College Examination Result Processing System

A PROJECT REPORT
Submitted for the Subject
DATABASE MANAGEMENT SYSTEM (OSDBMS)

Submitted By:
Urmila Badve

Academic Year: 2025 – 2026

Established – 1961 Subject: OSDBMS

SEVA SADAN'S

R. K. TALREJA COLLEGE

OF

ARTS, SCIENCE & COMMERCE

ULHASNAGAR – 421 003

CERTIFICATE

This is to certify that Mr./Ms. Urmila Mahesh Badve of S.Y. Information Technology (SYIT) Roll No. 2542002 has satisfactorily completed the Open Source DataBase Management System Mini Project entitled College Examination Result Processing Database during the academic year 2025 – 2026, as a part of the practical requirement. The project work is found to be satisfactory and is approved for submission.

PROF. INCHARGE **HEAD OF DEPT**

_____ _____

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my project guide, **Prof. Sahil Shukla**, for his valuable guidance, constant encouragement, and continuous support throughout the development of my project titled *"College Examination Result Processing Database."* His expert advice and insightful suggestions helped me understand database concepts clearly and implement them effectively using MySQL.

I am also highly thankful to **R.K. Talreja Degree College** for providing the necessary facilities, resources, and a positive learning environment that enabled me to complete this project successfully. This project has greatly enhanced my practical knowledge of database design, SQL queries, and result processing systems.

I would like to express my heartfelt thanks to my parents and friends for their motivation, encouragement, and support during the completion of this project.

Finally, I would like to thank everyone who directly or indirectly contributed to the successful completion of this project.

Name: Urmila Badve
College: R.K. Talreja Degree College
Project Title: College Examination Result Processing Database
Date: 23/01/2026

INDEX

1. INTRODUCTION:

A College Examination Result Processing Database is a structured digital system designed to store, manage, and process students' academic result information efficiently. It uses database tables to organize data such as student details, subject information, marks, grades, and examination records. Each piece of information is stored in a systematic format using rows and columns, which allows easy access, updating, and retrieval. The system also maintains relationships between different entities, such as students and their results, ensuring data consistency and accuracy. This database system acts as a central repository where all examination-related data is securely stored and managed.

In traditional result processing, colleges often rely on manual record keeping using paper files or spreadsheets. This approach can lead to several problems, such as calculation errors, data duplication, loss of records, and difficulty in retrieving information quickly. Managing results for hundreds or thousands of students becomes time-consuming and inefficient. The database system solves these problems by automating result storage and processing. It allows fast entry of marks, automatic calculation of totals and grades, and quick generation of result reports. It also improves accuracy, reduces human errors, ensures data security, and makes it easy for administrators to search, update, or analyze student performance. Overall, it improves efficiency, reliability, and organization in examination result management.

MySQL is used because it is a powerful, reliable, and open-source relational database management system developed and maintained by Oracle Corporation. Being open-source means it is free to use, which makes it ideal for educational institutions with limited budgets. MySQL supports structured data storage using tables and relationships, which is essential for managing student and result data. It provides high performance, strong security features, and supports multiple users accessing the database at the same time. MySQL is also easy to learn, integrates well with programming languages like Python, Java, and PHP, and is widely used in real-world applications. Additionally, it ensures data integrity through constraints such as primary keys and foreign keys, which help maintain accurate and consistent records.

Another important reason for using MySQL as an open-source solution is its flexibility and strong community support. Since MySQL is open-source, its source code is freely available, allowing developers and institutions to modify and customize the database according to their specific requirements. This makes it highly adaptable for college examination systems, where different institutions may have different grading patterns or result formats. In addition, MySQL has a large global community supported by Oracle and independent developers who continuously improve its performance, security, and features. Regular updates, detailed documentation, and online support make it easier to troubleshoot issues and maintain the system. This ensures long-term reliability and sustainability of the examination result processing database.

In summary, the College Examination Result Processing Database is an essential system that helps colleges store, manage, and process examination results efficiently. It replaces manual systems with an automated, accurate, and secure solution. By using MySQL as an open-source database management system, the project becomes cost-effective, reliable, and scalable, making it suitable for modern educational institutions.

