Data Modelling, Management, and Governance

Sports Club Management System Implementation Report

Contents

INTRODUCTION

Today, it is trite knowledge that most routine business processes are still executed manually in many quarters. This means business operations are predominantly processed using ancient paper technology.

Thus, many heaps of paper documents containing essential or supposedly confidential information are common in most business offices and organisations. But this approach has its known associated drawbacks. Some of these include data inaccuracy and or error proneness due to the human factor involved, data maintenance difficulties – it is impossible to update data without reproducing the entire document, data integrity, etc.

This report is for automating a system that would otherwise be manually implemented. It is for digitising and managing the activities and basic operations of a Sports Club.

SPORTS CLUB MANAGEMENT ANALYSIS

The provided scenario is a sports club wherein members and staff carry out certain functions. Typically, people sign up to become members of a club even as provided in the given scenario. On becoming members, they can participate in the club's various activities, including signing up for club activities such as taking a sports class, updating personal information upon registration, booking a sports class, and viewing daily or weekly activities or bookings.

Also, this club has staff members that help with managing its operations. These staff members, particularly instructors, are expected to search for daily running activities and record attendance for a class, in addition to registering and being able to update their profile.

However, the Sports Club Management System is expected to store information about these three entities: members, staff, and classes. For members, it would capture details such as member id, first name, last name, address, telephone, email address, date of birth, and medical condition. It would capture details such as staff id, first name, last name, role, and contact number for staff members. And for class, it would capture details such as class code, title, delivery day, and delivery time.

SYSTEM ANALYSIS AND DESIGN

According to Pressman (2015), understanding the requirements of a problem is among the most challenging tasks that face a software engineer. This is a truism. Software engineers always face the challenge of understanding what a client's system should do for the client, so they

spend sufficient time analysing the system requirements and functionalities before developing. This process is often referred to as Requirements Engineering.

The whole idea of analysing a system is to elicit the functional specifications of the system. This is so that the right design can be built and built the right way. These requirements are divided into functional and non-functional requirements.

Functional Requirements

In straightforward terms, a Functional Requirement (FR) describes the service that the software must offer. In other words, a functional requirement represents a software system or its component. The term "functional" describes a function – what the system must or should do, referring to inputs made to the software system, its behaviour, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality that defines what function a system is likely to perform. Functional Requirements in Software Engineering are also called Functional specifications**.**

According to Ian Sommerville (2016) - one of the foremost authorities in software engineering, Functional Requirements are the system's behaviour – how it should react to inputs and the services it should provide. In other words, what the system should do. Thus, in our case, the functional requirements or specifications of the Sports Club Management System refer to the functions and behaviour of the system. It describes what the system must allow users to do and their outputs. Accordingly, we have elicited some of the system's functionalities as clearly defined in the given scenario. Thus, the Functional Requirements Document for the Sports Club Management System is given below.

Critical Decision

It is assumed that the Sports Club Management System should be a web application, which means users would be required to sign up and sign in.

Also, the addition of *password change* and *password retrieval* is premised upon the fact that web applications generally include such functionalities to enable users to quickly retrieve forgotten or lost passwords and change the same for any good reason on authentication. This is the reason behind the inclusion of these functionalities in the Functional Requirements Documents.

FUNCTIONAL REQUIREMENTS N0.	FUNCTIONAL REQUIREMENTS DESCRIPTION
FR 1	**Sign Up:** The system shall allow new users to sign up by providing an email address, user details, and a password.
FR 2	**Sign In**: The system shall allow users to sign in by providing an email and a password.
FR 3	**Password Change:** The system shall allow users to change their password when provided with an old password and a One Time Password
FR 4	**Password Retrieval:** The system shall provide "Forgot your password?" functionality and email users their passwords when provided with a verified email address.
FR 5	**Account Updating:** The system shall allow users to update personal account details
FR 6	Search: The system shall allow staff (instructor) to search for activities running on a particular day using class codes
FR 7	**Attendance Recording:** The system shall allow staff (Instructors) to record attendance
FR 8	**View:** The system shall allow members to view daily/weekly activities and bookings
FR 9	**Booking:** The system shall allow members to book up to five Sports classes per week

Table 1 - Functional Requirements Specification

In addition to the Functional Requirements Document, we have also provided a Requirements Diagram, which clearly and distinctly defines the various identified functionalities and their relationships in terms of association, dependencies, or generalisation.

The diagram shows that to log in, the user must sign up and have login credentials to perform all other functions such as changing or retrieving passwords, booking a class, updating the user profile, etc.

Critical Decision

Regarding tools for modelling the Functional Requirements, the choice was Visual Paradigm Online. Other tools such as Creatly.com, ArgoUML, Rational Rose, Modello, etc., exist and

could have done a great job, but only Visual Paradigm provides for a Requirements diagram modelling and is free.

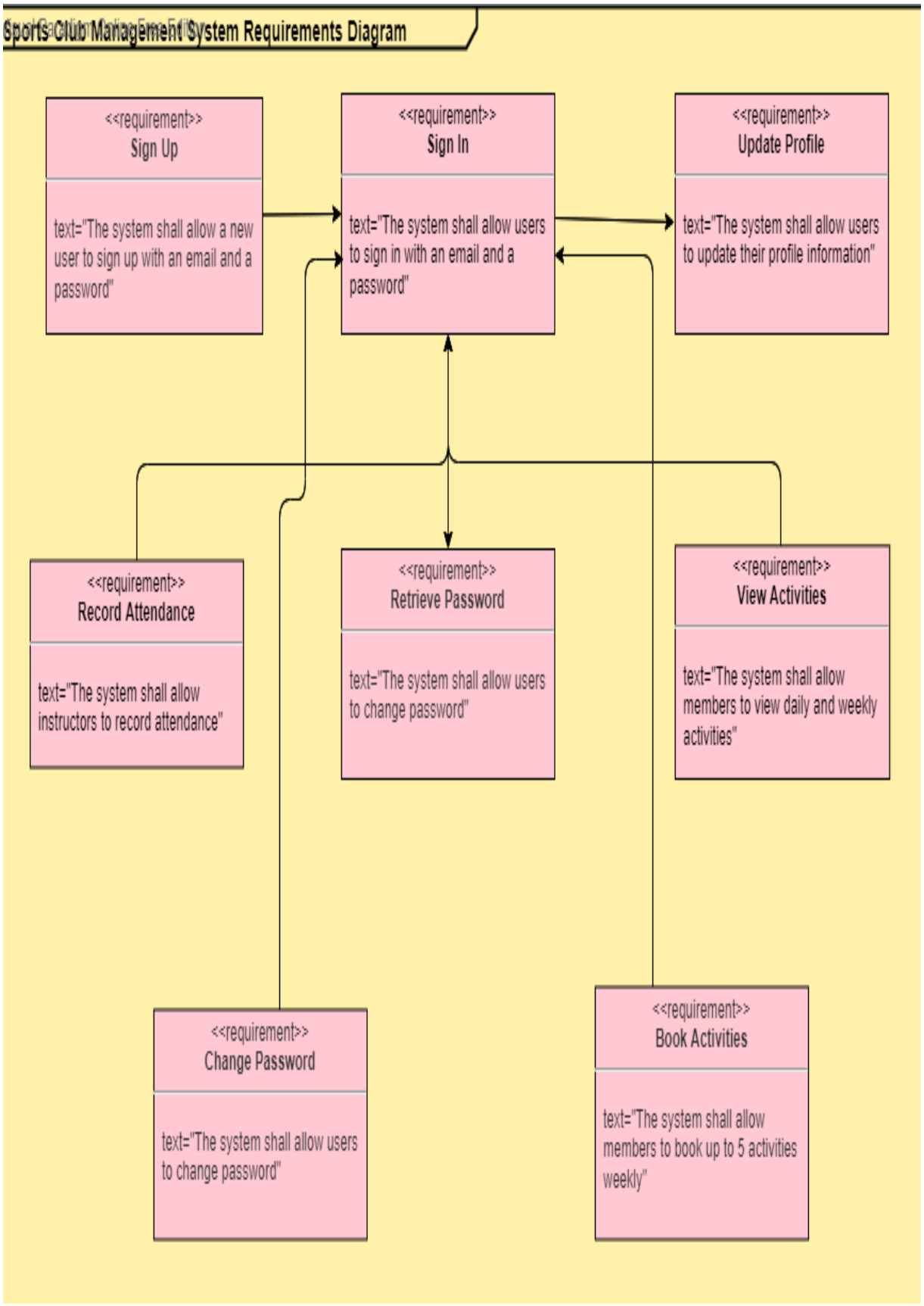


Figure 1 - Requirements Diagram

Usecase Diagram

Essentially, a use-case diagram describes the high-level functions and scope of a system while also identifying the interactions between the system and its actors. Thus, the diagram below shows use-cases and actors in the Sports Club Management System, but not the system's internal operations.

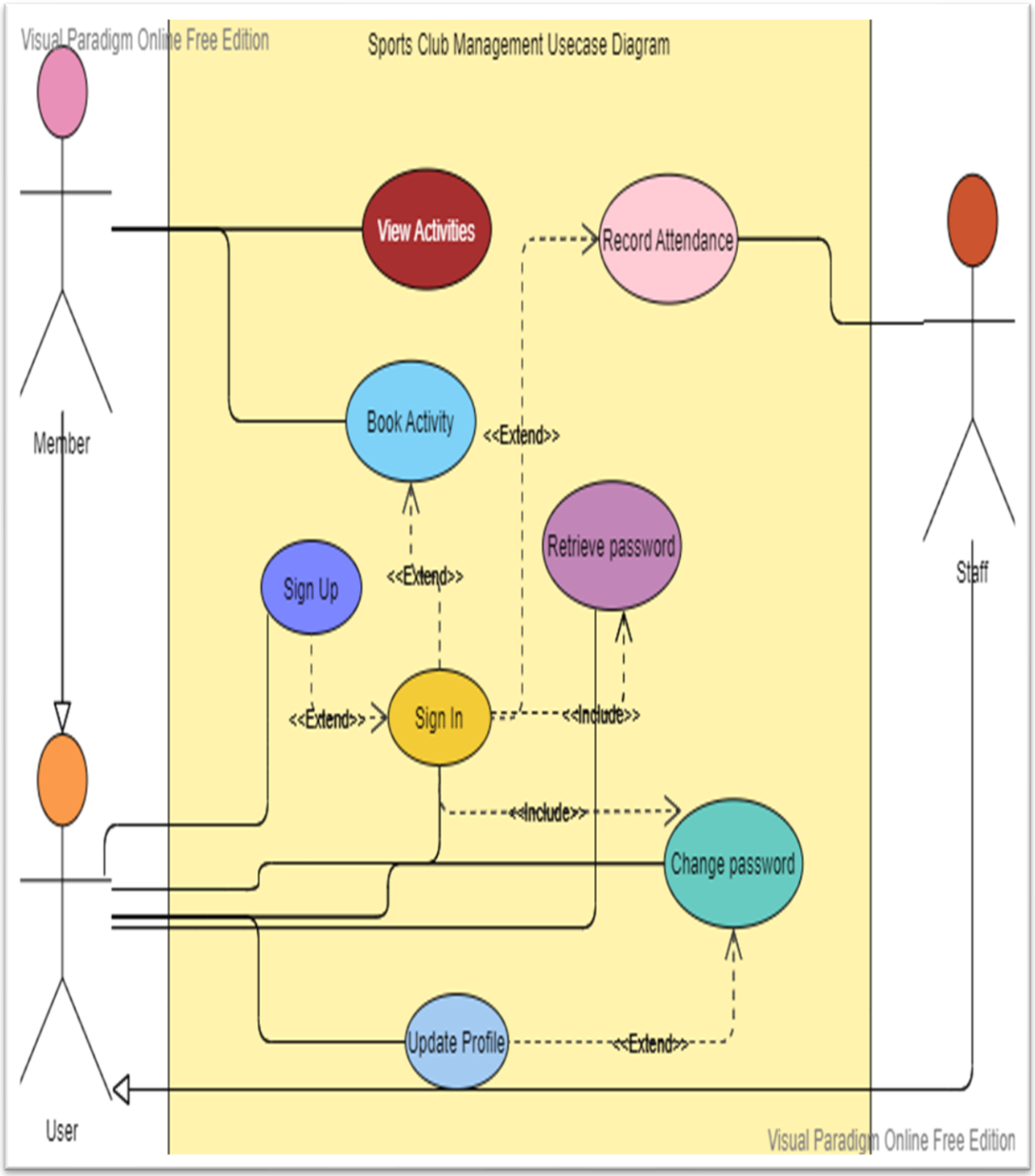


Figure 2 - Usecase Diagram

Explanation: The Usecase diagram has three actors - user, member, and staff. The user actor can sign up, sign in (login), and update the user profile. Both member and staff are child actors

of the user parent actor and inherit those use-cases as shown in the diagram. However, the member actor has two use-cases – he can book and view activity (class), while the staff actor can record attendance.

