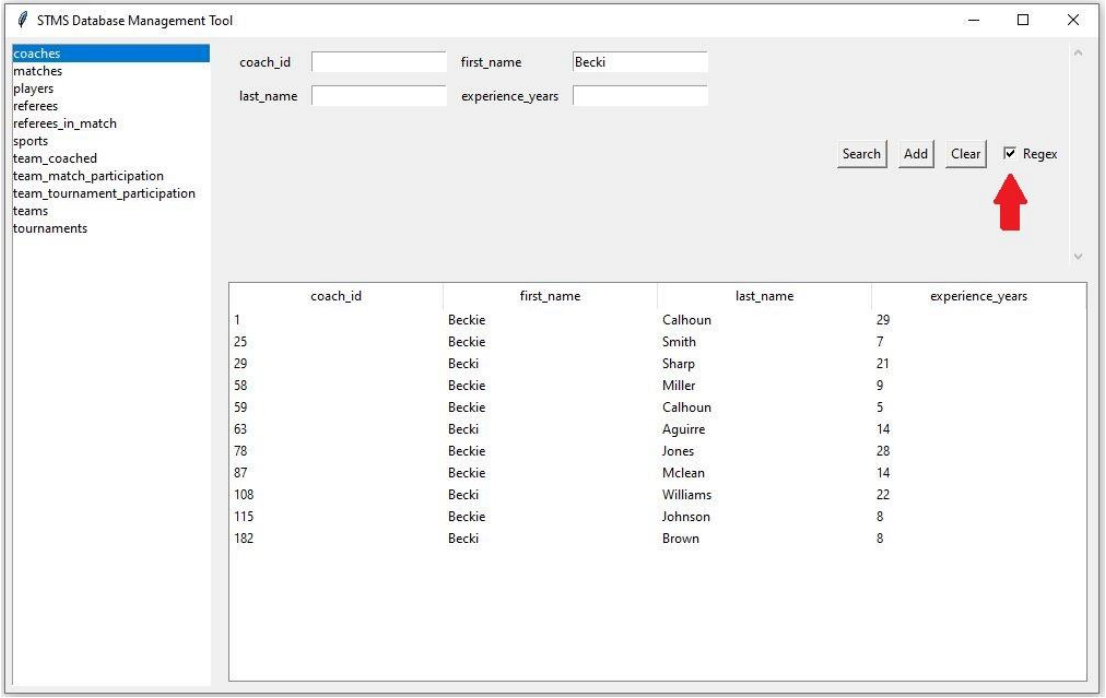
Key Features: **Real-Time Search**

Regex Integration:

- When the "Regex" checkbox is selected, the search becomes more flexible by allowing partial matches or patterns
- If regex is disabled, the search ensures precise results, only returning records with exact matches.