2. PROBLEM DEFINITION:

In many colleges, examination results are managed using manual methods such as paper records, registers, or basic spreadsheet files. These traditional systems are not efficient for handling large volumes of student data and often lead to errors, delays, and poor data management. Manual result processing makes it difficult to maintain accuracy, ensure data security, and retrieve information quickly. As the number of students and subjects increases, the limitations of the manual system become more serious. Therefore, there is a strong need for a database-driven solution that can store, process, and manage examination results in a structured, secure, and efficient manner.

Issues in the existing manual / unstructured system:

Data Redundancy: Same student information is recorded multiple times, leading to duplication of data.
Data Inconsistency: Duplicate records may contain different or incorrect information, causing confusion.
Calculation Errors: Manual calculation of total marks, percentage, and grades increases chances of mistakes.
Poor Data Security: Paper records or simple files can be easily lost, damaged, or accessed by unauthorized persons.
Difficulty in Data Retrieval: Searching for a specific student's result is time-consuming and inefficient.
Lack of Proper Organization: Records are not systematically structured, making management difficult.
Limited Scalability: Manual systems cannot efficiently handle large numbers of students and examination records.

Need for a database-driven solution:

To store student and result data in a structured and organized manner.
To reduce data redundancy and maintain data consistency.
To automate result calculations and improve accuracy.
To provide secure access and protect sensitive student information.
To allow fast retrieval, updating, and management of examination results.
To improve overall efficiency, reliability, and performance of result processing.

CONCLUSION:

The existing manual examination result system is inefficient, error-prone, and difficult to manage, especially when handling large amounts of student data. Problems such as data redundancy, inconsistency, lack of security, and slow data retrieval reduce the reliability of the system. These limitations highlight the need for a database-driven solution. By using a structured database system, colleges can store and manage examination results accurately, securely, and efficiently. This improves data organization, reduces errors, and ensures faster and more reliable result processing, making the system suitable for modern educational institutions.

3. OBJECTIVES OF THE PROJECT:

The main objective of this project is to design and implement a College Examination Result Processing Database using MySQL that can efficiently store, manage, and process student examination records. The system organizes student information, subject details, exam data, and marks in separate tables and links them using relationships. It also automates result generation, ensures data accuracy, and provides secure and structured data management.

Specific Objectives:

- **To design a structured database system:**
  To create properly organized tables such as *student, subject, exam,* and *marks* to store examination data in a systematic and relational format.
- **To implement SQL queries for data management:**
  To use SQL commands such as INSERT, SELECT, and JOIN to add student records, store marks, and retrieve examination results efficiently.
- **To ensure data integrity using constraints:**
  To apply constraints such as PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, and NOT NULL to maintain accuracy and consistency of data. For example, marks are restricted between 0 and 100, and relationships between student, subject, and exam tables are maintained using foreign keys.
- **To automate result generation using database features:**
  To create a stored procedure (*generate_result*) that automatically calculates total marks, percentage, and result classification such as Distinction, First Class, Pass, or Fail.
- **To enforce data validation using triggers:**
  To use a trigger (*validate_marks*) that prevents insertion of marks if the student and subject semesters do not match, ensuring correct and valid data entry.
- **To create views for easy result access:**
  To create a view (*result_view*) that allows easy retrieval of student marks and subject information without complex queries.
- **To gain hands-on experience with MySQL:**
  To develop practical knowledge of creating databases, tables, procedures, triggers, and views using MySQL.

This project helps in understanding how a relational database system can be used to manage college examination results efficiently. It ensures accurate result processing, maintains data integrity, and automates calculations using procedures and triggers. The use of MySQL provides practical experience in database design and implementation, making the system reliable, secure, and suitable for real-world educational institutions.

