# PROJECT NAME: SCHOOL TIMETABLE SCHEDULING SYSTEM

| GROUP MEMBERS | Adm. Number |
|---|---|
| Lucas Nancy Auma- | 1045410 |
| Caro Kariuki Wanjiku - | 1049097 |
| Trevor Serwanga- | 1049421 |
| Evans Raila- | 1049468 |
| Dennis Chivilli- | 1049151 |

22nd November, 2024

# Overview of the Project

A School Timetable Scheduling system is a system that focuses on automating the scheduling of classes and exams in advanced tools and which simplifies and optimizes the academic planning. It ensures that teachers, students, classrooms, and resources are allocated efficiently without institutional policy conflicts and disruptions. It also offers analysis tools for efficiency monitoring to support evidence-based decision-making for administrators.

# Rationale

Due to the significant burden of the administration that generates the schedule of classes manually this imposes the importance of implementing a School Timetable Schedule System that automates the scheduling of the classes and exams. The system ensures a well-organized and smooth academic environment to the students and teachers of a school. The automation saves resources and time allowing the users of the system focus on their crucial tasks and improving the overall performance of the system.

# Objectives

- To assign students, time slots and teachers classes and exam and confusion among the system users
- To avoid clashes between classes and exams and any other activities scheduled simultaneously.
- To avoid overburdening among stuff by distributing teaching assignment equally.
- To reduce resource conflict among the system users
- To enable easy input of constraints and changes by the administrators.
- To enforce break times maximizing the use of school work long hours
- To maximize the optimal utilization of school resources such as whiteboards, projectors and classrooms available in the school.
- To ensure no teacher or stuff is doubled booked in the same time slot.

-Companies that use a School Timetable Scheduling System in the world include;

i. **Zunia**
ii. **Fedena**
iii. **MyStudyLife**
iv. **Untis**
v. **Untis Express**

# System Design

The tool used for the design of the Entity-Relation diagram Luna Modeler. Luna Modeler is a data modelling and database design tool for SQL Server, PostgreSQL, MariaDB and other relational databases that is used to generate most of the popular databases draw ER diagrams, generate documentation or SQL scripts easily. The reason why we used this tool is because of the following advantages

## The Advantages of Luna Modeler

- It is user-friendly since it simplifies the process of designing ER diagrams by introducing drag and drop functionalities which are used to create and modify tables, rows, columns and relationships.
- It also supports multiple popular databases such as PostgreSQL, MySQL and SQLite.
- It offers efficient tools for creation of ER diagrams that visually gives a clear representation of database schema, which makes it easier for defining primary keys, foreign keys, indexes and relationships.
- It also has built-in validation tool that detect any inconsistence or errors in the schema design e.g. invalid relationships, missing keys.
- It also allows customization of relationship cardinalities, table attributes and column data fields to ensure that they meet database requirements.
- It is accessible to developers across every operating system since it available in windows, MacOs and Linux.

-We used Luna Modeler to design most of our database diagrams such as

## 1. Table Structures

Our system had fourteen tables in total and below include all the tables that are interconnected and interact to manage all the system data.

### ✔ Users Table

It stores information about all the users interacting with the system.

| USERS | | |
|---|---|---|
| User_id | integer | NN |
| Username | varchar(50) | NN |
| Password | varchar(50) | NN |
| Role | varchar(20) | NN |
| Created_at | timestamp | NN |
| Update_at | timestamp | NN |
| Status | varchar(50) | NN |

Figure 1.1: User Table

-The table has password and username attributes to store user credentials

-It handles all the login and access control

-User_id is the primary key

It also maintains user roles such as Admin, Teachers and Students

-The created_at and update_at attributes makes the table to track at what time the user logged in the system and at what time the account was updated.

-The status attribute shows if a user account is active or inactive incase a student dropouts or a teacher resigns

## ✔ Admin Table

| ADMINISTRATORS | | |
|---|---|---|
| Admin_id | integer | NN |
| Admin_name | varchar(100) | NN |
| Email | varchar(100) | NN |
| Phone_number | integer(15) | NN |
| User_id | integer | NN |

Figure 1.2: Admin Table

It stores the information of all the administrators managing the system.

The table facilitates adding, deleting and updating the users of the system

It also tracks the activity logs of the system for monitoring and troubleshooting it

It verifies user login credential

## ✔ Student Table

| STUDENT | | |
|---|---|---|
| Student_id | integer | NN |
| First_Name | varchar(255) | NN |
| Last_Name | varchar(255) | NN |
| Gender | varchar(255) | NN |
| Email | varchar(255) | NN |
| Course | varchar(255) | NN |
| User_id | integer | NN |
| Date_of_birth | date | NN |

Figure 1.3: Student Table

Student_id is the unique identifier

It stores all details about all the student in the system.

## ✔ Teachers Table

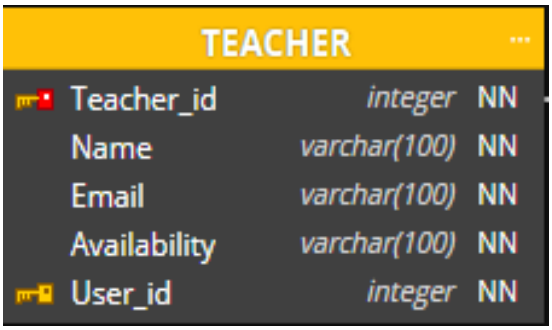


Figure 1.4: Teachers Table

Teacher_id is the unique identifier

It stores details about all the teachers in the system

## ✔ Rooms Table

| ROOM | | |
|---|---|---|
| Room_id | integer | NN |
| Room_Name | varchar(50) | NN |
| Capacity | integer(50) | NN |
| Building | varchar(50) | NN |

Figure 1.5: Room Table

Room_id is the unique identifier of the table.