Further, these use cases have some relationships. The signup usecase *extends* sign-in; while sign-in *extends* record attendance and book activity; it also *includes* retrieve password and change password.

Class Diagram

A class diagram is a Unified Modelling Language (UML) structural diagram that describes the structure of a system by showing the system's classes, their attributes, behaviours (operations or methods), and the relationships among objects. Thus, this class diagram shows the various objects, their respective attributes and operations, and their relationships in the Sports Club Management System. This structure can be converted directly into a software class during development.

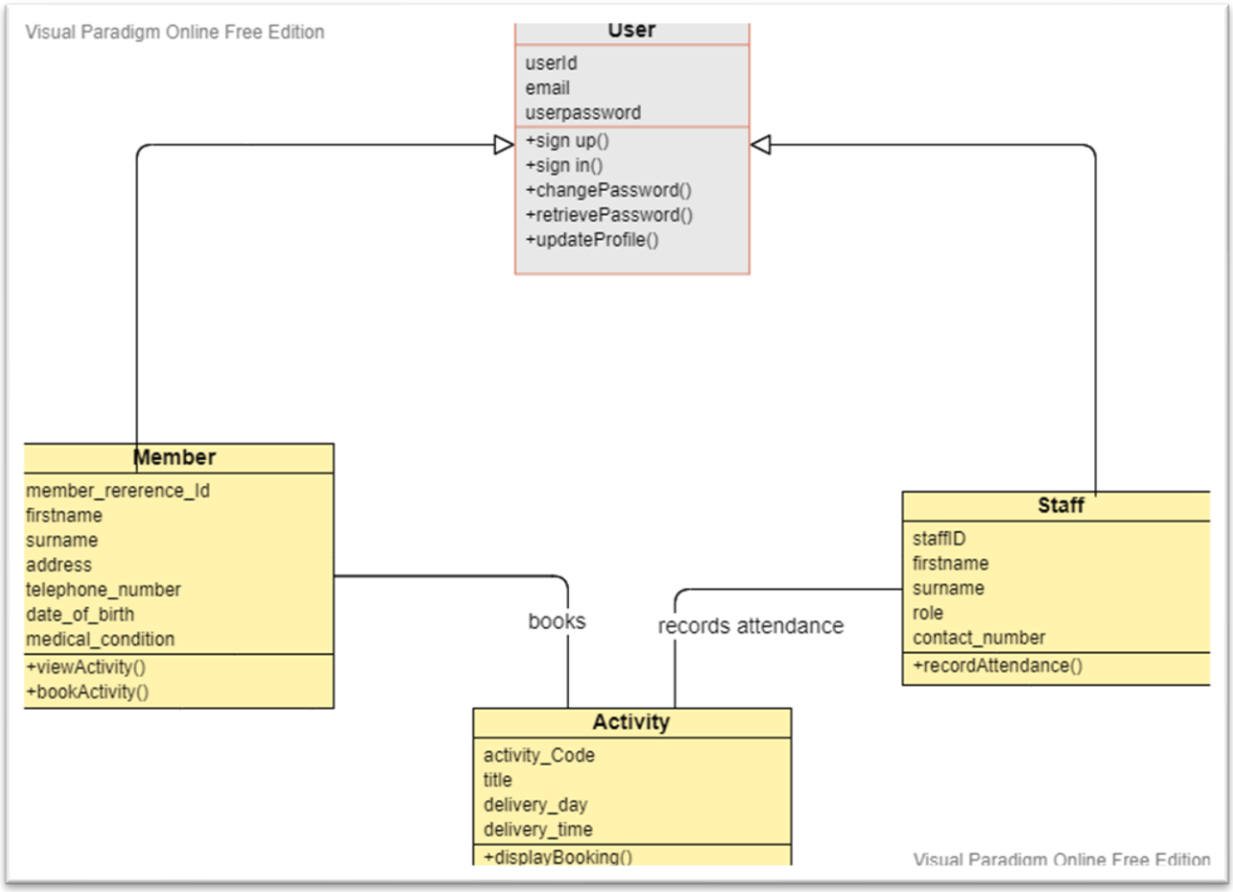


Figure 3 - Class Diagram

As seen above, classes are entities which could be roles, persons, or objects. If they are entities or identified objects in the system, they should have attributes and behaviours. Ideally, the

characteristics of classes converted into software classes are variables and should also have their data types defined in the class. This feature is not available in the tool used. Other tools such as ArgoUML will force the designer to define attribute datatypes.

Non-Functional Requirements

Non-Functional Requirements are constraints or requirements imposed on the system. They specify the quality attributes of the software. They are concerned with issues like performance (considering whether it is fast or not fast in its execution); scalability (considering whether it can grow and accommodate growth as more business processes are integrated); portability (assuming whether the software can be installed in a new platform); security (considering whether the system is secure against external attacks); reliability (considering whether the system can produce the expected results, and perform as expected always); maintainability (assuming whether the system can be easily maintained or modularised for easy maintenance); and many more.

Non-Functional Requirements address vital issues of quality for software systems. If not adequately addressed, the results could be that users, clients, and developers are unsatisfied, the software might be inconsistent, and time and cost could be overrun.

DATABASE DESIGN

This section discusses three different types of data models. These models depict and demonstrate three levels of abstraction. Design thoughts are crucial to the implementation of the system. If the design architecture is incorrect, it leads to incorrect database implementation and incorrect system development. Thus, presented below are the conceptual, logical, and physical models of the Sports Club Management System.

Conceptual Data model

The conceptual Entity-Relationship Diagram (ERD) models the business objects in a system and their relationships. It presents a bird view or an overall system picture by recognising and defining the existing business objects, albeit without providing their attributes and or data type details. Business analysts mostly use this model for a non-technical system presentation to stakeholders. Thus, a Conceptual Diagram of the Sports Club Management System presents only entities without their respective attributes or further details.

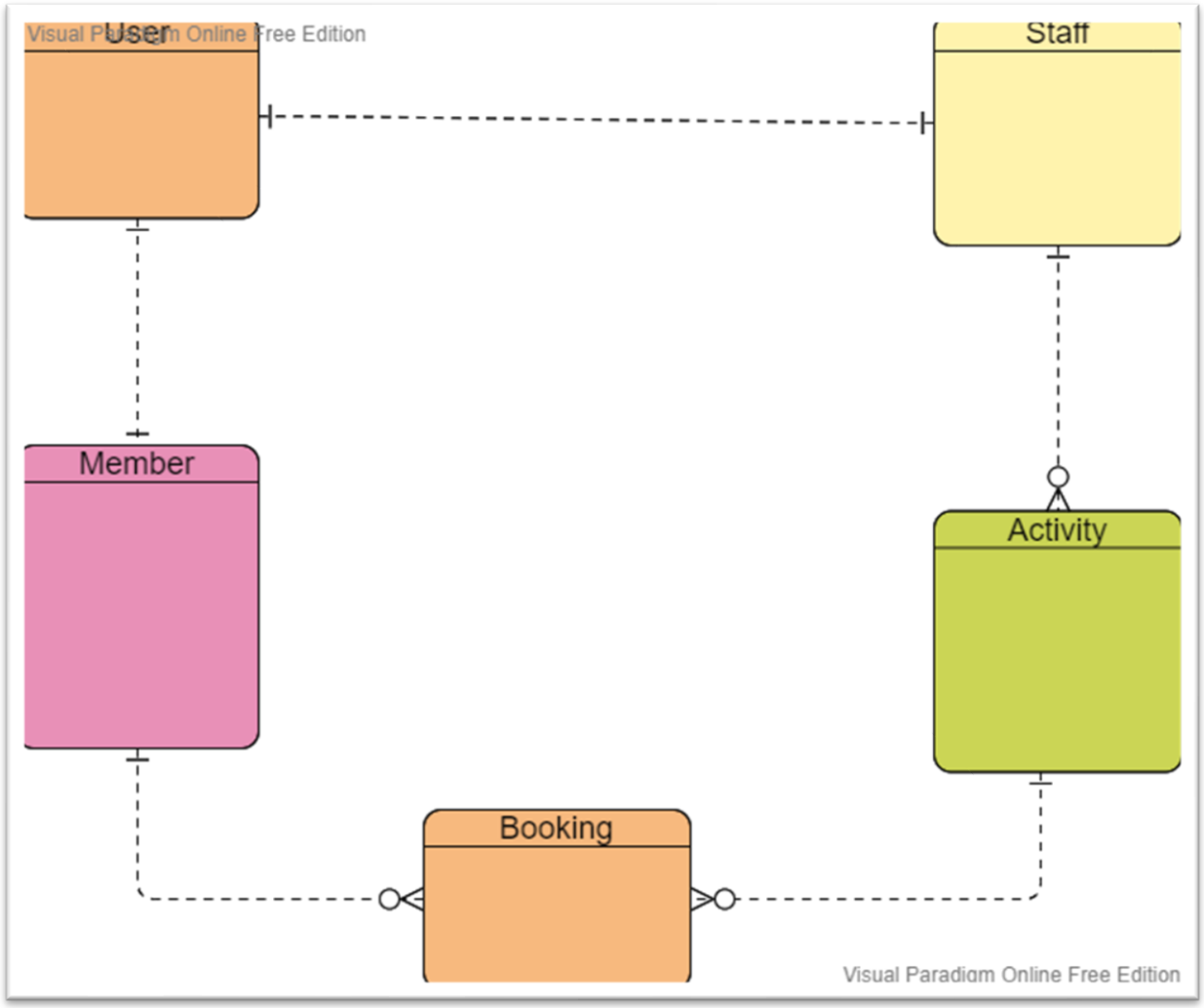


Figure 4 -Conceptual Data Model

Logical Data Model

A Logical Entity-Relationship Diagram (ERD) is a detailed version of a Conceptual ERD. It is developed to enrich a conceptual model by defining the columns explicitly in each entity. It is a dive into the internal structure of the identified entities, although, unlike the physical ERD, it only defines the attributes of these entities.