4. SCOPE OF THE PROJECT:

The College Examination Result Processing Database system is designed to store, manage, and process student examination results in a structured and efficient manner. This system can be used in colleges, universities, and other educational institutions to maintain accurate records of students, subjects, exams, and marks. It helps automate result generation, ensures data accuracy through constraints and triggers, and allows easy retrieval of result information using views and SQL queries. The system improves efficiency, reduces manual work, and provides secure data management.

Users of the system:

- Administrator:
  The admin manages the database, creates tables, inserts student and subject data, and controls overall system operations.
- Staff / Faculty:
  Faculty members can enter student marks, update records, and view examination results.
- Students:
  Students can view their marks, total score, percentage, and result status through result reports or result views.
- Institution / Business User:
  Educational institutions use the system to maintain academic records, analyze student performance, and generate official result reports.

Subdomains of the Project Scope:

- Educational Use:
  Used in colleges and universities to store student details, subject information, and examination marks, and to generate results automatically.
- Administrative Use:
  Helps administrators manage student records, enter marks, and generate result reports efficiently.
- Institutional / Business Use:
  Helps educational institutions maintain academic records, monitor student performance, and manage examination data systematically

Academic Limitations:

- The system is designed for a limited number of tables such as student, subject, exam, and marks.
- It does not include advanced features such as online result access through a web application.
- The system is suitable for small to medium-sized educational institutions.
- It requires basic knowledge of MySQL to operate and manage the database.

5. REQUIREMENT SPECIFICATION:

5.1 Hardware Requirements

The following hardware components are required to implement and run the College Examination Result Processing Database system:

- **Computer / Laptop:**
  A computer or laptop with a minimum Intel Core i3 processor or equivalent is required to run MySQL and manage the database efficiently.
- **Minimum RAM:**
  At least **4 GB RAM** is required to ensure smooth operation of the database system and MySQL server.
- **Storage:**
  A minimum of **20 GB free disk space** is required to store the MySQL software, database files, and related project data.
- **Input Devices:**
  Keyboard and mouse are required for data entry and system operation.
- **Output Devices:**
  Monitor is required to view database tables, queries, and result reports.

5.2 Software Requirements

The following software is required to develop and operate the College Examination Result Processing Database system:

Table 5.2:Software Requirements

Software	Purpose
MySQL Server	Stores and manages all database tables (student, subject, exam, marks), supports procedures, triggers, and views for result processing.
MySQL Workbench	GUI tool for writing and executing SQL queries, creating tables, and managing the database visually.
Operating System (Windows/Linux)	Provides the platform and system resources to run MySQL Server and Workbench.
SQL (Structured Query Language)	Language used to create tables, insert and retrieve data, calculate results, and maintain data integrity.

6. SYSTEM DESIGN:

6.1 System Architecture (Conceptual)

The system architecture of the College Examination Result Processing Database is designed to manage and process student examination results efficiently using a database-driven approach. It focuses on how users interact with the database and how MySQL handles queries.

- **User Interaction with Database:**
  Users such as administrators, faculty, and staff interact with the system through SQL queries or a graphical interface like MySQL Workbench. They can insert student details, subject information, exam data, and marks into the database. Users can also retrieve results, calculate totals, and view reports through queries or pre-defined views.
- **Role of MySQL Server:**
  MySQL Server acts as the core database engine. It stores all the data in structured tables (student, subject, exam, marks) and enforces data integrity using constraints, triggers, and relationships. It processes user requests, executes SQL commands, and manages data securely and efficiently. Stored procedures like `generate_result` automate result calculation, while triggers like `validate_marks` ensure data accuracy.
- **Query Execution Flow:**

1. The user submits a query (e.g., SELECT, INSERT, UPDATE) via MySQL Workbench or a query interface.
2. The MySQL server parses the query to check syntax and validates it against database rules.
3. The server optimizes the query and executes it on the relevant tables.
4. The result is returned to the user, either as a report, table view, or confirmation of data insertion.
5. For automated tasks, stored procedures and triggers are executed as part of the flow to maintain consistency and calculate results.

This architecture ensures efficient communication between users and the database, maintaining secure, accurate, and reliable management of examination results.

