## Brief Introduction to the Problem

Managing academic data in a university manually or using unstructured spreadsheets is inefficient and prone to errors. Faculty members face challenges in tracking student attendance, recording marks, calculating eligibility for exams, and generating results. Additionally, managing user access for administrators, lecturers, technical officers, and students is cumbersome without a centralized system.

To address these challenges, we developed a database management system specifically for the Faculty of Technology at the University of Ruhuna. This system efficiently handles student details, attendance, marks, results, and eligibility verification, while providing secure access to different user roles. It ensures accurate record-keeping, reduces manual effort, and allows faculty and students to access necessary academic information quickly and reliably.

## Brief Introduction to the Solution

To overcome the challenges of manual academic management, we designed and implemented a centralized database management system for the Faculty of Technology at the University of Ruhuna. The system uses MySQL as the backend and incorporates tables, views, stored procedures, and triggers to automate key processes.

* **Views** provide simplified access to aggregated data, such as attendance summaries, final marks, grades, and eligibility reports, reducing the need for manual calculations.
* **Triggers** ensure data consistency and automatically update related tables when certain actions occur, such as marking attendance or updating exam results.
* **Stored Procedures** streamline repetitive tasks, like calculating final marks, checking eligibility for exams, and generating grade reports for students and batches.

By leveraging these database features, the system minimizes manual work, ensures data accuracy, and provides role-based access for administrators, lecturers, technical officers, and students. It allows faculty members to efficiently manage student information, attendance, assessments, and results, while students can securely view their own academic progress.

## Proposed ER/EER Diagram

### Entities and Attributes

| **Entity** | **Primary Key** | **Key Attributes** |
| --- | --- | --- |
| users | user\_id | name, email, password, role |
| student | user\_id | reg\_no, batch, status, department\_id |
| lecture | user\_id | specialization, designation |
| dean | lecture\_id | term\_start, term\_end |
| tech\_officer | user\_id | – |
| department | department\_id | name, faculty\_name |
| course | course\_id | name, credit, academic\_year, semester, total\_hours, weekly\_hours |
| session | session\_id | course\_id, session\_date, session\_hours, type |
| attendance | attendance\_id | student\_id, session\_id, status, medical, hours\_attended |
| marks | marks\_id | student\_id, course\_id, quiz1\_marks, quiz2\_marks, quiz3\_marks, assessment\_marks, mid\_marks, final\_theory, final\_practical, ca\_marks, final\_marks, ca\_eligible, final\_eligible, grade |
| medical | medical\_id | student\_id, course\_id, exam\_type, date\_submitted, status |
| result | result\_id | student\_id, academic\_year, semester, sgpa, cgpa, total\_credits |

### Relationships

* **1:1**: users → student/lecture/tech\_officer, lecture → dean
* **1:N**: student → attendance, student → marks, course → session, student → result, student → medical
* **M:N (junction tables)**:
  + student\_course (students enrolled in courses)
  + lecture\_course (lecturers teaching courses)
  + department\_course (courses offered by departments)
  + lecture\_department (lecturers assigned to departments)

**Points**: - Attendance and marks are linked to students and courses, supporting medical handling. - Result tracks SGPA/CGPA per student per semester. - Constraints enforce referential integrity and data consistency.

## Proposed Relational Mapping Diagram

| **Table** | **FK References** | **Relationship Type** |
| --- | --- | --- |
| student | users(user\_id), department(department\_id) | One-to-One, Many-to-One |
| lecture | users(user\_id) | One-to-One |
| dean | lecture(user\_id) | One-to-One |
| tech\_officer | users(user\_id) | One-to-One |
| session | course(course\_id) | Many-to-One |
| attendance | student(user\_id), session(session\_id) | Many-to-One |
| marks | student(user\_id), course(course\_id) | Many-to-One |
| medical | student(user\_id), course(course\_id) | Many-to-One |
| result | student(user\_id) | Many-to-One |
| student\_course | student(user\_id), course(course\_id) | Many-to-Many (junction) |
| lecture\_course | lecture(user\_id), course(course\_id) | Many-to-Many (junction) |
| department\_course | department(department\_id), course(course\_id) | Many-to-Many |
| lecture\_department | lecture(user\_id), department(department\_id) | Many-to-Many |

