Cambodia Academy of Digital Technology

Institute of Digital Technology

Faculty of Digital Engineering

Department of Computer Science

Database Analysis and Design

Project: Academic Skill Development Center

lecturer Kak Soky

Teaching Assistant Heang Sopagna

Submitted By

Chhun Sivheng	(IDTB100035)
Sat Panha	(IDTB100033)
Phy Vathanak	(IDTB100037)

G3 (SE GEN 10)

Date: 03 March, 2025

PHNOM PENH

Table of Contents

1 Introduction 1
1.1 Project description 1
1.2 Objective 1
1.3 Project Timeline 1
1.4 Responsible task 1
2 Database Analysis 2
2.1 User Requirements 2
2.2 ER Diagram 3
2.3 The Relational model 5
3 Database Implementation 5
3.1 DDL 6
3.2 Data Dictionaries (desc table) 7
3.3 DML (Insert, Update, Delete) 12
3.4 DQL 13
3.5 Views 14
3.6 Store Procedures and Functions 16
3.7 Trigger 20
4 Conclusion 21
4.1 Outcome and completed 21
4.2 Strong and Weak point 21
4.3 Challenges 22
4.4 Future work 22
4.5Conclusion 22
References 23

1. Introduction

1.1 Project description

This project is designed Database for **Academic Skill Development Center** to support the efficient management of school operations by organizing and storing essential data. It ensures seamless handling of student and teacher information, course enrollments, attendance tracking, and academic performance. With a structured and scalable approach, the database provides a reliable foundation for school administration, enabling smooth data access, security, and future expansion.

1.2 Objective

The objective of this project is to design and develop a robust school database system that efficiently handles student enrollment, teacher management, course administration, attendance tracking, and performance analytics.

1. **User Registration & Management**
    a. Maintain accurate records of students, teachers, and guardians, including personal details and location information.
2. **Course & Enrollment Management**
    a. Enable administrators to create and manage courses with levels, fees, and descriptions.
    b. Allow students to enroll in courses, track payment status, and assign them to classroom groups.
3. **Attendance & Performance Tracking**
    a. Implement attendance tracking for each class session with real-time updates on student participation.
    b. Store student performance data, including quiz scores and overall grades, for generating reports.
4. **Learning Management System Integration**
    a. Support quiz creation, student attempts, and automated grading.
    b. Provide detailed performance analytics for students and teachers.
5. **Data Integrity & Scalability**
    a. Design a relational database using MySQL to ensure efficient data storage and retrieval.
    b. Structure tables to optimize performance while supporting future implementation.

By achieving these objectives, the System will streamline administrative tasks, enhance student learning experiences, and provide valuable insights for academic decision-making.

1.3 Project Timeline

Week 1:

- Initial planning and requirement gathering
- Design the database schema (ER, Rational Medel)

Week 2:

- Develop core database tables
- Implement basic queries for data retrieval and manipulation
- Test the database structure with sample data

Week 3:

- Insert Dumi data into all tables
- Begin creating stored procedures, triggers, and views
- Conduct unit testing of the database functions

Week 4:

- Finalize the database schema and structure
- Complete data core academic operation
- Conduct system testing to ensure data consistency and integrity

1.4 Responsible task

- Sat Panha: Defined DQL (Data Query Language), created views, and implemented functions for efficient data retrieval and manipulation.
- Phy Vathanak: Defined project requirements, designed the ER diagram, developed the relational model, and implemented DDL (Data Definition Language) statements and procedures.
- Chhun Sivheng: Created triggers, implemented DML (Data Manipulation Language) operations, and prepared data dictionaries to ensure clear documentation of the data structure.

2. Database Analysis

2.1 User Requirements

The School System is designed to efficiently handle multiple aspects of academic and administrative processes. For user management, it is essential that the system allows the registration of both students and teachers/staff with comprehensive personal details. This includes basic information like first name, last name, date of birth, gender, contact information, national ID, and detailed location information (village, commune, district, and province). Teachers should also have their professional details, such as their major and employment status (active or inactive), tracked within the system. This ensures that the system can support full management of both student and staff data.

Course management must enable administrators to create and manage various courses. This includes inputting detailed information for each course, such as the short name (e.g., "MTH" for Mathematics), the full course name, the course level (e.g., "Beginner," "Intermediate," or "Advanced"), the course fee, and a description of what the course covers. The ability to manage courses flexibly is essential for accommodating different levels of study and ensuring proper fee structures are in place.

In enrollment and attendance management, the system should allow students to be enrolled in courses, tracking critical details like the student's ID, course instance ID (which links the student to a specific course and term), payment status, and enrollment date. The system should also handle student group management, ensuring that students are assigned to specific groups based on the course instance. Each group should be allocated to a classroom with a unique ID and capacity. The system should also track attendance for every session, capturing details such as whether a student was present, late, absent, or given permission to be absent. Attendance records must be linked to both the student and the specific course instance.

The learning management system functionality must allow teachers to create and manage quizzes for their courses, including the title, description, due date, and time limit for each quiz. The system should be capable of tracking student attempts on quizzes, including capturing the score, time remaining during each attempt, and the date of the attempt. This ensures that teachers can track and assess student progress through quizzes.

Finally, the reporting and analytics features should provide robust insights into student performance and attendance. The system must generate student performance reports that include data on overall grades, quiz scores, attendance, and performance trends over time. These reports should provide administrators and teachers with detailed insights into individual student achievements, helping them make informed decisions about student support and development. Additionally, the system should support attendance and behavior analytics, providing insights into trends such as the total number of sessions attended, absences, and late arrivals. The system should also offer the ability to track behavioral patterns like participation levels, which can be useful for identifying students who may need additional support or encouragement.

The overall goal of the system is to ensure efficient and accurate management of student and staff data, seamless course management, detailed tracking of enrollment and attendance, comprehensive learning assessment tools, and powerful reporting features to monitor and improve the learning experience.