It contains the details of classroom or exam rooms.

It also  allocates rooms for activities classes and exams

## ✔ Course Table

| COURSE | | |
|---|---|---|
| Course_id | integer | NN |
| Course_name | varchar(100) | NN |
| Department | varchar(100) | NN |
| Credits | integer | NN |
| Description | text(100) | NN |

Figure 1.6: Course Table

The table confirms all the courses offered by the school

The table tracks all the courses assigned to classes

## ✔ Class table

| CLASS | | |
|---|---|---|
| Class_id | integer | NN |
| Room_id | integer | NN |
| Schedule_id | integer | NN |
| Semester | varchar(100) | NN |
| Year | integer | NN |
| Status | varchar(100) | NN |
| Time_slot | varchar(100) | NN |
| Teacher_id | integer | NN |
| Course_id | integer | NN |

Figure 1.7: Class Table

Class_id is the primary key

Contains the details about classes of specific courses

## ✔ Exam Table

| Exams | | |
|---|---|---|
| Exam_id | integer | NN |
| Exam_date | date | NN |
| Start_time | time | NN |
| End_time | time | NN |
| Status | varchar | NN |
| Course_id | integer | NN |
| Room_id | integer | NN |

Figure 1.8: Exam Table

Has the exam_id as the unique address for the exams.

The table stores information about the exams i.e. the exam date, time and the subject.

## ✔ Schedule Table

| SCHEDULE | | |
|---|---|---|
| Schedule_id | integer | NN |
| Start_time | time | NN |
| End_time | time | NN |
| Day_of_the_week | integer | NN |
| Schedule_type | varchar(20) | NN |
| Status | varchar(50) | NN |

Figure 1.9: Schedule Table

The table stores details of all the scheduled activities throughout the week.

It also tracks the weekly schedule of classes in the system.

## ✔ Notification Table

| Notification | | |
|---|---|---|
| Notification_id | integer | NN |
| Message | text(255) | NN |
| Notification_type | varchar(255) | NN |
| Created_at | time | NN |
| Status | varchar(200) | NN |
| USERS_User_id | integer | NN |

Figure 1.10: Notification Table

Stores all the notification sent to the user

It tracks the delivery status of every notification through the status field.

It also logs messages for references in future

## ✔ Student Enrollment

| Student Enrollment | | |
|---|---|---|
| Enrollment_id | integer | NN |
| Class_id | integer | NN |
| Enrollment_date | integer | NN |
| Student_id | integer | NN |

Figure 1.11: Student Enrollment Table

Enrollment_id is the unique identifier in the table.

It keeps track of which student are enrolled in which class

- ✔ Classroom Availability

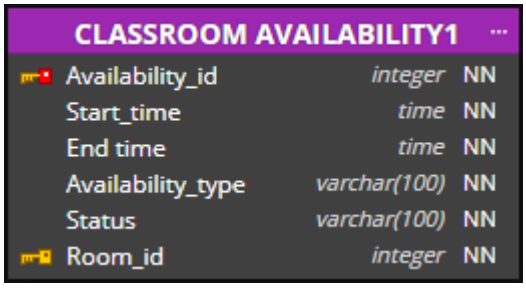


Figure 1.12: Classroom Availability

It keeps track of when the classes are available for class and exams scheduling.

- ✔ Teachers Assignment

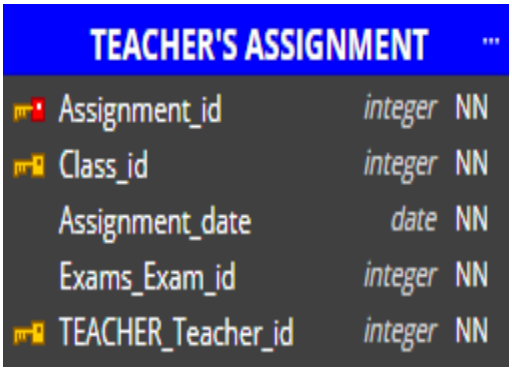


Figure 1.13: Teachers Assignment

Stores information about which teachers are assigned to teach the classes and supervise the exams

The table also tracks changes in teacher's assignment

- ✔ Student Exam Registration

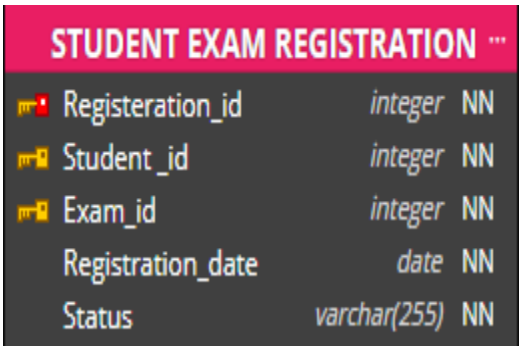


Figure 1.14: Student Exam Registration

It keeps records of which student has registered for which exams.

# 2. Entity-Relationship Diagram

This refers to a key tool used in database design to model and visualizes the connection of entities and relationships in a database. The system relied on a well-structured diagram that involved the connection of different entities such as Student, Teachers, Users, Rooms, Exams and many others.

Example of the relationships that we had in our project include;

- **One to Many Relationships**

  **Teacher – Classes**

  A teacher can teach multiple classes,

  but a class is only taught by one teacher

  **Subject – Class**

  A subject can be studied in several classes,

  But a class can only be studied by one subject at a time

  **Student – Notification**

  A student can get multiple notifications

  A notification can be sent to multiple student at one time

- **Many to Many Relationships**

  **Student – Course**

  Multiple student can enroll in multiple courses

  Multiple courses can be taken by multiple students.

