THE UNIVERSITY OF TEXAS AT DALLAS
Naveen Jindal School of Management

Course:	BUAN 6320.006
Course Title:	Database Foundations for Business Analytics
Instructor:	Dr. Lindong Wu
Term:	Fall 2023
Project Title:	Comet Attends
Group Members:	Anuja Abhay Thakar; axt230040@utdallas.edu
	Apoorva Shukla; axs220598@utdallas.edu
	Shradha Sharma; sxs230150@utdallas.edu
	Siddhi Javalkar; sxj230040@utdallas.edu

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Introduction

This project is implemented in three phases.
Phase I – Project Proposal
Phase II – Design and Modeling
Phase III – Implementation

Phase I Project Proposal

1.1 BRIEF
Comet Attends aims to be an attendance tracking system which will enable a professor teaching multiple courses and sections centrally track attendance for all courses in one system.

1.2 OPPORTUNITY
At present there are multiple ways a professor can track attendance such as below:

1. Manually through MS excel or on paper
2. Swiping Comet Card which requires card swiping apparatus
3. Scan QR

Keeping above in mind Comet Attends aims to solve the need of trying multiple ways of tracking to allow a professor an opportunity to centrally track and view attendance digitally.

1.3 SOLUTION
To create a digital central system with two authorized views i.e., professor and student. Logging for the same will work based on UT Dallas NET ID email and password.
Professor View

1. Can generate unique link for each lecture
2. Can generate link using dropdowns for subjects and sections
3. Can decide time for link to remain active e.g. First 15 minutes of class duration
4. Can view attendance throughout course tenure

Student View

1. Click on a login link created for respective lecture
2. Login using UT Dallas NET ID and password
3. Confirm subject and section
4. Mark attendance for respective lecture
5. Can view attendance for lectures

1.4 INFORMATION REQUIRED
Using the information Comet Attends will track attendance for respective course code & Section code by mapping respective NET ID.

1. Professor NET ID

2. Student NET ID
3. Department Code
4. Course Title
5. Course Code
6. Course Section

1.5 INITIAL LIST OF ENTITIES

DFBA Project Fall 2023

professor	student	department	course	attendance
professor_id(p)	student_id(p)	department_id(p)	course_id(p)	student_id
net_id	net_id	department_name	course_title	course_id
professor_firstname	student_firstname	department_code	department_id	status
professor_lastname	student_lastname		course_code	trace_id
			course_section	
			timestamp	
			professor_id	

1.6 ROLES AND RESPONSIBILITIES

1. Implementation - Anuja and Apoorva
2. System Analyst – Apoorva
3. Frontend- Anuja
4. Presentation – Siddhi and Shradha
5. Data Collection – Shradha
6. Research - Siddhi
7. Documentation – Siddhi and Shradha

Phase II. Design and Modelling

2.1 Executive Summary

In this project report, we delve into the logic design and modeling of our project Attendance Management System.

Section 1 , provides an introduction to the project.

Section 2 unveils our ER/EER diagram, along with all underlying assumptions, derived from Section 1.

Continuing to Section 3, we present the relational schema, resulting from the transformation of the aforementioned ER/EER diagram.

In Section 4, we meticulously document functional dependencies and normalize all tables to meet the third normal form (3NF) standards. To conclude, a concise summary is offered at the end of this report.

2.2 Conceptual Design

Here is the EER diagram generated based on both our project description and real-life experiences.

2.3 EER diagram with all assumptions

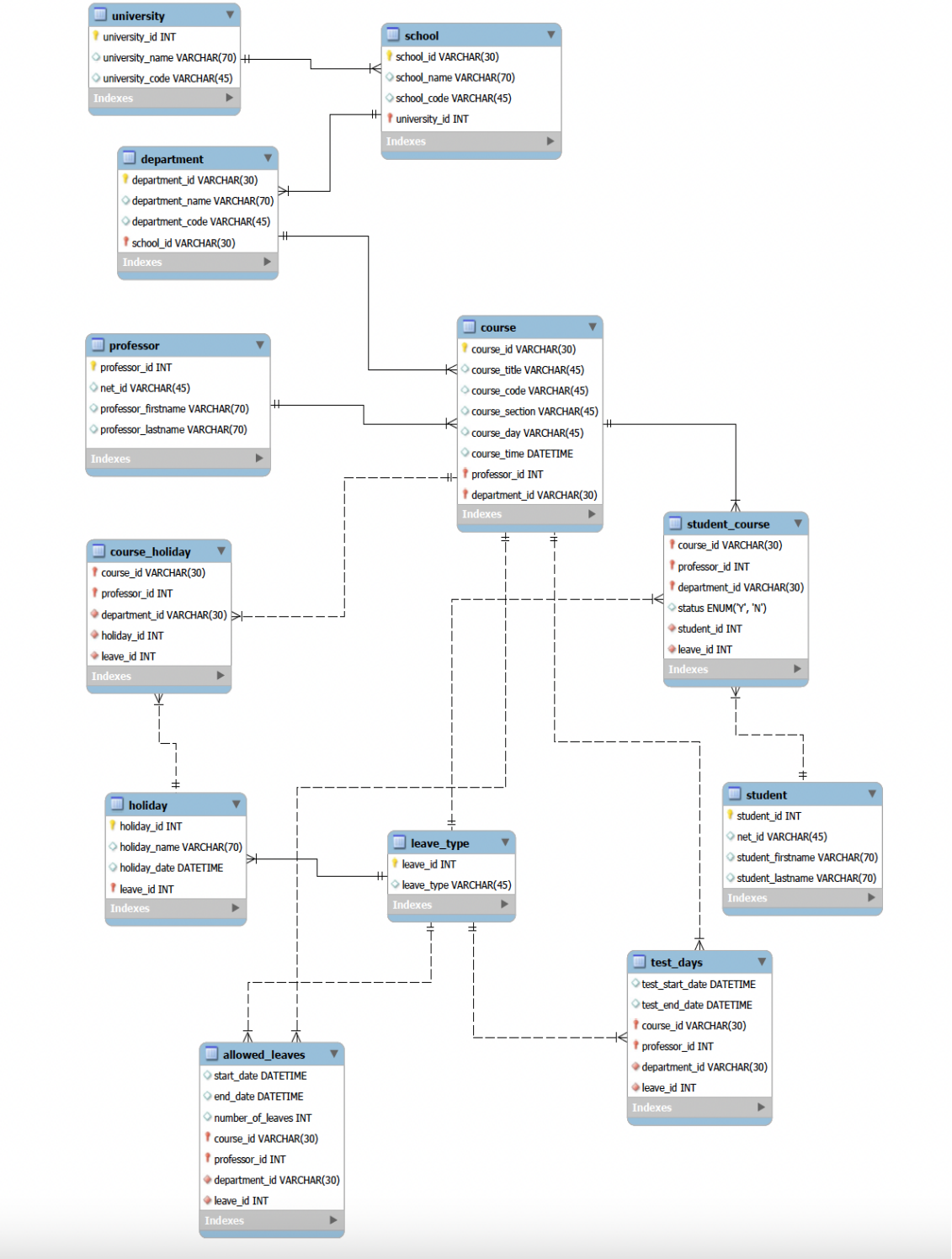

Figure 1 : ER Diagram

2.4 (Min, Max) Notation for Relationship

In this part we discuss the (min, max) notation for several important relationships that exist in our EER diagram. Table 1 clearly specifies how the numerical expression corresponds to the relationship between two entities.