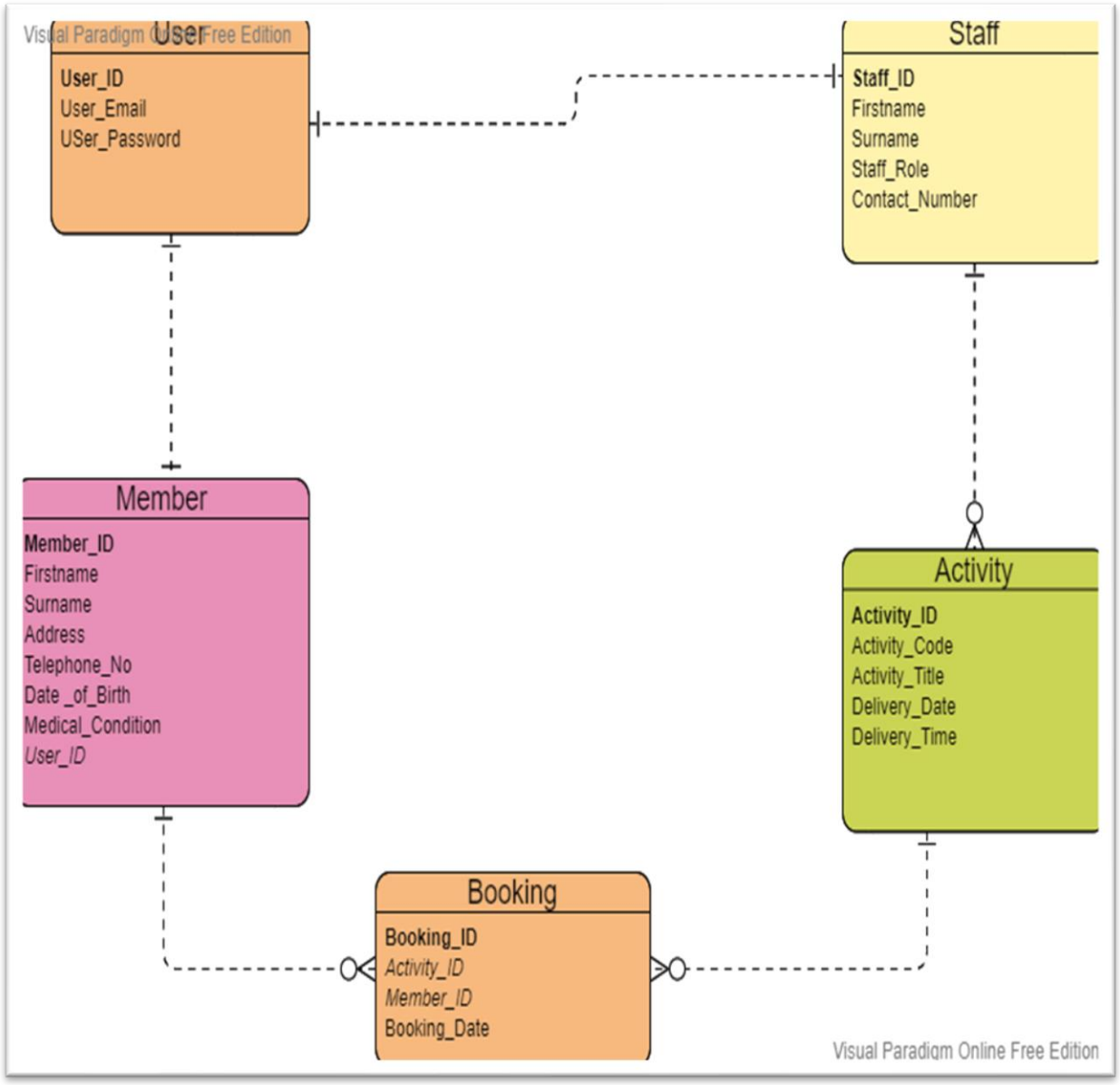


Figure 5 - Logical Data Model

Physical Data Model

Physical Entity-Relationship Diagram (ERD) represents the actual design blueprint of a relational database. A physical data model elaborates on the logical data model by assigning each column type, length, nullable, etc. Since a physical ERD represents how data should be structured and related in a specific DBMS, the convention, and restrictions of the actual database system – MySQL, the chosen DMBS, were considered in this design.

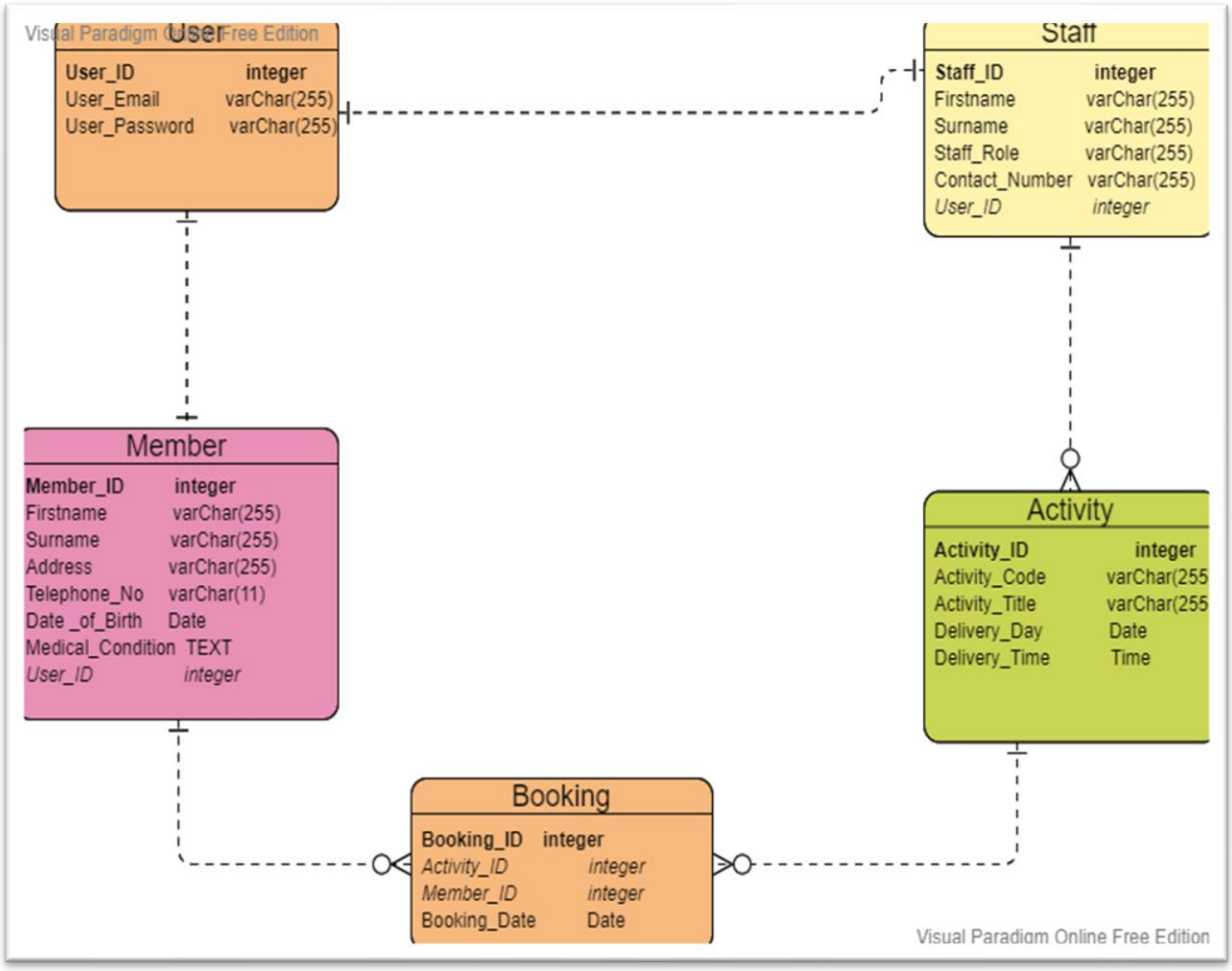


Figure 6 - Physical Data Model

Critical Decisions

IDs

A quick look at the entities and their attributes show that all IDs are defined as integer data type. This is so that IDs can be whole numbers randomly generated or automatically incremented for a user to be uniquely identified with. This auto-increment setting can be set to start with any number, but the default is one (1). Thus, it is ideal for entities identifying records with such unique numbers, to begin with the number 1. But it might be more appropriate to use a bigger integer number for others where such unique IDs are also used to identify people and their records uniquely.

String Column

Also, string columns are defined as a *varchar* data type that allows 255 characters. But columns such as medical_condition, which require more characters, are assigned a data type of Text. Date of Birth is set as a type of Date because our chosen Database Management System –

MySQL, has different types for a date - including DateTime and Timestamp, and both set date and time. Meanwhile, the Date type sets only Date in the format of YYYY-MM-DD. Thus, it is the best date data type for the Date of Birth.

Delivery Date

Again, for the *delivery_date* column of the activity entity, data type *Time* was assigned because MySQL data type TIME sets the time in HH: MM(SS) format, with seconds being optional. It will allow the user to set the time when creating an activity or searching for an activity.

Relationships

User – Member: Entity User's Primary Key is added to Member as a Foreign Key, and they have a one-to-one relationship. One user becomes one member. Besides, User is the parent entity while Member is the child entity and, as such, holds the Primary Key of User.

Member - Activity: This is a *many-to-many* relationship. Many members can sign up or book many activities, and many activities can have many members. Thus, this relationship was divided into two *one-to-many* relationships instead for ease and simplicity by creating a new table according to best practice. This new table is the *Booking* table with a one-to-many relationship between member and activity, as shown in the diagram.

Activity - Booking: This is a one-to-many relationship. And given that *activity* is the base entity, *Booking* has the Primary Key of *activity* as its Foreign Key.

Member - Booking: This is a one-to-many relationship. And given that *member* is the base entity, *Booking* has the Primary Key of *member* as its Foreign Key.

User – Staff: Entity User's Primary Key is added to Member as a Foreign Key, and they have a one-to-one relationship. One user becomes one member. Besides, User is the parent entity while Staff is the child entity and, as such, holds the Primary Key of User.

Staff - Activity: This is a one-to-many relationship. And given that *staff* is the base entity, *activity_staff* has the Primary Key of *staff* as its Foreign Key.

DATABASE CREATION USING STRUCTURED QUERY LANGUAGE

This section creates the Sports Club Management System database using the Data manipulation Language called SQL - Structured Query Language. This is the standard language for interrogating and interacting with relational database management systems. It is the standard language for storing, manipulating, and retrieving data from relational databases.

It allows us to create databases, tables, and records, and to read, update, and delete them. Examples of relational database management systems using SQL are MS Access, MS SQL Server, MySQL, Oracle, etc. Although Structured Query Language is the standard language for all Relational Database Management Systems, some still have proprietary differences in their application. But for the database implementation of the Sports Club Management System database, the relational database management system chosen is MySQL.