  **Student – Exam**

  Multiple student can sit for multiple exams

  And multiple exams can be seated by several students.

  **Student – Student Exam Registration**

  One student can register for many exams and

  An exam can have multiple students registered to it.

- **One to One Relationships**

  **Admin – Users**

  Each admin is a user,

  while a user with admin privileges can only be for only one admin.

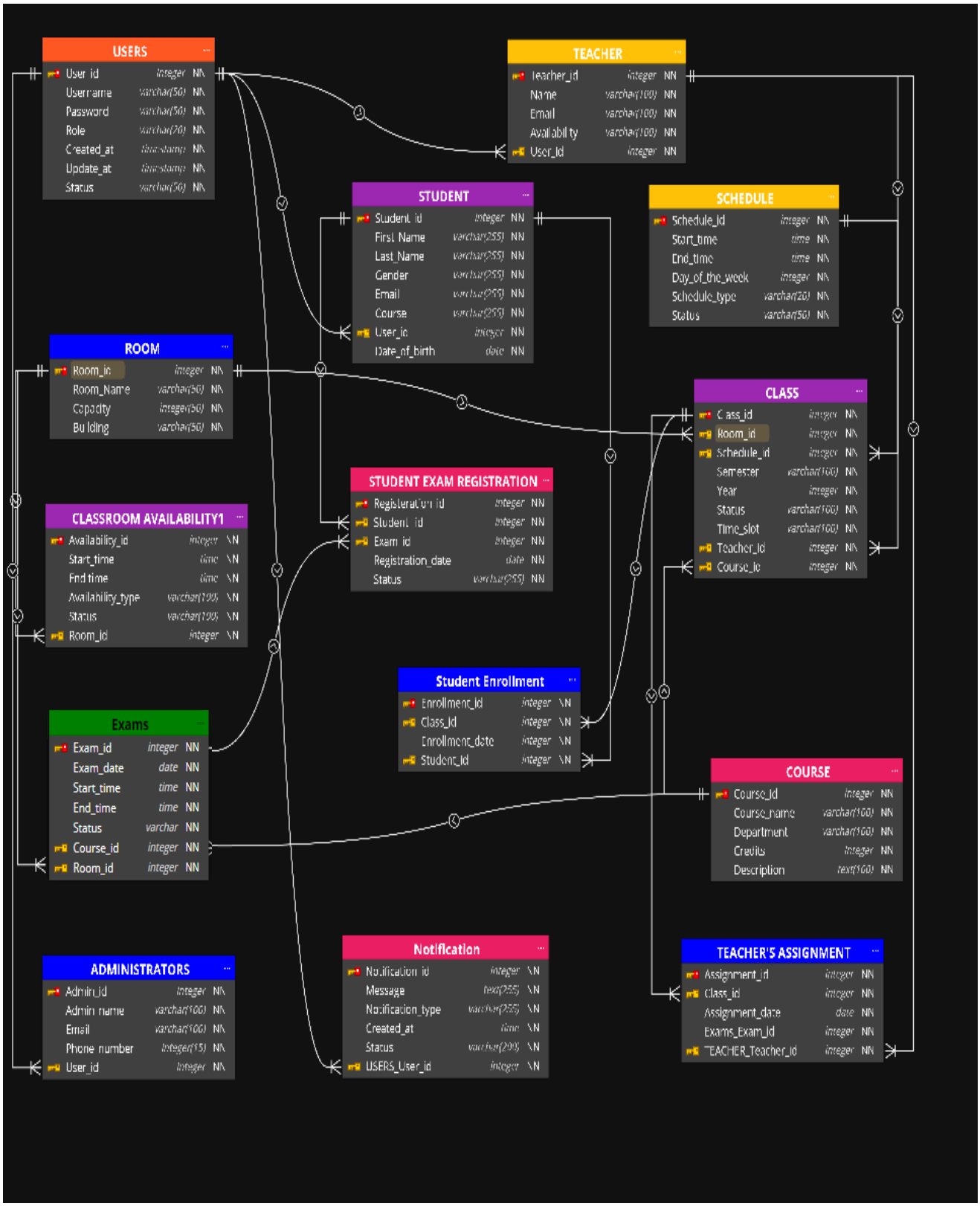

Figure 2.1: Entity- Relationship Diagram

## 3. Use Case Diagram

A use case diagram represents the behavior of a system. It incorporates the use case, actors and their relationship to model the functionality of the system. It also models the actions, services and operations needed by a system. Will use a draw.io to model use case diagrams