7. DATABASE DESIGN:

7.1 Entity Description

Student Table:
This table stores information about each student, including their name, roll number, course, and semester. Each student is uniquely identified by a student_id. The table ensures that duplicate student entries are avoided, and data such as semester is restricted to valid values using constraints.

Subject Table:
The subject table contains details of each course subject, including subject name, semester, and maximum marks. It is linked with the student and marks tables to ensure the correct subjects are assigned to students in the appropriate semester.

Exam Table:
This table records information about each exam, such as the type of exam (e.g., End Semester) and the exam year. It allows multiple exams to be conducted for different years and types, which can then be linked with student marks.

Marks Table:
The marks table stores the marks obtained by students in each subject for a particular exam. It links the student, subject, and exam tables using foreign keys, ensuring data consistency. A trigger ensures that marks are only inserted if the student and subject semester match.

Views and Procedures:
A view (result_view) allows easy retrieval of student results along with subject names, while the stored procedure (generate_result) calculates total marks, percentage, and final result for each student automatically.

7.2 Table Structure

1. Student Table

Figure 7.2.1: Student Table

Attribute	Data Type	Description	Constraints
Student_id	INT	Unique ID for each student	PRIMARY KEY, AUTO_INCREMENT
Name	VARCHAR(50)	Full name of the student	NOT NULL
Roll_no	VARCHAR(20)	Unique roll number	UNIQUE
Course	VARCHAR(30)	Course enrolled by student	NULL allowed
Semester	INT	Current semester of student	CHECK(semester BETWEEN 1 AND 6)

2.Subject Table

Figure 7.2.2: Subject Table

Attribute	Data Type	Description	Constraints
subject_id	INT	Unique ID for each subject	PRIMARY KEY, AUTO_INCREMENT
subject_name	VARCHAR(50)	Name of the subject	NULL allowed
semester	INT	Semester in which the subject is taught	NULL allowed
max_marks	INT	Maximum marks for the subject	DEFAULT 100

3. Exam Table

Figure 7.2.3: Exam Table

Attribute	Data Type	Description	Constraints
exam_id	INT	Unique ID for each exam	PRIMARY KEY, AUTO_INCREMENT
exam_type	VARCHAR(20)	Type of exam (e.g., End Semester)	NULL allowed
exam_year	YEAR	Year in which the exam is conducted	NULL allowed

4. Marks Table

Figure 7.2.4: Marks Table

Attribute	Data Type	Description	Constraints
marks_id	INT	Unique ID for each marks record	PRIMARY KEY, AUTO_INCREMENT
student_id	INT	ID of the student	FOREIGN KEY REFERENCES student(student_id)
subject_id	INT	ID of the subject	FOREIGN KEY REFERENCES subject(subject_id)
exam_id	INT	ID of the exam	FOREIGN KEY REFERENCES exam(exam_id)
marks_obtained	INT	Marks obtained by the student	CHECK (marks_obtained BETWEEN 0 AND 100)

7.3 Constraints Used

- **PRIMARY KEY:**
  Each table has a primary key (student_id, subject_id, exam_id, marks_id) to uniquely identify each record and prevent duplicate entries.
- **FOREIGN KEY:**
  The marks table uses foreign keys (student_id, subject_id, exam_id) to link each mark to the correct student, subject, and exam. This ensures referential integrity.
- **NOT NULL:**
  Columns such as name in the student table are set to NOT NULL to ensure mandatory information is always provided.
- **UNIQUE:**
  The roll_no in the student table is unique to prevent multiple students from having the same roll number.
- **CHECK:**
  Columns like semester and marks_obtained use CHECK constraints to ensure valid values (e.g., semester between 1 and 6, marks between 0 and 100).

8. UML DIAGRAMS:

In this project, various UML diagrams are used to represent and understand the system effectively. The **ER Diagram** shows the main entities—Student, Subject, Exam, and Marks—and their relationships. The **Use Case Diagram** illustrates how Admin, Faculty/Staff, and Students interact with the system for operations like inserting, updating, viewing, and generating results. The **Activity Diagram** represents the workflow of database operations from data entry to result generation. Finally, the **Sequence Diagram** shows the flow of query execution in MySQL, including parsing, execution, triggers, procedures, and returning results. Together, these diagrams provide a clear visual representation of the system's structure, operations, and data flow.

