UNIVERSITY STUDENT INFORMATION SYSTEM (USIS)

GROUP MEMBERS

JOSEPH OMUYONGA - 1048247

KUNGU TIMOTHY MAINA - 1049047

NOBERT MUMA ONDIEKI - 1049075

OTIENO CHRIS HENK - 1049481

KABI RICHARD GITHUA – 1052075

SUBMISSION DATE: 22/11/2024

INTRODUCTION

Overview

We were tasked with creating a University Student Information System (USIS). This system represents a real life university setting database. We have various tables in our database that store all the relevant information of a student. To give a high level view of the database, the tables we have include: student, grade, assessment, assessment_type, enrollment, course, semester, unit_registration, faculty, unit, department. This system allows us to manage student registrations, courses and grades.

Rationale

The tables in our university student information system database are related so that the student information is consistently stored and accessed across various entities such as courses, grades and departments enabling efficient data management and retrieval.

Objectives

Our objectives are as follows;

Student management:

1. To efficiently manage student records including personal information and contact details
2. Facilitate student enrollment and registration processes

Course management:

1. Manage the courses that are offered
2. Assign faculty to courses
3. Track course enrollment

Grade management:

1. To record and manage student grades for various assessments
2. Calculate students' overall performance

SYSTEM DESIGN

Table Structures

1. student(student_id, reg_number, first_name, last_name, date_of_birth, gender, email, phone_number, next_of_kin_name, next_of_kin_phone, next_of_kin_relationship)
2. grade(grade_id, student_id, assessment_id, score, grade_letter, remarks, graded_date)
3. enrollment(enrollment_id, student_id, course_id, session_id, current_year, current_semester, enrollment_date, status)
4. course(course_id, course_name, dept_id, duration_years, fee_per_semester, total_units)
5. semester(session_id, session_name, start_date, end_date, is_current)
6. unit(unit_id, unit_code, unit_name, course_id, credit_hours, year_offered, semester_offered, is_core, prerequisites)
7. unit_registration(registration_id, student_id, unit_id, session_id, registration_date, status)
8. department(dept_id, dept_name, faculty_id, hod_name)
9. faculty(faculty_id, faculty_name)
10. assessment(assessment_id, unit_id, type_id, assessment_name, max_score, weight_percentage, assessment_date)
11. assessment_type(type_id, type_name, description)

The foreign keys include:

1. Grade

   student_id → References student.student_id

   assessment_id → References assessment.assessment_id
2. Enrollment

   student_id → References student.student_id

   course_id → References course.course_id

   session_id → References semester.session_id
3. Course

   dept_id → References department.dept_id
4. Unit

   course_id → References course.course_id
5. Unit Registration

student_id → References student.student_id

unit_id → References unit.unit_id

session_id → References semester.session_id

6. Department

   faculty_id → References faculty.faculty_id

7. Assessment

   unit_id → References unit.unit_id

   type_id → References assessment_type.type_id

Entity Relationship in the USIS

1. Student to Grade:

   Relationship: Awarded

   Cardinality: A student can have many grades, but each grade is associated with only one student.

2. Student to Unit Registration:

   Relationship: Register

   Cardinality: A student can register for many units, and a unit can have many students registering for it.

3. Unit Registration to Unit:

   Relationship: Has

   Cardinality: A unit can have many registrations, and each registration can be linked to one unit.

4. Unit to Course:

   Relationship: Assessed

   Cardinality: A unit is assessed by many assessments, and each assessment belongs to one unit.

5. Department to Faculty:

   Relationship: Have

   Cardinality: A department belongs to one faculty, but a faculty can have many departments.

6. Department to Course:

   Relationship: Have

Cardinality: A department can have many courses, and each course belongs to one department.

7. Unit to Assessment:

   Relationship: Assessed

   Cardinality: A unit can be assessed by many assessments, and each assessment is tied to one unit.

8. Assessment to Assessment Type:

   Relationship: Has

   Cardinality: An assessment can have many assessment types, and each assessment type can have one assessment.