Table 1. Explanation for (Min, Max) Notation

Relation	Discussion
University and school	• A University can have one or more Schools. (1, N). • A School belongs to exactly one University. (1, 1).
School and Department	• A school can have one or more departments. (1, N). • A department belongs to exactly one school. (1, 1).
Department and course	• A department can have one or more courses. (1, N). • A course belongs to exactly one department. (1, 1).
Professor and course	• A professor can teach one or more courses. (1, N). • A course is taught exactly by one professor. (1, 1).
Course and student	• A student can enroll in many courses. (1, N). • A course can have many students enrolled. (1, N).
Student_course	• Student_Course serves as the associative entity to represent the many-to-many relationship between Student and Course. This will also track the attendance of the students
Holiday and course	• A course can be associated with zero or more holidays. (0, N). • A holiday can be associated with zero or more courses. (0, N).
Holiday_course	• The Course_Holidays table acts as an associative entity connecting Course and Holidays.
Allowed leaves and course	• A course can be associated with zero or more allowed leaves. (0, N). • A set of allowed leaves will be part of zero or

	1 course. (0, 1).
course and test days	• A course can be associated with zero or more test days. (0, N). • A test day can be associated with one and only courses. (0, 1).
Leave type	• Considered as a reference entity for categorizing different types of absence

2.5 Relational Schema

Strong entities:
University → [university_id, university_name, university_code]
School → [school_id, school_name, school_code, university_id(FK)]
Department → [department_id, department_name, department_code, school_id(FK)]
Professor → [professor_id, net_id, professor_firstname, professor_lastname]
Course → [course_id, course_title, course_code, course_section, date, time, professor_id(FK), department_id(FK)]
Student → [student_id, net_id, student_firstname, student_lastname]
Leave_type → [leave_id, leave_type]
Holiday → [holiday_id, holiday_name, holiday_date, course_id(FK), professor_id(FK), department_id(FK), leave_id(FK)]

Weak entities:
student_course → [course_id(FK), professor_id(FK), student_id(FK), department_id(FK), leave_id(FK), status]
allowed_leaves → [course_id(FK), professor_id(FK), department_id(FK), leave_id(FK), start_date, end_date, number_of_leaves]
test_days → [course_id(FK), professor_id(FK), department_id(FK), leave_id(FK), test_start_date, test_end_date]
course_holiday→ [course_id(FK), professor_id(FK), department_id(FK),holiday_id, leave_id(FK)]

2.6 Data Format for Every Relation

-DATABASE
```
CREATE DATABASE AttendanceManagementSystem;
-- DROP DATABASE AttendanceManagementSystem;
use AttendanceManagementSystem;
-- SHOW TABLES;
-- Table structure for `university`
-- DROP TABLE IF EXISTS `university`;
```
-UNIVERSITY
```
CREATE TABLE AttendanceManagementSystem.university(
university_id INT NOT NULL AUTO_INCREMENT,
university_name VARCHAR(70) NOT NULL,
```

university_code VARCHAR(45) NOT NULL,
PRIMARY KEY(university_id));

Relation Name	Attributes	Data Type
university	university_id	integer
	university_name	string <= 70 chars
	university_code	string <= 45 chars

Table 2 : University

-SCHOOL

```
CREATE TABLE AttendanceManagementSystem.school(
school_id VARCHAR(30) NOT NULL,
school_name VARCHAR(70) NOT NULL,
school_code VARCHAR(45) NOT NULL,
university_id INT NOT NULL,
PRIMARY KEY (school_id),
-- FOREIGN KEY for school TABLE
INDEX university_idx (university_id ASC) VISIBLE,
  CONSTRAINT university_id
    FOREIGN KEY (university_id)
    REFERENCES AttendanceManagementSystem.university (university_id)
    -- ON DELETE SET DEFAULT 1234
    ON UPDATE CASCADE);
```

Relation Name	Attributes	Data Type
school	school_id	string<=30
	school _name	string <= 70 chars
	school_code	string <= 45 chars
	university_id	integer

Table 3: School

-DEPARTMENT

```
CREATE TABLE AttendanceManagementSystem.department(
department_id VARCHAR(30) NOT NULL,
department_name VARCHAR(70) NOT NULL,
department_code VARCHAR(45) NOT NULL,
school_id VARCHAR(30) NOT NULL,
PRIMARY KEY (department_id),
-- FOREIGN KEY for DEPARTMENT TABLE
INDEX school_idx (school_id ASC) VISIBLE,
  CONSTRAINT school_id
    FOREIGN KEY (school_id)
    REFERENCES AttendanceManagementSystem.school (school_id)
    -- ON DELETE SET DEFAULT 1234
    ON UPDATE CASCADE);
```

Relation Name	Attributes	Data Type
department	department_id	string<=30
	department _name	string <= 70 chars

	department _code	string <= 45 chars
	school_id	string<=30

Table 4 : Department

- PROFESSOR

CREATE TABLE AttendanceManagementSystem.professor(
professor_id INT NOT NULL AUTO_INCREMENT,
net_id VARCHAR(45) NOT NULL,
professor_firstname VARCHAR(70) NOT NULL,
professor_lastname VARCHAR(70) NOT NULL,
PRIMARY KEY (professor_id));

Relation Name	Attributes	Data Type
professor	professor_id	integer
	net_id	string<=45
	professor_firstname	string <= 70 chars
	professor_lastname	string <= 70 chars

Table 5: Professor

-STUDENT

CREATE TABLE AttendanceManagementSystem.student(
student_id INT NOT NULL AUTO_INCREMENT,
net_id VARCHAR(45) NOT NULL,
student_firstname VARCHAR(70) NOT NULL,
student_lastname VARCHAR(70) NOT NULL,
PRIMARY KEY (student_id));

Relation Name	Attributes	Data Type
student	student_id	integer
	net_id	string <=45
	student_firstname	string <= 70 chars
	student_lastname	string <= 70 chars

Table 6: Student

-COURSE

CREATE TABLE AttendanceManagementSystem.course(
course_id VARCHAR(30) NOT NULL,
course_title VARCHAR(45) NOT NULL,
course_code VARCHAR(45) NOT NULL,
course_section VARCHAR(45) NOT NULL,
course_day VARCHAR(10) NOT NULL,
course_time TIME NOT NULL,
PRIMARY KEY (course_id));

Relation Name	Attributes	Data Type
course	course_id	string<=30 chars
	course_title	string<=45 chars
	course_code	string <= 45 chars
	course_section	string<=45 chars
	course_day	string<=10 chars
	course_time	time

Table 7: Course

-LEAVE TYPE

```
CREATE TABLE AttendanceManagementSystem.leave_type(
leave_id INT NOT NULL AUTO_INCREMENT,
leave_type VARCHAR(45) NOT NULL,
PRIMARY KEY (leave_id));
```

Relation Name	Attributes	Data Type
leave_type	leave_id	integer
	leave_type	string<=45 chars

Table 8: Leave Type

-HOLIDAY

```
CREATE TABLE AttendanceManagementSystem.holiday (
  holiday_id VARCHAR(30) NOT NULL,
  holiday_name VARCHAR(70) NOT NULL,
  holiday_date DATETIME NOT NULL,
  course_id VARCHAR(30) NOT NULL,
  professor_id INT NOT NULL,
  department_id VARCHAR(30) NOT NULL,
  leave_id INT NOT NULL,
  PRIMARY KEY (course_id, professor_id),
  -- FOREIGN KEY for HOLIDAY TABLE
  INDEX course_idx (course_id ASC) VISIBLE,
  CONSTRAINT course_id
    FOREIGN KEY (course_id)
    REFERENCES AttendanceManagementSystem.course (course_id)
    ON UPDATE CASCADE,
  INDEX department_idx (department_id ASC) VISIBLE,
  CONSTRAINT department_id
    FOREIGN KEY (department_id)
    REFERENCES AttendanceManagementSystem.department (department_id)
    ON UPDATE CASCADE,
  INDEX leave_idx (leave_id ASC) VISIBLE,
  CONSTRAINT leave_id
```

```
    FOREIGN KEY (leave_id)
    REFERENCES AttendanceManagementSystem.leave_type (leave_id)
    ON UPDATE CASCADE
);
```

Relation Name	Attributes	Data Type
holiday	holiday_id	string<=30 chars
	holiday_name	string<=70 chars
	holiday_date	datetime
	course_id	string<=30 chars
	professor_id	integer
	department_id	string<=30 chars
	leave_id	integer