## Table Structure of the Solution

### Table: users

+----------+----------------------------------------------------------+------+-----+---------+-------+  
| Field | Type | Null | Key | Default | Extra |  
+----------+----------------------------------------------------------+------+-----+---------+-------+  
| user\_id | varchar(10) | NO | PRI | NULL | |  
| name | varchar(50) | NO | | NULL | |  
| email | varchar(50) | NO | UNI | NULL | |  
| password | varchar(100) | NO | | NULL | |  
| role | enum('Admin','Dean','Lecturer','Tech\_Officer','Student') | NO | | NULL | |  
+----------+----------------------------------------------------------+------+-----+---------+-------+

### Table: department

+---------------+--------------+------+-----+---------+-------+  
| Field | Type | Null | Key | Default | Extra |  
+---------------+--------------+------+-----+---------+-------+  
| department\_id | varchar(10) | NO | PRI | NULL | |  
| name | varchar(100) | NO | | NULL | |  
| faculty\_name | varchar(100) | YES | | NULL | |  
+---------------+--------------+------+-----+---------+-------+

### Table: student

+---------------+-------------------------------------+------+-----+---------+-------+  
| Field | Type | Null | Key | Default | Extra |  
+---------------+-------------------------------------+------+-----+---------+-------+  
| user\_id | varchar(10) | NO | PRI | NULL | |  
| reg\_no | varchar(15) | NO | UNI | NULL | |  
| batch | varchar(10) | YES | | NULL | |  
| status | enum('Proper','Repeat','Suspended') | YES | | Proper | |  
| department\_id | varchar(10) | YES | MUL | NULL | |  
+---------------+-------------------------------------+------+-----+---------+-------+

### Table: lecture

+---------------+-------------+------+-----+---------+-------+  
| Field | Type | Null | Key | Default | Extra |  
+---------------+-------------+------+-----+---------+-------+  
| user\_id | varchar(10) | NO | PRI | NULL | |  
| specialization| varchar(50) | YES | | NULL | |  
| designation | varchar(50) | YES | | NULL | |  
+---------------+-------------+------+-----+---------+-------+

### Table: dean

+------------+-------------+------+-----+---------+-------+  
| Field | Type | Null | Key | Default | Extra |  
+------------+-------------+------+-----+---------+-------+  
| lecture\_id | varchar(10) | NO | PRI | NULL | |  
| term\_start | date | NO | | NULL | |  
| term\_end | date | YES | | NULL | |  
+------------+-------------+------+-----+---------+-------+

### Table: tech\_officer

+---------+-------------+------+-----+---------+-------+  
| Field | Type | Null | Key | Default | Extra |  
+---------+-------------+------+-----+---------+-------+  
| user\_id | varchar(10) | NO | PRI | NULL | |  
+---------+-------------+------+-----+---------+-------+

### Table: course

+---------------+---------------+------+-----+---------+-------+  
| Field | Type | Null | Key | Default | Extra |  
+---------------+---------------+------+-----+---------+-------+  
| course\_id | varchar(10) | NO | PRI | NULL | |  
| name | varchar(100) | NO | | NULL | |  
| credit | int | NO | | NULL | |  
| academic\_year | int | YES | | NULL | |  
| semester | enum('1','2') | NO | | NULL | |  
| total\_hours | decimal(5,2) | YES | | 50.00 | |  
| weekly\_hours | decimal(4,2) | YES | | 3.00 | |  
+---------------+---------------+------+-----+---------+-------+

### Table: session

+---------------+----------------------------+------+-----+---------+----------------+  
| Field | Type | Null | Key | Default | Extra |  
+---------------+----------------------------+------+-----+---------+----------------+  
| session\_id | int | NO | PRI | NULL | auto\_increment |  
| course\_id | varchar(10) | NO | MUL | NULL | |  
| session\_date | date | NO | | NULL | |  
| session\_hours | decimal(4,2) | YES | | 3.00 | |  
| type | enum('Theory','Practical') | YES | | Theory | |  
+---------------+----------------------------+------+-----+---------+----------------+

### Table: attendance

+----------------+--------------------------+------+-----+---------+----------------+  
| Field | Type | Null | Key | Default | Extra |  
+----------------+--------------------------+------+-----+---------+----------------+  
| attendance\_id | int | NO | PRI | NULL | auto\_increment |  
| student\_id | varchar(10) | NO | MUL | NULL | |  
| session\_id | int | NO | MUL | NULL | |  
| status | enum('Present','Absent') | NO | | NULL | |  
| medical | boolean | YES | | FALSE | |  
| hours\_attended | decimal(4,2) | YES | | 0.00 | |  
+----------------+--------------------------+------+-----+---------+----------------+