8.1 ER Diagram

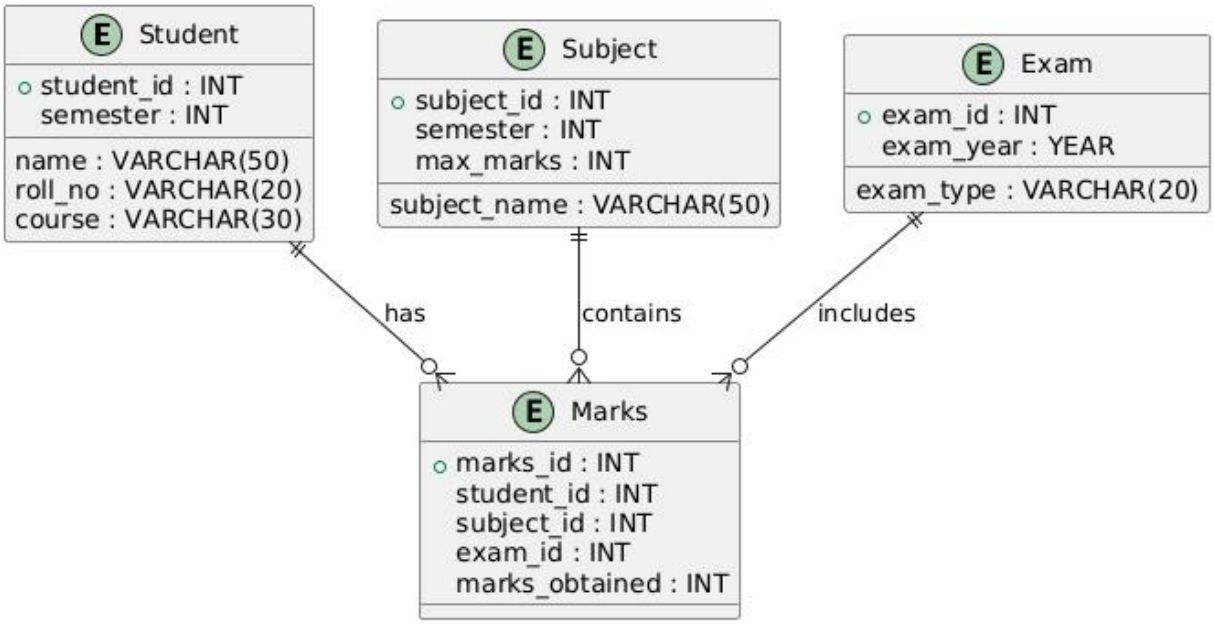


Figure 9.1: ER Diagram

The ER Diagram represents the main entities of the system—Student, Subject, Exam, and Marks—and their relationships. It shows how students, subjects, and exams are linked through the Marks table, ensuring data consistency and proper mapping of examination results.

8.2 Use Case Diagram

Figure 8.1: Use Case Diagram

The Use Case Diagram shows how different users interact with the system. Admin manages students, subjects, and exams; Faculty enters and updates marks; and Students view their results. It highlights the key operations like insert, update, view, and generate report.