Table 9: Holiday

```
-ALLOWED LEAVES
CREATE TABLE AttendanceManagementSystem.allowed_leaves (
  start_date DATETIME NOT NULL,
  end_date DATETIME NOT NULL,
  number_of_leaves INT NOT NULL,
  course_id VARCHAR(30) NOT NULL,
  professor_id INT NOT NULL,
  department_id VARCHAR(30) NOT NULL,
  leave_id INT NOT NULL,
  PRIMARY KEY (course_id, professor_id),
  -- FOREIGN KEY for ALLOWED_LEAVES TABLE
  INDEX course_idx (course_id ASC) VISIBLE,
  CONSTRAINT fk_course_id
    FOREIGN KEY (course_id)
    REFERENCES AttendanceManagementSystem.course (course_id)
    ON UPDATE CASCADE,
  INDEX professor_idx (professor_id ASC) VISIBLE,
  CONSTRAINT fk_professor_id
    FOREIGN KEY (professor_id)
    REFERENCES AttendanceManagementSystem.professor (professor_id)
    ON UPDATE CASCADE,
  INDEX department_idx (department_id ASC) VISIBLE,
  CONSTRAINT fk_department_id
    FOREIGN KEY (department_id)
    REFERENCES AttendanceManagementSystem.department (department_id)
    ON UPDATE CASCADE,
  INDEX leave_idx (leave_id ASC) VISIBLE,
  CONSTRAINT fk_leave_id
    FOREIGN KEY (leave_id)
```

```
    REFERENCES AttendanceManagementSystem.leave_type (leave_id)
    ON UPDATE CASCADE
);
```

Relation Name	Attributes	Data Type
allowed_leaves	start_date	datetime
	end_date	datetime
	number_of_leaves	integer
	course_id	string<=30 chars
	professor_id	integer
	department_id	string<=30 chars
	leave_id	integer

Table 10: Allowed Leaves

```
-TEST DAYS
CREATE TABLE AttendanceManagementSystem.test_days(
test_start_date DATETIME NOT NULL,
test_end_date DATETIME NOT NULL,
course_id VARCHAR(30) NOT NULL,
professor_id INT NOT NULL,
department_id VARCHAR(30) NOT NULL,
leave_id INT NOT NULL,
PRIMARY KEY(course_id,professor_id),
-- FOREIGN KEY for TEST_DAYS TABLE
INDEX course_idx (course_id ASC) VISIBLE,
  CONSTRAINT fk1_course_id
    FOREIGN KEY (course_id)
    REFERENCES AttendanceManagementSystem.course (course_id)
    ON UPDATE CASCADE,
    INDEX professor_idx (professor_id ASC) VISIBLE,
  CONSTRAINT fk1_professor_id
    FOREIGN KEY (professor_id)
    REFERENCES AttendanceManagementSystem.professor (professor_id)
    ON UPDATE CASCADE,
    INDEX department_idx (department_id ASC) VISIBLE,
  CONSTRAINT fk1_department_id
    FOREIGN KEY (department_id)
    REFERENCES AttendanceManagementSystem.department (department_id)
    ON UPDATE CASCADE,
    INDEX leave_idx (leave_id ASC) VISIBLE,
  CONSTRAINT fk1_leave_id
    FOREIGN KEY (leave_id)
    REFERENCES AttendanceManagementSystem.leave_type (leave_id)
    ON UPDATE CASCADE);
```

Relation Name	Attributes	Data Type
test_days	test_start_date	datetime
	test_end_date	datetime
	course_id	string<=30 chars

	professor_id	integer
	department_id	string<=30 chars
	leave_id	integer

Table 11: Test Days

```
-STUDENT COURSE
-- Table structure for `student_course`
-- DROP TABLE IF EXISTS `student_course`;
CREATE TABLE AttendanceManagementSystem.student_course(
course_id VARCHAR(30) NOT NULL,
professor_id INT NOT NULL,
department_id VARCHAR(30) NOT NULL,
attendance_status ENUM('Y','N'),
leave_id INT NOT NULL,
student_id INT NOT NULL,
PRIMARY KEY(course_id,professor_id,student_id),
-- FOREIGN KEY for STUDENT_COURSE TABLE
INDEX course_idx (course_id ASC) VISIBLE,
  CONSTRAINT fk2_course_id
    FOREIGN KEY (course_id)
    REFERENCES AttendanceManagementSystem.course (course_id)
    ON UPDATE CASCADE,
    INDEX professor_idx (professor_id ASC) VISIBLE,
  CONSTRAINT fk2_professor_id
    FOREIGN KEY (professor_id)
    REFERENCES AttendanceManagementSystem.professor (professor_id)
    ON UPDATE CASCADE,
    INDEX department_idx (department_id ASC) VISIBLE,
  CONSTRAINT fk2_department_id
    FOREIGN KEY (department_id)
    REFERENCES AttendanceManagementSystem.department (department_id)
    ON UPDATE CASCADE,
    INDEX student_idx (student_id ASC) VISIBLE,
  CONSTRAINT fk2_student_id
    FOREIGN KEY (student_id)
    REFERENCES AttendanceManagementSystem.student (student_id)
    ON UPDATE CASCADE,
    INDEX leave_idx (leave_id ASC) VISIBLE,
  CONSTRAINT fk2_leave_id
    FOREIGN KEY (leave_id)
    REFERENCES AttendanceManagementSystem.leave_type (leave_id)
    ON UPDATE CASCADE);
```

Relation Name	Attributes	Data Type
student_course	student_id	integer

	course_id	string<=30 chars
	professor_id	integer
	department_id	string<=30 chars
	leave_id	integer

Table 12: Student Course

2.7 Normalization

In this part, we apply the principles of normalization to ensure all the tables conform to 3NF. To do this, we document all functional dependencies and indicate how the normalization is performed.

Each table is in at least the Third Normal Form (3NF) and is free of transitive dependencies. This schema is designed to adhere to the principles of normalization, ensuring that data redundancy is minimized, and data integrity is maintained.

2.8 Conclusion

In this report, we discuss and design the relational schema of the AttendanceManagement-System Database. Our EER diagram and its associated relational schema show the conceptual and logical designs of the system. We also define data types and formats for each attribute in the relation. The next step is to implement this database. In the future, we may change some designs due to practical difficulties and other requirements.

Phase III. Implementation

This phase includes an improvised version of ERD and tables.

3.1 Pre-Illumination

This report describes the database project's implementation phase, with an emphasis on database construction, table setup, data population, SQL queries. The project utilizes the MySQL database management system. Part 1 is the modified relational schema. Part 2 is the creation of the database, including tables, all other structures as well as constraints, data type and format, Part 3 is the query scenario design and implementation.

3.2 Modified Relational Schema

In accordance with the requirements outlined in the previous phase and aiming to streamline the relational model for this database, we implemented the following modifications compared to the original relational models.

1) Redundant Data Removal (student_course): We identified and eliminated redundant data stored in the 'student_course' table. Redundancies often lead to unnecessary storage consumption and can cause data inconsistencies or anomalies.
2) Data Type Modifications: We made alterations to the data types within the database schema. This adjustment involved changing the data type of certain attributes to better suit the data being stored.
3) Primary Key Addition (Holiday) and Foreign Key (course_holiday): In the 'holiday' table, we enhanced the primary key by incorporating 'holiday_date.' This adjustment aims to provide a more comprehensive identification of the tuple. Simultaneously, we established a foreign key reference in the 'course_holiday' table, linking it to the newly added primary key in the 'holiday' table. This association helps maintain data consistency across tables and enables relational connections between the two entities.
4) Added Surrogate key in student_course, course_holiday, allowed_leaves and test_days inorder to maintain the uniqueness of each tuple. This addition ensures data integrity, simplifies relational connections, and facilitates efficient data management across the tables.

The modified relational schema is shown in Figure 1.