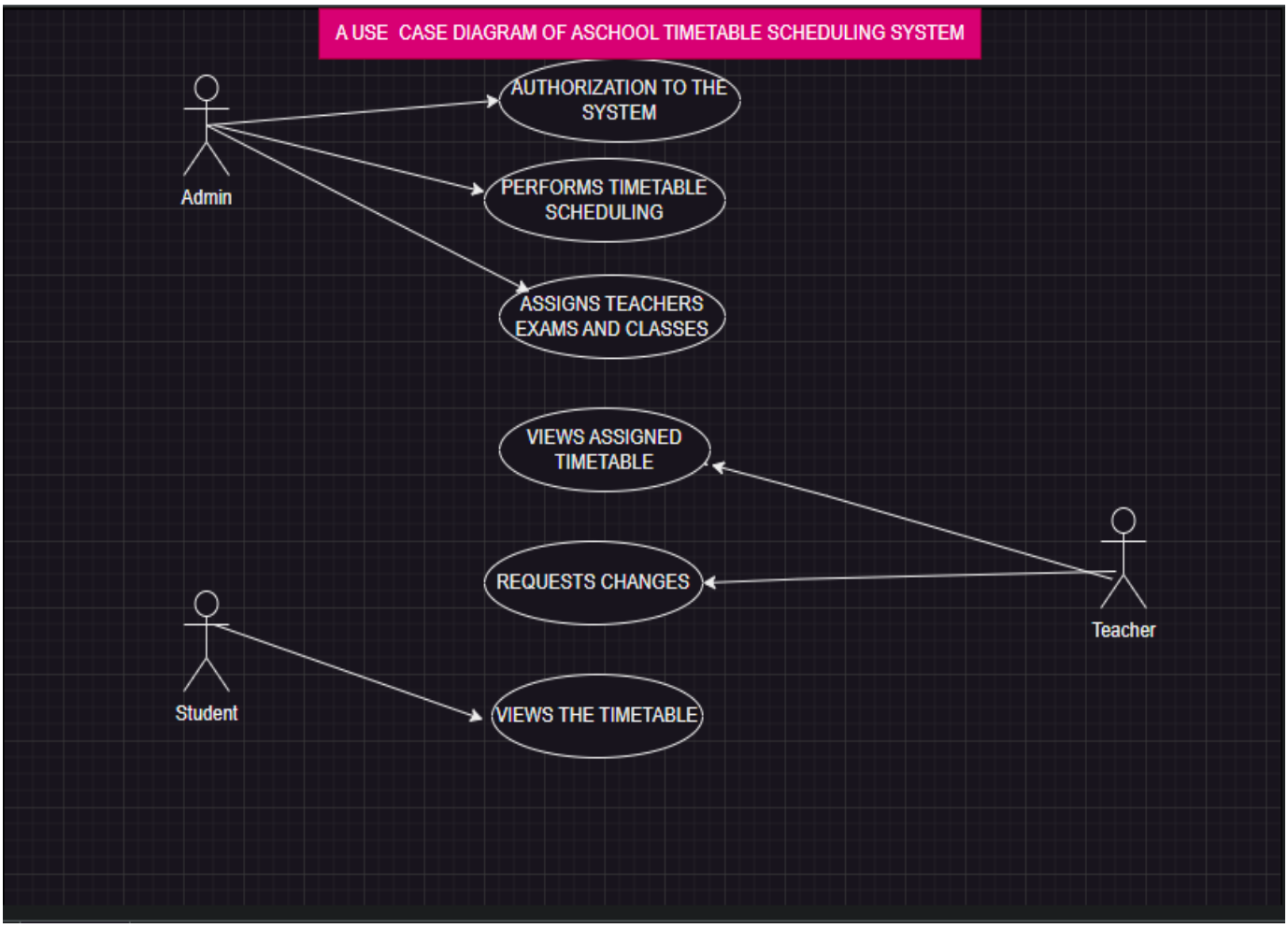


Figure 3.1: A use case diagram

## Implementation

### i. CRUD Operations

We were able to implement the CRUD Operations through writing the following codes

The codes we wrote implements all the designs of the tables.

## These are the operations that we used to create

## Create

This is a query to create a table named as User with a UserID as its unique address.

```sql
USE SchoolTimetableSchedulingSystem;
CREATE TABLE User (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50),
    Password VARCHAR(50),
    Role VARCHAR(20)
    );
INSERT INTO User (UserID, Username, Password, Role ) VALUES
(1, 'admin', 'password123', 'Administrator'),
(2, 'johndoe', 'johnpass', 'Student', 'Active'),
(3, 'janedoe', 'janepass', 'Teacher', 'Suspended'),
(4, 'michael', 'mikepass', 'Student', 'Suspended'),
(5, 'emily', 'emilypass', 'Teacher', 'Active');
SELECT * FROM User;
```

## Read1

This query extracts and reads the values of the user in row1 and displays it

```sql
-- CRUD operations
SELECT *FROM User
WHERE UserID=1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| UserID | Username | Role | Password |
|---|---|---|---|
| 1 | admin | Administrator | password123 |
| NULL | NULL | NULL | NULL |

## Read 2

This reads and extracts data from a Student table whose gender is female and does Physics as its major

```
SELECT *FROM Student
WHERE Gender='Female' AND Major='Physics' ;
```

- **The results**

| StudentID | FirstName | LastName | Gender | Email | DateOfBirth | Major |
|---|---|---|---|---|---|---|
| 3 | Carol | Williams | Female | carol.williams@example.com | 2001-07-30 | Physics |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

## Update1

It shows how the table was updated with a new row

| TeacherID | FirstNAme | LastName | Email | Availability | Department | UserID |
|---|---|---|---|---|---|---|
| 1 | John | Bob | bob@gmail.com | Full-Time | Computer Science | 1 |
| 2 | Jane | Doe | jane.doe@example.com | Part-Time | Mathematics | 2 |
| 3 | Michael | Johnson | michael.johnson@example.com | Full-Time | Physics | 3 |
| 4 | Emily | Brown | emily.brown@example.com | Full-Time | Chemistry | 4 |
| 5 | Robert | Davis | robert.davis@example.com | Part-Time | Biology | 5 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

```
UPDATE Teacher SET LastName='Andrew ',Email='andrew@gmail.com'
WHERE TeacherID=1;
SELECT * FROM Teacher WHERE TeacherID = 1;
```

- **The Results**

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| TeacherID | FirstNAme | LastName | Email | Availability | Department | UserID |
|---|---|---|---|---|---|---|
| 1 | John | Andrew | andrew@gmail.com | Full-Time | Computer Science | 1 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

## Update2

It shows how the table was updated with a new column