Database Creation

Database creation using the Structured Query Language can be done using the SQL CREATE DATABASE statement. The syntax is as follows:

CREATE DATABASE IF NOT EXISTS *database_name;*

Thus, for the sports club management system databases, it can be created using SQL with the following command:

CREATE DATABASE IF NOT EXISTS sports_club_management_system;

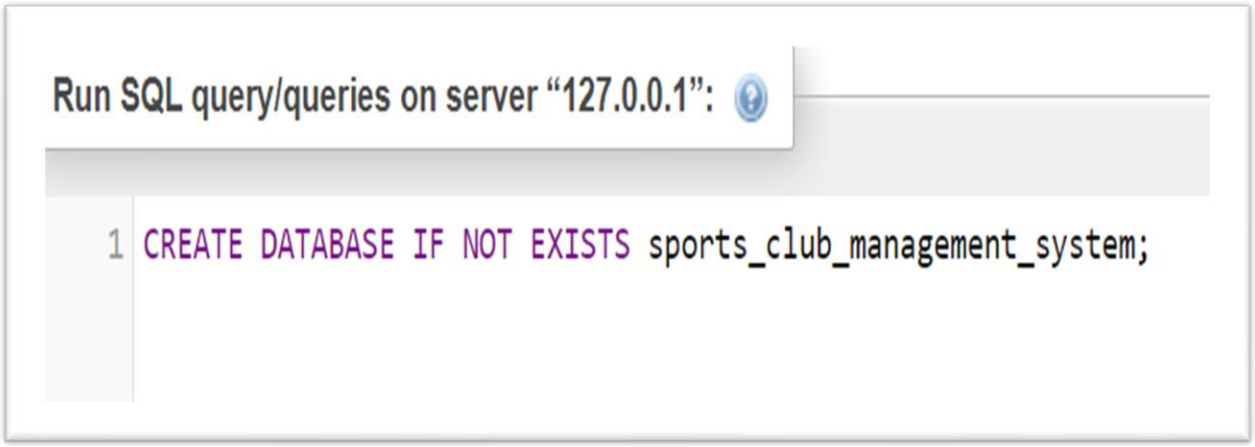


Figure 7 - SQL Database Creation Using MySQL

The above statement without the "If not exists" will still work if the database does not exist in MySQL RDBMS. But if it does exist, MySQL will throw an error saying the database already exists.

Tables Creation

We will create tables to represent and precisely reflect the entities on the Entity Relationship Diagram. These tables are User, Member, Staff, Activity, Activity_Member, and Activity_Staff.

The syntax for table creation using the Structured Query Language is as follows:

```
CREATE TABLE table_name (
  column1 data type,
  column2 data type,
  column3 data type,
 ....
);
```

Thus, we will create the required tables as follows:

1. User Table

```
CREATE TABLE IF NOT EXISTS user_table (
  user_id integer,
  user_email varchar (255),
  user_password varchar (255)
);
```

Figure 8- User Table SQL Command

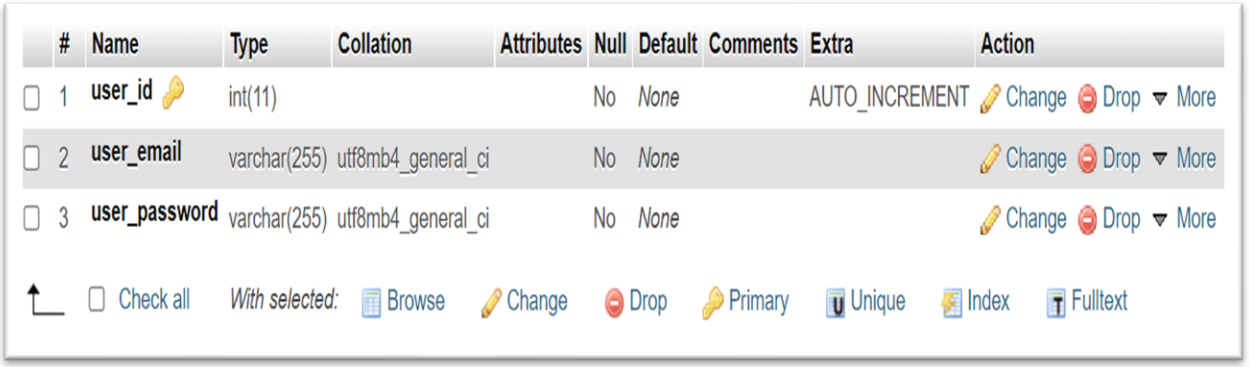


Figure 9 - User Table Structure

The above table is somewhat differently named because the word “user” is a reserved word in MySQL, so we had to rename it to use it. We could have just modified the Physical ERD model by using this exact name, but that would mean we would not have had the opportunity to explain this vital information about MySQL RDMS.

2. Member Table

```
CREATE TABLE IF NOT EXISTS member (
        member_id integer AUTO_INCREMENT PRIMARY KEY,
        firstname varchar (255) NOT NULL,
        surname varchar (255) NOT NULL,
        address varchar (255) NOT NULL,
        telephone_no varchar (11) NOT NULL,
        date_of_birth Date NOT NULL,
        medical_condition TEXT NOT NULL,
        user_id integer NOT NULL,
        FOREIGN KEY (user_id)
        REFERENCES user_table (user_id)
        ON UPDATE RESTRICT
        ON DELETE CASCADE
);
```

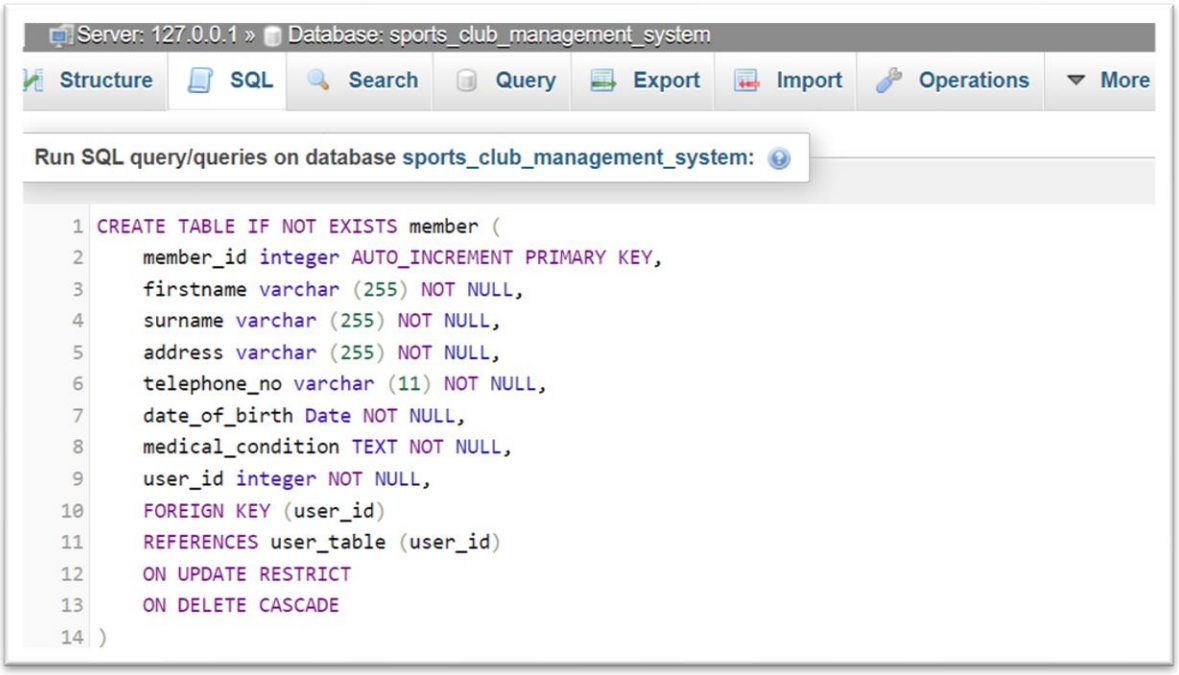


Figure 10 - Create Staff Table SQL Command

3. Staff Table

CREATE TABLE IF NOT EXISTS *staff* (

staff_id integer AUTO_INCREMENT PRIMARY KEY,
firstname varchar (255) NOT NULL,
surname varchar (255) NOT NULL,
staff_role varchar (255) NOT NULL,
contact_no varchar (11) NOT NULL,
user_id integer NOT NULL,
FOREIGN KEY (user_id)
REFERENCES user_table (user_id)
ON UPDATE RESTRICT
ON DELETE CASCADE

);

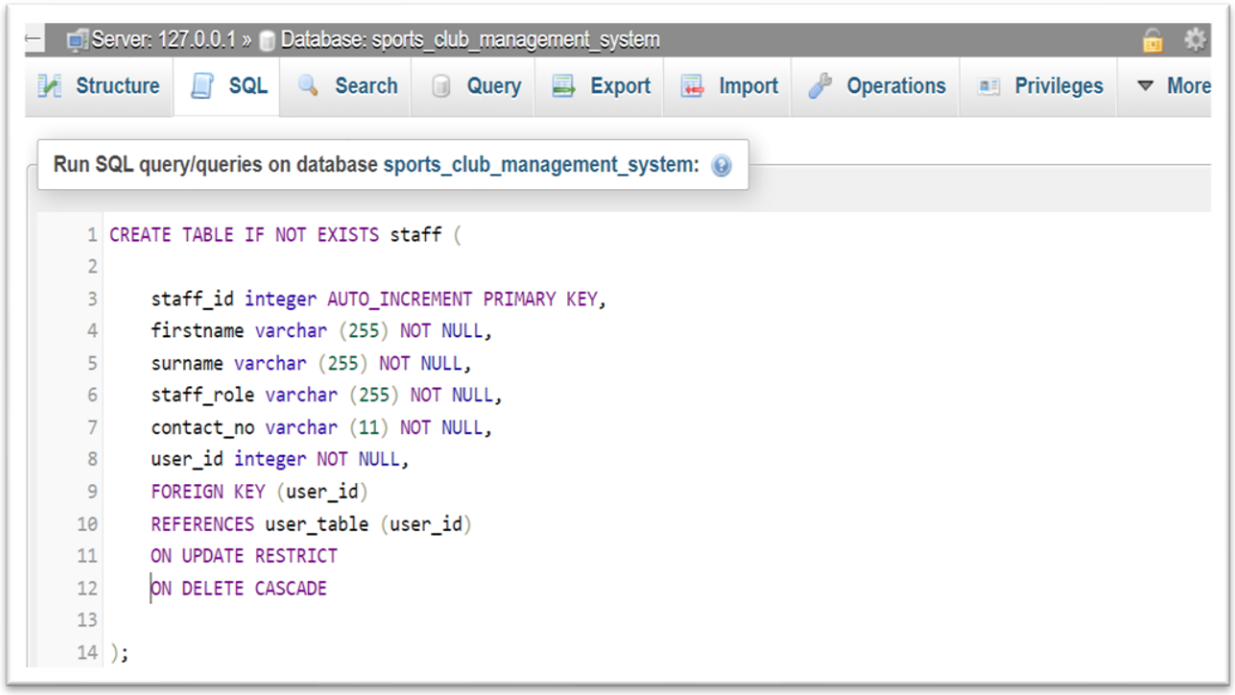


Figure 11 - Create Staff Table SQL Command

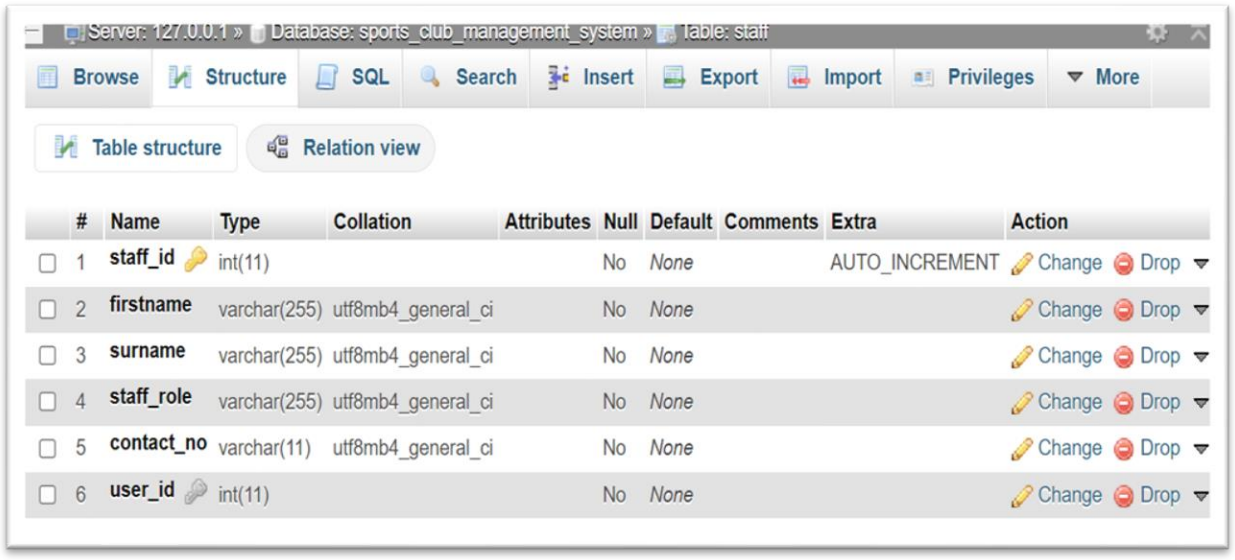


Figure 12 - Staff Table Structure

4. Activity Table

```
CREATE TABLE IF NOT EXISTS activity (
    activity_id integer NOT NULL AUTO_INCREMENT PRIMARY KEY,
    activity_code varchar (255) NOT NULL,
    activity_title varchar (255) NOT NULL,
    delivery_day   DATE NOT NULL,
    delivery_time Time NOT NULL
);
```

