BİL 372

Sports Tournament Management System

Software Layers

Utku CEYLAN	*221101067*
Kıvanç TAŞ	*221401022*
Ali Berk KARAARSLAN	*221401010*
Mustafa TUFAN	*221401029*

Table of Contents

Introduction

This report outlines the software layers of the **Sports Tournament Management System (STMS)**, detailing the technologies, tools, and methodologies used in its development. STMS is designed to efficiently manage sports tournaments, covering functionalities such as tournament management, player/team information and match scheduling.

The report focuses on key components, including the user interface built with Tkinter, the API developed using Flask, and database interactions handled through PyMySQL with a MySQL backend. Each layer is discussed to illustrate how they work together to deliver a scalable, efficient, and user-friendly system, meeting both technical and functional requirements.

Technologies Used in STMS Software Layers

The **Sports Tournament Management System (STMS)** uses a range of modern technologies to ensure the system is reliable, efficient, and easy to use. Below is an explanation of the key tools and frameworks used in the application:

Python

Python is the core programming language of the STMS application. It was chosen for its simplicity, flexibility, and rich library support. Python allows developers to write clean and readable code, making the application easier to maintain and extend. Its compatibility with different operating systems ensures that the STMS application can run on various platforms without modifications. Additionally, Python provides built-in tools and external libraries for tasks such as database management, web development, and user interface creation.

Tkinter

Tkinter is used to build the graphical user interface (GUI) for the desktop version of the STMS application. It comes pre-installed with Python, removing the need for additional setup. Tkinter allows developers to create user-friendly interfaces with buttons, menus, and forms. Its simplicity makes it an ideal choice for building lightweight desktop applications. Through Tkinter, users can easily navigate and interact with the system.

Flask

Flask is the web framework used in the STMS application to develop its API. It is a minimal framework, which means it only provides essential tools, allowing developers to customize it with specific components as needed. Flask's lightweight nature helps keep the application fast and efficient. Furthermore, its seamless integration with Python libraries makes it a powerful choice for handling backend operations and creating flexible, web-based solutions.

PyMySQL

PyMySQL connects the STMS application to the MySQL database. It enables the application to store, retrieve, and manage tournament data efficiently. The library is easy to use, offering simple methods for executing SQL queries and managing transactions. PyMySQL also supports secure database practices, such as parameterized queries, to protect against SQL injection attacks. This ensures the application handles data interactions reliably and securely.

aiohttp

aiohttp is used for managing asynchronous HTTP requests in the STMS application. It allows the application to handle multiple network operations at the same time without performance delays. aiohttp supports both client-side (sending requests) and server-side (receiving requests) functionality, making it a versatile choice for real-time communication. Its support for web sockets ensures the application can provide live updates, making it ideal for tasks that require constant interaction with external services or users.

Database Access and API Endpoints

In this project, we utilized the **PyMySQL** library to interact with the **MySQL database**. A connection to the database is established using PyMySQL, with the following parameters: hostname, username, password, and database name. These connection parameters are stored in a structured format using a dictionary data type for ease of configuration and maintenance.

The API facilitates communication between the database and the client by leveraging these connection parameters. For each API endpoint, a control structure known as a **cursor** is obtained from the connection; this cursor is used to execute SQL queries on the database.

The API creates a connection between database and client with the connection parameters. For each API Endpoint, a control structure -cursor- is obtained from connection and used to perform queries on the database.

API Endpoints of STMS

The following API endpoints are implemented to interact with the database:

get_tables() [GET]

- **Description**: Fetches the list of all tables available in the database.
- **Functionality**:
- Executes the SQL SHOW TABLES query to retrieve all table names from the database.
    - Returns the list of table names in JSON format.

get_table_schema(table_name) [GET]

- **Description**: Retrieves the schema of a specified table, detailing its structure.
- **Functionality**:

    - Executes the SQL DESCRIBE <table_name> query to fetch metadata about the specified table.
    - Returns details such as column names, data types, nullability, primary keys, and additional properties.

get_table_data(table_name) [GET]

- **Description**: Retrieves all rows of data from a specified table.
- **Functionality**:
    - Executes the SQL SELECT * FROM <table_name> query to fetch all records in the table.
    - Supports tables of varying structures, dynamically adapting to the table's columns.

add_data() [POST]

- **Description**: Inserts a new record into a specified table.
- **Functionality**:
    - Accepts a JSON payload containing the target table name and the data to be inserted.
    - Executes a parameterized SQL INSERT query to securely insert the data.