GUI

Key Features: **Real-Time Search**



GUI

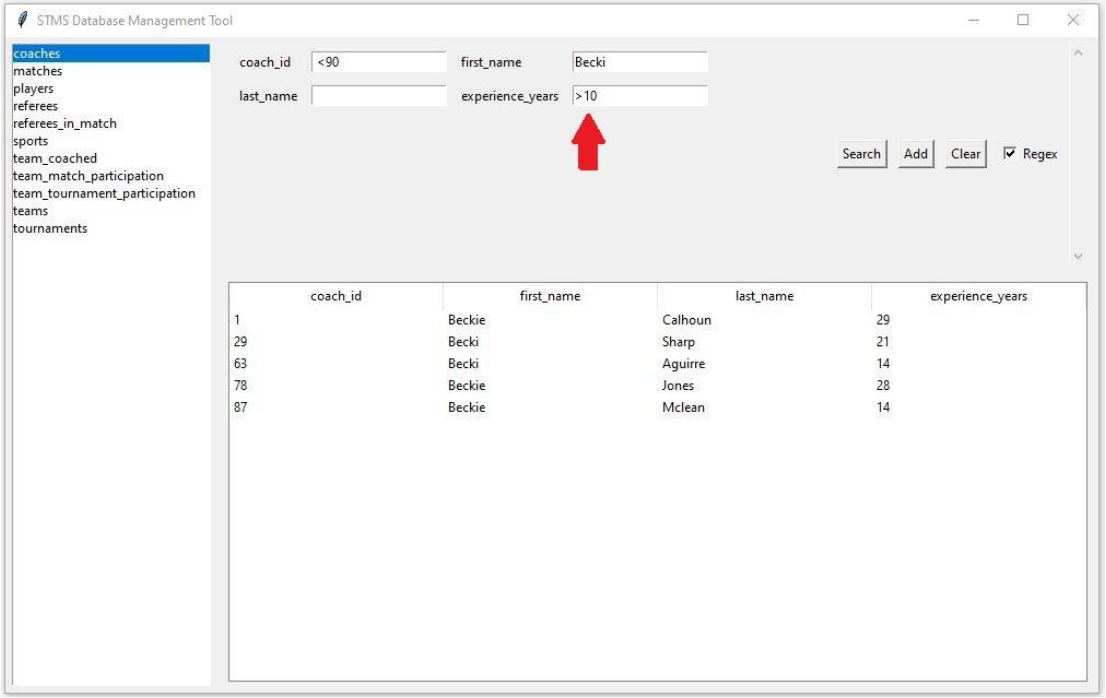
Key Features: **Real-Time Search**

Advanced Comparison Operators:

- Search supports advanced numeric filtering using comparison operators like >, <, >=, <=

GUI

Key Features: **Real-Time Search**



GUI

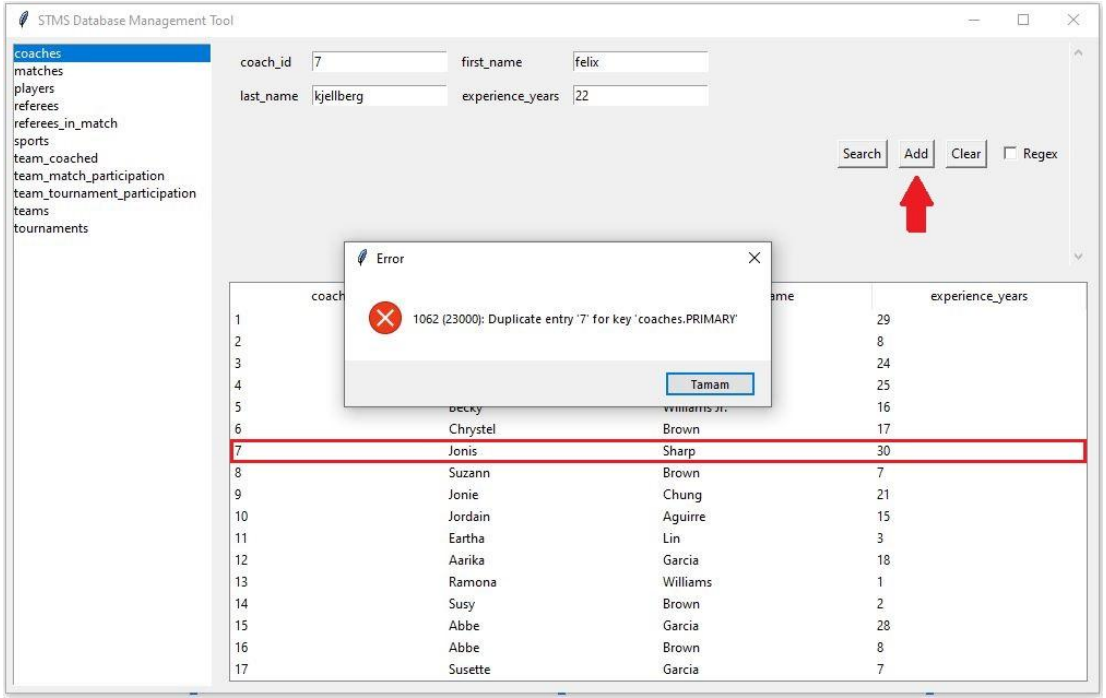
Highlights

Error Resilience:

- Handles empty or invalid inputs with clear error messages.

GUI

Highlights: **Error Resilience**



Recommendations

Even though STMS is for Tournament Organisers, we can add an option for user view with sensitive informations are hidden.

Using Unit Tests and Version Control methods can help us to maintain a stable database.

Regular database backups are needed to prevent data loss.

Implementing archiving strategies for old data would improve performance.

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.