8.3 Activity Diagram

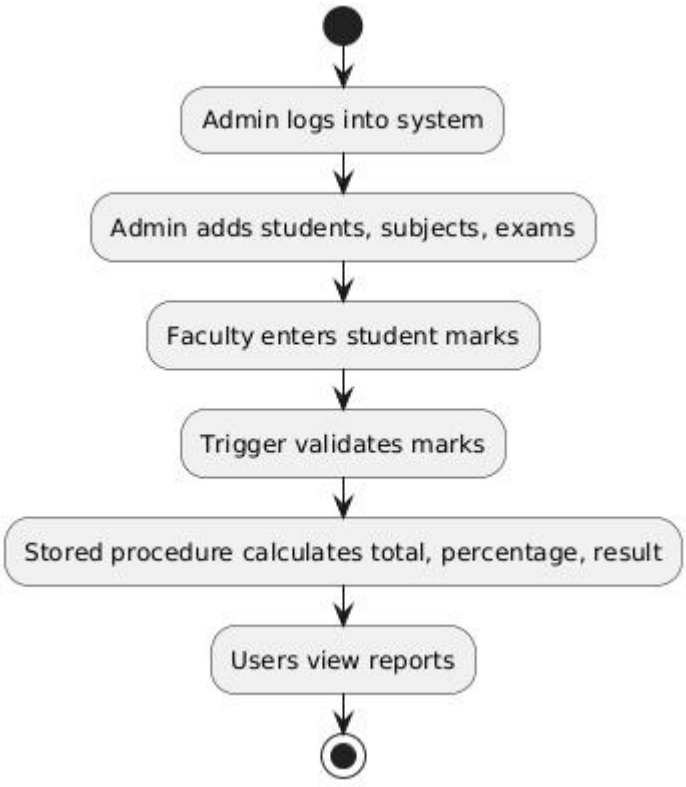


Figure 8.3: Activity Diagram

The Activity Diagram illustrates the workflow of the system. It shows the step-by-step process from adding student, subject, and exam data, to entering marks, validating data with triggers, calculating results with procedures, and finally generating reports for users.

8.4 Sequence Diagram

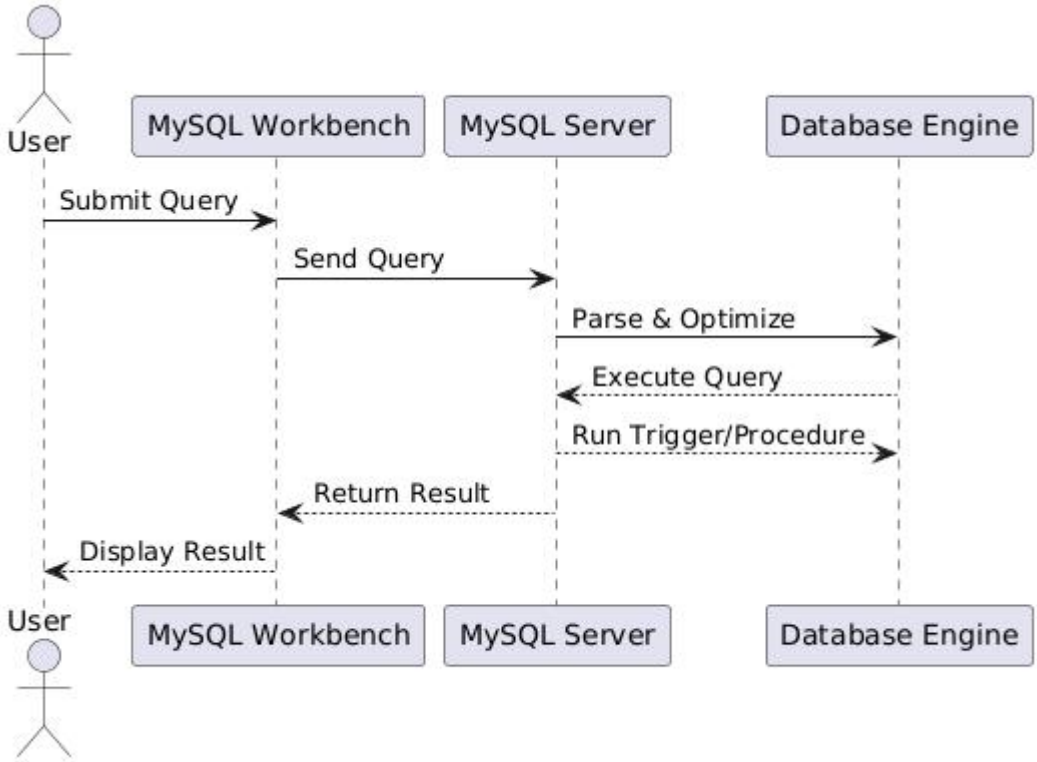


Figure 8.4: Sequence Diagram

The Sequence Diagram shows the flow of query execution. It depicts how a user sends a query through MySQL Workbench, how the MySQL Server parses and executes it, triggers or procedures run automatically if needed, and results are returned back to the user.