search_data() [POST]

- **Description**: Searches for records in a specified table based on user-defined criteria.
- **Functionality**:
    - Accepts a JSON payload with the table name and the search criteria.
    - Executes a parameterized SQL SELECT query with WHERE conditions based on the criteria.
    - Returns matching records.

update_data() [PUT]

- **Description**: Updates an existing record in a specified table.
- **Functionality**:
    - Accepts a JSON payload containing the table name, the record's primary key (or identifying columns), and the updated data.
    - Executes a parameterized SQL UPDATE query to modify the specified record.

delete_data() [DELETE]

- **Description**: Deletes a record from a specified table based on a unique identifier or set of criteria.
- **Functionality**:
    - Accepts a JSON payload specifying the table name and the record's identifying columns.
    - Executes a parameterized SQL DELETE query to remove the specified record.

Database Management GUI Documentation

This documentation describes the implementation, functionality, and usage of a Python-based GUI for database management. The GUI interacts with a MySQL database, allowing users to perform **CRUD** (Create, Read, Update, Delete) operations and execute advanced searches with both exact matches and regular expressions. 2 video demo are also added as a YouTube link below:

https://www.youtube.com/watch?v=jSTzLS7pyww

https://www.youtube.com/watch?v=TSFd6GDuPyk&feature=youtu.be

The application is designed to:

- Be user-friendly for both technical and non-technical users.
- Offer advanced features like regex-based searches and comparison operator handling.
- Handle edge cases gracefully with appropriate error feedback.

Key Features

1. Dynamic Table Management

- Tables are automatically detected, and schemas are dynamically loaded, reducing hardcoding and ensuring scalability.

2. Full CRUD (Create, Read, Update, Delete) Support

- Intuitive buttons (Add, Edit, Remove, Clear) enable users to manipulate data with ease.

3. Advanced Search Capabilities

Regex-Based Search:

- Enables pattern matching for string fields.
- Example:
    - Search: name = 'Jo' (regex mode enabled).
    - Results: Rows with name='John', name='Johnny', or name='Johnson'.

Exact Match Search:

- Ensures only precise matches are returned.
- Example:
    - Search: team_name = 'Team A' (regex mode disabled).
    - Results: Rows where team_name is exactly Team A.

Numeric Comparisons:

- Supports operators for numeric filtering.
- Example:
    - Search: score > 50.
    - Results: Rows where the score exceeds 50.

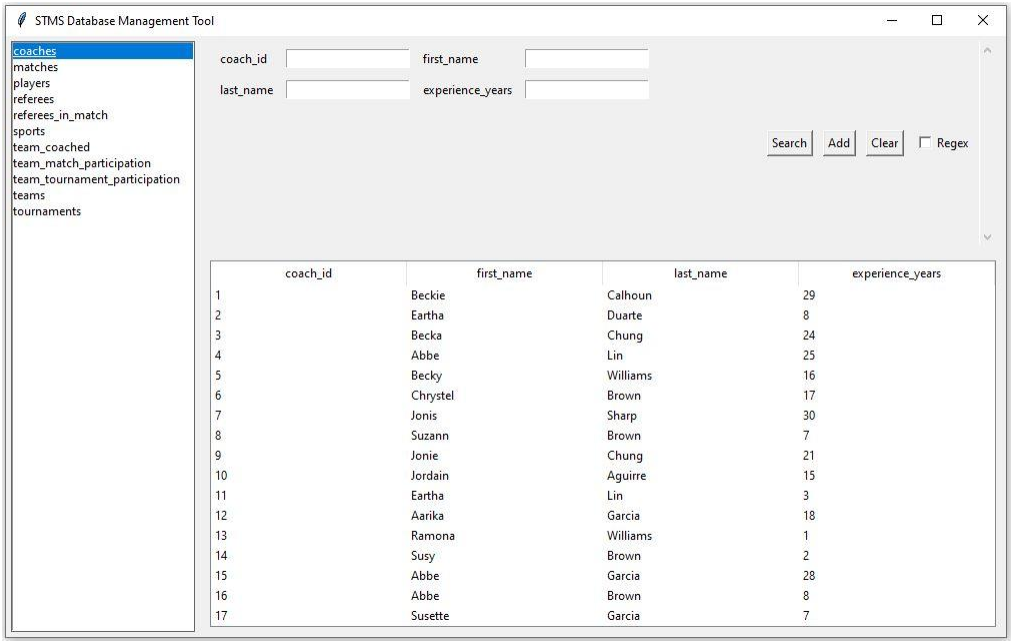
4. Error Handling and Validation

- Prevents invalid SQL operations and provides meaningful feedback.