```
UPDATE Teacher SET UserID =1 WHERE TeacherID=1;
UPDATE Teacher  SET UserID =2 WHERE TeacherID=2;
UPDATE Teacher  SET UserID =3 WHERE TeacherID=3;
UPDATE Teacher SET UserID =4 WHERE TeacherID=4;
UPDATE Teacher  SET UserID =5 WHERE TeacherID=5;
SELECT * FROM Teacher;
```

- **The results**

| TeacherID | FirstNAme | LastName | Email | Availability | Department | UserID |
|---|---|---|---|---|---|---|
| 1 | John | Bob | bob@gmail.com | Full-Time | Computer Science | 1 |
| 2 | Jane | Doe | jane.doe@example.com | Part-Time | Mathematics | 2 |
| 3 | Michael | Johnson | michael.johnson@example.com | Full-Time | Physics | 3 |
| 4 | Emily | Brown | emily.brown@example.com | Full-Time | Chemistry | 4 |
| 5 | Robert | Davis | robert.davis@example.com | Part-Time | Biology | 5 |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

## Delete1 Operation

It shows how row3 was deleted from the table

| UserID | Username | Role | Password |
|---|---|---|---|
| 1 | admin | Administrator | password123 |
| 2 | johndoe | Student | johnpass |
| 3 | peter | Teacher | janepass |
| 4 | michael | Student | mikepass |
| 5 | emily | Teacher | emilypass |

```
DELETE FROM User
WHERE UserID=3;
```

- **The Result**

Result Grid | Filter Rows: | Edit: | Export

| UserID | Username | Role | Password |
|---|---|---|---|
| 1 | admin | Administrator | password123 |
| 2 | johndoe | Student | johnpass |
| 4 | michael | Student | mikepass |
| 5 | emily | Teacher | emilypass |

## Delete2 Operation

It shows how a column2 was deleted from the table

```
SELECT *FROM Room;
```

| RoomID | Location | Capacity |
|---|---|---|
| 101 | Building A, Room 101 | 30 |
| 102 | Building A, Room 102 | 25 |
| 201 | Building B, Room 201 | 50 |
| 202 | Building B, Room 202 | 40 |
| 301 | Building C, Room 301 | 35 |
| NULL | NULL | NULL |

```
ALTER TABLE Room
DROP COLUMN Location;
SELECT *FROM Room;
```

- **The Results**

| RoomID | Capacity |
|---|---|
| 101 | 30 |
| 102 | 25 |
| 201 | 50 |
| 202 | 40 |
| 301 | 35 |
| NULL | NULL |

## ADVANCED OPERATIONS

### First Operation

```sql
-- Advanced operations
SELECT  s.StudentID,s.FirstName,s.LastName
FROM Student s
RIGHT JOIN Enrollment e
ON s.StudentID=e.StudentID
GROUP BY  s.StudentID;
```

| StudentID | FirstName | LastName |
|---|---|---|
| 1 | Alice | Johnson |
| 2 | Bob | Smith |
| 3 | Carol | Williams |
| 4 | David | Brown |
| 5 | Emma | Jones |

### Second Operation

```sql
SELECT  s.StudentID,s.FirstName,s.LastName,GROUP_CONCAT(e.ClassID) AS Class
FROM Student s
LEFT JOIN Enrollment e
ON s.StudentID=e.StudentID
GROUP BY  s.StudentID;
```

| StudentID | FirstName | LastName | Class |
|---|---|---|---|
| 1 | Alice | Johnson | 1,2 |
| 2 | Bob | Smith | 2,3 |
| 3 | Carol | Williams | 3 |
| 4 | David | Brown | 5 |
| 5 | Emma | Jones | 5,1 |

## Third Operation

```sql
SELECT t.FirstName ,LastName,c. ClassID,c.CourseCode,c.RoomID,c.TimeSlot
FROM Class c
INNER JOIN Course co
ON c.CourseCode=co.CourseCode
INNER JOIN Teacher t
ON t.TeacherID=c.TeacherID
ORDER BY c.ClassID
;
```

| FirstName | LastName | ClassID | CourseCode | RoomID | TimeSlot |
|---|---|---|---|---|---|
| John | Andrew | 1 | CS101 | 101 | Monday 09:00 - 11:00 |
| Jane | Doe | 2 | MATH201 | 102 | Tuesday 10:00 - 12:00 |
| Michael | Johnson | 3 | PHYS101 | 201 | Wednesday 11:00 - 13:00 |
| Emily | Brown | 4 | CHEM101 | 202 | Thursday 14:00 - 16:00 |
| Robert | Davis | 5 | BIO101 | 301 | Friday 15:00 - 17:00 |

## Fourth Operation

```sql
-- Subqueries
SELECT ClassID,CourseCode,RoomID
FROM Class
WHERE ClassID=(
SELECT ClassID
FROM Enrollment
GROUP BY ClassID
ORDER BY COUNT(*) DESC
LIMIT 1
);
```

| ClassID | CourseCode | RoomID |
|---|---|---|
| 1 | CS101 | 101 |
| NULL | NULL | NULL |

## TESTING AND VALIDATION

System testing was conducted where we validated end to end functionalities to protect the system.

```sql
-- Encrypt data
UPDATE User
SET TempPassword = AES_ENCRYPT(TempPassword, 'YourSecretKey');


DELETE FROM User WHERE UserID IN (1, 2, 3, 4, 5);
INSERT INTO User (UserID, Username, TempPassword, Role)
VALUES

(1, 'admin',  AES_ENCRYPT('password123', 'YourSecretKey'), 'Administrator'),
(2, 'johndoe',  AES_ENCRYPT('johnpass', 'YourSecretKey'), 'Student'),
(3, 'janedoe',  AES_ENCRYPT('janepass', 'YourSecretKey'), 'Teacher'),
(4, 'michael',  AES_ENCRYPT('mikepass', 'YourSecretKey'), 'Student'),
(5, 'emily', AES_ENCRYPT('emilypass', 'YourSecretKey'), 'Teacher');
;
```

| UserID | Username | Role | TempPassword |
|---|---|---|---|
| 1 | admin | Administrator | BLOB |
| 2 | johndoe | Student | BLOB |
| 3 | janedoe | Teacher | BLOB |
| 4 | michael | Student | BLOB |
| 5 | emily | Teacher | BLOB |

- Password is stored in BLOB because we have used the AES -ENCRYPT function to encrypt the password.
- To put in a readable form, we decrypt the encrypted password using AES-DECRYPT.