2.2 ER Diagram

Detail diagram

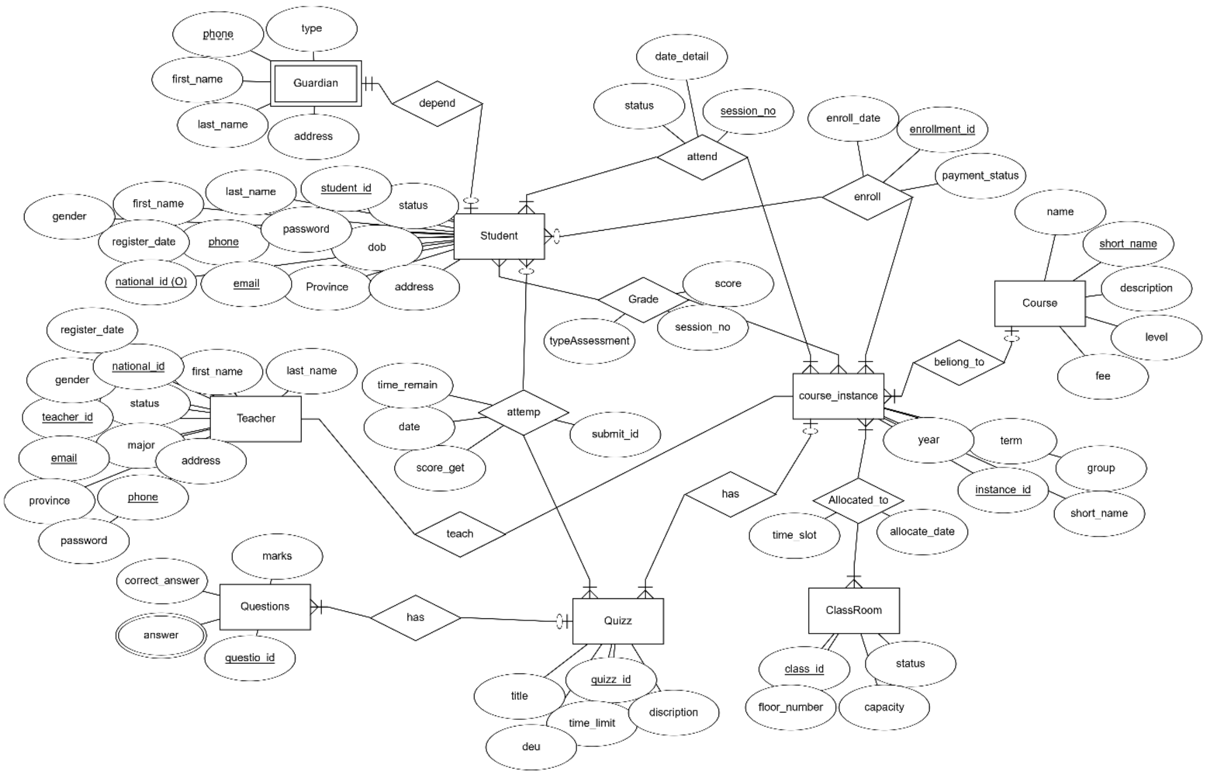
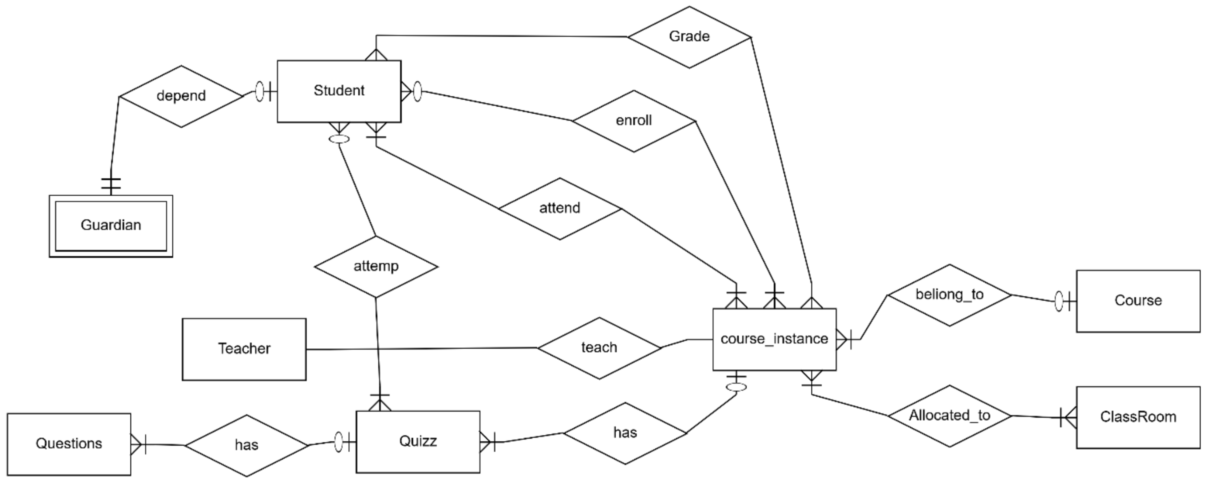


Diagram (focus on relationship)



Relationships:

Student → Enrollment:

- A student can enroll in multiple courses, but each enrollment is associated with one specific student. The relationship is one-to-many.

Student → Student_Attendance:

- A student can have multiple attendance records, but each attendance is for a specific student. The relationship is one-to-many.
- Foreign Key: student_id references Student(student_id).

Student → Grade:

- A student can have multiple grades, but each grade corresponds to a specific student. The relationship is one-to-many.
- Foreign Key: student_id references Student(student_id).

Enrollment → Course_instance:

- An enrollment corresponds to a single course instance, but a course instance can have multiple enrollments. The relationship is one-to-many.
- Foreign Key: instance_id references Course_instance(course_instance_id).

Grade → Course_instance:

- A grade is linked to one specific course instance, but each course instance can have multiple grades. The relationship is one-to-many.
- Foreign Key: course_instance_id references Course_instance(course_instance_id).

Course_instance → Quiz:

- A course instance can have multiple quizzes, but each quiz is linked to one specific course instance. The relationship is one-to-many.
- Foreign Key: instance_id references Course_instance(course_instance_id).

Quiz → QuizzAttempt:

- A quiz can have multiple attempts, but each attempt corresponds to one specific quiz. The relationship is one-to-many.
- Foreign Key: quizz_id references Quizz(quizz_id).

QuizAttempt → Student:

- A student can attempt multiple quizzes, but each quiz attempt is linked to one student. The relationship is one-to-many.
- Foreign Key: student_id references Student(student_id).

Student → Guardian:

- A student can have one guardian, but each guardian is associated with only one student. The relationship is one-to-one.
- Foreign Key: student_id references Student(student_id).

Teacher → Course_instance:

- A teacher can teach multiple course instances, but each course instance is taught by one teacher. The relationship is one-to-many.
- Foreign Key: teacher_id references Teacher(teacher_id).

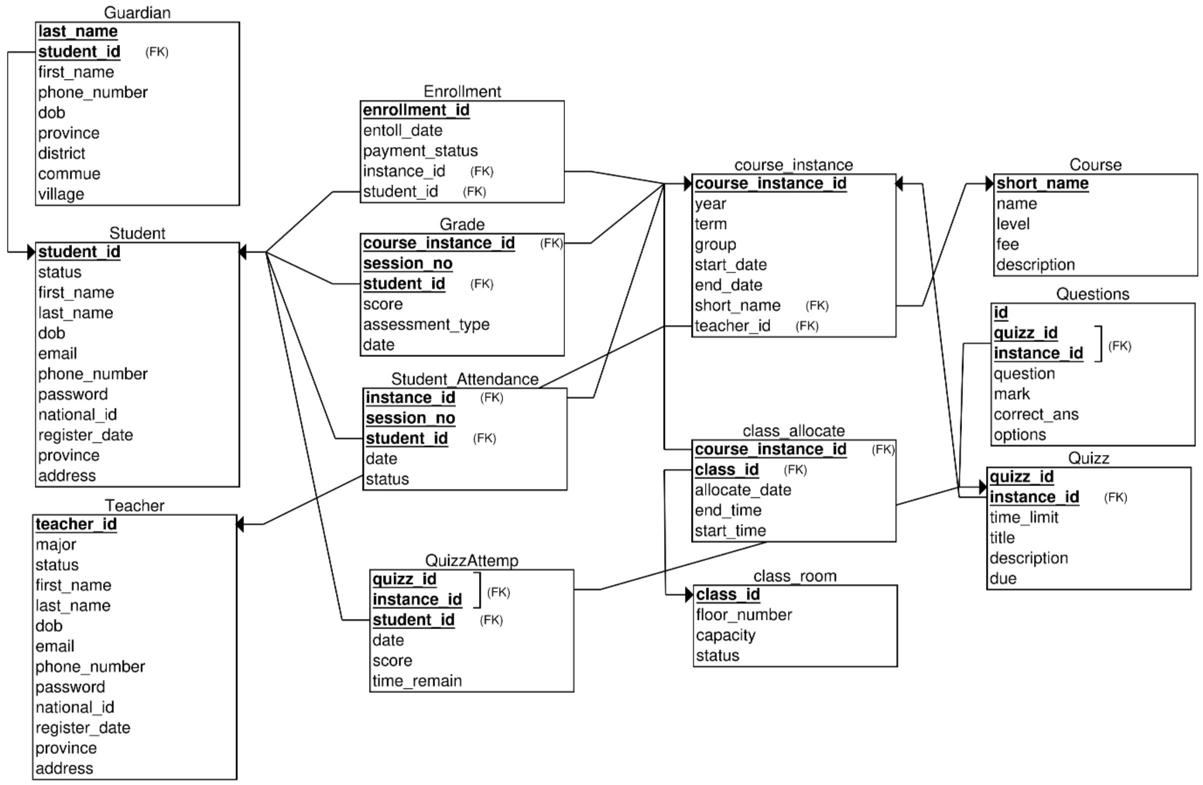
Quiz → Questions:

- A quiz can have multiple questions, but each question is linked to one specific quiz. The relationship is one-to-many.
- Foreign Key: quizz_id references Quizz(quizz_id).

Course → Course_instance:

- A course can have multiple instances, but each course instance is linked to one course. The relationship is one-to-many.
- Foreign Key: short_name references Course(short_name).

2.3 Relational Model



3. Database Implementation

3.1 DDL (Data Definition Language)

The DDL (Data Definition Language) is responsible for defining and managing the structure of the database. This includes creating tables, specifying constraints, defining relationships between entities, and setting default values. The schema for the School Management System was carefully designed to ensure data integrity, scalability, and normalized structure.

Database Initialization
After we analysis user requirements and draw the ER diagram and perform normalization we can create this table for the school database system.

Students Table
Stores student information including:

- Personal details (name, DOB, gender)
- Contact info (email, phone)
- Identity (student ID, national ID)
- Credentials (password)
- Registration details (date, province, address, status)

Teacher Table
Stores teacher information:

- Similar to the Students table with additional field major to indicate their teaching specialty.

Guardians Table
Links each student to a guardian:

- Each guardian is tied to one student (student_id)
- Stores guardian contact and address info
- ON DELETE CASCADE: If a student is deleted, their guardian entry is also deleted

Course
Defines courses offered:

- short_name as unique ID (e.g., "ENG", "MAT")
- Includes level, fee, and description

Course instance
Represents a specific instance of a course for a particular year, term, and group:

- Linked to Course and assigned a Teacher
- Generates a unique course_instance_id automatically by year,term,group,and shortname of course
- Tracks start and end dates

Enrollment
Tracks student enrollment into Course_instance:

- Links Students and Course_instance
- Contains enrollment date and payment status

Quiz
Represents a quiz created for a course instance:

- Includes time limit, title, due date
- Linked to a Course_instance

Question

Holds questions for each quiz:

- Includes multiple-choice options and correct answer
- Linked to a Quiz

Student Attendance

Tracks student attendance per course session:

- Includes session number and attendance status (present, late, etc.)
- Linked to Students and Course_instance

Grade

Stores scores of students for specific assessments in course sessions:

- Assessment types can be "QA", "assignment", etc.
- Linked to Students and Course_instance

Quiz Attempt

Records each attempt of a student on a quiz:

- Includes attempt date, score, and remaining time
- Linked to Students, Quiz, and Course_instance

Class room

Defines physical classrooms:

- Includes floor number, capacity, and status (available or not)

Class allocate

Schedules course instances into classrooms:

- Links Class_room and Course_instance
- Includes time range for the class (start and stop time)

The table that we created is to supports a complete academic system including:

- User management (students, teachers, guardians)
- Course offerings and scheduling
- Enrollment and grading
- Attendance tracking
- Classroom management
- Quiz system

3.2 Data Dictionaries

The data directory consists of multiple entities that define the structure of the e-learning platform. The **Location** table stores geographical details for users and guardians. The **User** table manages general user information, which branches into **Students**, **Teachers**, and **Guardians**, each with role-specific attributes. The **Course** and **Course Instance** tables define academic offerings, while **Enrollment** records student registrations. Assessment-related data is managed through **Quiz**, **Questions**, **Quiz Attempts**, and **Grades**. **Student Attendance** tracks presence in sessions. The **Classroom** and **Class Allocation** tables handle physical learning spaces. Each table follows strict validation rules to ensure data consistency and integrity across the platform.

Location					
Attribute Name	Data Type	Description	Constraints	Example Value	Validation
id	INT	Unique identifier for the location	Primary Key, AUTO_INCREMENT	1	Must be a unique identifier
village	VARCHAR(100)	Name of the village	Maximum length: 100 characters	Doun Meas	Length must not exceed 100 characters
commune	VARCHAR(100)	Name of the commune	Maximum length: 100 characters	Sre ja	Length must not exceed 100 characters
district	VARCHAR(100)	Name of the district	Maximum length: 100 characters	Snoul	Length must not exceed 100 characters
province	VARCHAR(100)	Name of the province	Maximum length: 100 characters	Kratie	Length must not exceed 100 characters
User					
Attribute Name	Data Type	Description	Constraints	Example Value	Validation
id	VARCHAR(10)	Unique user identifier	Primary Key; maximum length 10	U001	Must be unique and follow the format (U[Number])
first_name	VARCHAR(30)	User’s first name	NOT NULL; maximum length 30	Chea	Must not be null, length must not exceed 30 characters
last_name	VARCHAR(30)	User’s last name	NOT NULL; maximum length 30	Sopheaktra	Must not be null, length must not exceed 30 characters
dob	DATE	Date of birth	NOT NULL; must be a valid date	1/1/1990	Must follow a valid date format
District	VARCHAR	User's district name	NULLABLE	Sompumeas	Only letters and spaces, max 100 characters
province	VARCHAR	User's province name	NULLABLE	Pusart	Only letters and spaces, max 100 characters
location_id	INT	Reference to the user’s location	Foreign Key → Location(id); can be NULL	1	Foreign Key reference to Location
email	VARCHAR(50)	User email address	NOT NULL; UNIQUE; should follow valid email format	chea.sopheaktra@example.com	Must be a valid email format
phone_number	VARCHAR(15)	User phone number	NOT NULL; UNIQUE; valid phone format	12345678	Must follow valid phone number format
password	VARCHAR(255)	User password (hashed)	NOT NULL; stored securely (e.g., hashed)	hashedpassword123	Must be hashed for security
national_id	VARCHAR(15)	National ID number	Optional; maximum length 15	1.23457E+14	Maximum length 15 characters
Register_id	INT	Account registration date	NOT NULL, DEFAULT CURRENT_DATE	2/22/2020	Must be a valid date, cannot be in the future
Students					
Attribute Name	Data Type	Description	Constraints	Example Value	Validation
student_id	VARCHAR	student id numebr	FOREIGN KEY REFERENCES User(id),NOT NULL, PRIMARY KEY	S1 (Student)	Must follow the format S[Number]
status	BOOLEAN	status of student is active ot not	Defaults to false; must be a boolean	FALSE	Must be either true or false
Teachers					
Attribute Name	Data Type	Description	Constraints	Example Value	Validation

teacher_id	VARCHAR	teacher id numebr	FOREIGN KEY REFERENCES User(id),NOT NULL, PRIMARY KEY	T1 (Teacher)	Must follow the format T[Number]
status	BOOLEAN	status of teacher is active ot not	Defaults to false; must be a boolean	FALSE	Must be either true or false