How to Use

1. Table Selection

- Select a table from the left-hand Listbox.
- The table's data appears in the bottom Treeview, and input fields for its attributes appear at the top.
- In this example, coaches table has selected.



2. Adding Data

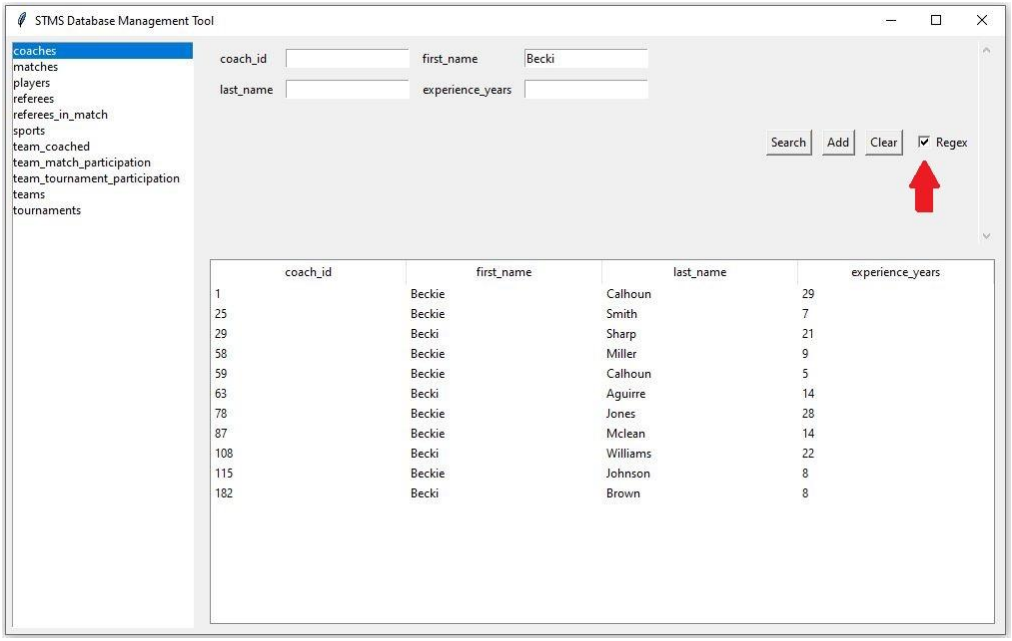
- Fill in the required fields in the input panel.
- Click the Add button to insert the data.
- Results appear in the Treeview below.

3. Searching Data

- Enter a condition in the search field.
- Click the Search button. Matching rows appear in the Treeview.

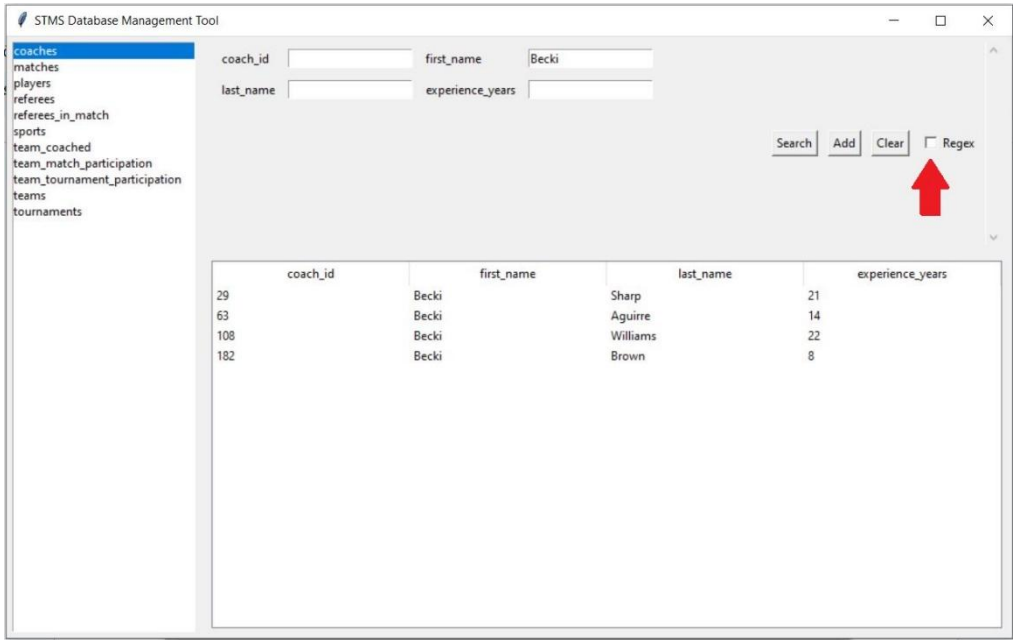
Regex-Based Search:

- Enables pattern matching for string fields.
- In this example, “Becki” as first_name has searched. Because the regex option was selected, all first_names containing “Becki” has returned as result.