```sql
-- Decrypt data
SELECT
    UserID,
    Username,
    CAST(AES_DECRYPT(TempPassword, 'YourSecretKey') AS CHAR(50)) AS DecryptedPassword,
    Role
FROM User;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| UserID | Username | DecryptedPassword | Role |
|---|---|---|---|
| 1 | admin | password123 | Administrator |
| 2 | johndoe | johnpass | Student |
| 3 | janedoe | janepass | Teacher |
| 4 | michael | mikepass | Student |
| 5 | emily | emilypass | Teacher |

```sql
SET SQL_SAFE_UPDATES = 0;

DESCRIBE User;
-- Hash
ALTER TABLE User DROP COLUMN PasswordHash ;

ALTER TABLE User ADD COLUMN PasswordHash CHAR(64);

DELETE FROM User WHERE UserID IN (1, 2, 3, 4, 5);

DELETE FROM User WHERE UserID IN (1, 2, 3, 4, 5);

INSERT INTO User (UserID, Username, PasswordHash, Role)
VALUES
(1, 'admin', SHA2('password123', 256), 'Administrator'),
(2, 'johndoe', SHA2('johnpass', 256), 'Student'),
(3, 'peter', SHA2('peterpass', 256), 'Teacher'),
(4, 'michael', SHA2('mikepass', 256), 'Student'),
(5, 'emily', SHA2('emilypass', 256), 'Teacher');

DESCRIBE User;
-- Select all data to view the table contents
SELECT * FROM User;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| UserID | Username | Role | PasswordHash |
|---|---|---|---|
| 1 | admin | Administrator | ef92b778bafe771e89245b89ecbc08a44a4e166... |
| 2 | johndoe | Student | dcffce09862520d2eb2c98534ee8caf446a6664e... |
| 3 | peter | Teacher | e86529d2f2386c55c57b67a1a05b08abd1e4898... |
| 4 | michael | Student | d6ab7a4ba46690f83961f28f7d537f4f8db309d7... |
| 5 | emily | Teacher | 46c993a2c3c6d362a09f08ea58c5744dc2e5a99... |

## References


Datensen. (2024, January 12). *Luna Modeler: A Powerful Database Design Tool*. Datensen. https://www.datensen.com/

GeeksforGeeks. (2024, February 14). *Timetable Generating System UML diagram*. GeeksforGeeks. https://www.geeksforgeeks.org/timetable-generating-system-uml-diagram/

*Types of Relationship in DBMS - javatpoint*. (n.d.). www.javatpoint.com. https://www.javatpoint.com/types-of-relationship-in-database-table

Altvater, A. (2024, April 18). *What are CRUD Operations: How CRUD Operations Work, Examples, Tutorials & More*. Stackify. https://stackify.com/what-are-crud-operations/

*MySQL :: MySQL Workbench*. (n.d.). https://www.mysql.com/products/workbench/

# Appendix

## List of Figures

# TABLE CODE SNIPPET

We created the following tables in our project using the following SQL codes with populated data in them.

## Student Table.

- The SQL Code

```sql
create database schooltimetableschedulingsystem;
SHOW DATABASES;
USE schooltimetableschedulingsystem;
select database();
CREATE TABLE Student(
StudentID INT PRIMARY KEY,
FirstName VARCHAR(100),
LastName VARCHAR(100),
Gender VARCHAR(10),
Email VARCHAR (100),
DateOfBirth DATE,
Major VARCHAR (100)
);
INSERT INTO Student (StudentID, FirstName, LastName, Gender, Email, DateOfBirth, Major) VALUES
(1, 'Alice', 'Johnson', 'Female', 'alice.johnson@example.com', '2000-01-15', 'Computer Science'),
(2, 'Bob', 'Smith', 'Male', 'bob.smith@example.com', '1999-03-22', 'Mathematics'),
(3, 'Carol', 'Williams', 'Female', 'carol.williams@example.com', '2001-07-30', 'Physics'),
(4, 'David', 'Brown', 'Male', 'david.brown@example.com', '1998-11-10', 'Chemistry'),
(5, 'Emma', 'Jones', 'Female', 'emma.jones@example.com', '2002-05-05', 'Biology');
DESCRIBE Student;
SELECT * FROM Student;
```

- The Output

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| StudentID | FirstName | LastName | Gender | Email | DateOfBirth | Major |
|---|---|---|---|---|---|---|
| 1 | Alice | Johnson | Female | alice.johnson@example.com | 2000-01-15 | Computer Science |
| 2 | Bob | Smith | Male | bob.smith@example.com | 1999-03-22 | Mathematics |
| 3 | Carol | Williams | Female | carol.williams@example.com | 2001-07-30 | Physics |
| 4 | David | Brown | Male | david.brown@example.com | 1998-11-10 | Chemistry |
| 5 | Emma | Jones | Female | emma.jones@example.com | 2002-05-05 | Biology |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

Result Grid

Form Editor

# Teacher Table

## The SQL code