### Table: marks

+------------------+------------------------------------------------+------+-----+--------------+----------------+  
| Field | Type | Null | Key | Default | Extra |  
+------------------+------------------------------------------------+------+-----+--------------+----------------+  
| marks\_id | int | NO | PRI | NULL | auto\_increment |  
| student\_id | varchar(10) | NO | MUL | NULL | |  
| course\_id | varchar(10) | NO | MUL | NULL | |  
| quiz1\_marks | decimal(5,2) | YES | | NULL | |  
| quiz2\_marks | decimal(5,2) | YES | | NULL | |  
| quiz3\_marks | decimal(5,2) | YES | | NULL | |  
| assessment\_marks | decimal(5,2) | YES | | NULL | |  
| mid\_marks | decimal(5,2) | YES | | NULL | |  
| final\_theory | decimal(5,2) | YES | | NULL | |  
| final\_practical | decimal(5,2) | YES | | NULL | |  
| ca\_marks | decimal(5,2) | YES | | NULL | |  
| final\_marks | decimal(5,2) | YES | | NULL | |  
| ca\_eligible | enum('Eligible','Not Eligible','MC','WH') | YES | | Not Eligible | |  
| final\_eligible | enum('Eligible','Not Eligible','MC','WH','E\*') | YES | | Not Eligible | |  
| grade | char(10) | YES | | NULL | |  
+------------------+------------------------------------------------+------+-----+--------------+----------------+

### Table: medical

+----------------+---------------------------------------+------+-----+---------+----------------+  
| Field | Type | Null | Key | Default | Extra |  
+----------------+---------------------------------------+------+-----+---------+----------------+  
| medical\_id | int | NO | PRI | NULL | auto\_increment |  
| student\_id | varchar(10) | NO | MUL | NULL | |  
| course\_id | varchar(10) | YES | MUL | NULL | |  
| exam\_type | enum('Mid','Final','Attendance') | NO | | NULL | |  
| date\_submitted | date | NO | | NULL | |  
| status | enum('Pending','Approved','Rejected') | YES | | Pending | |  
+----------------+---------------------------------------+------+-----+---------+----------------+

### Table: result

+---------------+---------------+------+-----+---------+----------------+  
| Field | Type | Null | Key | Default | Extra |  
+---------------+---------------+------+-----+---------+----------------+  
| result\_id | int | NO | PRI | NULL | auto\_increment |  
| student\_id | varchar(10) | NO | MUL | NULL | |  
| academic\_year | int | YES | | NULL | |  
| semester | enum('1','2') | NO | | NULL | |  
| sgpa | decimal(3,2) | YES | | NULL | |  
| cgpa | decimal(3,2) | YES | | NULL | |  
| total\_credits | int | YES | | 0 | |  
+---------------+---------------+------+-----+---------+----------------+

### Table: student\_course

+------------+-------------+------+-----+---------+-------+  
| Field | Type | Null | Key | Default | Extra |  
+------------+-------------+------+-----+---------+-------+  
| student\_id | varchar(10) | NO | PRI | NULL | |  
| course\_id | varchar(10) | NO | PRI | NULL | |  
+------------+-------------+------+-----+---------+-------+

### Table: lecture\_course

+------------+-------------+------+-----+---------+-------+  
| Field | Type | Null | Key | Default | Extra |  
+------------+-------------+------+-----+---------+-------+  
| lecture\_id | varchar(10) | NO | PRI | NULL | |  
| course\_id | varchar(10) | NO | PRI | NULL | |  
+------------+-------------+------+-----+---------+-------+

### Table: lecture\_department

+---------------+-------------+------+-----+---------+-------+  
| Field | Type | Null | Key | Default | Extra |  
+---------------+-------------+------+-----+---------+-------+  
| lecture\_id | varchar(10) | NO | PRI | NULL | |  
| department\_id | varchar(10) | NO | PRI | NULL | |  
+---------------+-------------+------+-----+---------+-------+