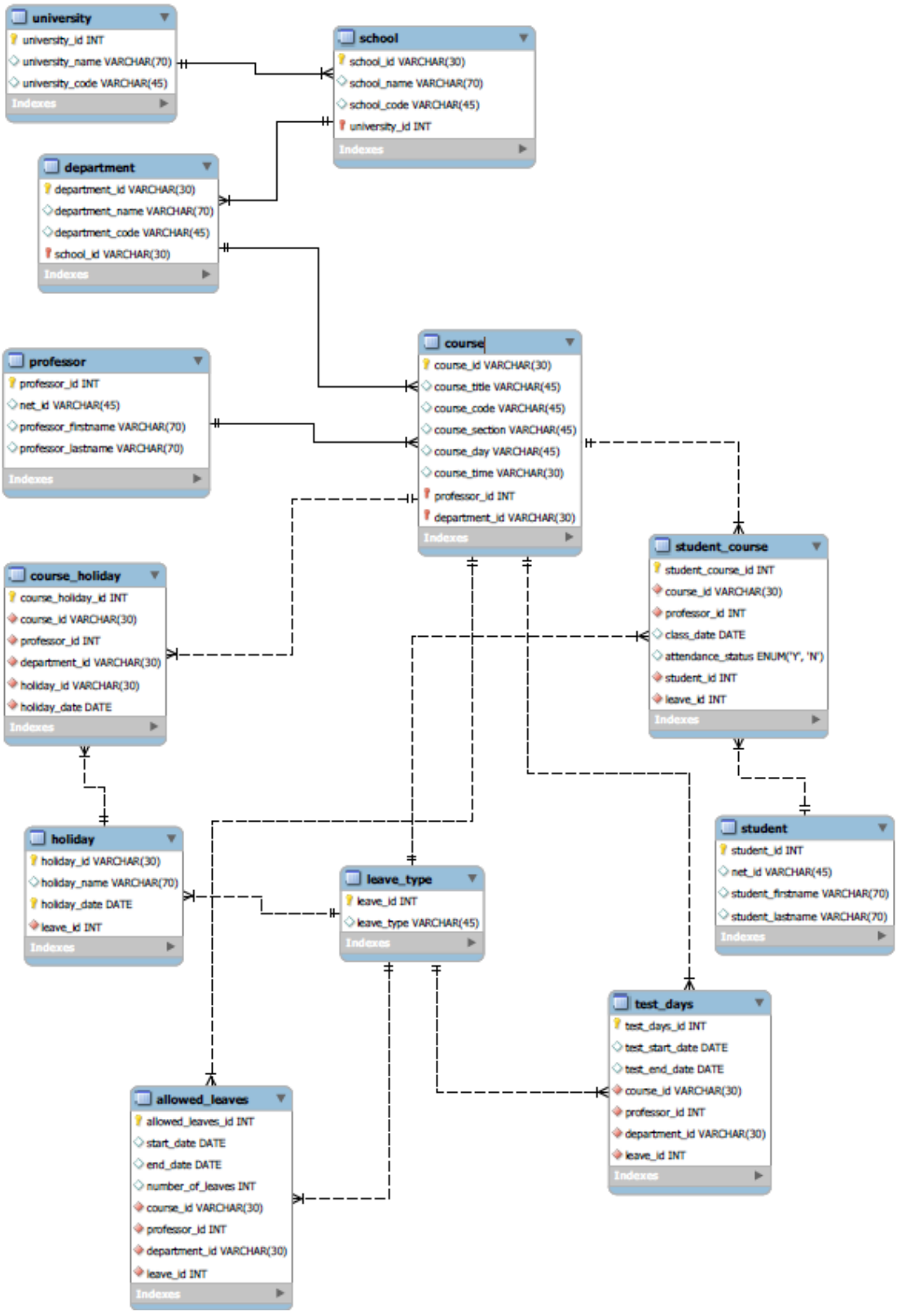

Figure 2

The following section shows how we created our database in MySQL.

3.3 Creation of Database with SQL Statements

```
CREATE DATABASE AttendanceManagementSystem;
-- DROP DATABASE AttendanceManagementSystem;
use AttendanceManagementSystem;
-- SHOW TABLES;
```

3.3a Table Creation

First, we created University table using the following SQL statement:

- **University:**

```
CREATE TABLE University
-- Table structure for `university`
-- DROP TABLE IF EXISTS `university`;
CREATE TABLE AttendanceManagementSystem.university(
university_id INT NOT NULL AUTO_INCREMENT,
university_name VARCHAR(70) NOT NULL,
university_code VARCHAR(45) NOT NULL,
PRIMARY KEY(university_id));
```

- **School:**

```
CREATE TABLE School
-- Table structure for `school`
-- DROP TABLE IF EXISTS `school`;
CREATE TABLE AttendanceManagementSystem.school(
school_id VARCHAR(30) NOT NULL,
school_name VARCHAR(70) NOT NULL,
school_code VARCHAR(45) NOT NULL,
university_id INT NOT NULL,
PRIMARY KEY (school_id),
-- FOREIGN KEY for school TABLE
INDEX university_idx (university_id ASC) VISIBLE,
  CONSTRAINT university_id
    FOREIGN KEY (university_id)
    REFERENCES AttendanceManagementSystem.university (university_id)
    -- ON DELETE SET DEFAULT 1234
    ON UPDATE CASCADE);
```

- **Department:**

```
CREATE TABLE Department
    -- Table structure for `department`
-- DROP TABLE IF EXISTS `department`;
CREATE TABLE AttendanceManagementSystem.department(
department_id VARCHAR(30) NOT NULL,
department_name VARCHAR(70) NOT NULL,
department_code VARCHAR(45) NOT NULL,
school_id VARCHAR(30) NOT NULL,
PRIMARY KEY (department_id),
-- FOREIGN KEY for DEPARTMENT TABLE
INDEX school_idx (school_id ASC) VISIBLE,
  CONSTRAINT school_id
    FOREIGN KEY (school_id)
    REFERENCES AttendanceManagementSystem.school (school_id)
    -- ON DELETE SET DEFAULT 1234
    ON UPDATE CASCADE);
```

- **Course:**

```
CREATE TABLE Course
-- Table structure for `course`
-- DROP TABLE IF EXISTS `course`;
CREATE TABLE AttendanceManagementSystem.course(
course_id VARCHAR(30) NOT NULL,
course_title VARCHAR(45) NOT NULL,
course_code VARCHAR(45) NOT NULL,
course_section VARCHAR(45) NOT NULL,
course_day VARCHAR(45) NOT NULL,
course_time VARCHAR(30) NOT NULL,
professor_id INT NOT NULL,
department_id VARCHAR(30) NOT NULL,
PRIMARY KEY (course_id),
-- FOREIGN KEY for COURSE TABLE
    INDEX professor_idx (professor_id ASC) VISIBLE,
    CONSTRAINT fk4_professor_id
        FOREIGN KEY (professor_id)
        REFERENCES AttendanceManagementSystem.professor (professor_id)
        ON UPDATE CASCADE,
INDEX department_idx (department_id ASC) VISIBLE,
    CONSTRAINT fk4_department_id
        FOREIGN KEY (department_id)
        REFERENCES AttendanceManagementSystem.department (department_id)
        ON UPDATE CASCADE
        );
```

- **Professor:**

```
CREATE TABLE Professor
-- Table structure for `professor`
-- DROP TABLE IF EXISTS `professor`;
CREATE TABLE AttendanceManagementSystem.professor(
professor_id INT NOT NULL AUTO_INCREMENT,
net_id VARCHAR(45) NOT NULL,
professor_firstname VARCHAR(70) NOT NULL,
professor_lastname VARCHAR(70) NOT NULL,
PRIMARY KEY (professor_id));
```

- **Student:**

```
-- Table structure for `student`
-- DROP TABLE IF EXISTS `student`;
CREATE TABLE Student
   CREATE TABLE AttendanceManagementSystem.student(
student_id INT NOT NULL AUTO_INCREMENT,
net_id VARCHAR(45) NOT NULL,
student_firstname VARCHAR(70) NOT NULL,
student_lastname VARCHAR(70) NOT NULL,
PRIMARY KEY (student_id));
```