9. SQL IMPLEMENTATION

SQL (Structured Query Language) was used in this project to create and manage the College Examination Result Processing Database. SQL helps in creating the database, defining tables, inserting data, retrieving information, and performing advanced result analysis. It ensures structured storage and efficient processing of examination data.

9.1 Database Creation

The database was created using the CREATE DATABASE statement. This command creates a new database that stores all examination-related data such as student details, subject information, exam records, and marks. After creating the database, the USE command selects the database for further operations.

```
CREATE DATABASE college_exam_db;
USE college_exam_db;
```

This database acts as the main storage area for all tables and ensures proper organization of examination data.

9.2 Table Creation

Tables were created using the CREATE TABLE statement to store different types of information in a structured format. Each table contains attributes with appropriate data types and constraints such as PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE, and CHECK to maintain data integrity.

Student Table:

```
CREATE TABLE student (
  student_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  roll_no VARCHAR(20) UNIQUE,
  course VARCHAR(30),
  semester INT CHECK (semester BETWEEN 1 AND 6)
);
```

This table collect the deatils of the student.

Subject Table:

```
CREATE TABLE subject (
  subject_id INT AUTO_INCREMENT PRIMARY KEY,
  subject_name VARCHAR(50),
  semester INT,
  max_marks INT DEFAULT 100
);
```

The subject table stores information about subjects, including subject ID, subject name, semester, and maximum marks. The subject_id is the primary key and is automatically incremented. The max_marks field has a default value of 100.

Exam Table:

```
CREATE TABLE exam (
  exam_id INT AUTO_INCREMENT PRIMARY KEY,
  exam_type VARCHAR(20),
  exam_year YEAR
);
```

The exam table stores information about examinations, including exam ID, exam type, and exam year. The exam_id is the primary key and uniquely identifies each exam.

Marks Table:

```
CREATE TABLE marks (
  marks_id INT AUTO_INCREMENT PRIMARY KEY,
  student_id INT,
  subject_id INT,
  exam_id INT,
  marks_obtained INT CHECK (marks_obtained BETWEEN 0 AND 100),
  FOREIGN KEY (student_id) REFERENCES student(student_id),
  FOREIGN KEY (subject_id) REFERENCES subject(subject_id),
  FOREIGN KEY (exam_id) REFERENCES exam(exam_id)
);
```

The marks table connects student, subject, and exam tables using foreign keys, ensuring proper relationships between entities.

9.3 Data Insertion

Data was inserted into the tables using the INSERT INTO statement. This allows storing records such as student details, subject names, exam types, and marks obtained.

Insert data into Student Table

```
INSERT INTO student (name, roll_no, course, semester)
VALUES
('Amit Patil','IT401','BSc IT',4),
('Neha Sharma','IT402','BSc IT',4);
```

Insert data into Subject Table

```
INSERT INTO subject (subject_name, semester)
VALUES
('DBMS',4),
('Operating System',4);
```

Insert data into Exam Table

```
INSERT INTO exam (exam_type, exam_year)
VALUES ('End Semester',2025);
```

Insert data into Marks Table

```
INSERT INTO marks (student_id, subject_id, exam_id, marks_obtained)
VALUES
(1,1,1,78),
(1,2,1,85),
(2,1,1,66),
(2,2,1,72);
```

9.4 Data Retrieval

Data was retrieved using SELECT statements. The SELECT command allows users to view stored data from one or more tables.