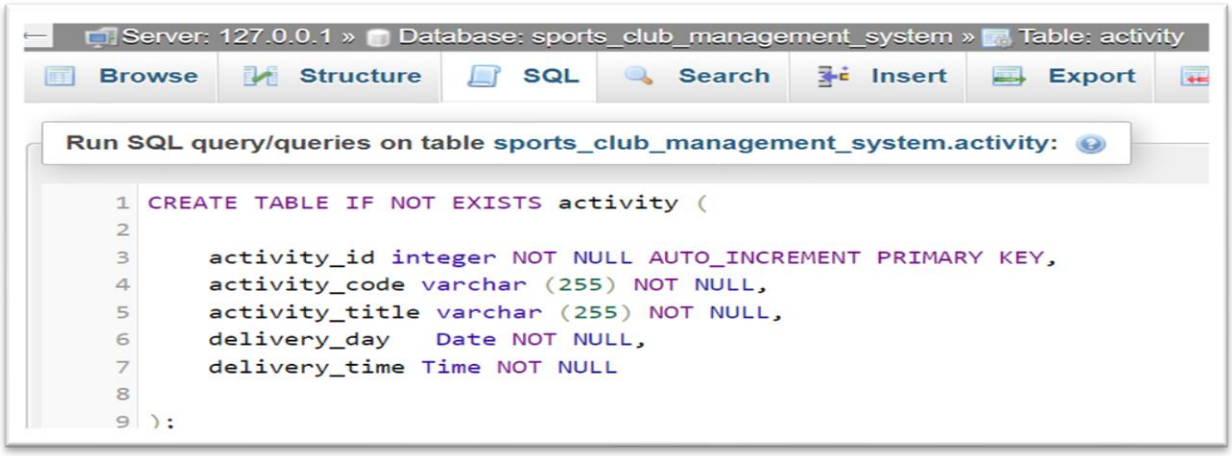


Figure 13 - Activity Table Creation

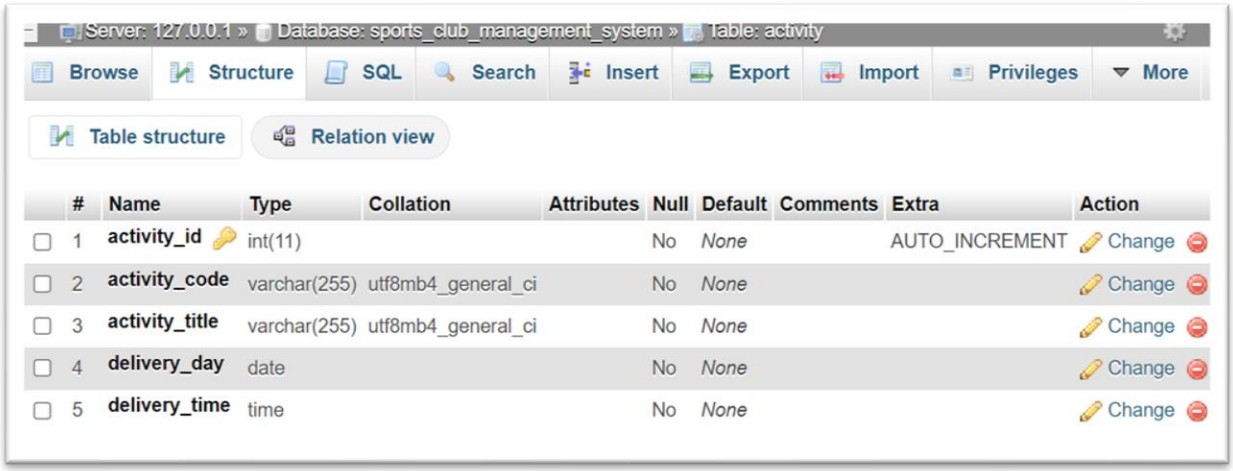


Figure 14 - Activity Table Structure

5. Booking Table

```
CREATE TABLE IF NOT EXISTS booking (

    booking_id integer NOT NULL AUTO_INCREMENT PRIMARY KEY,
    booking_date Date
    member_id integer NOT NULL,
    activity_id integer NOT NULL,
    FOREIGN KEY (member_id) REFERENCES member (member_id),
    FOREIGN KEY (activity_id) REFERENCES activity (activity_id)
    ON UPDATE RESTRICT
    ON DELETE CASCADE

    );
```

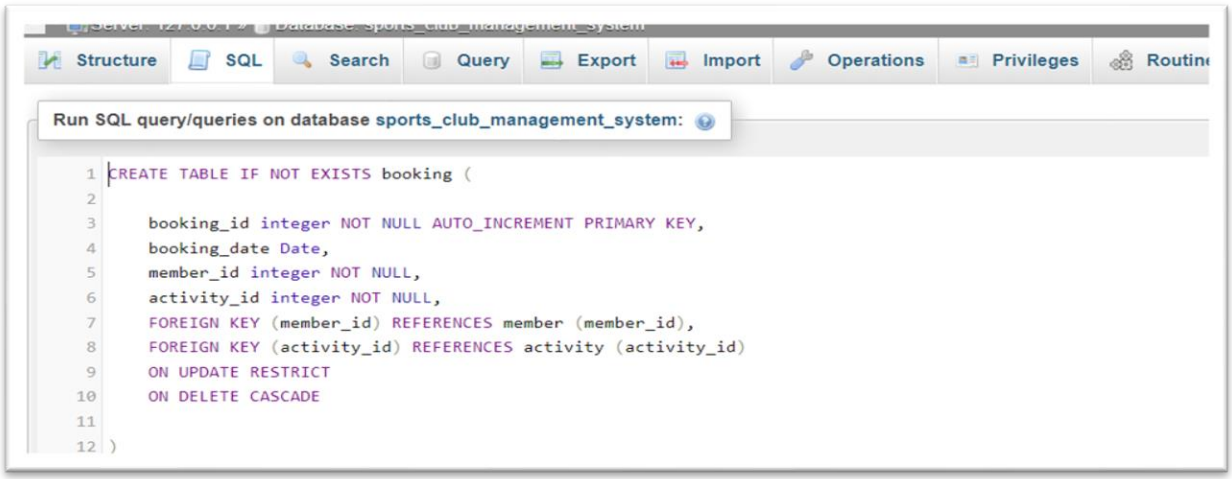


Figure 15 - Booking Table Creation

Figure 16 - Booking Table Structure

SQL Queries

This section demonstrates the use of the Structured Query Language in the Sports Club Management System. It queries the database for some defined information. However, these tables will be populated first to have the desired results. Thus, each table will have an INSERT INTO command, populating the table with some dummy records before the actual SELECT query is performed.