### Table: department\_course

+---------------+-------------+------+-----+---------+-------+  
| Field | Type | Null | Key | Default | Extra |  
+---------------+-------------+------+-----+---------+-------+  
| department\_id | varchar(10) | NO | PRI | NULL | |  
| course\_id | varchar(10) | NO | PRI | NULL | |  
+---------------+-------------+------+-----+---------+-------+

## Architecture of the Solution

### Three-Tier Architecture

1. **Presentation Layer (Frontend)**:
   * Web interface or CLI for users to enter/view data.
   * Role-based views: Student, Lecturer, Technical Officer, Dean, Admin.
2. **Application Layer (Business Logic)**:
   * **Triggers**: Update attendance, marks, and eligibility automatically.
   * **Stored Procedures**: Calculate CA marks, final marks, grades, GPA/CGPA.
   * **Views**: Summarize attendance, marks, and results for batch or individual student.
3. **Data Layer (Backend Database)**:
   * MySQL database with normalized tables (3NF).
   * Referential integrity via foreign keys and cascading updates/deletes.
   * Role-based access control:
     + **Admin**: Full privileges
     + **Dean**: Full without grant
     + **Lecturer**: CRUD on course/student/marks
     + **Technical Officer**: CRUD on attendance
     + **Student**: Read-only on results

### Automation Highlights

* Automatic eligibility calculation for attendance & marks
* Medical handling integrated for CA, mid, final exams
* Dynamic GPA/CGPA calculation per semester/year
* Views for easy reporting and analytics

## Tools and Technologies Used

1. **MySQL**
   * Relational database management system used as the backend.
   * Provides robust support for tables, relationships, constraints, triggers, stored procedures, and views.
   * Ensures data consistency, integrity, and normalization.
2. **Navicat for MySQL**
   * Assisted in database development, data modeling, query execution, and user management.
   * Used for importing/exporting data, backup, and running administrative tasks.
3. **XAMPP**
   * Provided a local server environment to run MySQL and test database operations.
   * Helped integrate and test database connectivity for future backend applications.
4. **Draw.io**
   * Used to create clear and professional ER/EER diagrams and relational mapping diagrams.
   * Allowed easy visualization of entities, relationships, and cardinality for the project report.
5. **ChatGPT**
   * Assisted in design guidance, query optimization, database design review.
   * Used as a reference tool to clarify DBMS concepts and implement efficient database solutions.

## Security Measures

### Security Measures Taken to Protect the Database

1. **Role-Based Access Control (RBAC)**:
   * Users have specific roles with tailored privileges:
     + **Admin**: Full privileges with GRANT option
     + **Dean**: Full privileges without GRANT
     + **Lecturer**: CRUD privileges on academic tables but cannot create users
     + **Technical Officer**: CRUD on attendance tables/views only
     + **Student**: Read-only access to final marks and attendance
   * Ensures users only access data relevant to their responsibilities.
2. **User Authentication**:
   * Each user has a unique username and password.
   * Ensures only authorized users can log in and access the database.
3. **Data Validation and Constraints**:
   * Use of NOT NULL, UNIQUE, ENUM, and CHECK constraints to prevent invalid entries.
   * Example: Marks are restricted to 0–100; student status is restricted to proper, repeat, or suspended.
4. **Foreign Key Constraints and Referential Integrity**:
   * Enforced relationships between tables using foreign keys with cascading updates/deletes.
   * Prevents orphaned records and maintains consistent data across the database.
5. **Stored Procedures and Triggers**:
   * Automate sensitive operations such as:
     + Calculating attendance percentages
     + Updating eligibility for exams
     + Calculating final marks and grades
   * Minimizes human error and prevents unauthorized manual changes.
6. **Medical & Eligibility Handling**:
   * Sensitive academic data like medical exemptions, CA eligibility, and final exam eligibility are controlled via tables and triggers.
   * Ensures accurate, secure handling of exceptions.
7. **Principle of Least Privilege**:
   * Users only have the minimum permissions needed to perform their tasks.

### Database Accounts / Users

**Definition:**  
A database account (or user) is an identity created in a database system that allows a person or application to connect to and interact with the database. Each account is assigned specific privileges that define what actions it can perform, such as reading data, inserting records, updating information, or managing other users.