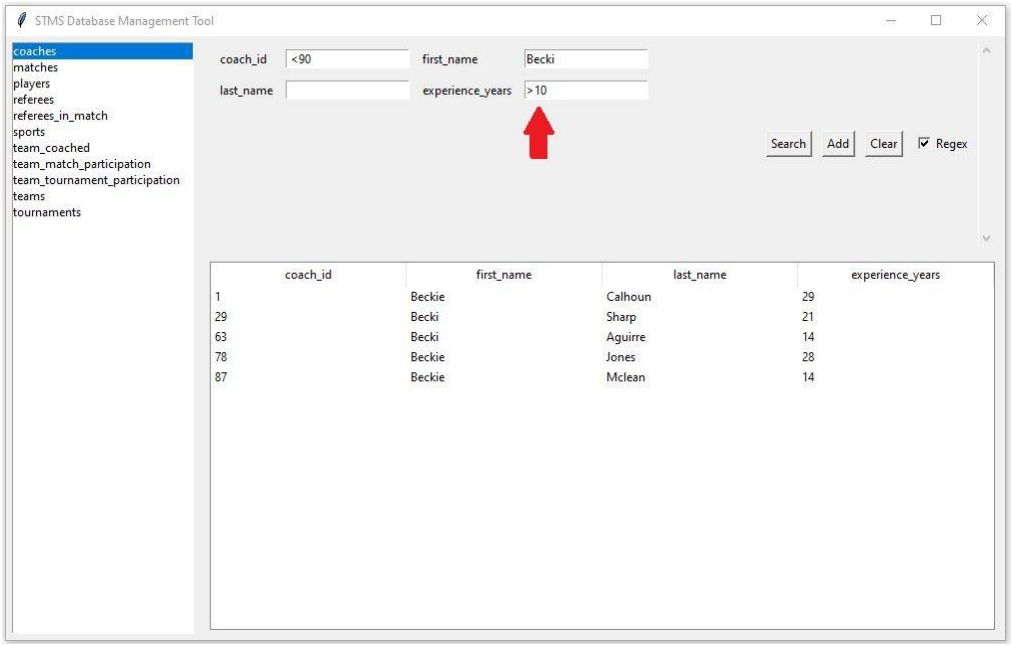
Exact Match Search:

- Ensures only precise matches are returned.
- In this example, “Becki” as first_name has been searched. Because the regex option was not selected, all first_names which are precisely matched with “Becki” have returned as result.



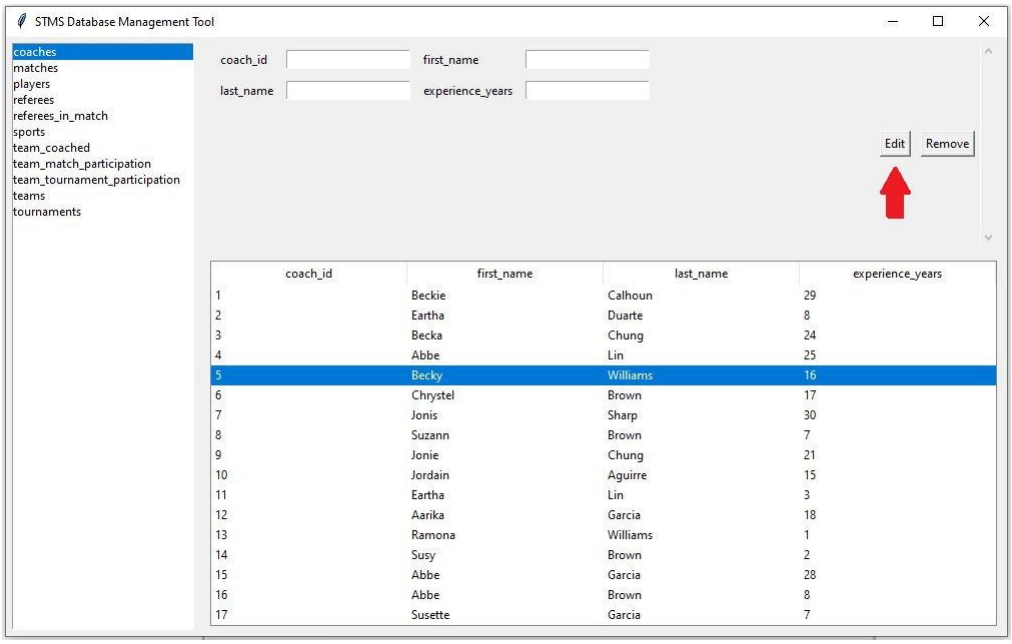
Numeric Comparisons:

- Supports operators for numeric filtering.



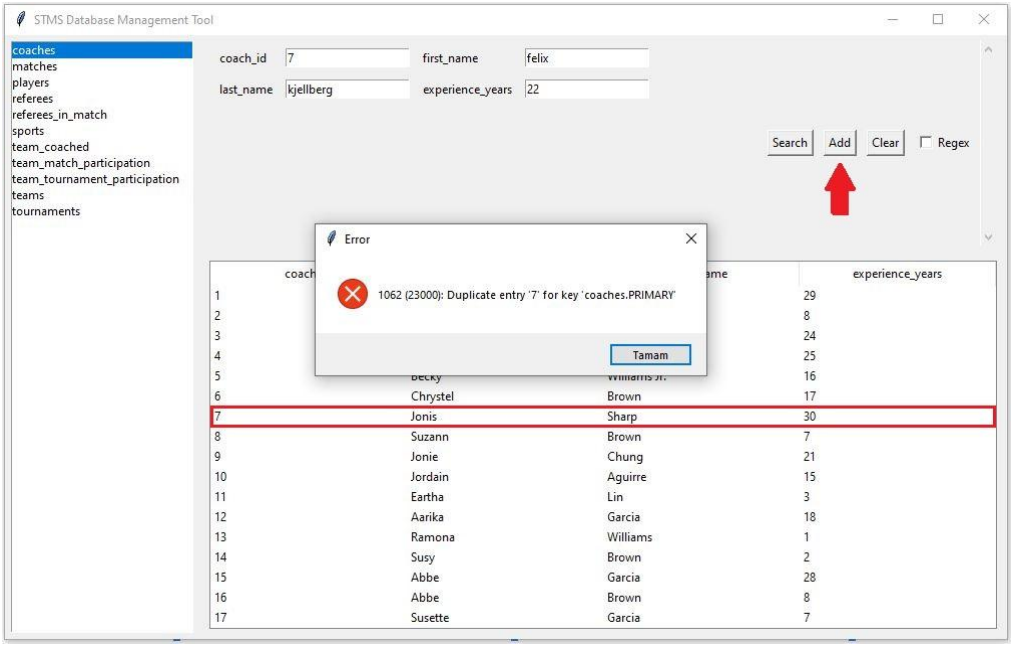
4. Editing or Removing Data

- Select a row in the Treeview.
- Click Edit to modify the data or Remove to delete it.
- Confirm changes, and the Treeview updates automatically.



5. Error Handling

- If an invalid SQL operation is tried to execute, GUI shows this error as an error box and blocks the operation.
- In this example, a coach with coach_id 7 tried to add to the database. However, because another coach with coach_id 7 present, the operation has blocked and showed this error as error_box.



Conclusion

The **Sports Tournament Management System (STMS)** is built using Python and integrates Tkinter for the user interface, Flask for API development, and PyMySQL for secure database interactions with a MySQL backend. This combination provides a user-friendly, scalable, and efficient solution for managing tournament data.

The API endpoints include:

- get_tables() [GET]: Retrieves all database tables.
- get_table_schema(table_name) [GET]: Provides the schema of a specific table.
- get_table_data(table_name) [GET]: Fetches all rows from a table.
- add_data() [POST]: Adds new records to a table.
- search_data() [POST]: Searches records based on criteria.
- update_data() [PUT]: Updates existing records.
- delete_data() [DELETE]: Deletes records from a table.

The integration of these layers enables smooth communication between the user interface, API, and database, fulfilling the project's requirements and providing a solid foundation for future improvements.

GUI serves as a comprehensive tool for managing a MySQL database. Its dynamic nature, advanced search capabilities, and robust error handling make it suitable for both developers and non-technical users. By focusing on user experience and functionality, this application significantly simplifies database interactions.