```sql
USE schooltimetableschedulingsystem;
CREATE TABLE Teacher (
    TeacherID INT PRIMARY KEY,
    FirstName VARCHAR(100),
    LastName VARCHAR(100),
    Email VARCHAR(100),
    Availability VARCHAR(100),
    Department VARCHAR (100)
);
INSERT INTO Teacher (TeacherID, FirstName, LastName, Email, Availability, Department) VALUES
(1, 'John', 'Smith', 'john.smith@example.com', 'Full-Time', 'Computer Science'),
(2, 'Jane', 'Doe', 'jane.doe@example.com', 'Part-Time', 'Mathematics'),
(3, 'Michael', 'Johnson', 'michael.johnson@example.com', 'Full-Time', 'Physics'),
(4, 'Emily', 'Brown', 'emily.brown@example.com', 'Full-Time', 'Chemistry'),
(5, 'Robert', 'Davis', 'robert.davis@example.com', 'Part-Time', 'Biology');

SELECT * FROM Teacher;
```

- The Output

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| TeacherID | FirstName | LastName | Email | Availability | Department |
|---|---|---|---|---|---|
| 1 | John | Smith | john.smith@example.com | Full-Time | Computer Science |
| 2 | Jane | Doe | jane.doe@example.com | Part-Time | Mathematics |
| 3 | Michael | Johnson | michael.johnson@example.com | Full-Time | Physics |
| 4 | Emily | Brown | emily.brown@example.com | Full-Time | Chemistry |
| 5 | Robert | Davis | robert.davis@example.com | Part-Time | Biology |

Result Grid

Form Editor

# User Table

- The SQL Code

```sql
USE SchoolTimetableSchedulingSystem;
CREATE TABLE User (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50),
    Password VARCHAR(50),
    Role VARCHAR(20)
    );
INSERT INTO User (UserID, Username, Password, Role ) VALUES
(1, 'admin', 'password123', 'Administrator'),
(2, 'johndoe', 'johnpass', 'Student', 'Active'),
(3, 'janedoe', 'janepass', 'Teacher', 'Suspended'),
(4, 'michael', 'mikepass', 'Student', 'Suspended'),
(5, 'emily', 'emilypass', 'Teacher', 'Active');
SELECT * FROM User;
```

- The Output

Result Grid | Filter Rows:

| UserID | Username | Password | Role |
|---|---|---|---|
| 1 | admin | password123 | Administrator |
| 2 | johndoe | johnpass | Student |
| 3 | janedoe | janepass | Teacher |
| 4 | michael | mikepass | Student |
| 5 | emily | emilypass | Teacher |

# Course Table

- The SQL Code

```sql
USE SchoolTimetableSchedulingSystem;
CREATE TABLE Course (
    CourseCode VARCHAR(10) PRIMARY KEY,
    CourseName VARCHAR(100),
    Description TEXT,
    Credits INT
);
INSERT INTO Course (CourseCode, CourseName, Description, Credits) VALUES
('CS101', 'Introduction to Computer Science', 'Basics of computer science including programming and algorithms.', 4
('MATH201', 'Calculus I', 'Introduction to differential and integral calculus.', 3),
('PHYS101', 'General Physics', 'Fundamentals of physics, covering mechanics, heat, and sound.', 4),
('CHEM101', 'General Chemistry', 'Basic concepts of chemical principles and laboratory techniques.', 4),
('BIO101', 'Introduction to Biology', 'Study of living organisms, including structure, function, growth, and evolut
```

- The Output

| CourseCode | CourseName | Description | Credits |
|---|---|---|---|
| BIO101 | Introduction to Biology | Study of living organisms, including structure, f... | 3 |
| CHEM101 | General Chemistry | Basic concepts of chemical principles and labora... | 4 |
| CS101 | Introduction to Computer Science | Basics of computer science including programmi... | 4 |
| MATH201 | Calculus I | Introduction to differential and integral calculus. | 3 |
| PHYS101 | General Physics | Fundamentals of physics, covering mechanics, ... | 4 |
| NULL | NULL | NULL | NULL |

# Room Table

- The SQL Code

```sql
USE schooltimetableschedulingsystem;
SHOW databases;
CREATE TABLE Room (
    RoomID INT PRIMARY KEY,
    Location VARCHAR(100),
    Capacity INT
);
INSERT INTO Room (RoomID, Location, Capacity) VALUES
(101, 'Building A, Room 101', 30),
(102, 'Building A, Room 102', 25),
(201, 'Building B, Room 201', 50),
(202, 'Building B, Room 202', 40),
(301, 'Building C, Room 301', 35);
```

- The Output

| RoomID | Location | Capacity |
|---|---|---|
| 101 | Building A, Room 101 | 30 |
| 102 | Building A, Room 102 | 25 |
| 201 | Building B, Room 201 | 50 |
| 202 | Building B, Room 202 | 40 |
| 301 | Building C, Room 301 | 35 |
| NULL | NULL | NULL |

# Exam Table

- The SQL Code

```sql
USE SchoolTimetableSchedulingSystem;
CREATE TABLE Exam (
    ExamID INT PRIMARY KEY,
    CourseCode VARCHAR(10),
    RoomID INT,
    Date DATE,
    TimeSlot VARCHAR(50),
    FOREIGN KEY (CourseCode) REFERENCES Course(CourseCode),
    FOREIGN KEY (RoomID) REFERENCES Room(RoomID)
);
INSERT INTO Exam (ExamID, CourseCode, RoomID, Date, TimeSlot) VALUES
(1, 'CS101', 101, '2024-12-15', '09:00 - 11:00'),
(2, 'MATH201', 102, '2024-12-16', '10:00 - 12:00'),
(3, 'PHYS101', 201, '2024-12-17', '11:00 - 13:00'),
(4, 'CHEM101', 202, '2024-12-18', '14:00 - 16:00'),
(5, 'BIO101', 301, '2024-12-19', '15:00 - 17:00');
SELECT * FROM Exam;
```

## The Output

| ExamID | CourseCode | RoomID | Date | TimeSlot |
|---|---|---|---|---|
| 1 | CS101 | 101 | 2024-12-15 | 09:00 - 11:00 |
| 2 | MATH201 | 102 | 2024-12-16 | 10:00 - 12:00 |
| 3 | PHYS101 | 201 | 2024-12-17 | 11:00 - 13:00 |
| 4 | CHEM101 | 202 | 2024-12-18 | 14:00 - 16:00 |
| 5 | BIO101 | 301 | 2024-12-19 | 15:00 - 17:00 |
| NULL | NULL | NULL | NULL | NULL |

# Class Table

## The SQL Code