SQL Schema

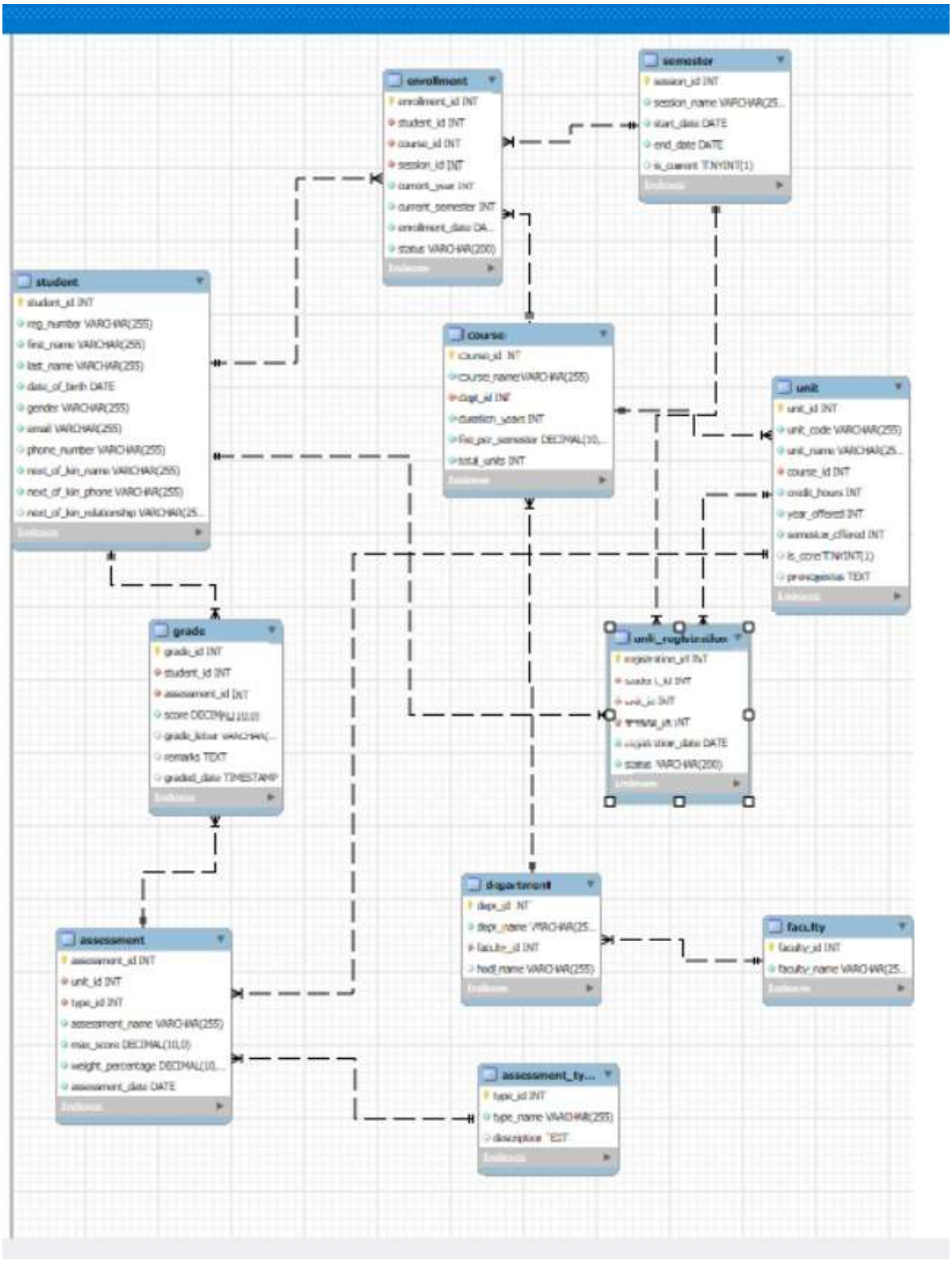

Figure 1: Database Schema made using MySQL Workbench

IMPLEMENTATION

CRUD Operations

The Data Definition Language, which is known for defining the structure of a database, was used in the following operations:

1. Creating database tables using the CREATE command.
2. Modifying the database structure by adding foreign keys, modifying columns and adding constraints which was achieved by the use of ALTER command.
3. Deleting an unnecessary table was achieved by the use of the DROP command

The Data Manipulation Language, was used to manipulate the database in the following ways:

1. Inserting new students, courses, or enrollments into the database involved using INSERT command which enables the database admin to populate data in the tables.
2. The UPDATE command was used to modify existing records in the database.
3. In the case where a department was dissolved, all its existing records were removed using the DELETE command.
4. In order to retrieve records from the database, the SELECT command was put to use thus enabling users to access the data.

Advanced SQL Queries

The database also implemented the use of advanced sql queries such as stored procedures, views, and triggers which perform complex data analysis and manipulations in the database.

The following operations were included in the database:

1. Stored Procedures

   Stored procedures were created to automate frequent operations:

   a. addStudent: Automates adding a new student.
   b. enrollStudent: Simplifies enrolling a student in a course.
   c. updateStudentContact: Updates a student's email and phone number.
   d. getStudentsByCourse: Retrieves all students enrolled in a specific course.
   e. getStudentGrades: Fetches grades for a particular student.
   f. deleteStudent: Removes a student's records from all related tables.