- **Student_course:**

```
CREATE TABLE Student_course
-- Table structure for `student_course`
-- DROP TABLE IF EXISTS `student_course`;
CREATE TABLE AttendanceManagementSystem.student_course(
student_course_id INT NOT NULL,
course_id VARCHAR(30) NOT NULL,
professor_id INT NOT NULL,
class_date DATE NOT NULL,
attendance_status ENUM('Y','N'),
leave_id INT NOT NULL,
student_id INT NOT NULL,
PRIMARY KEY(student_course_id),
-- FOREIGN KEY for STUDENT_COURSE TABLE
INDEX course_idx (course_id ASC) VISIBLE,
  CONSTRAINT fk2_course_id
    FOREIGN KEY (course_id)
    REFERENCES AttendanceManagementSystem.course (course_id)
    ON UPDATE CASCADE,
    INDEX professor_idx (professor_id ASC) VISIBLE,
  CONSTRAINT fk2_professor_id
    FOREIGN KEY (professor_id)
    REFERENCES AttendanceManagementSystem.professor (professor_id)
    ON UPDATE CASCADE,
    INDEX student_idx (student_id ASC) VISIBLE,
  CONSTRAINT fk2_student_id
    FOREIGN KEY (student_id)
```

```
    REFERENCES AttendanceManagementSystem.student (student_id)
    ON UPDATE CASCADE,
    INDEX leave_idx (leave_id ASC) VISIBLE,
  CONSTRAINT fk2_leave_id
    FOREIGN KEY (leave_id)
    REFERENCES AttendanceManagementSystem.leave_type (leave_id)
    ON UPDATE CASCADE);
```

- **Holiday:**

```
CREATE TABLE Holiday
-- Table structure for `holiday`
-- DROP TABLE IF EXISTS `holiday`;
CREATE TABLE AttendanceManagementSystem.holiday(
    holiday_id VARCHAR(30) NOT NULL,
    holiday_name VARCHAR(70) NOT NULL,
    holiday_date DATE NOT NULL,
    leave_id INT NOT NULL,
    PRIMARY KEY (holiday_id,holiday_date),
    -- FOREIGN KEY for HOLIDAY TABLE
    INDEX leave_idx (leave_id ASC) VISIBLE,
    CONSTRAINT leave_id
        FOREIGN KEY (leave_id)
        REFERENCES AttendanceManagementSystem.leave_type (leave_id)
        ON UPDATE CASCADE
);
```

- **Course_holiday:**

```
CREATE TABLE Course_holiday
-- Table structure for `course_holiday`
-- DROP TABLE IF EXISTS `course_holiday`;
CREATE TABLE AttendanceManagementSystem.course_holiday(
course_holiday_id int NOT NULL,
course_id VARCHAR(30) NOT NULL,
professor_id INT NOT NULL,
department_id VARCHAR(30) NOT NULL,
holiday_id VARCHAR(30) NOT NULL,
holiday_date DATE NOT NULL,
leave_id INT NOT NULL,
PRIMARY KEY(course_holiday_id),
-- FOREIGN KEY for STUDENT_COURSE TABLE
INDEX course_idx (course_id ASC) VISIBLE,
  CONSTRAINT fk3_course_id
    FOREIGN KEY (course_id)
    REFERENCES AttendanceManagementSystem.course (course_id)
    ON UPDATE CASCADE,
    INDEX professor_idx (professor_id ASC) VISIBLE,
  CONSTRAINT fk3_professor_id
    FOREIGN KEY (professor_id)
    REFERENCES AttendanceManagementSystem.professor (professor_id)
    ON UPDATE CASCADE,
    INDEX department_idx (department_id ASC) VISIBLE,
  CONSTRAINT fk3_department_id
    FOREIGN KEY (department_id)
    REFERENCES AttendanceManagementSystem.department (department_id)
```

```
    ON UPDATE CASCADE,
    INDEX holiday_idx (holiday_id ASC) VISIBLE,
  CONSTRAINT fk3_holiday_id
    FOREIGN KEY (holiday_id,holiday_date)
    REFERENCES AttendanceManagementSystem.holiday (holiday_id,holiday_date)
    ON UPDATE CASCADE,
    INDEX leave_idx (leave_id ASC) VISIBLE,
  CONSTRAINT fk3_leave_id
    FOREIGN KEY (leave_id)
    REFERENCES AttendanceManagementSystem.leave_type (leave_id)
    ON UPDATE CASCADE
    );
```

- **Leave_type:**

```
CREATE TABLE Leave_type
-- Table structure for `leave_type`
-- DROP TABLE IF EXISTS `leave_type`;
CREATE TABLE AttendanceManagementSystem.leave_type(
leave_id INT NOT NULL AUTO_INCREMENT,
leave_type VARCHAR(45) NOT NULL,
PRIMARY KEY (leave_id));
```

- **Test_days:**

```
CREATE TABLE Test_days
-- Table structure for `test_days`
-- DROP TABLE IF EXISTS `test_days`;
CREATE TABLE AttendanceManagementSystem.test_days(
test_days_id INT NOT NULL,
test_start_date DATE NOT NULL,
test_end_date DATE NOT NULL,
course_id VARCHAR(30) NOT NULL,
professor_id INT NOT NULL,
department_id VARCHAR(30) NOT NULL,
leave_id INT NOT NULL,
PRIMARY KEY(test_days_id),
-- FOREIGN KEY for TEST_DAYS TABLE
INDEX course_idx (course_id ASC) VISIBLE,
  CONSTRAINT fk1_course_id
    FOREIGN KEY (course_id)
    REFERENCES AttendanceManagementSystem.course (course_id)
    ON UPDATE CASCADE,
    INDEX professor_idx (professor_id ASC) VISIBLE,
  CONSTRAINT fk1_professor_id
    FOREIGN KEY (professor_id)
    REFERENCES AttendanceManagementSystem.professor (professor_id)
    ON UPDATE CASCADE,
    INDEX department_idx (department_id ASC) VISIBLE,
  CONSTRAINT fk1_department_id
    FOREIGN KEY (department_id)
    REFERENCES AttendanceManagementSystem.department (department_id)
    ON UPDATE CASCADE,
    INDEX leave_idx (leave_id ASC) VISIBLE,
  CONSTRAINT fk1_leave_id
    FOREIGN KEY (leave_id)
    REFERENCES AttendanceManagementSystem.leave_type (leave_id)
    ON UPDATE CASCADE);
```

- **Allowed_leaves:**

```
CREATE TABLE Allowed_leaves
-- Table structure for `allowed_leaves`
-- DROP TABLE IF EXISTS `allowed_leaves`;
CREATE TABLE AttendanceManagementSystem.allowed_leaves(
allowed_leaves_id INT NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    number_of_leaves INT NOT NULL,
    course_id VARCHAR(30) NOT NULL,
    professor_id INT NOT NULL,
    department_id VARCHAR(30) NOT NULL,
    leave_id INT NOT NULL,
    PRIMARY KEY (allowed_leaves_id),
    -- FOREIGN KEY for ALLOWED_LEAVES TABLE
    INDEX course_idx (course_id ASC) VISIBLE,
    CONSTRAINT fk_course_id
        FOREIGN KEY (course_id)
        REFERENCES AttendanceManagementSystem.course (course_id)
        ON UPDATE CASCADE,
    INDEX professor_idx (professor_id ASC) VISIBLE,
    CONSTRAINT fk_professor_id
        FOREIGN KEY (professor_id)
        REFERENCES AttendanceManagementSystem.professor (professor_id)
        ON UPDATE CASCADE,
    INDEX department_idx (department_id ASC) VISIBLE,
    CONSTRAINT fk_department_id
        FOREIGN KEY (department_id)
        REFERENCES AttendanceManagementSystem.department (department_id)
        ON UPDATE CASCADE,
    INDEX leave_idx (leave_id ASC) VISIBLE,
    CONSTRAINT fk_leave_id
        FOREIGN KEY (leave_id)
        REFERENCES AttendanceManagementSystem.leave_type (leave_id)
        ON UPDATE CASCADE
);
```

3.3b A Database State

For testing and development needs, the database was populated with sample data inserted into every table. The inserted records were meticulously added to maintain both data consistency and validity across all tables.