Guardian

Attribute Name	Data Type	Description	Constraints	Example Value	Validation
stu_id	VARCHAR(10)	Student identifier for which the guardian is assigned	Primary Key; Foreign Key → Students(user_id)	S001	Foreign Key reference to Students and format (S[Num])
last_name	VARCHAR(30)	Guardian’s last name	Maximum length 30; optional	Aao	Length must not exceed 30 characters
first_name	VARCHAR(40)	Guardian’s first name	Maximum length 40; optional	Sokunkanha	Length must not exceed 40 characters
phone_number	VARCHAR(15)	Guardian’s phone number	NOT NULL; UNIQUE; must be valid	987654321	Must follow valid phone number format
dob	DATE	Guardian’s date of birth	Must be a valid date; optional	5/15/1970	Must follow a valid date format
location_id	INT	Reference to guardian’s location	Foreign Key → Location(id)	1	Foreign Key reference to Location

Course

Attribute Name	Data Type	Description	Constraints	Example Value	Validation
short_name	VARCHAR(5)	Short identifier for the course	Primary Key; maximum length 5	GDS	Length must not exceed 5 characters
name	VARCHAR(50)	Full name of the course	NOT NULL; maximum length 50	Graphic Design	Length must not exceed 50 characters
level	VARCHAR(20)	Course level (e.g., Beginner, Intermediate)	NOT NULL; maximum length 20	Beginner	Length must not exceed 20 characters
fee	INT	Fee for the course	NOT NULL; must be a non-negative integer	625	Must be a non-negative integer
description	VARCHAR(255)	Brief description of the course	Maximum length 255; optional	Introduction to Drawing on DIgitals	Length must not exceed 255 characters

Course_instance

Attribute Name	Data Type	Description	Constraints / Validation	Example Value	Validation
year	INT	Academic year of the course instance	NOT NULL	2025	Must be a valid academic year
term	INT	Academic term (e.g., 1, 2)	NOT NULL	1	Must be a valid term (1 or 2)
group_s	VARCHAR(10)	Group identifier for the instance	NOT NULL; maximum length 10	G1	Length must not exceed 10 characters and follow the format (G[Num])
short_name	VARCHAR(5)	Identifier for the course (links to Course table)	NOT NULL; Foreign Key → Course(short_name)	GDS	Foreign Key reference to Course
course_instance_id	VARCHAR(20)	Unique course instance identifier generated as a concatenation	Generated ALWAYS AS (CONCAT(year, '-', term, '-', group_s, '-', short_name)); UNIQUE; stored value	T1-Y1-GDS	Must follow the generated format (Term1-Year1-Graphic Design)
teacher_id	VARCHAR(5)	Identifier of the teacher assigned to the instance	Foreign Key → Teachers(user_id); optional; maximum length 5	T001	Foreign Key reference to Teachers and follw format (T[num])
start_date	DATE	Start date for the course instance	Defaults to CURRENT_DATE	9/1/2023	Must follow a valid date format

end_date	DATE	End date for the course instance	Must be a valid date; optional	12/15/2023	Must follow a valid date format
Enrollment					
Attribute Name	Data Type	Description	Constraints / Validation	Example Value	Validation
enrollment_id	INT	Unique enrollment record identifier	Primary Key, AUTO_INCREMENT	1	Must be unique
enroll_date	DATE	Date of enrollment	Defaults to CURRENT_DATE	9/1/2023	Must follow a valid date format
student_id	VARCHAR(10)	Identifier of the enrolled student	Foreign Key → Students(user_id); optional	S001	Foreign Key reference to Students
payment_status	BOOLEAN	Payment status for the enrollment	Boolean value (true/false)	FALSE	Must be either true or false
course_instance_id	VARCHAR(20)	Associated course instance	NOT NULL; Foreign Key → Course_instance(course_instance_id)	T1-Y1-GDS	Foreign Key reference to Course_instance
Quizz					
Attribute Name	Data Type	Description	Constraints / Validation	Example Value	Validation
quizz_id	INT	Unique identifier for the quiz	Primary Key	1	Must be a unique identifier and format
time_limit	TIME	Time limit allowed to complete the quiz	Must be a valid time (format HH:MM:SS)	0:30:00	Must follow the time format (HH:MM:SS)
title	VARCHAR(25)	Title of the quiz	NOT NULL; maximum length 25	Quiz 1	Length must not exceed 25 characters
description	VARCHAR(70)	Brief description of the quiz	NOT NULL; maximum length 70	design, drawing two Apples.	Length must not exceed 70 characters
instance_id	VARCHAR(20)	Course instance to which the quiz belongs	NOT NULL; Foreign Key → Course_instance(course_instance_id)	T1-Y1-GDS	Foreign Key reference to Course_instance
due	DATE	Due date for taking the quiz	Must be a valid date; optional	10/15/2023	Must follow a valid date format
Question					
Attribute Name	Data Type	Description	Constraints / Validation	Example Value	Validation
id	INT	Question identifier within a quiz	Part of Composite Primary Key with quizz_id	1	Must be unique within the quiz
quizz_id	INT	Associated quiz identifier	Part of Composite Primary Key; Foreign Key → Quizz(quizz_id)	101	Foreign Key reference to Quizz
question	VARCHAR(70)	The question text	Maximum length 70	What is 2+2?	Length must not exceed 70 characters
mark	INT	Points awarded for the question	Must be an integer	5	Must be a positive integer
correct_ans	VARCHAR(255)	Correct answer text	Maximum length 70	4	Length must not exceed 70 characters
Option_a	Varchar(255)	Answer a for question	Maximum length 255	1	Length must not exceed 255 characters
Option_b	Varchar(255)	Answer b for question	Maximum length 255	2	Length must not exceed 255 characters
Option_c	Varchar(255)	Answer c for question	Maximum length 255	3	Length must not exceed 255 characters
Option_d	Varchar(255)	Answer d for question	Maximum length 255	4	Length must not exceed 255 characters
StudentAttendance					
Attribute Name	Data Type	Description	Constraints / Validation	Example Value	Validation