User Table

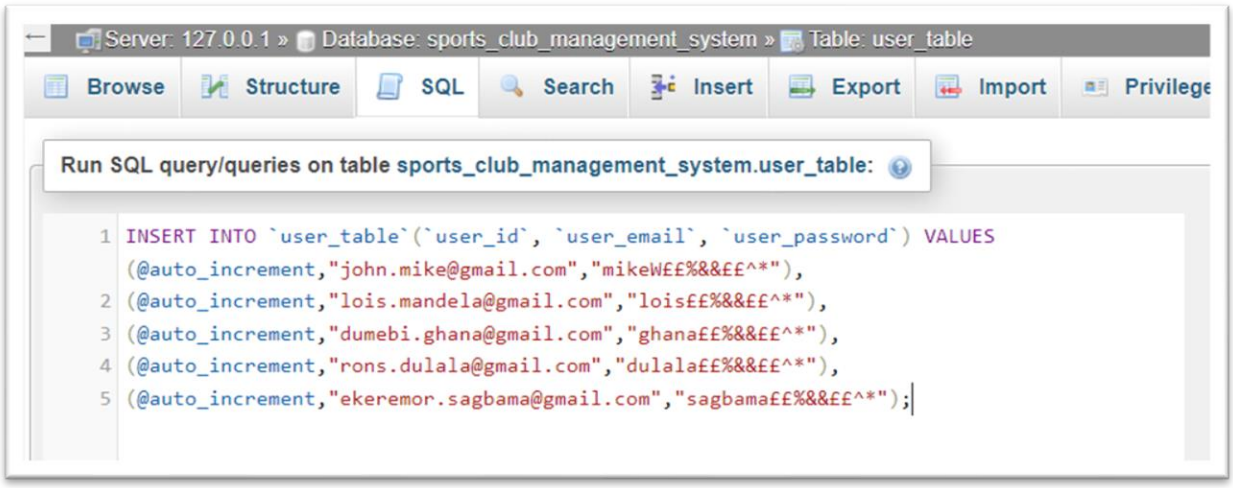


Figure 17 - User Table Multiple Records Creation

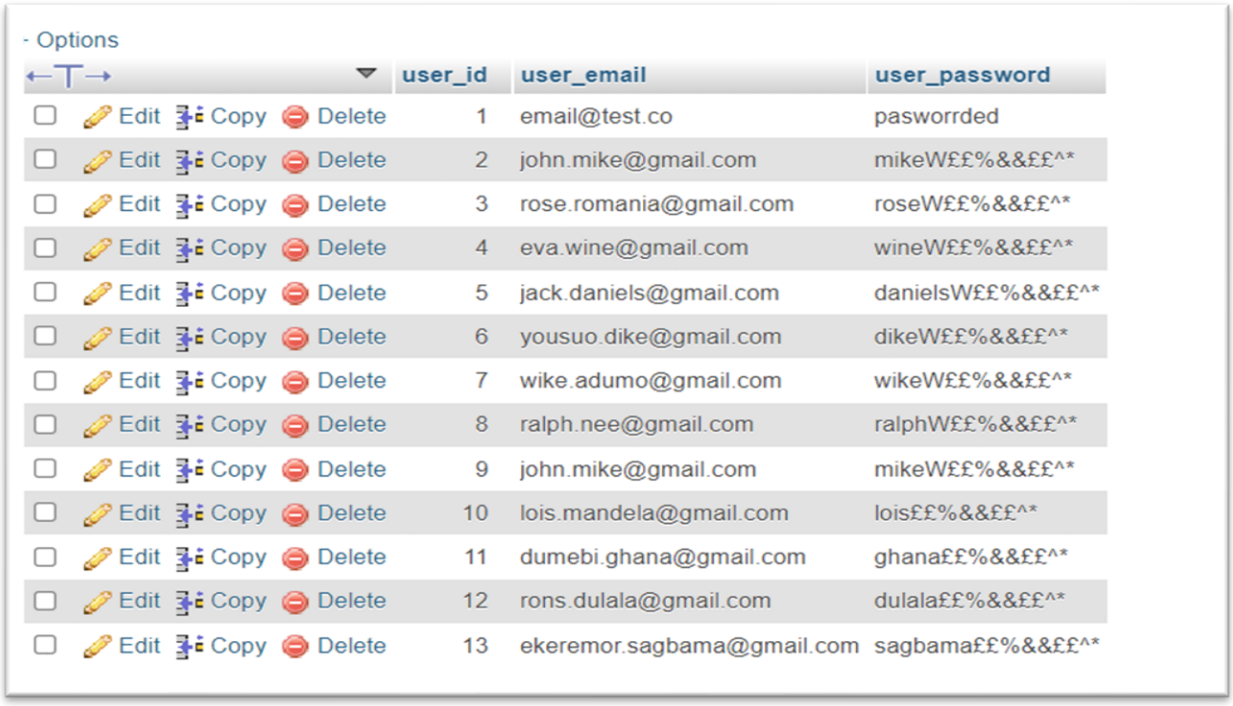
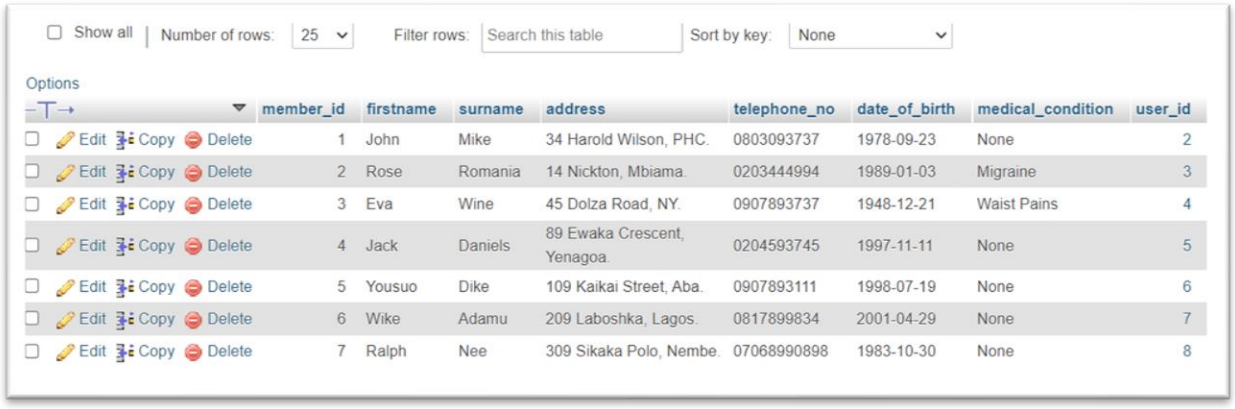


Figure 18 - User Table Multiple New Records Creation

Member Table



Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Options

	member_id	firstname	surname	address	telephone_no	date_of_birth	medical_condition	user_id
Edit Copy Delete	1	John	Mike	34 Harold Wilson, PHC.	0803093737	1978-09-23	None	2
Edit Copy Delete	2	Rose	Romania	14 Nickton, Mbiama.	0203444994	1989-01-03	Migraine	3
Edit Copy Delete	3	Eva	Wine	45 Dolza Road, NY.	0907893737	1948-12-21	Waist Pains	4
Edit Copy Delete	4	Jack	Daniels	89 Ewaka Crescent, Yenagoa.	0204593745	1997-11-11	None	5
Edit Copy Delete	5	Yousuo	Dike	109 Kaikai Street, Aba.	0907893111	1998-07-19	None	6
Edit Copy Delete	6	Wike	Adamu	209 Laboshka, Lagos.	0817899834	2001-04-29	None	7
Edit Copy Delete	7	Ralph	Nee	309 Sikaka Polo, Nembe.	07068990898	1983-10-30	None	8

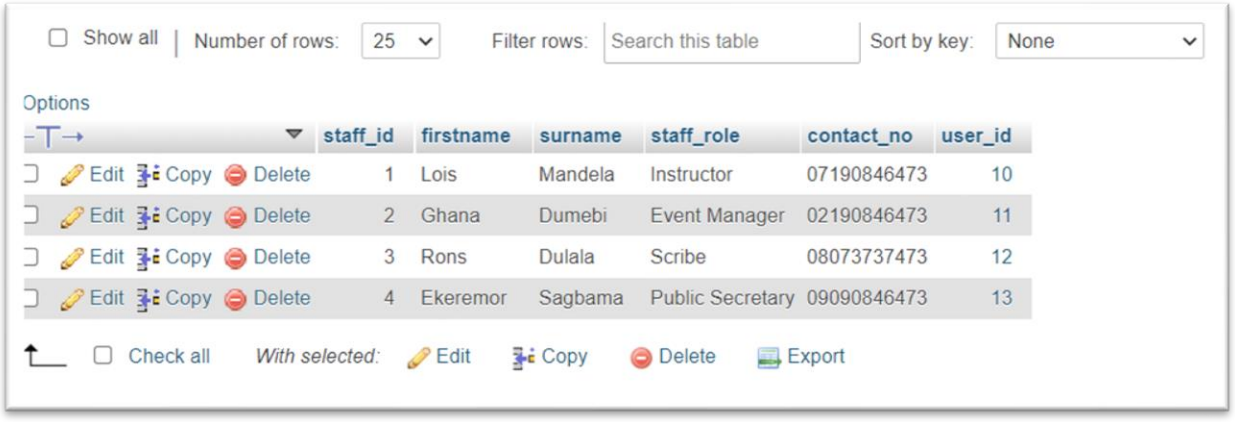
Figure 19 - Member table New Records

Staff Table

Run SQL query/queries on table sports_club_management_system.staff:

```
1 INSERT INTO `staff`(`staff_id`, `firstname`, `surname`, `staff_role`, `contact_no`,
  `user_id`) VALUES
2 (@auto_increment,"Lois","Mandela","Instructor","07190846473"),
3 (@auto_increment,"Ghana","Dumebi","Event Manager","02190846473"),
4 (@auto_increment,"Rons","Dulala","Scribe","08073737473"),
5 (@auto_increment,"Ekeremor","Sagbama","Public Secretary","09090846473");
```

Figure 20 - Staff Table Multiple Records Creation



Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Options

	staff_id	firstname	surname	staff_role	contact_no	user_id
Edit Copy Delete	1	Lois	Mandela	Instructor	07190846473	10
Edit Copy Delete	2	Ghana	Dumebi	Event Manager	02190846473	11
Edit Copy Delete	3	Rons	Dulala	Scribe	08073737473	12
Edit Copy Delete	4	Ekeremor	Sagbama	Public Secretary	09090846473	13