- **INSERTION OF TABLE University**

```
-- Inserting into university using table import data wizard
-- SELECT * FROM AttendanceManagementSystem.university;
INSERT    INTO    AttendanceManagementSystem.university    (university_id    ,
```

```
university_name, university_code)
VALUES ('2401', 'The University of Texas at Dallas', 'UTD');
```

- **INSERTION OF TABLE School**

```
-- Inserting into school using table import data wizard
-- SELECT * FROM AttendanceManagementSystem.school;
INSERT INTO AttendanceManagementSystem.school
(school_id , school_name, school_code, university_id)
VALUES ('3651', 'Naveen Jindal School of Management', 'JSOM','2401');
```

- **INSERTION OF TABLE Department**

```
-- Inserting into department using table import data wizard
-- SELECT * FROM AttendanceManagementSystem.department;
INSERT INTO AttendanceManagementSystem.department
(department_id , department_name, department_code, school_id)
VALUES ('1251',   'Business Analytics', 'BUAN', '3651');
```

- **INSERTION OF TABLE course**

```
INSERT INTOAttendanceManagementSystem.course
(course_id,course_title,course_code,course_section,course_day,course_time,p
rofessor_id,department_id)
VALUES ('FIN6368.0001', 'Financial Information and
Analysis','FIN6368','0001','Tuesday','4pm-7:45pm','15285748','1251');
-- Inserting into course using table import data wizard
-- SELECT * FROM AttendanceManagementSystem.course;
```

- **INSERTION OF TABLE professor**

```
INSERT INTO AttendanceManagementSystem.professor
(professor_id,net_id,professor_firstname,professor_lastname)
VALUES ('15285748','KDP730374','Fayth','Caldero');
-- Inserting into professor using table import data wizard
-- SELECT * FROM AttendanceManagementSystem.professor;
```

- **INSERTION OF TABLE student**

```
INSERT INTO AttendanceManagementSystem.student
(student_id,net_id,student_firstname,student_lastname)
VALUES ('14708852','AVR224375','Carson','Betton');
-- Inserting into student using table import data wizard
-- SELECT * FROM AttendanceManagementSystem.student;
```

- **INSERTION OF TABLE student_course**

```
INSERT INTO AttendanceManagementSystem.student_course
(course_id,professor_id,attendance_status,leave_id,student_id)
VALUES ('FIN6368.0001','15285748','Y','1','14708852');
-- Inserting into student_course using table import data wizard
-- SELECT * FROM AttendanceManagementSystem.student_course;
```

- **INSERTION OF TABLE holiday**

```
INSERT INTO AttendanceManagementSystem.holiday
(holiday_id,holiday_name,holiday_date,leave_id)
VALUES ('hol_17','Spring break','2024-03-17','1');
-- Inserting into holiday using table import data wizard
-- SELECT * FROM AttendanceManagementSystem.holiday;
```

INSERTION OF TABLE course_holiday

```
INSERT INTO AttendanceManagementSystem.course_holiday
(course_id,professor_id,department_id,holiday_id,holiday_date,leave_id)
VALUES ('FIN6368.0001','15285748','1251','hol_17','2024-03-17','1');
-- Inserting into course_holiday using table import data wizard
-- SELECT * FROM AttendanceManagementSystem.course_holiday;
```

- **INSERTION OF TABLE leave_type**

```
INSERT INTO AttendanceManagementSystem.leave_type
(leave_id,leave_type)
VALUES ('1','holiday');
-- Inserting into leave_type using table import data wizard
```

```
-- SELECT * FROM AttendanceManagementSystem.leave_type;
```

- **INSERTION OF TABLE test_days**

```
INSERT INTO AttendanceManagementSystem.test_days
(test_start_date,test_end_date,course_id,professor_id,department_id,leave_i
d)
VALUES ('2023-10-03','2023-10-04','FIN6368.0001','15285748','1251','1');
-- Inserting into test_days using table import data wizard
-- SELECT * FROM AttendanceManagementSystem.test_days;
```

- **INSERTION OF TABLE allowed_leaves**

```
INSERT INTO AttendanceManagementSystem.allowed_leaves
(start_date,end_date,number_of_leaves,course_id,professor_id,department_id,
leave_id)
VALUES ('2023-08-21','2024-01-
15','2','FIN6368.0001','15285748','1251','1');
-- Inserting into allowed_leaves using table import data wizard
-- SELECT * FROM AttendanceManagementSystem.allowed_leaves;
```

We inserted the remaining data using the 'table data import wizard' command in MySQL.

DataTables

3.4 Query Scenario Design (including questions and answers)

Query 01: Retrieve the count of allowed leaves for each course:

SELECT c.course_id,COUNT(al.allowed_leaves_id) AS allowed_leaves_count

FROM AttendanceManagementSystem.course c

LEFT JOIN AttendanceManagementSystem.allowed_leaves al ON c.course_id = al.course_id

GROUP BY c.course_id;

Result of Query 01:

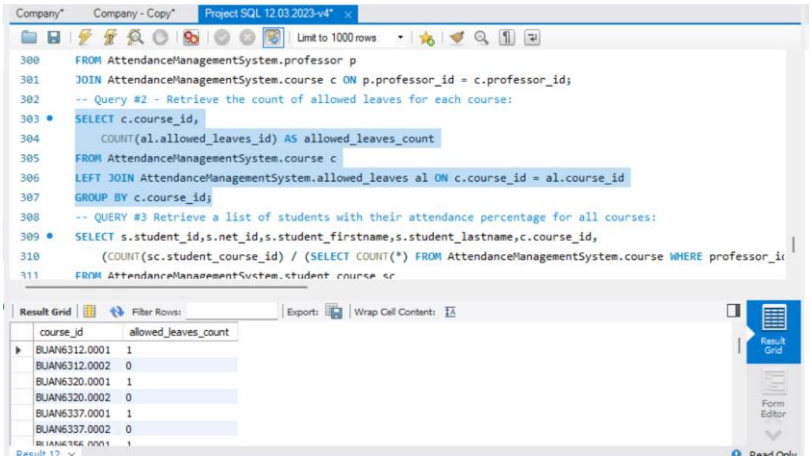

FIGURE 3

Query 02: -- Retrieve a list of students with their attendance percentage for all courses:

```
SELECT s.student_id,s.net_id,s.student_firstname,s.student_lastname,c.course_id,

    (COUNT(sc.student_course_id) / (SELECT COUNT(*) FROM AttendanceManagementSystem.course WHERE professor_id = '15285748')) * 100 AS attendance_percentage

FROM AttendanceManagementSystem.student_course sc

JOIN AttendanceManagementSystem.student s ON sc.student_id = s.student_id

JOIN AttendanceManagementSystem.course c ON sc.course_id = c.course_id

WHERE sc.attendance_status = 'Y'

GROUP BY s.student_id, c.course_id;
```

Result of Query 02:

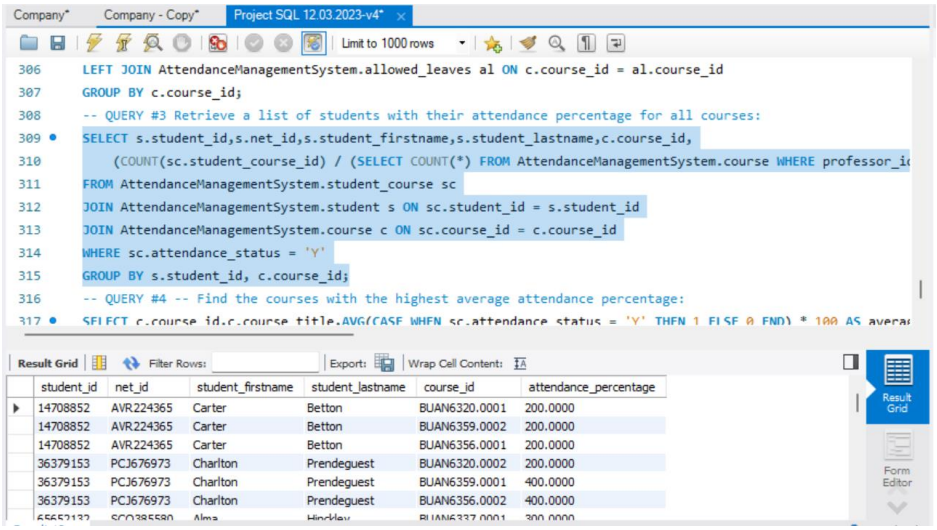

Figure 4

Query 03: -- -- Find the courses with the highest average attendance percentage:

```
SELECT c.course_id,c.course_title,AVG(CASE WHEN sc.attendance_status = 'Y' THEN 1 ELSE 0 END) * 100 AS average_attendance_percentage

FROM AttendanceManagementSystem.course c

LEFT JOIN AttendanceManagementSystem.student_course sc ON c.course_id = sc.course_id

GROUP BY c.course_id, c.course_title

ORDER BY average_attendance_percentage DESC;
```

Result of Query 03:

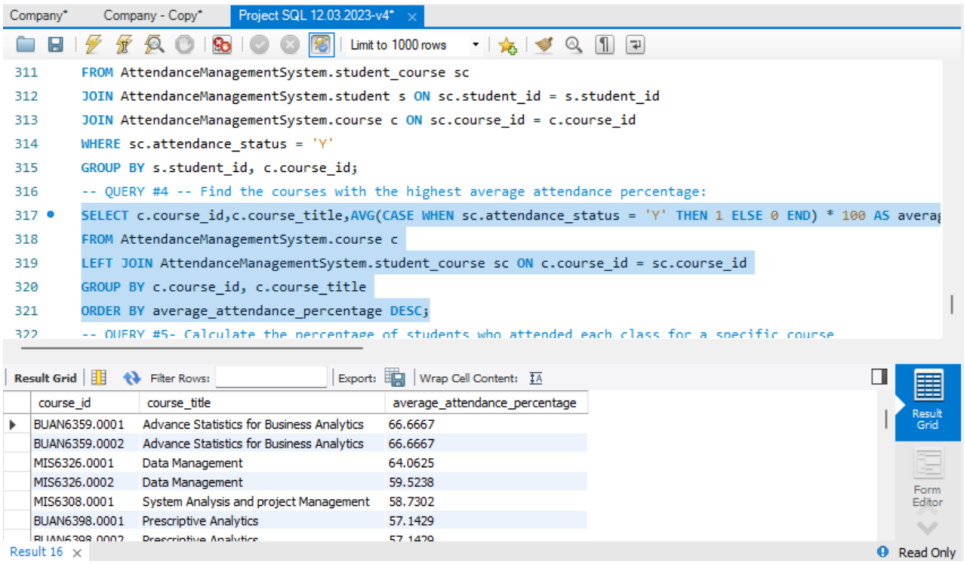

Figure 5

Query 04: -- Calculate the percentage of students who attended each class for a specific course

```
SELECT c.course_id,c.course_title,sc.class_date,

    (COUNT(sc.student_id) / (SELECT COUNT(*) FROM AttendanceManagementSystem.student WHERE student_id IN (SELECT student_id FROM AttendanceManagementSystem.student_course WHERE course_id = c.course_id))) * 100 AS attendance_percentage

FROM AttendanceManagementSystem.course c

LEFT JOIN AttendanceManagementSystem.student_course sc ON c.course_id = sc.course_id

GROUP BY c.course_id, c.course_title, sc.class_date

ORDER BY c.course_id, sc.class_date;
```

Result of Query 04:

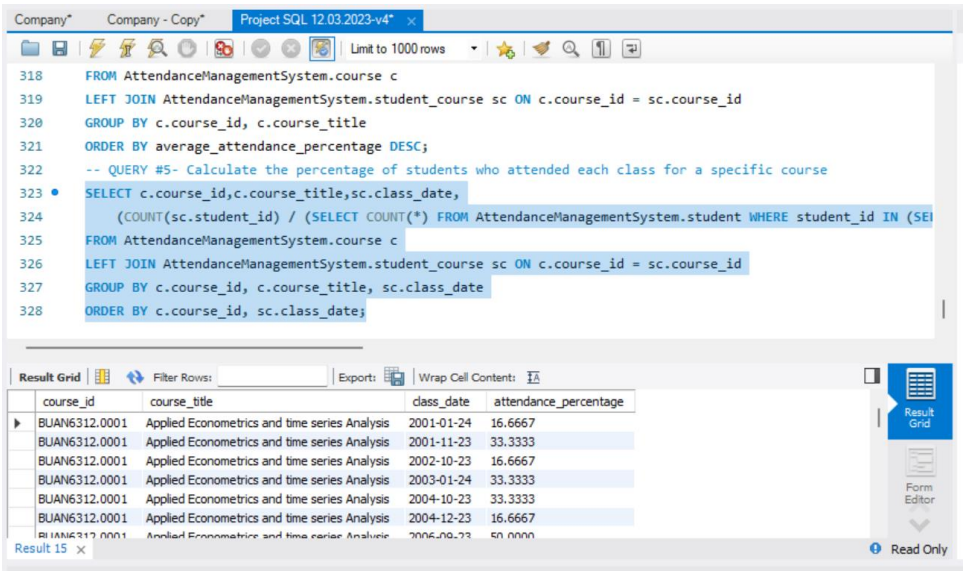

Figure 6

Query 05: -- Retrieve Leaves and holidays of a particular student('27059590') in all their courses

```
SELECT student_id,course_id,attendance_status, COUNT(*) AS absent_days
FROM student_course
WHERE attendance_status = 'N' and student_id = '27059590' AND leave_id IN (1,2)
GROUP BY student_id,course_id;
```

Result of Query 05:

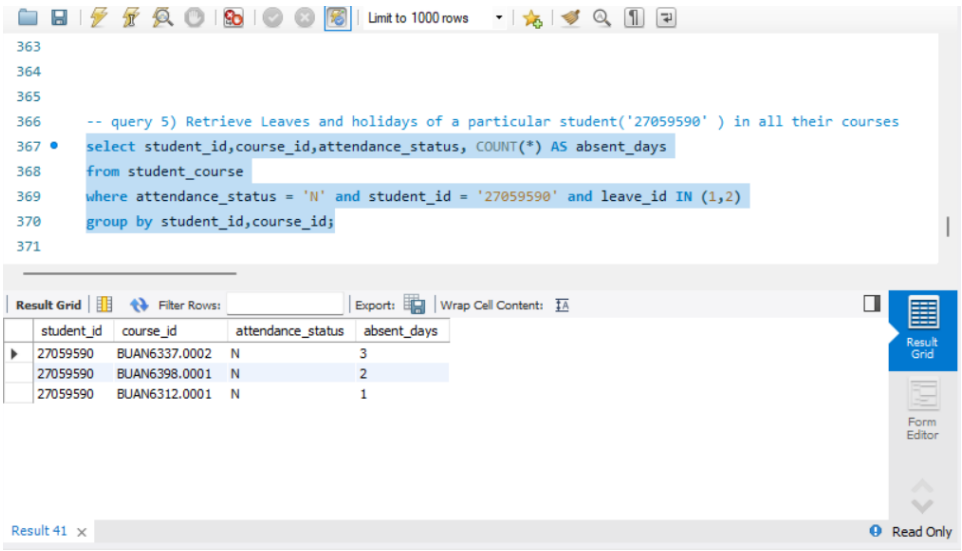

Figure 7

Query 06: -- Attendance in a particular month (21 Aug 2021 -21 Sept 2023) for department 1377.

```
SELECT d.department_id, d.department_code, sc.student_id, c.course_id,
sc.attendance_status
FROM student_course sc INNER JOIN course c ON sc.course_id = c.course_id
INNER JOIN department d ON c.department_id = d.department_id
WHERE d.department_id = 1377 AND sc.class_date BETWEEEN '2023-08-21' AND '2023-09-
21';
```