instance_id	VARCHAR(20)	Associated course instance	NOT NULL; Foreign Key → Course_instance(course_instance_id)	T1-Y1-GDS	Foreign Key reference to Course_instance
stu_id	VARCHAR(10)	Student identifier	Foreign Key → Students(user_id)	S001	Foreign Key reference to Students
session_no	INT	Session number for the attendance record	Part of Composite Primary Key	1/1/1900	Must be a positive integer
status	ENUM('present', 'late', 'absent', 'permission')	Attendance status (e.g., present, late)	Must be one of the specified ENUM values	present	Must be one of the values: present, late, absent, permission
Grade					
Attribute Name	Data Type	Description	Constraints / Validation	Example Value	Validation
instance_id	VARCHAR(20)	Associated course instance	NOT NULL; Foreign Key → Course_instance(course_instance_id)	2023-1-A-MATH	Foreign Key reference to Course_instance
stu_id	VARCHAR(10)	Student identifier	Foreign Key → Students(user_id)	S001	Foreign Key reference to Students
score	FLOAT	Score achieved by the student	Must be a valid floating-point number	85.5	Must be a floating-point number between 0 and 100
assessment_type	VARCHAR(200)	Type of assessment (e.g., exam, quiz, assignment)	Maximum length 200	quizz	Length must not exceed 200 characters
session_no	INT	Session number for the grade record	Part of Composite Primary Key	1	Must be a positive integer
QuizzAttempt					
Attribute Name	Data Type	Description	Constraints / Validation	Example Value	Validation
attempt_id	INT	Unique identifier for the quiz attempt	Primary Key, AUTO_INCREMENT	1	Must be unique
quizz_id	INT	Associated quiz identifier	Foreign Key → Quizz(quizz_id)	101	Foreign Key reference to Quizz
student_id	VARCHAR(10)	Student identifier	Foreign Key → Students(user_id)	S001	Foreign Key reference to Students
time_taken	TIME	Time taken to complete the quiz	Must be a valid time (format HH:MM:SS)	0:20:00	Must follow the time format (HH:MM:SS)
score	FLOAT	Score obtained on the quiz	Must be a floating-point number between 0 and 100	85.5	Must be between 0 and 100
Class_room					
Attribute Name	Data Type	Description	Constraints / Validation	Example Value	Validation
class_id	VARCHAR(10)	Unique classroom identifier	Primary Key	A001	Must be unique and follow the format CR[Number]
capacity	INT	Maximum number of students the classroom can accommodate	Must be a positive integer	30	Must be a positive integer
floor_number	INT	Reference to classroom's location	Foreign Key → Location(id)	1	Foreign Key reference to Location
Class_allocate					
Attribute Name	Data Type	Description	Constraints / Validation	Example Value	Validation
class_id	VARCHAR (10)	Classroom identifier	Foreign Key → Class_room(class_room_id)	A204	Foreign Key reference to Class_room

instance_id	VARCHAR (20)	Associated course instance identifier	Foreign Key → Course_instance(course_instance_id)	2023-1-1-MATH	Foreign Key reference to Course_instance
allocation_date	DATE	Date of allocation	Defaults to CURRENT_DATE	4/1/2025	Must follow a valid date format (YYYY-MM-DD)
start_time	TIME	Class start time	NOT NULL	8:00:00	Valid time format
end_time	TIME	Class end time	OT NULL	10:00:00	Valid time format

3.3 DML (Data Manipulation Language)

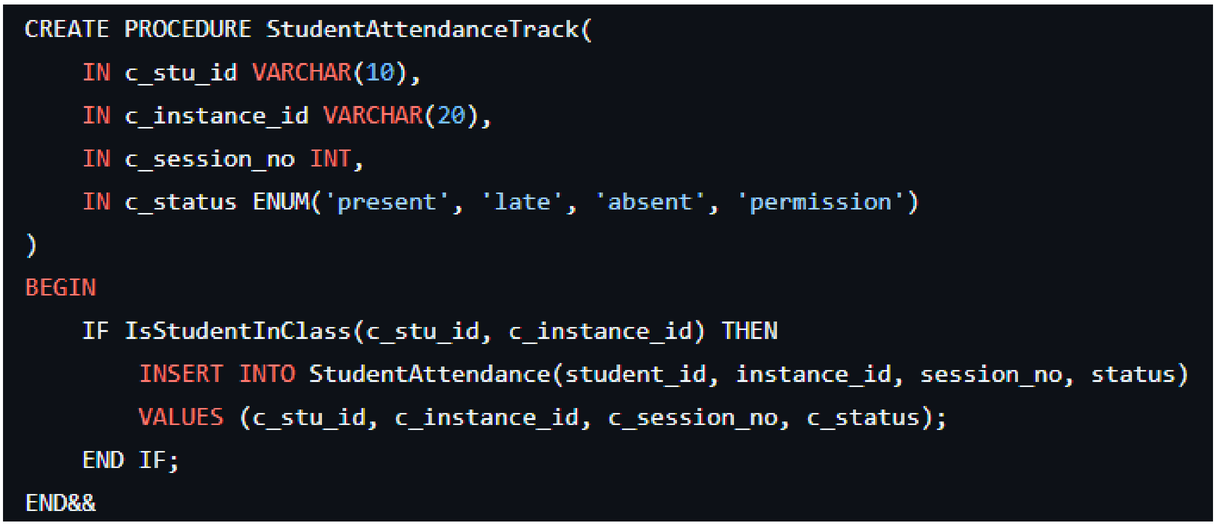
In our analysis of the academic operations of the system, we identified several critical Data Manipulation Language (DML) operations that are essential for managing and modifying the data within the database. DML is a subset of SQL (Structured Query Language) focused on manipulating and maintaining the data stored in the database. This includes operations such as INSERT, UPDATE, and DELETE, which are used to add, modify, and remove data as required by the academic system.

The INSERT statement allows for the addition of new records into various tables such as students, teachers, and courses, while the UPDATE statement helps modify existing records, such as updating student contact details or course information. The DELETE statement is used to remove unnecessary or outdated records from the database, ensuring that the data remains clean and up-to-date.

These DML operations are crucial for maintaining the integrity of academic data and ensuring that the system remains accurate, current, and effective. By employing these operations, administrators can manage student registrations, update course schedules, remove inactive records, and ensure the proper functioning of the academic system.

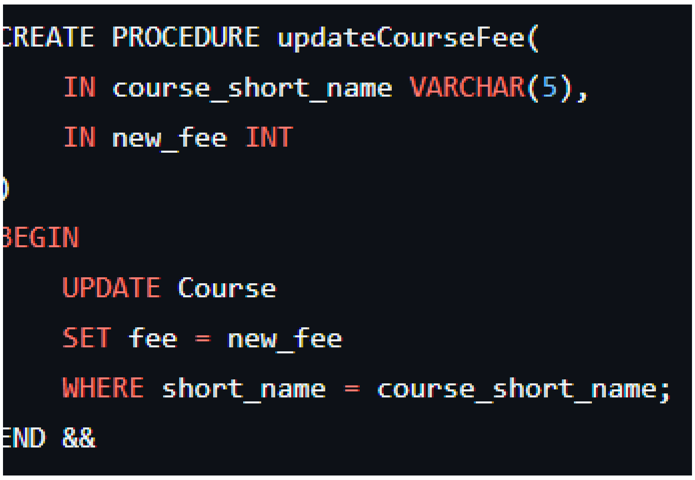
The following section outlines the specific DML operations used in our system, with examples provided for each type of operation and for the detail is in our git hub repository . and here is the basic of our DML