Check all *With selected:* Edit Copy Delete Export

Figure 21 - Staff Table New Records

Activity Table

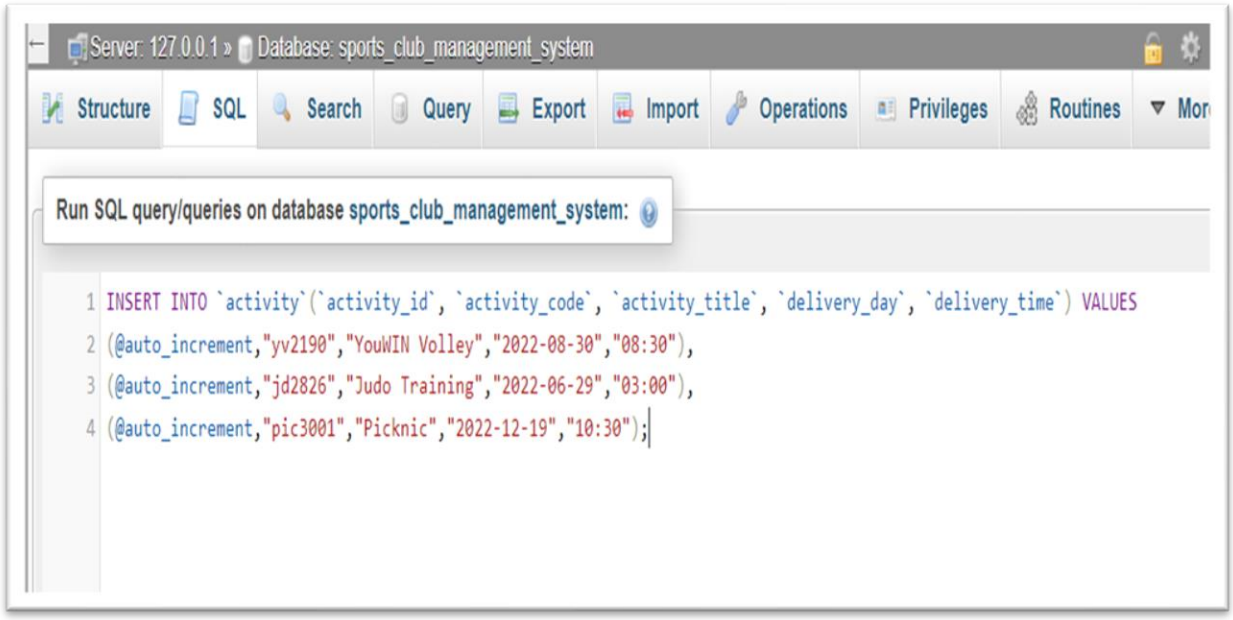


Figure 22 - Activity Multiple New Records Creation

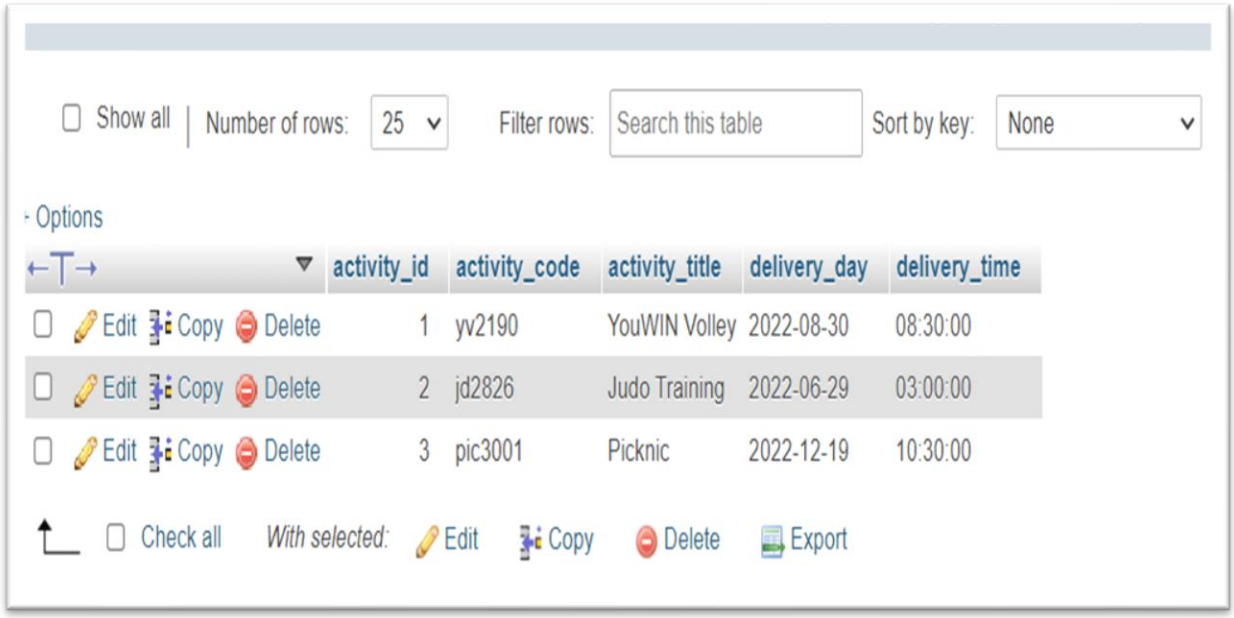


Figure 23 - Activity Table New Records

Booking Table

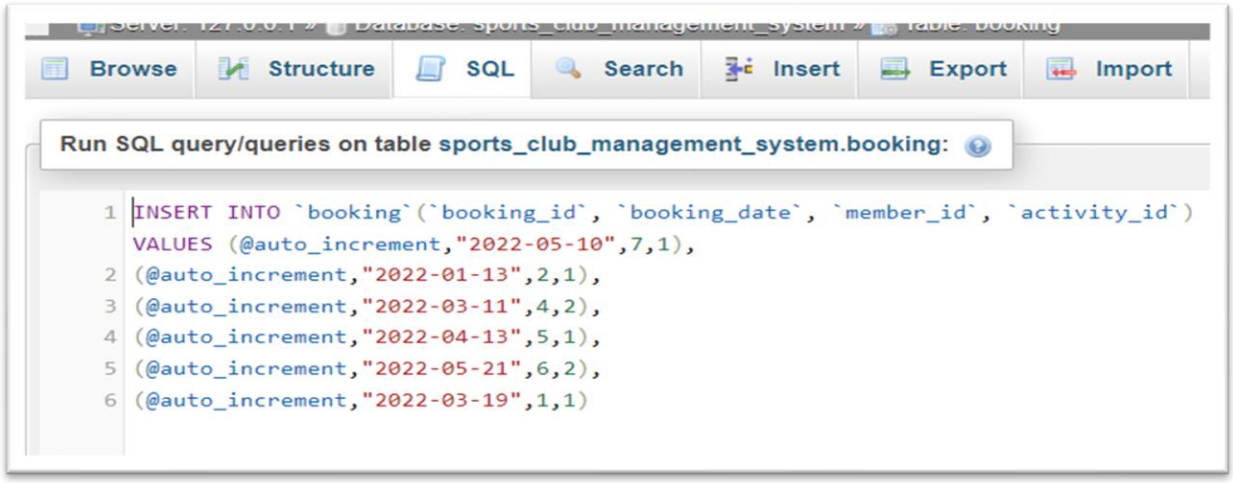


Figure 24 - Booking Table Multiple Records Creation

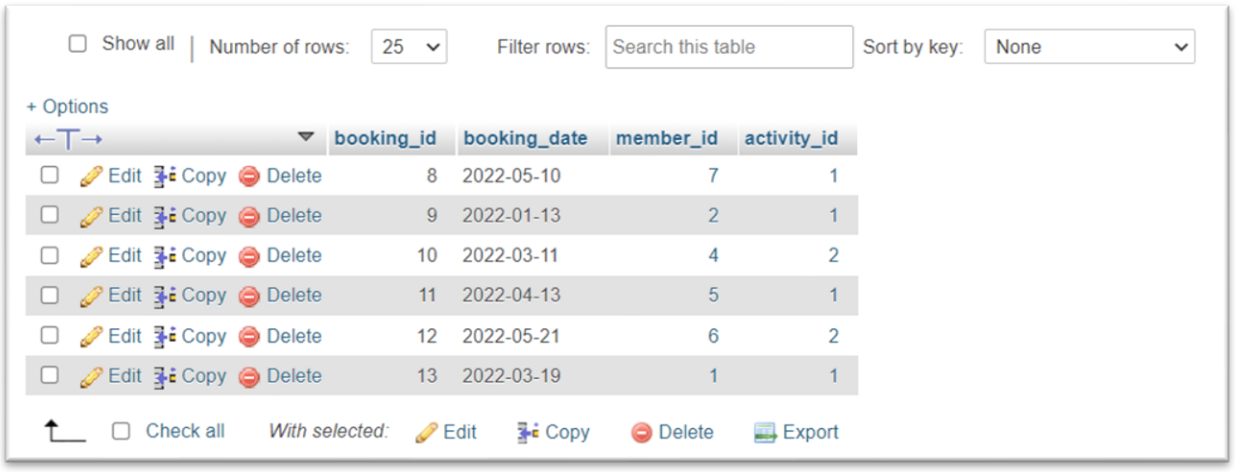


Figure 25 -Booking Table New Records

Query To View Current Weekly Bookings

To query the Sports Club Management System to display current weekly bookings, we will use the week() and now() MySQL functions and a double INNER JOIN clause to query three tables – booking, member and activity using Alias to have the result as we want it.

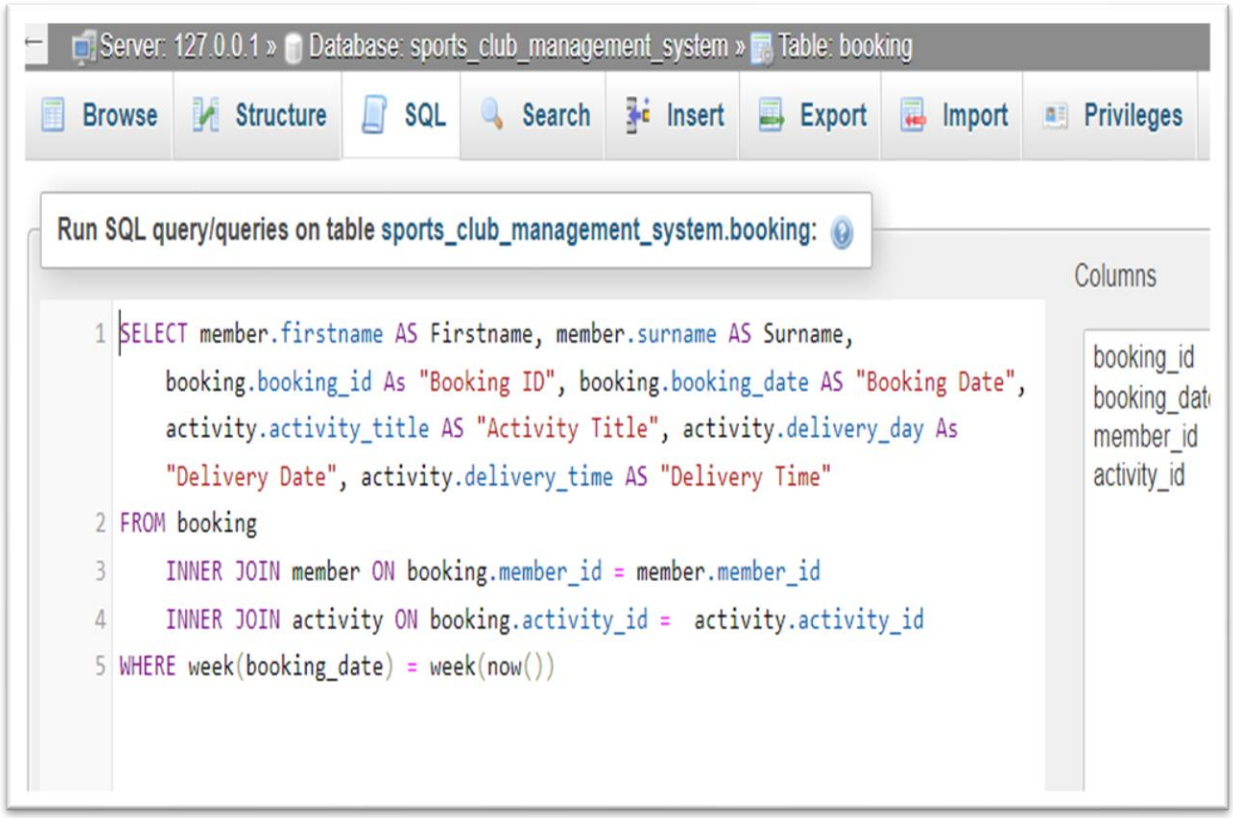


Figure 26 - Viewing Current Weekly Booking SQL

The output of the above SQL query is as shown below:

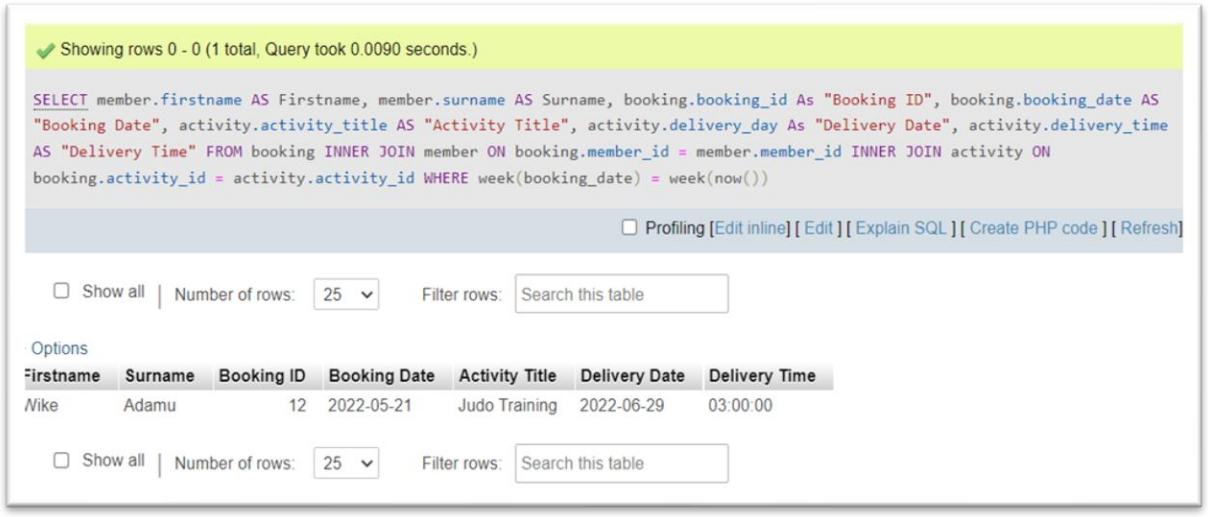


Figure 27 - View Current Weekly Booking Output

Query To Display Current Weekly Activities

To display the current weekly activities, we still use the week and now MySQL functions in selecting all columns from activity.

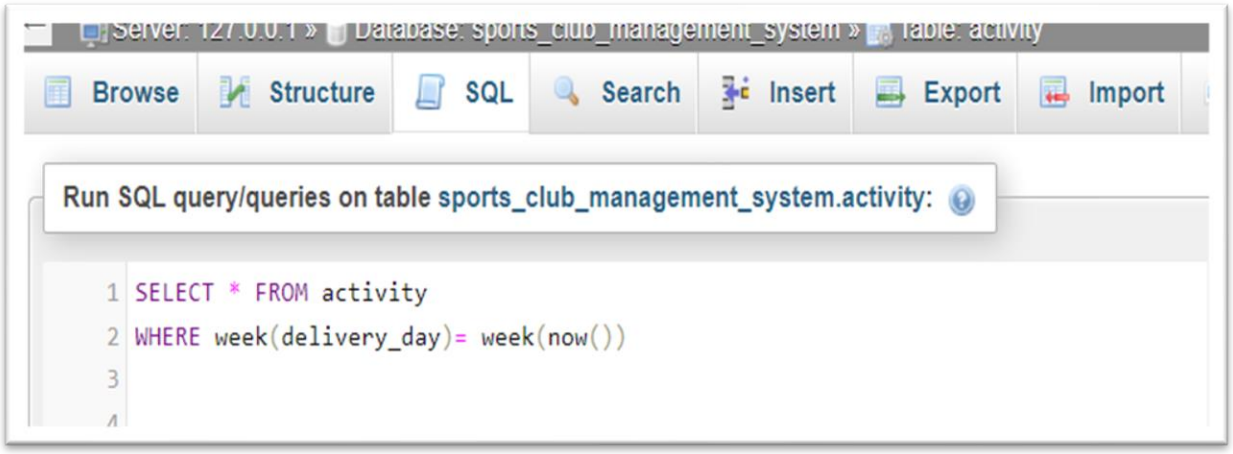


Figure 28 - Display Current Weekly Activities SQL

The output shows no records as the query was performed on 16 May 2022, whereas there are no activities slated for May on the *Activity* table.