```sql
USE SchoolTimetableSchedulingSystem;
CREATE TABLE Class (
    ClassID INT PRIMARY KEY,
    CourseCode VARCHAR(10),
    TeacherID INT,
    RoomID INT,
    TimeSlot VARCHAR(50),
    FOREIGN KEY (CourseCode) REFERENCES Course(CourseCode),
    FOREIGN KEY (TeacherID) REFERENCES Teacher(TeacherID),
    FOREIGN KEY (RoomID) REFERENCES Room(RoomID)
);
INSERT INTO Class (ClassID, CourseCode, TeacherID, RoomID, TimeSlot) VALUES
(1, 'CS101', 1, 101, 'Monday 09:00 - 11:00'),
(2, 'MATH201', 2, 102, 'Tuesday 10:00 - 12:00'),
(3, 'PHYS101', 3, 201, 'Wednesday 11:00 - 13:00'),
(4, 'CHEM101', 4, 202, 'Thursday 14:00 - 16:00'),
(5, 'BIO101', 5, 301, 'Friday 15:00 - 17:00');
SELECT * FROM
```

## The Output

| ClassID | CourseCode | TeacherID | RoomID | TimeSlot |
|---|---|---|---|---|
| 1 | CS101 | 1 | 101 | Monday 09:00 - 11:00 |
| 2 | MATH201 | 2 | 102 | Tuesday 10:00 - 12:00 |
| 3 | PHYS101 | 3 | 201 | Wednesday 11:00 - 13:00 |
| 4 | CHEM101 | 4 | 202 | Thursday 14:00 - 16:00 |
| 5 | BIO101 | 5 | 301 | Friday 15:00 - 17:00 |
| NULL | NULL | NULL | NULL | NULL |

# Schedule Table

- **The SQL CODE**

```sql
USE SchoolTimetableSchedulingSystem;
CREATE TABLE Schedule (
    ScheduleID INT PRIMARY KEY,
    ClassID INT,
    ExamID INT,
    Day VARCHAR(20),
    Period VARCHAR(20),
    FOREIGN KEY (ClassID) REFERENCES Class(ClassID),
    FOREIGN KEY (ExamID) REFERENCES Exam(ExamID)
);
INSERT INTO Schedule (ScheduleID, ClassID, ExamID, Day, Period) VALUES
(1, 1, 1, 'Monday', '09:00 - 11:00'),
(2, 2, 2, 'Tuesday', '10:00 - 12:00'),
(3, 3, 3, 'Wednesday', '11:00 - 13:00'),
(4, 4, 4, 'Thursday', '14:00 - 16:00'),
(5, 5, 5, 'Friday', '15:00 - 17:00');
SELECT * FROM Schedule;
```

- **The results**

| ScheduleID | ClassID | ExamID | Day | Period |
|---|---|---|---|---|
| 1 | 1 | 1 | Monday | 09:00 - 11:00 |
| 2 | 2 | 2 | Tuesday | 10:00 - 12:00 |
| 3 | 3 | 3 | Wednesday | 11:00 - 13:00 |
| 4 | 4 | 4 | Thursday | 14:00 - 16:00 |
| 5 | 5 | 5 | Friday | 15:00 - 17:00 |
| NULL | NULL | NULL | NULL | NULL |

# Student Enrollment Table

- **The SQL codes**

```sql
USE SchoolTimetableSchedulingSystem;
CREATE TABLE StudentEnrollment (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    ClassID INT,
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (ClassID) REFERENCES Class(ClassID)
);
INSERT INTO StudentEnrollment (EnrollmentID, StudentID, ClassID) VALUES
(1, 1, 1),
(2, 2, 2),
(3, 3, 3),
(4, 4, 4),
(5, 5, 5),
(6, 1, 2),
(7, 2, 3),
(8, 3, 4),
(9, 4, 5),
(10, 5, 1);
```

- **The Output**

| EnrollmentID | StudentID | ClassID |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 3 |
| 4 | 4 | 4 |
| 5 | 5 | 5 |
| 6 | 1 | 2 |
| 7 | 2 | 3 |
| 8 | 3 | 4 |
| 9 | 4 | 5 |
| 10 | 5 | 1 |
| NULL | NULL | NULL |

# Conclusion and Recommendation

***Recommendation**

Functional Requirements

User Roles.: Full access to create, update, and delete schedules

Teachers: View their schedules and suggest changes.

Students: View their class schedules.

Schedule Creation: Automatically generate timetables based on input data (classes, teachers, subjects, and constraints).

Conflict Detection: Identify and prevent scheduling conflicts (e.g., double-booked teachers, overlapping classes,

**Conclusion**

The system simplifies the process throughout this presentation we have discussed the School Timetable Scheduling System, mentioning its aims to utilize resources effectively, avoid clashes and be dynamic. The design of the system has also been covered where its architecture, major modules and elements has been described. In the final section, some significant SQL commands that help to control and retrieve the information necessary for effective and correct scheduling were presented.