Implement Insert with Procedure



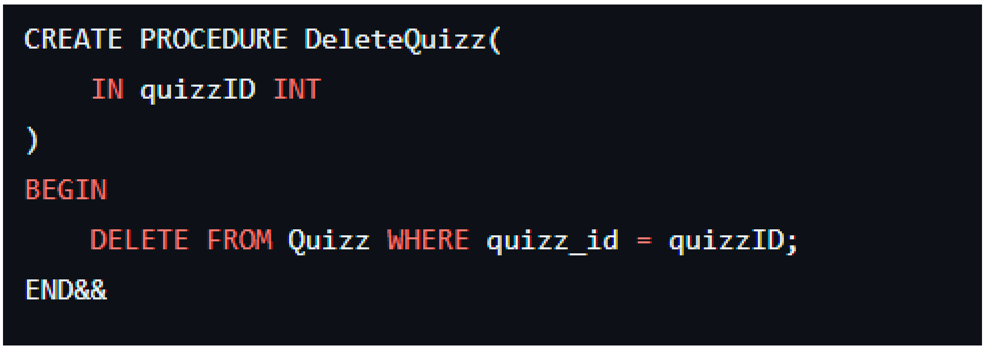
```
CREATE PROCEDURE StudentAttendanceTrack(
    IN c_stu_id VARCHAR(10),
    IN c_instance_id VARCHAR(20),
    IN c_session_no INT,
    IN c_status ENUM('present', 'late', 'absent', 'permission')
)
BEGIN
    IF IsStudentInClass(c_stu_id, c_instance_id) THEN
        INSERT INTO StudentAttendance(student_id, instance_id, session_no, status)
        VALUES (c_stu_id, c_instance_id, c_session_no, c_status);
    END IF;
END&&
```

Implement Update With procedure



```
CREATE PROCEDURE updateCourseFee(
    IN course_short_name VARCHAR(5),
    IN new_fee INT
)
BEGIN
    UPDATE Course
    SET fee = new_fee
    WHERE short_name = course_short_name;
END &&
```

Implement Delete with Procedure



```
CREATE PROCEDURE DeleteQuizz(
    IN quizzID INT
)
BEGIN
    DELETE FROM Quizz WHERE quizz_id = quizzID;
END&&
```

3.4 DQL (Data Query Language)

In this analysis, we have examined the academic operations within our system and identified several essential retrievals queries necessary for efficient data access and reporting. These queries serve various purposes, from retrieving basic student and teacher information to more complex data, such as tracking enrollments, attendance, and financial transactions. Below is a comprehensive list of the most critical retrieval statements for our system's operations and the detail about script are in our git hub repository in schoolDQL.sql :

1. Retrieve all columns from all tables in the database.
2. Get the name, phone number, and address from the Guardian table.
3. Get the ID, name, date of birth (DOB), phone number, and address from the student table.
4. Get the ID, name, date of birth (DOB), phone number, address, and major from the Teacher table.
5. Retrieve the courses that students have enrolled in.
6. Retrieve the courses that teachers have taught.
7. Get all students who have studied in a class (join Enrollment and Course Instance tables).
8. Retrieve all courses offered in a specific term and year.
9. Get the student’s name, ID, guardian’s name, and phone number.
10. Get the number of students in each class.
11. Retrieve students with unpaid enrollment fees.
12. Get questions from quizzes in a specific course instance.
13. Retrieve scores when students attempt quizzes.

14. Get attendance records for a specific student in a class.
15. Retrieve the student's report card, including grade letters and average scores.
16. Get the classes (course instances) associated with a course.
17. Retrieve class ID, course instance ID, start time, and end time from class allocation.
18. Get the number of students enrolled in a specific term and year.
19. Get the number of students enrolled in a specific course.
20. Retrieve all quizzes in a course instance.
21. Retrieve the total amount of money earned from students.
22. Retrieve the total amount of money earned in each year and term.
23. Retrieve the total amount of money earned from each course for a specific term and year.
24. Get a list of students based on gender (male or female).
25. Retrieve the number of students per province.
26. Get the top 5 youngest students.
27. Get the most common province among students.
28. Find students who have not enrolled in any course.
29. Retrieve students who have enrolled in more than 3 courses.
30. Find students who have failed.
31. Retrieve students with at least 90% attendance.
32. Retrieve the top 10 students based on their average grade.
33. Find students with the highest total marks in each course.
34. Generate an academic report for students in each course, including attendance percentage, grade (A-F), average score, and rank.

3.5 Views

A **view** is a virtual table created by a `SELECT` query, allowing us to simplify complex joins and improve data readability. In this e-learning system, a view is used to consolidate and present student-related information efficiently. In our school system view is very useful and here our detail about view that we have implemented in the project.

1. AvgGrades
Purpose:
This view calculates the average score and total score for each student in each course instance. It aggregates the grades of students from the Grade table.
Advantage:
The view simplifies the retrieval of average and total scores, making it easy to track and report on student performance over multiple sessions.
2. AttendanceScores
Purpose:
This view calculates the total attendance score for each student in each course instance. It uses the getAttendanceScore function to assign points based on the student's attendance status (present, late, absent, or permission).
Advantage:
It centralizes attendance scoring, ensuring consistency in how attendance is tracked and reported across the system. This makes it easier to query total attendance scores for any student.

3. StudentPerformance

Purpose:

This view combines grade data and attendance data, calculating each student's overall performance. The score is a weighted sum of the grade score and attendance percentage.

Advantage:

The view consolidates multiple data points (grade scores and attendance) into a single performance metric, allowing for easy access to a student's overall performance. It is useful for generating reports on student performance, factoring in both grades and attendance.

4. StudentReport

Purpose:

This view generates a detailed student report that includes the student's final score (a weighted average of their grades and attendance) and their letter grade. It also ranks students by performance within each course instance.

Advantage:

It simplifies the generation of student reports by calculating final scores and grades in one query, ensuring consistency in how student performance is evaluated and ranked.

5. listStudentInClass

Purpose:

This view retrieves a list of students enrolled in each class, providing information such as student names and course names.

Advantage:

It allows easy querying of all students enrolled in a specific class or course instance. This can be helpful for tracking attendance, managing class schedules, and monitoring student participation.

6. student_guardians

Purpose:

This view links each student with their guardian(s), showing the student's and guardian's names and guardian contact information.

Advantage:

It simplifies the process of retrieving information about students and their guardians. It can be useful for communication purposes, such as sending notifications or updates to guardians.

7. students_not_enrolled

Purpose:

This view identifies students who are not currently enrolled in any course by performing a LEFT JOIN between the Students and Enrollment tables.

Advantage:

It helps quickly identify students who need to be enrolled in courses, ensuring no student is left out of the registration process.

8. revenue_per_term

Purpose:

This view calculates the total revenue earned for each academic term, based on course fees and payment status. It joins the Enrollment, Course_instance, and Course tables to compute this value.

Advantage:

It provides a quick way to analyze the total revenue generated from course enrollments per term. This can help administrators manage budgets and financial reports more efficiently.

9. quiz_with_questions
Purpose:
This view combines quiz and question details, showing all the questions related to each quiz in a specific course instance. It joins the Quizz, Question, and Course_instance tables.
Advantage:
It simplifies the management and retrieval of quiz and question data. This view makes it easier to track which questions belong to which quiz in a given course instance.

3.6 Store Procedures and Functions

Store Procedures

The academic operation system uses several stored procedures to manage and manipulate data in an efficient and structured way. These stored procedures allow us to streamline various actions such as inserting, updating, deleting, and retrieving data, while ensuring the integrity and efficiency of the system. Below, we explain the purpose and advantages of each stored procedure used in the system.