### Reasons for Creating Database Accounts / Users

1. **Security:**
   * Restrict access to sensitive data by giving only authorized users the required permissions.
   * Prevent accidental or malicious changes to important tables.
2. **Role-Based Access Control:**
   * Different users have different responsibilities in the system.
   * Example:
     + **Admin:** Full control including creating other users and granting privileges.
     + **Dean:** Full control over the database but cannot create users.
     + **Lecturer:** Can manage course data but cannot grant privileges.
     + **Technical Officer:** Limited access to attendance-related tables for updating or inserting records.
     + **Student:** Read-only access to their own grades and attendance.
3. **Accountability:**
   * Actions in the database can be traced back to specific users.
   * Helps in auditing and tracking changes.
4. **Minimize Risks:**
   * Limiting privileges reduces the risk of data corruption, loss, or unauthorized access.
   * For example, students cannot modify grades or attendance.
5. **Organized Database Management:**
   * Assigning roles makes managing large databases easier.
   * Helps in maintaining consistency and following the principle of least privilege.

- **MySQL User Accounts for DB**

**1) Admin:** Full privileges with GRANT OPTION

CREATE USER 'admin'@'localhost' IDENTIFIED BY 'Admin@123';

GRANT ALL PRIVILEGES ON db\_project.\* TO 'admin'@'localhost' WITH GRANT OPTION;

**2) Dean:** Full privileges without GRANT OPTION

CREATE USER 'dean'@'localhost' IDENTIFIED BY 'Dean@123';

GRANT ALL PRIVILEGES ON db\_project.\* TO 'dean'@'localhost';

**3) Lecturer:** Full privileges on all tables, can create users

CREATE USER 'lecturer'@'localhost' IDENTIFIED BY 'Lecturer@123';

GRANT ALL PRIVILEGES ON db\_project.\* TO 'lecturer'@'localhost';

GRANT CREATE USER ON \*.\* TO 'lecturer'@'localhost';

**4) Technical Officer:** Read, write, update permissions on attendance-related tables/views

CREATE USER 'technical\_officer'@'localhost' IDENTIFIED BY 'Tech@123';

GRANT SELECT, INSERT, UPDATE ON db\_project.attendance TO 'technical\_officer'@'localhost';

GRANT SELECT, INSERT, UPDATE ON db\_project.session TO 'technical\_officer'@'localhost';

GRANT SELECT, INSERT, UPDATE ON db\_project.student TO 'technical\_officer'@'localhost';

GRANT SELECT, INSERT, UPDATE ON db\_project.attendance\_detailed TO 'technical\_officer'@'localhost';

GRANT SELECT, INSERT, UPDATE ON db\_project.attendance\_combined TO 'technical\_officer'@'localhost';

GRANT SELECT, INSERT, UPDATE ON db\_project.student\_attendance\_summary TO 'technical\_officer'@'localhost';

GRANT SELECT, INSERT, UPDATE ON db\_project.student\_attendance\_details TO 'technical\_officer'@'localhost';

**5) Student**: Read-only permission for final attendance and marks/grades tables/views

CREATE USER 'student'@'localhost' IDENTIFIED BY 'Student@123';

**Attendance views**

GRANT SELECT ON db\_project.attendance\_detailed TO 'student'@'localhost';

GRANT SELECT ON db\_project.attendance\_combined TO 'student'@'localhost';

GRANT SELECT ON db\_project.student\_attendance\_summary TO 'student'@'localhost';

GRANT SELECT ON db\_project.student\_attendance\_details TO 'student'@'localhost';

**Marks & results views**

GRANT SELECT ON db\_project.student\_results TO 'student'@'localhost';

GRANT SELECT ON db\_project.semester\_pass\_fail TO 'student'@'localhost';

GRANT SELECT ON db\_project.student\_class TO 'student'@'localhost';

GRANT SELECT ON db\_project.v\_progressive\_cgpa TO 'student'@'localhost';

-- Batch & overall summaries (optional for student view)

GRANT SELECT ON db\_project.batch\_department\_marks TO 'student'@'localhost';

GRANT SELECT ON db\_project.batch\_marks\_summary TO 'student'@'localhost';

GRANT SELECT ON db\_project.student\_marks\_summary TO 'student'@'localhost';

GRANT SELECT ON db\_project.student\_overall\_eligibility TO 'student'@'localhost';

GRANT SELECT ON db\_project.batch\_overall\_eligibility TO 'student'@'localhost';

Apply changes

FLUSH PRIVILEGES;