Result of Query 06:

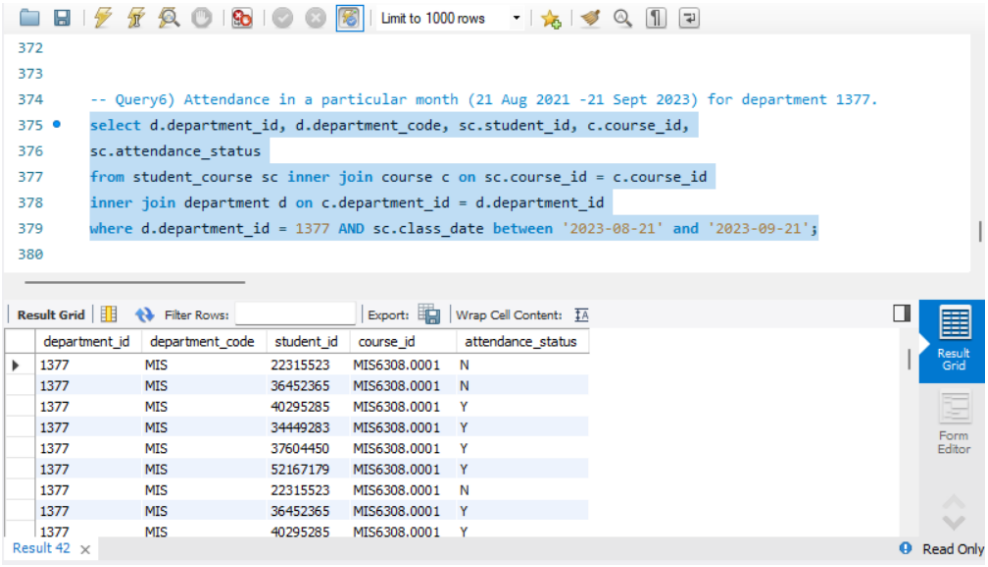

Figure 8

Query 07: -- Number of actual leaves (other than test days, allowed leaves, holidays) for student in BUAN6320.0001

```
SELECT student_id,course_id, COUNT(*) AS actual_leaves
FROM student_course
WHERE course_id = 'BUAN6320.0001' AND
attendance_status = 'N' AND leave_id NOT IN (SELECT test_days_id FROM test_days
WHERE course_id = 'BUAN6320.0001' UNION SELECT allowed_leaves_id FROM
allowed_leaves WHERE course_id = 'BUAN6320.0001' UNION SELECT holiday_id
FROM holiday)
group by student_id;
```

Result of Query 07:

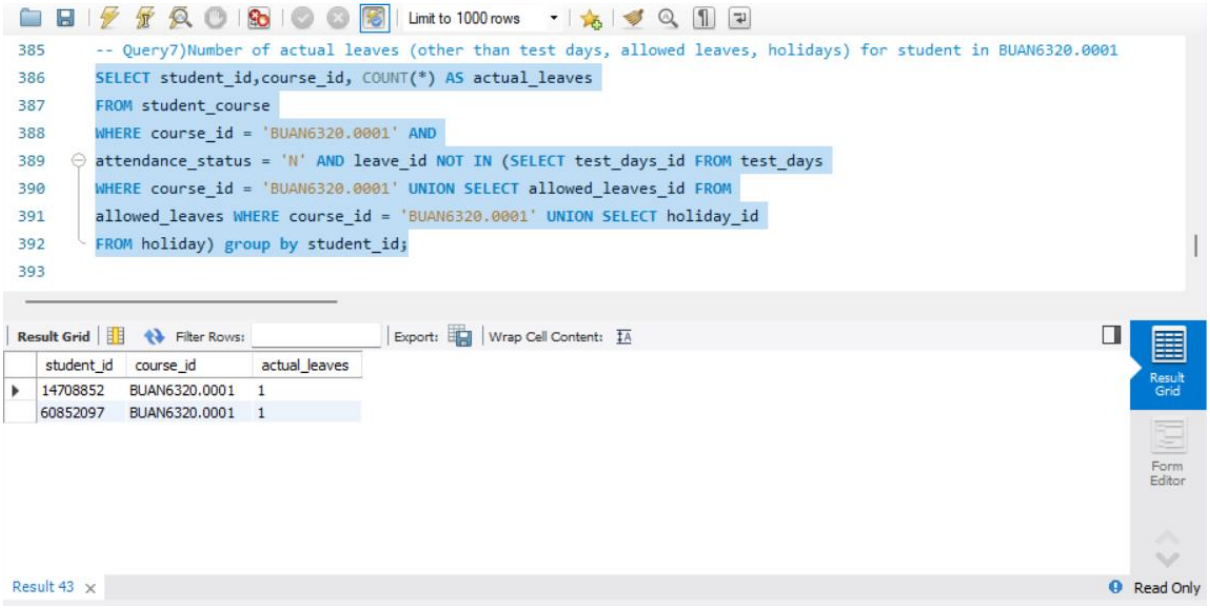

Figure 9

Query 08: -- Retrieve the average attendance for each student in each course within department 1251 during a specified date range ('2023-08-21' to '2023-12-15').

SELECT sc.course_id, s.student_id, AVG(CASE WHEN sc.attendance_status = 'Y' THEN 1 ELSE 0 END) AS average_attendance FROM student s INNER JOIN student_course sc ON s.student_id = sc.student_id inner join course c on sc.course_id = c.course_id WHERE c.department_id = 1251 AND sc.class_date BETWEEN '2023-08-21' AND '2023-12-15' GROUP BY sc.course_id,s.student_id;

Result of Query 08:

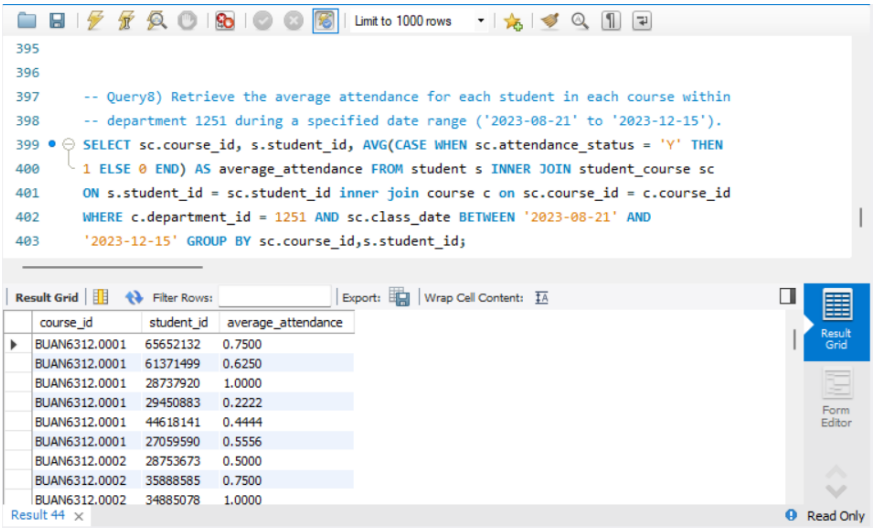

Figure 10

3.6 User Interface System

Following technologies were used in designing the user interface for this system:

HTML - To create pages containing forms, form elements like textbox, button, drop downs, labels, page titles and other messages

CSS - Used to maintain consistent styling of elements like headings, form elements across all the pages in this system

JavaScript - used to display the current date in the system pages, handle button click and redirect to respective pages and populate data in the attendance report table using eventListeners

3.7 Conclusion

The report thoroughly explains how we set up and used our database. It covers each step we took to build the database, add information to it, and find important details using specific queries.