Using INSERT procedures in the system brings several benefits, including ensuring data integrity, centralizing logic, reducing complexity, improving security, optimizing performance, and minimizing human error. They also enable better transaction control, ensuring consistent and accurate data insertion across related tables. This approach streamlines data management and enhances overall system reliability.
Operations Implemented in INSERT Procedures:

1. **Insert Teacher**: Adds a new teacher record to the Teachers table.
2. **Insert Guardian**: Adds guardian details linked to a student in the Guardians table.
3. **Register Student**: Registers both a student and their guardian in the respective tables with a single call.
4. **Insert Course**: Adds a new course to the Course table.
5. **Insert Course Instance**: Adds details about a course offering, including the course year, term, and assigned teacher.
6. **Insert Enrollment**: Registers a student in a specific course instance, including their payment status.
7. **Insert Quizz**: Adds a new quiz linked to a specific course instance.
8. **Insert Question**: Inserts questions related to a quiz.
9. **Insert Grade**: Records grades for students in a specific course instance and session.
10. **Insert Quizz Attempt**: Tracks a student's attempt at a quiz, including their score and time taken.
11. **Insert Class Allocation**: Allocates class timings for a specific course instance.

Implementing DELETE procedures in the system offers several advantages, such as improving data integrity, simplifying the deletion process, and reducing the risk of errors. By centralizing deletion logic, it ensures that related data across multiple tables is consistently removed. Additionally, using stored procedures for deletion improves security by restricting direct access to the database and enhances performance through optimized queries. This approach also makes it easier to maintain and update the system by encapsulating deletion logic within reusable procedures.

Delete Teacher:
Data Required: Teacher ID
Explanation: The Teacher ID is used to identify the specific teacher record to be deleted. It ensures that the deletion is targeted at the correct teacher, especially when there are multiple teachers in the system.

Delete Guardian:
Data Required: Guardian ID or Student ID
Explanation: The Guardian ID is used to delete the specific guardian record, or alternatively, the Student ID can be used to identify the guardian linked to that student. Ensuring the correct relationship between students and guardians is maintained during deletion.

Delete Student:
Data Required: Student ID
Explanation: The Student ID is necessary to identify the specific student whose record needs to be deleted. Deleting a student typically involves removing records from the Students table, along with any associated data (such as enrollments, attendance, and grades).

Delete Course:
Data Required: Course Short name
Explanation: The Short name is used to identify the specific course to be deleted. The deletion will cascade to any related course instances, enrollments, quizzes, and grades tied to that course.

Delete Course Instance:
Data Required: Course Instance ID (combination of year term group short_name (course unique id))
Explanation: The Course Instance ID is needed to identify which specific instance of the should be deleted. This operation also affects related data such as class allocation and enrollments, enrollments, quizzes, and grades and attendance.

Delete Enrollment:
Data Required: Student ID and Course Instance ID
Explanation: Both the Student ID and the Course Instance ID are required to remove the student's enrollment in a specific course instance. This ensures that the student's enrollment record is properly deleted from the system.

Delete Quizz:
Data Required: Quiz ID
Explanation: The Quiz ID is necessary to identify the specific quiz to be deleted. Deleting a quiz will also remove all related data such as questions and student attempts for that quiz.

Delete Question:
Data Required: Question ID and Quiz ID
Explanation: The Question ID is required to delete a specific question from a quiz. The Quiz ID ensures that the deletion is tied to the correct quizzes, and only the intended question is removed.

Delete Grade:
Data Required: Student ID, Course Instance ID, and Grade ID and session number
Explanation: Those above are needed to identify and delete a specific grade record for a student in a particular course instance.

Delete Attendance:
Data Required: Student ID, Course Instance ID, and Grade ID and session number
Explanation: Those above are needed to identify and delete a specific attendance record for a student in a particular course instance.
Delete Quizz Attempt:
Data Required: and Attempt ID
Explanation: Attempt ID are required to delete a specific quiz attempt made by a student. This helps ensure that only the relevant attempt record is deleted, preserving the integrity of other student attempts.
Delete Class Allocation:
Data Required: Class Allocation ID or Course Instance ID
Explanation: The Class Allocation ID or Course Instance ID is needed to remove class schedule data tied to a specific course instance. This helps maintain accurate scheduling information in the system.

In system where data frequently changes or needs to be modified, it is crucial to have efficient and accurate methods for updating records. The UPDATE procedures implemented in our system are designed to handle these changes in a structured and reliable manner. These procedures ensure that the system's data remains consistent and accurate, reducing the risk of errors and ensuring that updates are applied without disrupting the overall database integrity. By encapsulating the update logic within stored procedures, we make the process more maintainable, prevent direct table manipulation, and ensure that updates are performed in a standardized manner. This is the implementation of update procedure in our system

Update Payment Status:

- Purpose: This procedure is used to update the payment status for a specific student's enrollment.
- Data Required: eid (enrollment ID), pay_status (payment status, a boolean value).
- Explanation: It updates the payment status of a student in the Enrollment table by setting the payment_status field based on the provided boolean value. This is useful for marking a student's enrollment as paid or pending based on their payment status.

Update Grade:

- Purpose: This procedure updates the grade of a student for a particular course instance and session.
- Data Required: stu_id (student ID), instance (course instance), session (session number), new_score (the new grade score).
- Explanation: It updates the score of a student in the Grade table. This is useful when grades need to be corrected or updated after a re-evaluation or other changes.

Update Course Fee:

- Purpose: This procedure updates the fee for a particular course.
- Data Required: course_short_name (short name or code of the course), new_fee (the new fee amount).
- Explanation: This procedure modifies the fee of a course in the Course table. This is useful when course fees change, either due to adjustments in pricing or financial policy updates.

Update Course Instance End Date:

- Purpose: This procedure updates the end date of a particular course instance.
- Data Required: instance_id_in (course instance ID), new_end_date (the new end date for the course instance).
- Explanation: It modifies the end_date of a course instance in the Course_instance table. This update is important when a course's end date needs to be extended or shortened due to unforeseen circumstances.

Update Student Attendance:

- Purpose: This procedure updates the attendance status of a student for a specific class session.
- Data Required: stu_id (student ID), instance (course instance), section (session number), new_status (the updated attendance status, which could be 'present', 'late', 'absent', or 'permission').
- Explanation: It modifies the attendance status of a student in the StudentAttendance table, allowing attendance records to be updated whenever necessary.

Update Grade:

- Purpose: This procedure updates the grade of a student for a particular course instance and session.
- Data Required: stu_id (student ID), instance (course instance), session (session number), new_score (the new grade score).
- Explanation: It updates the score of a student in the Grade table. This is useful when grades need to be corrected or updated after a re-evaluation or other changes.

Store Function

functions are used to handle specific tasks and calculations. These functions help make the system work more efficiently by allowing us to reuse the same logic whenever needed. They help with tasks like checking if a student is enrolled in a class, calculating attendance scores, finding course details, and ensuring class schedules don't overlap. Using functions in the system makes the code cleaner, faster, and easier to maintain. Here's a breakdown of each function used in the system.