2. Triggers

   Triggers were implemented to enforce data integrity and automate updates:

   a. after_student_delete**:** Logs every deletion of a student into a `student_deletions` table.
   b. before_enrollment_insert: Prevents future enrollment dates.
   c. before_update_semester: Ensures only one semester is marked as current at a time.
   d. before_grade_insert: Ensures grades do not exceed the maximum score set for an assessment.
   e. after_grade_insert: Automatically sets remarks (Excellent, Good, Needs Improvement) based on the student's score.

3. Views

   Views were created to simplify data retrieval and provide summarized information:

   a. student_course_view: Displays student details along with their enrolled courses.
   b. student_grades_view: Shows students' assessments and the grades they received.
   c. course_overview_view: Summarizes courses by their departments and faculties.
   d. unit_registration_view: Lists students and their registered units.
   e. semester_summary_view: Summarizes semester details, including the number of students enrolled.
   f. faculty_enrollment_view: Shows the total number of students enrolled under each faculty.

TESTING AND VALIDATION

The database system was tested to ensure that it met its functional requirements.

Testing

Unit testing was done on some of the advanced sql operations to ensure they behaved as expected.

The following tests were conducted on various queries:

1. Stored procedures were tested as follows:
   a. addStudent Procedure: Tested by inserting different student records with both valid and invalid data to check if students were correctly added to the database and errors were properly handled.
   b. deleteStudent Procedure: Tested by deleting student records and checking if associated records (grades, unit registrations, and enrollments) were also removed.
   c. getStudentsByCourse: Tested to confirm it returns accurate lists of students enrolled in specific courses.

   The data was inserted successfully when the correct details were keyed in. The queried details in getStudentByCourse were retrieved as requested and invalid inputs were handled as expected.

2. Triggers were tested as follows:
   a. before_enrollment_insert was tested to ensure that future enrollment dates are rejected by the system.

   The before_enrollment_insert trigger successfully blocked any inserted invalid future dates for enrollment.

Validation

The views were validated by running sample queries and comparing the results with the expected data.

The following views were tested:

a. student_course_view: This was tested by fetching data and verifying if student and course details were correctly displayed for enrolled students.
b. semester_summary_view: This was tested by verifying that the number of students enrolled in each semester is accurately calculated.

The views provided were accurate and proved to be relevant as the results matched the expected outputs.

The database system was tested thoroughly so as to ensure correctness and integrity of the data. CRUD operations were handled as well, and some of the errors that occurred during the testing phase were dealt with. In the end, all tests were passed proving that the system functions as expected.

CONCLUSION AND RECOMMENDATIONS

Conclusion

The team concluded that the development of the project was a success as it handled basic operations such as adding a student, enrolling students, registering them for other units and other operations. Advanced queries were included to ease the burden on the database administrator when inserting new data or when updating student information by the use of stored procedures and triggers while views improved the readability and simplified the use of complex queries. The system is robust enough to handle typical academic scenarios, and with future enhancements, it can be expanded to support larger-scale operations in a real-world environment. This project has demonstrated the importance of designing databases that not only meet functional requirements but also prioritize data accuracy, efficiency, and maintainability.

Recommendations

Based on the development and testing of the University Student Information System (USIS), several recommendations can be made for future improvements of the system. These are:

1. Email Validation

   Improve data integrity by validating emails that is, validate the email and verify if it exists to avoid adding non-existent emails in the system.
2. User Interface

   Develop a user interface to improve the user's interaction with the system thus removing the need for manual entry of data.
3. Data security

   Implement the use of log files so as to know when data is added, updated or deleted in the system. This helps the database administrator to know when changes were made and who made them improving the overall security of the system.