## Code Snippets and Instructions

### Steps to Set Up the Database

1. **Run Triggers** (Found in the triggers folder):
   * trg\_attendance\_before\_insert
   * trg\_attendance\_before\_update
   * trg\_ca\_marks\_before\_insert
   * trg\_ca\_marks\_before\_update
   * trg\_marks\_eligibility\_before\_insert
   * trg\_marks\_eligibility\_before\_update
2. **Insert Data** in the following order (Found in the data folder):
   1. users
   2. department
   3. lecture
   4. tech\_officer
   5. student
   6. dean
   7. course
   8. student\_course
   9. lecture\_department
   10. department\_course
   11. student\_course
   12. session
   13. medical
   14. attendance
3. **Add Attendance Views** (Can Found in the source Views file).
4. **Insert Marks Data**.
5. **Add calculate\_result() Procedure** (Can Found in the source ). Results will be automatically inserted into the result table.
6. **Add Remaining Views**.

### Sample Queries

### ****Procedures****

**1) CALL calculate\_result();**

**2) CALL get\_student\_course\_marks(‘U013’,’ICT1222’);**

**3) CALL get\_student\_eligibility(‘U013’,’ICT1222’);**

**4) CALL get\_batch\_marks\_summary\_by\_course(‘Database Management System’);**

**5) CALL generate\_student\_academic\_report(“TG/2023/1701”);**

**6) CALL**

### ****Attendance Details****

#### **a) Batch-wise attendance summary (give course code)**

SELECT

ad.reg\_no,

ad.course\_id,

ad.course\_name,

ad.session\_dates,

ad.attendance\_percentage,

ad.eligibility

FROM attendance\_detailed ad

WHERE ad.course\_id = 'ICT1222'

ORDER BY ad.reg\_no;

#### **b) All subjects for a batch**

SELECT

ac.reg\_no,

ac.course\_id,

ac.course\_name,

ac.attendance\_percentage,

ac.eligibility

FROM attendance\_combined ac

WHERE ac.academic\_year = '2025' AND ac.semester = '1'

ORDER BY ac.reg\_no, ac.course\_id;

#### **c) Individual student summary across all courses**

SELECT

sas.reg\_no,

sas.course\_id,

sas.course\_name,

sas.attendance\_percentage,

sas.medical\_percentage,

sas.eligibility

FROM student\_attendance\_summary sas

WHERE sas.reg\_no = 'TG/2023/1701'

ORDER BY sas.course\_id;

#### **d) Individual student for a specific course**

SELECT

sad.reg\_no,

sad.course\_id,

sad.course\_name,

sad.session\_date,

sad.session\_type,

sad.attendance\_status

FROM student\_attendance\_details sad

WHERE sad.reg\_no = 'TG/2023/1701'

AND sad.course\_id = 'ICT1222'

ORDER BY sad.session\_date;

#### **e) Theory / Practical / Combined Attendance**

-- Theory only

SELECT \* FROM attendance\_detailed

WHERE course\_id = 'ICT1222' AND session\_type = 'Theory';

-- Practical only

SELECT \* FROM attendance\_detailed

WHERE course\_id = 'ICT1222' AND session\_type = 'Practical';

-- Combined

SELECT \* FROM attendance\_combined

WHERE course\_id = 'ICT1222';

### ****Eligibility (Attendance + CA Marks)****

#### **a) Student-level overall eligibility**

SELECT

soe.reg\_no,

soe.course\_id,

soe.course\_name,

soe.attendance\_percentage,

soe.attendance\_eligibility,

m.ca\_marks,

m.ca\_eligible,

m.final\_eligible,

CASE

WHEN soe.attendance\_eligibility = 'Not Eligible' THEN 'Not Eligible (Attendance < 80%)'

WHEN m.ca\_eligible = 'Not Eligible' THEN 'Not Eligible (CA Failed)'

WHEN m.final\_eligible = 'Not Eligible' THEN 'Not Eligible (Final Failed)'

ELSE 'Fully Eligible'

END AS overall\_eligibility

FROM student\_overall\_eligibility soe

JOIN marks m

ON m.student\_id = soe.student\_id AND m.course\_id = soe.course\_id;