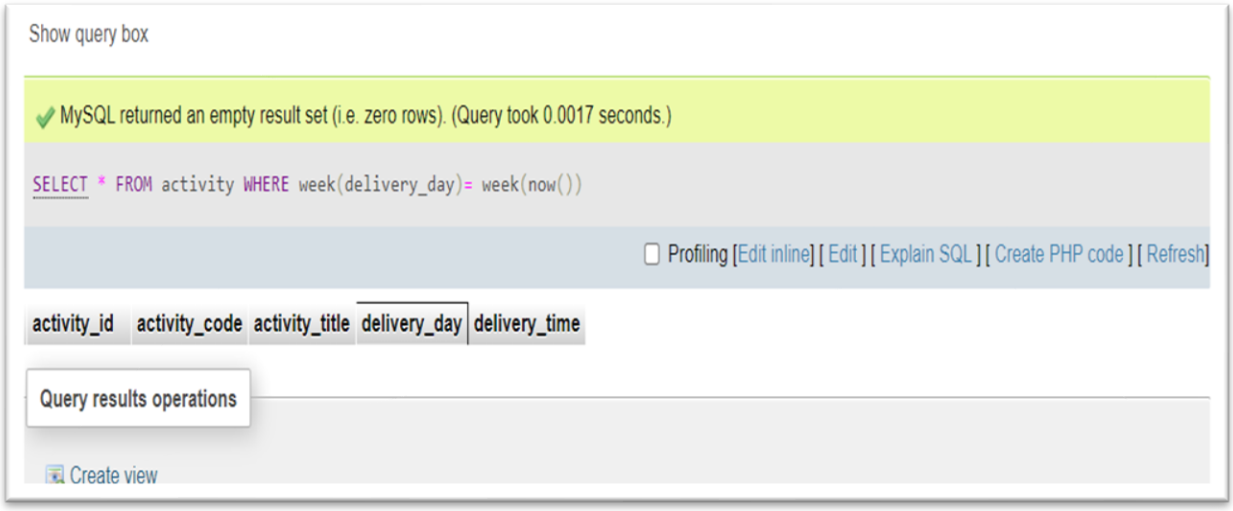


Figure 29 - Display Current Weekly Activities Output

Query To Show Most Active Members Monthly

To show the most active Sports Club Management System members, we use the *count* () MySQL function to query the booking table using the booking_id column to check the number

of bookings done within a month. The SQL command is given as follows:

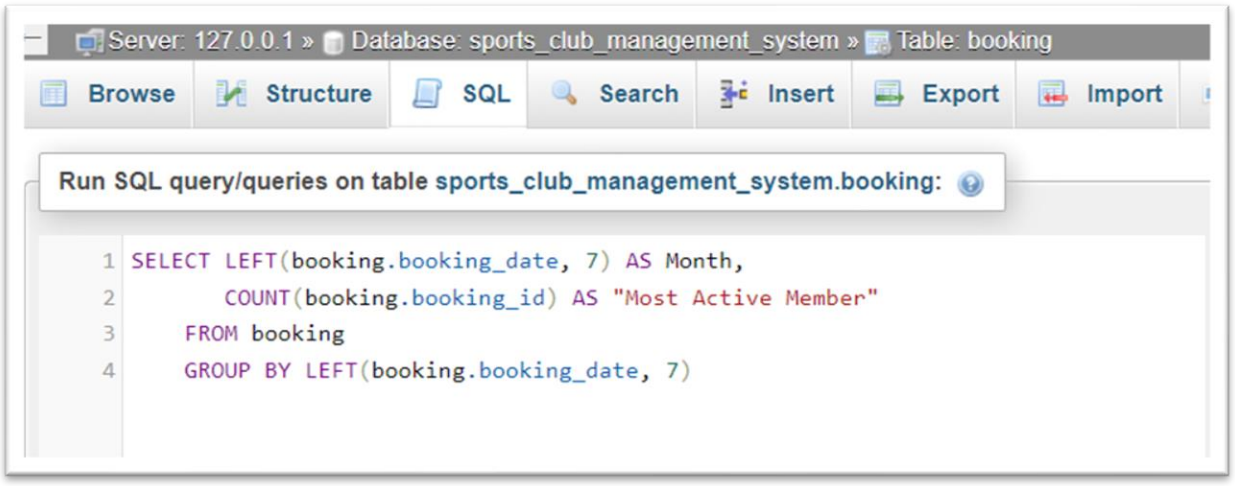


Figure 30 - Most Active Member SQL

The result of the above query is shown below:

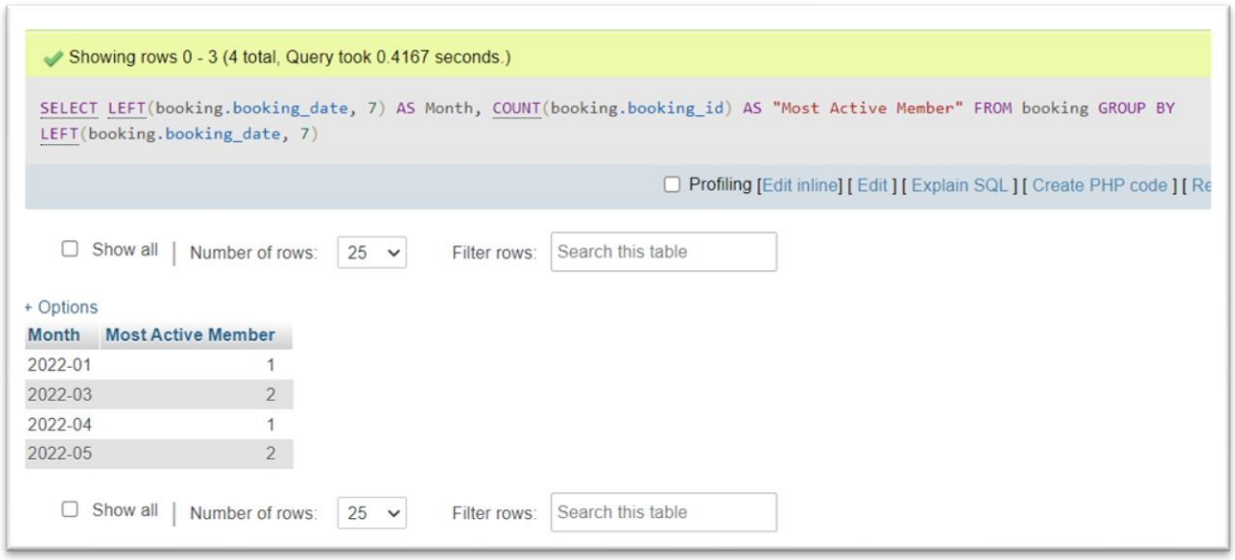


Figure 31 - Most Active Member Query Output

DESIGN DECISIONS AND CHALLENGES

This section discusses the various design decisions taken during the analysis, design, and implementation of the Sports Club Management System.

Entities Identification

The given scenario provides objects recognised as providing information and performing functions. These entities are Member, Staff, and Activity. It was observed from the scenario description that the entire system functions around these three entities. Thus, we decided to have a generic entity that captures the attributes and operations they all share. This produced

the User entity on the Usecase Diagram. Also, member entity and staff entities are identified as Actors. As such, they should inherit the user entity's usecases in addition to their respective usecases, as clearly shown on the usecase diagram.

Relational Database Management System Adoption

Although any choice amongst the popular Relational Database Management Systems such as Oracle, MS SQL Server, PostgreSQL, etc., would suffice for the implementation of the Sports Club Management System, the open-source MySQL RDBMS was adopted for the following reasons:

- ✓ *Cross-Platform Compatibility and Flexibility*

MySQL provides cross-platform flexibility that can run on macOS, Windows, Linux, Solaris, OS2, etc., with excellent API support for all major languages for easy integration with languages such as .NET, Java, PHP, Perl, Python, C++, etc. It is also part of the Linux Apache MySQL PHP (LAMP) server used worldwide for web applications. Thus, whichever programming language(s) is chosen for the backend development of the Sports Club Management System, it will find support for the MySQL database.

- ✓ *Open Source*

MySQL is open-source software. That means it is free and always will be. The source code is available for anyone interested in modifying it according to their needs, albeit a licence is required for distribution. In effect, this RDBMS is available for free to implement the Sports Club Management System.

- ✓ *Security*

One of the most critical aspects of digital systems is security. Adopted by well-known enterprises such as Facebook, Wikipedia, etc., MySQL provides a good security layer protecting sensitive information from intruders. For example, it uses plugin-specific algorithms to encrypt and store passwords. One of such is MD5. This is critical for the Sports Club Management system.

- ✓ *Scalability*

Another critical aspect of the business is growth possibility. Sports Club Management System will grow, which means more and more users will be added, and more activities created, among several other possible functionalities, in the future. But the question is, can the adopted

RDBMS allow scaling? MySQL can scale both horizontally and vertically and provide clusters to handle any possible load increase in the future.

CHALLENGES

During this work, few challenges were encountered. The brevity of information provided or the blank spaces in the scenario sort of made it a bit difficult to figure out or fill in. I was working with a developer's view and had to fill in some blank spaces with what I should obtain. For example, there is no mention of user login or authentication in the scenario, but properly designing a database for an application like this would require the same, so I filled it in.

CONCLUSION

The Sports Club Management System has been implemented within the scope of this assignment, yet it is not without omissions as always. By and large, all queries produced expected results.

REFERENCES

Ambler, S., *Relational Databases 101: Looking at the Whole Picture.* [Online]
Available at: http://www.agiledata.org/essays/relationalDatabases.html
[Accessed 15 5 2022].

Anon., n.d. *Visual Paradigm.* [Online]
Available at: https://www.visual-paradigm.com/
[Accessed 14 May 2022].

Hellerstein, J. M., Stonebraker, M. & Hamilton, J. R., 2007. *Architecture of a Database System.* [Online]
Available at: http://db.cs.berkeley.edu/papers/fntdb07-architecture.pdf
[Accessed 15 5 2022].

Pressman, R. S. & Maxim, B. R., 2015. *Software Engineering_A Practitioner's Approach.* Eighth ed. New York: McGraw-Hill Education.

Rosa, N. S., Cunha, P. R. F. & Justo, G. R. R., 2004. An approach for reasoning and refining non-functional requirements. *Journal of the Brazilian Computer Society,* 10(1), pp. 62-84.

Soler, J. et al., 2010. *A web-based e-learning tool for UML class diagrams.* [Online]
Available at: http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.ieee-000005492473
[Accessed 15 5 2022].

Sommerville, I., 2016. *Software Engineering.* Tenth ed. Essex: Pearson Education Limited.

Tsadimas, A., Nikolaidou, M. & Anagnostopoulos, D., 2009. *Handling Non-functional Requirements in Information System Architecture Design.* [Online]
Available at: http://galaxy.hua.gr/~tsadimas/papers/icsea09.pdf
[Accessed 15 5 2022].