APPENDIX

Appendix I: Entity Relationship Diagram

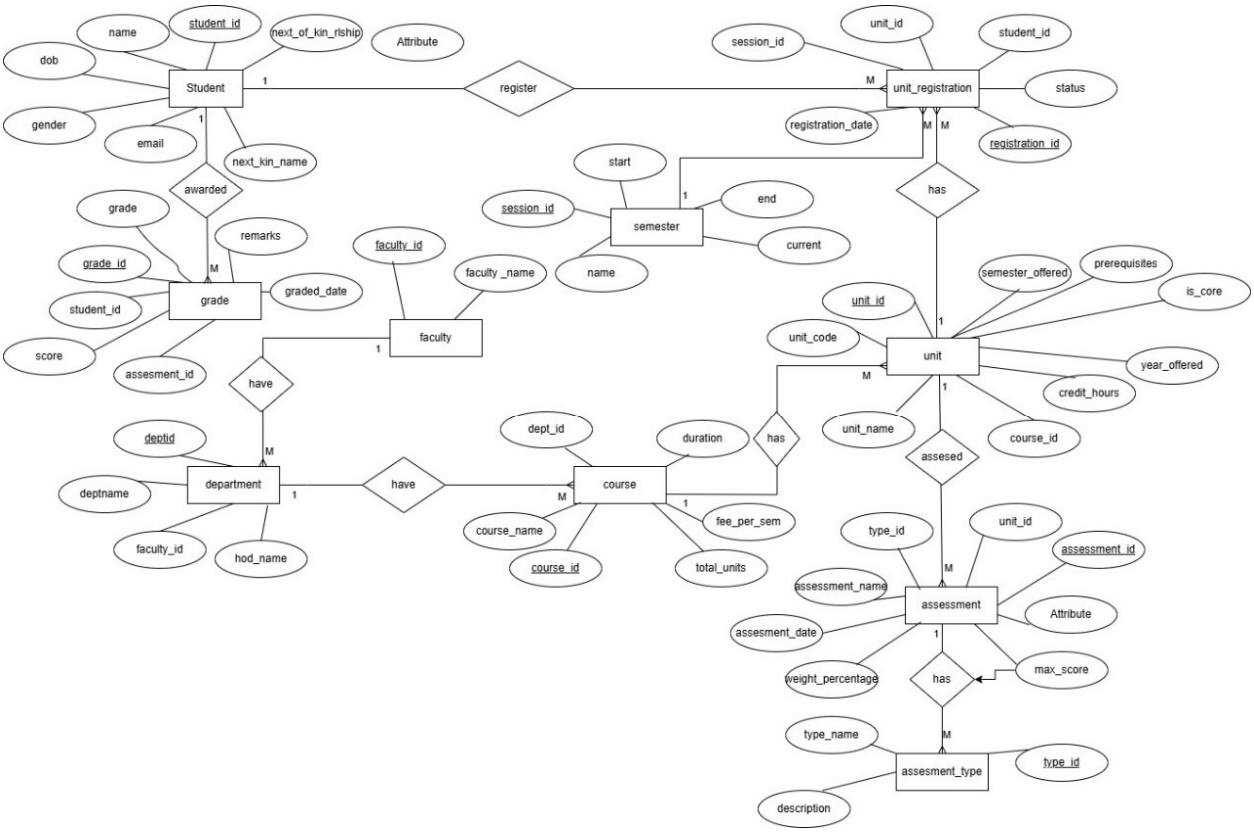

Figure 2: USIS Entity Relationship Diagram

Appendix II: SQL Schema

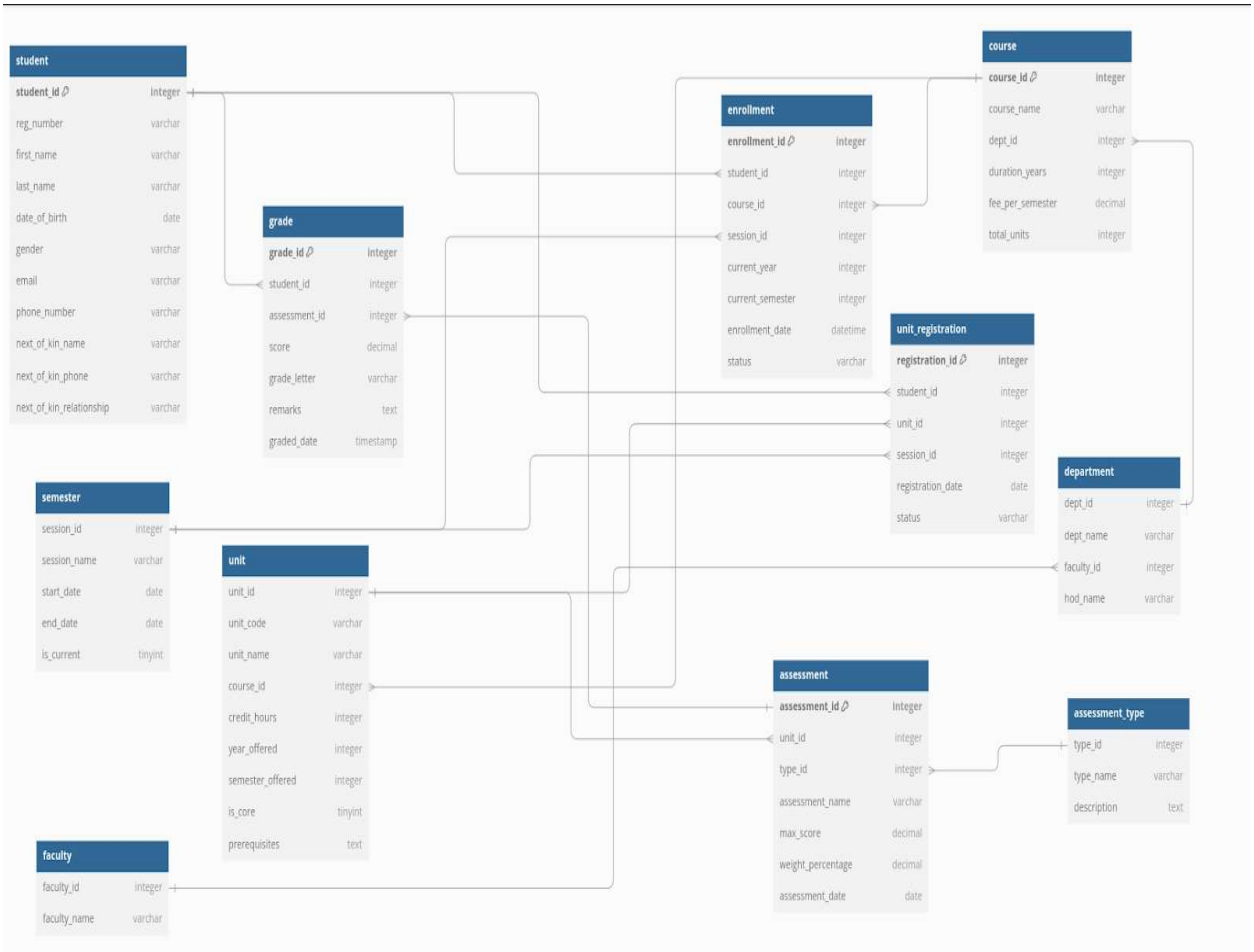

Figure 3: Database Schema made using DBDiagram.io

Appendix III: Code Snippets

Views

```
-- display the total number of students enrolled in courses under each faculty

CREATE VIEW faculty_enrollment_view AS

SELECT

   f.faculty_name,

   COUNT(e.enrollment_id) AS total_enrollments

FROM

   faculty f

JOIN department d ON f.faculty_id = d.faculty_id

JOIN course c ON d.dept_id = c.dept_id

JOIN enrollment e ON c.course_id = e.course_id

GROUP BY f.faculty_name;
```

Triggers

```
-- automatically set the is_current column of other semesters to false when a new semester is marked as current

CREATE TRIGGER before_update_semester

BEFORE UPDATE ON semester

FOR EACH ROW

BEGIN

   IF NEW.is_current = true THEN

      UPDATE semester

      SET is_current = false
```

```
    WHERE session_id != NEW.session_id;
  END IF;
END //
```

Stored Procedures

```
CREATE PROCEDURE addStudent (
IN reg_number VARCHAR(255),
IN firs_name VARCHAR(100),
IN last_name VARCHAR(100),
IN date_of_birth DATE,
IN gender VARCHAR(10),
IN email VARCHAR(100),