IsStudentInClass:

- Purpose: Checks if a student is enrolled in a specific course instance.
- Input: sid (student ID), instance_id (course instance ID).
- Output: Returns TRUE if the student is enrolled; FALSE otherwise.

getCourseInstanceID:

- Purpose: Retrieves the course instance ID based on year, term, group, and course short name.
- Input: syear (year), sterm (term), sgroup (group), shortNameCourse (course short name).
- Output: Returns the course instance ID for the given parameters.

getAttendanceScore:

- Purpose: Returns the attendance score based on the student's attendance status.
- Input: status (attendance status: 'present', 'late', 'absent', 'permission').
- Output: Returns a numerical score: 1.0 for present, 0.75 for late, 0.5 for permission, 0.0 for absent.

getCharGrade:

- Purpose: Converts an average score into a letter grade.
- Input: average (student's average score).
- Output: Returns a letter grade ('A', 'B', 'C', 'D', 'E', or 'F') based on the average score.

GetStudentCount:

- Purpose: Counts the number of students enrolled in a specific course instance.
- Input: instance_id (course instance ID).
- Output: Returns the number of students enrolled in the course instance.

findMaxGradeSection:

- Purpose: Finds the maximum session number in which students have grades for a specific course instance.
- Input: instance_id (course instance ID).
- Output: Returns the maximum session number with grades.

findMaxAttendanceSection:

- Purpose: Finds the maximum session number for attendance records in a specific course instance.
- Input: instance_id (course instance ID).
- Output: Returns the maximum session number with attendance data.

CheckClassAvailability:

- Purpose: Checks if a class is available at a specified start and stop time, ensuring no overlapping with existing classes.
- Input: classID (class ID), requested_start_time (start time), requested_stop_time (end time).
- Output: Returns TRUE if the class is available; FALSE otherwise.

3.7 Trigger

The DELETE operations in the school database management system are designed to ensure that any removal of records is logged for auditing purposes. For each entity (e.g., students, teachers, courses, enrollments), a trigger is created to log the details of the deleted record into a corresponding log table. These operations help in maintaining data integrity, tracking deletions, and providing an audit trail for any record removals. The key DELETE triggers implemented include:

- Student Deletion: Tracks when a student's record is deleted.
- Teacher Deletion: Logs deletions of teacher records.
- Course Deletion: Captures deletions of course records.
- Course Instance Deletion: Logs deletions of specific course instances.
- Enrollment Deletion: Tracks when a student's enrollment record is deleted.
- Quiz Deletion: Logs deletions of quizzes.
- Quiz Attempt Deletion: Captures deletions of quiz attempt records.
- Student Attendance Deletion: Logs deletions of student attendance records.
- Grade Deletion: Tracks when grade records are deleted.
- Class Allocation Deletion: Logs deletions of class allocation records.

Each of these DELETE operations includes a corresponding log table where the deleted record's key identifiers (e.g., student_id, teacher_id, course_id) are recorded along with a timestamp for when the deletion occurred.

The UPDATE operations in the school database system are designed to log any changes made to records. Whenever a record is modified (such as a student's grade or course enrollment), the previous state of the record is captured in an update log table. This is important for maintaining data integrity, tracking changes, and ensuring transparency. Each UPDATE operation is associated with a trigger that logs the before and after states of the updated records. The key UPDATE triggers include check for all column that have update so if student name is update it will insert into update log that have log id , record id, table that update, collum that update , old value and new value and also show the time stamp when the operation occurs.

For the delete operation, we create a new table to track deletions in our system. For example, the student table will have a corresponding student deletion table, which mirrors the structure of the student table but stores additional information, such as the time of deletion and the deletion log. We use this technique to prevent data loss. If an operation occurs unexpectedly, we can restore the deleted data.

For the update operation, we create only one table to track all records that are updated. This is because most applications allow users to update only one piece of data at a time, so there is no need to create multiple tables. Since only one column is changed at a time, we can efficiently track all updates in a single log table.

4. Conclusion and Future work

4.1 Outcome

The school database system provides an organized, efficient structure for managing user registration, course enrollment, attendance, and performance tracking. It utilizes a relational database built on MySQL, ensuring data integrity, scalability, and efficient querying. The system supports key features like maintaining accurate student, teacher, and guardian records, managing course details, tracking attendance and academic performance. It is designed to handle future growth and expansions in data storage needs.

4.2 Strong & Weak point

Strong Points:

- Data Structure Design: The database schema is well-structured with normalized tables, ensuring efficient data management and retrieval.
- Scalability: The design anticipates future expansion, supporting large numbers of users, courses, and records without compromising performance.
- Data Integrity: Proper constraints, such as primary and foreign keys, are in place to ensure data consistency and integrity.

Weak Points:

- Complexity of Queries: Some complex queries involving joins and aggregates might result in slower performance when dealing with large datasets, requiring further optimization.
- Limited Reporting: While performance data is stored, advanced reporting features, such as graphical analysis or customized reports, are not included at this stage.
- Limited validation**:** The system relies on basic database constraints (e.g., unique, not null) for validation, but there is a lack of more advanced business logic mechanisms (e.g., custom validation rules for specific fields or real-time input validation) that could help prevent incorrect or incomplete data from being entered.

4.3 Challenges

During the development school database system, several challenges were encountered in terms of performance, data consistency, and database design. These challenges required careful planning and optimization to ensure the system functioned efficiently and maintained data integrity across various modules. Below are the specific challenges faced:

Complex Relationships:

It was challenging to clearly define the relationships between various entities such as students, teachers, and courses. Determining whether to model certain relationships as entities or associations, and selecting appropriate primary keys for tables to ensure uniqueness and integrity, added complexity to the database design.

Data Redundancy Risks:

Ensuring that the database does not store duplicate records while maintaining data access efficiency was difficult. It required carefully balancing normalization to reduce redundancy, but without compromising the performance of queries, especially in large datasets.

Multi-Table Queries:

Writing complex SQL queries involving joins and aggregations across multiple tables became a challenge as the database grew. Handling large volumes of data while ensuring fast, accurate results from interconnected tables required advanced query optimization and indexing.

4.4 Future work

Looking ahead, several key improvements and extensions are planned to enhance the functionality and user experience of the system. These future developments aim to expand the database's capabilities and integrate it into more dynamic, user-friendly platforms. The following areas are targeted for future work:

- **Performance Optimization:** Implementing more advanced indexing strategies and query optimization techniques to handle larger datasets efficiently.
- **Need implementation :** add index in table for query optimized.
- **Reporting Features:** Expanding the database with features for generating advanced reports, such as attendance summaries and academic performance analytics.
- **Backup & Recovery:** Introducing a backup and recovery mechanism to ensure data safety and quick restoration in case of failure.
- **Implementation with Web Application:** Integrating this database with a web application to allow real-time interaction with the database, enabling functionalities like student registration, course enrollment, attendance tracking, and performance monitoring through a user-friendly interface.

4.5 Conclusion

The Academic Skill Development Center Database provides a foundation for managing school operations, ensuring efficient data handling, structured storage, and optimized performance. Despite challenges related to complex queries, and scalability, the system effectively supports most academic operations Future improvements will focus on enhancing performance, security, automation, and cloud integration, making the system more efficient and adaptable to evolving educational needs.

For detailed information about source code, please check the GitHub repository: https://github.com/CPF-CADT/Academic-skill-development-center-Database/.

Reference

https://www.w3schools.com/sql/

https://www.geeksforgeeks.org/window-functions-in-sql/