#### **b) Batch-level overall eligibility**

SELECT

course\_id,

course\_name,

COUNT(\*) AS total\_students,

SUM(CASE WHEN overall\_eligibility = 'Fully Eligible' THEN 1 ELSE 0 END) AS fully\_eligible,

SUM(CASE WHEN overall\_eligibility LIKE 'Not Eligible%' THEN 1 ELSE 0 END) AS not\_eligible

FROM student\_overall\_eligibility

GROUP BY course\_id, course\_name;

### ****Marks & Grades****

#### **a) Marks summary for batch by course**

SELECT

course\_id,

course\_name,

COUNT(\*) AS total\_students,

AVG(ca\_marks) AS avg\_ca\_marks,

AVG(final\_marks) AS avg\_final\_marks

FROM student\_marks\_summary

WHERE course\_id = 'ICT1222'

GROUP BY course\_id, course\_name;

#### **b) Individual marks**

SELECT

reg\_no,

course\_id,

course\_name,

ca\_marks,

final\_marks,

grade,

ca\_eligible,

final\_eligible

FROM student\_marks\_summary

WHERE reg\_no = 'TG/2023/1701';

#### **c) Check CA eligibility to sit for final exam**

SELECT

reg\_no,

course\_id,

course\_name,

ca\_marks,

ca\_eligible,

CASE

WHEN ca\_eligible = 'Eligible' THEN 'Can Sit for Final'

ELSE 'Cannot Sit for Final'

END AS final\_exam\_eligibility

FROM student\_marks\_summary

WHERE course\_id = 'ICT1222';

#### **d) Grade distribution for batch**

SELECT

course\_id,

course\_name,

grade,

COUNT(\*) AS students\_count

FROM student\_marks\_summary

WHERE course\_id = 'ICT1222'

GROUP BY course\_id, course\_name, grade;

### ****SGPA & CGPA****

#### **a) Individual SGPA & CGPA**

SELECT

reg\_no,

academic\_year,

semester,

sgpa,

cgpa

FROM v\_progressive\_cgpa

WHERE reg\_no = 'TG/2023/1701';

#### **b) Batch-wise SGPA & CGPA**

SELECT

reg\_no,

academic\_year,

semester,

sgpa,

cgpa

FROM v\_progressive\_cgpa

ORDER BY reg\_no, academic\_year, semester;

* Use views like batch\_attendance\_summary, student\_attendance\_summary, and attendance\_combined for simplified queries.
* Additional queries and procedure calls are in the requirements folder.

## Problems Faced During Development

1. **Team Collaboration Issues**:
   * Simultaneous modifications to database structures caused conflicts.
2. **Query and Logic Synchronization**:
   * Ensuring foreign key relations and joins worked correctly across multiple tables.
3. **Data Consistency and Automation**:
   * Required automatic updates for eligibility when marks or attendance changed.
4. **Combining Individual Work**:
   * Merging SQL scripts from team members caused version conflicts.
5. **Testing with Large Datasets**:
   * Validating queries with realistic data was time-consuming.

## Solutions to Overcome Problems

1. **Used GitHub for Collaboration**:
   * Set up a shared repository with branches and pull requests to avoid conflicts.
2. **Clear Communication and Role Division**:
   * Divided tasks (attendance, marks, reporting) and documented relationships.
3. **Introduced Triggers for Automation**:
   * Created triggers like trg\_marks\_eligibility for automatic updates.
4. **Continuous Testing**:
   * Ran stored procedures and views after integration to verify functionality.
5. **Used Views for Consistency**:
   * Isolated complex calculations in views for easier querying.

## Hosting the Backend Database

### Hosting Plan

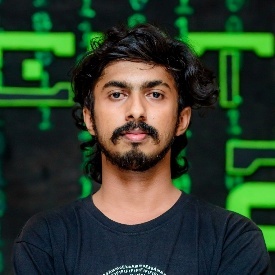
* Host the MySQL database on **Amazon Web Services (AWS) RDS**.
* **Reasons for Selecting AWS RDS**:
  + High availability, automatic backups, and scalability.
  + Free tier suitable for academic projects.
  + Encryption and access control via IAM roles.
  + Web-based management and MySQL Workbench integration.
* **Alternatives**: Railway, Clever Cloud, or Render for smaller-scale deployments.