IN phone_number VARCHAR(15),
IN next_of_kin_name VARCHAR(100),
IN next_of_kin_phone VARCHAR(15),
IN next_of_kin_relationship VARCHAR(30)
)
BEGIN
INSERT INTO student(
reg_number, first_name, last_name, date_of_birth, gender, email,
phone_number, next_of_kin_name, next_of_kin_phone, next_of_kin_relationship
)
VALUES(
reg_number, first_name, last_name, date_of_birth, gender, email,
phone_number, next_of_kin_name, next_of_kin_phone, next_of_kin_relationship
```

```
);

END //
```

Appendix IV: Validation Results

```
100
101 •  SELECT * FROM semester_summary_view WHERE is_current = 1;
```

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

session_id	session_name	start_date	end_date	is_current	total_students_enrolled
1	2024/2025 Semester 1	2024-09-01	2024-12-31	1	4

Figure 4: semester_summary_view

```
92
93 •  SELECT * FROM student_course_view;
```

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

student_id	reg_number	student_name	course_id	course_name	current_year	current_semester	enrollment_status
1	CS2021	John Paul	1	Bsc Computer Science	1	1	Graduated
2	CS2002	Alice Wangari	1	Bsc Computer Science	2	1	Active
3	CS2023	James Omondi	1	Bsc Computer Science	1	1	Active
5	CS2008	Manson	1	Bsc Computer Science	1	1	Active

Figure 5: student_course_view