### Changes Needed for Cloud Deployment

1. Export and import the schema to AWS RDS.
2. Update connection details (host, port, user, password) for the cloud endpoint.
3. Configure inbound rules in the RDS security group for authorized IPs.
4. Enable automated backups and snapshots.
5. Ensure UTF8MB4 character set for stored procedure compatibility.
6. Test stored procedures, triggers, and views on the cloud server.

## References

* MySQL 8.0 Official Documentation: <https://dev.mysql.com/doc/>
* AWS RDS Documentation: <https://docs.aws.amazon.com/rds/>
* Stack Overflow: For syntax clarification and optimization ideas
* GitHub Docs: For collaborative database project management

K.G.M.R.BANDARA 

TG/2023/1773

VIEWS

01.Batch Attendance Summary (All Students in a Course)

* **Main Purpose:** To provide a high-level attendance summary for the entire batch, aggregated on a per-course basis, directly fulfilling the project requirement to view attendance details "as a summary for whole batch".
* **Key Calculations:** For each course, the view calculates:
* The average attendance percentage.
* The total number of students.
* The specific count of *eligible* students (those with >= 80% attendance ).
* A formatted percentage of eligible students.

02.Batch-Level Eligibility (for Whole Course)

* **Purpose:** To provide an administrative summary of the entire batch's combined (attendance + CA) eligibility, broken down by course.
* **How it Works:** It aggregates data from your student\_overall\_eligibility view, grouping by course.
* **Key Outputs:** For each course, it shows:
* Total student enrollment.
* A specific count for each status: fully\_eligible, not\_eligible, medical\_cases, and withheld\_cases.
* The final percentage of fully\_eligible students.

03. Student\_Results

* **Main Purpose:** This view calculates the SGPA and CGPA for each student per semester, correctly implementing the project's rules for suspended students and medical certificates (MC).
* **How it Works**: It uses a two-step process:
  1. An inner query first calculates the numeric SGPA and adds flags to identify semesters affected by 'Suspended' or 'MC' status.
  2. An outer query then applies the final rules.
* **Key Rules:**
* SGPA: Displays 'WH' (Withheld) if the semester is suspended or has an MC.
* CGPA: Displays 'WH' if suspended. It displays NULL (blank) if there's an MC, correctly skipping that semester from the cumulative calculation.
* For all other normal semesters, it calculates the numeric, running CGPA.

04. Batch\_Marks\_Summary

* **Purpose:** Provides an administrative overview of the entire batch's performance and eligibility on a per-course basis, fulfilling the project's "summary for whole batch" requirement.
* **How it Works:** It groups data from the marks and course tables.
* **Key Outputs:** For each course, it shows:
* Total student count.
* Counts of students eligible for CA and the Final Exam.
* Average CA marks and average Final marks.
* The pass percentages for both CA and Final eligibility.

Trigger

These two triggers (trg\_marks\_eligibility\_before\_insert and trg\_marks\_eligibility\_before\_update) work together to **automatically enforce your project's business rules** for student results.

Their main purpose is to **calculate and set a student's eligibility status and final mark** every time a new mark record is added (INSERT) or an existing one is changed (UPDATE). This ensures your data is always consistent and correct.

**What they do:**

1. **Run Automatically:** They fire *before* any data is saved to the marks table.
2. **Gather Data:** For the specific student and course being updated, they first look up:
   * The student's status (e.g., 'Proper', 'Repeat', 'Suspended').
   * The student's final attendance\_percentage.
   * Any approved 'Mid' or 'Final' exam medicals.
3. **Enforce CA Eligibility Rules:** Based on the data, they set the ca\_eligible column to:
   * **'WH'** (Withheld) if the student is 'Suspended'.
   * **'MC'** (Medical) if they have an approved mid-exam medical.
   * **'Not Eligible'** if their ca\_marks are less than 20.
   * **'Eligible'** for everyone else.
4. **Enforce Final Eligibility Rules:** They set the final\_eligible column to:
   * **'WH'** (Withheld) if the student is 'Suspended'.
   * **'Eligible'** if the student is 'Repeat' (bypassing attendance checks).
   * **'E\*'** (Attendance Ineligible) if attendance is below 80% (and not a repeat/suspended student).
   * **'MC'** if they have an approved final exam medical.
   * **'Eligible'** for everyone else.
5. **Calculate Final Mark:** They automatically calculate the final\_marks using your project's formula: ( (final\_theory + final\_practical) \* 0.6 ) + ca\_marks