- SELECT statement

```
SELECT * FROM student;
```

- WHERE statement

```
SHOW PROCEDURE STATUS WHERE Db = 'college_exam_db';
```

- Join Query

In Procedure the join statement is used

```
FROM student s
JOIN marks m ON s.student_id = m.student_id
GROUP BY s.student_id;
```

In view the join statement is used

```
FROM student s
JOIN marks m ON s.student_id = m.student_id
JOIN subject sub ON m.subject_id = sub.subject_id;
```

JOIN statements are used in the stored procedure and view to combine data from student, marks, and subject tables. This allows the system to calculate results and display complete student examination information.

9.5 Advanced Queries

Advanced SQL queries were used for result processing and analysis.

GROUP BY

```
FROM student s
JOIN marks m ON s.student_id = m.student_id
GROUP BY s.student_id;
```

GROUP BY is used to calculate total marks and percentage for each student.

SUBQUERY

```
IF (SELECT semester FROM student WHERE student_id = NEW.student_id)
  !=
  (SELECT semester FROM subject WHERE subject_id = NEW.subject_id)
```

This is a subquery because:
SELECT is inside another statement.
It checks semester match before inserting marks.

VIEW

```
CREATE VIEW result_view AS
SELECT
  s.roll_no,
  s.name,
  sub.subject_name,
  m.marks_obtained
FROM student s
JOIN marks m ON s.student_id = m.student_id
JOIN subject sub ON m.subject_id = sub.subject_id;
```

View is used to display student results easily.

Conclusion

The SQL implementation successfully created and managed the College Examination Result Processing Database using MySQL. SQL commands were used to create the database and tables, insert student and examination data, and retrieve result information efficiently. Advanced features such as JOIN, GROUP BY, subqueries, and views were used to organize and analyze student marks. The stored procedure automated result calculation, and the trigger ensured data integrity by validating semester conditions. Overall, SQL provided a structured, accurate, and efficient method for managing and processing examination results in the system.

GANTT CHART:

College Examination Result Processing Database
- Project Schedule

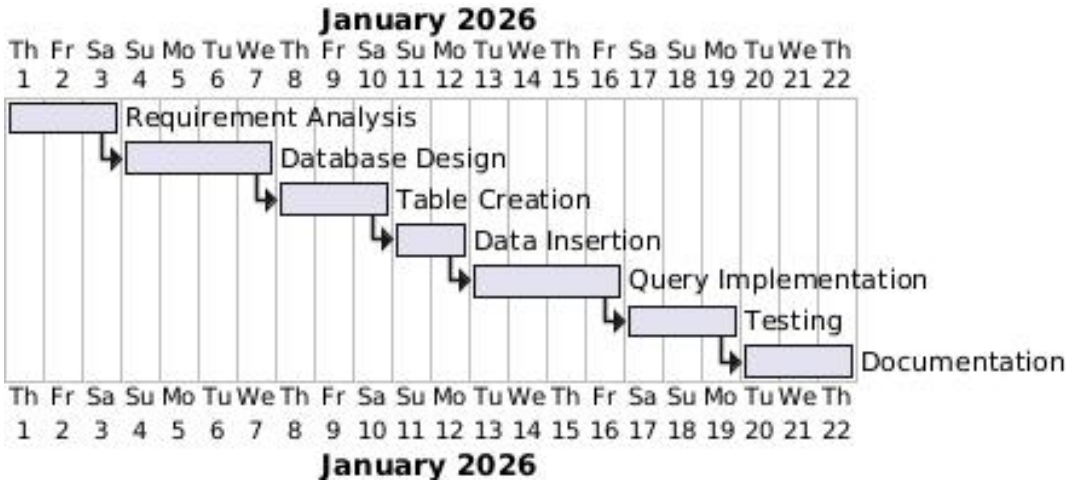


Figure : Gantt Diagram

10.System Testing And Result

System testing was performed to verify that the College Examination Result Processing Database functions correctly and produces accurate results. All SQL queries such as CREATE, INSERT, SELECT, VIEW, PROCEDURE, TRIGGER, and transaction commands were executed and tested. The system was checked for correct table creation, proper data insertion, data validation, and accurate result generation.

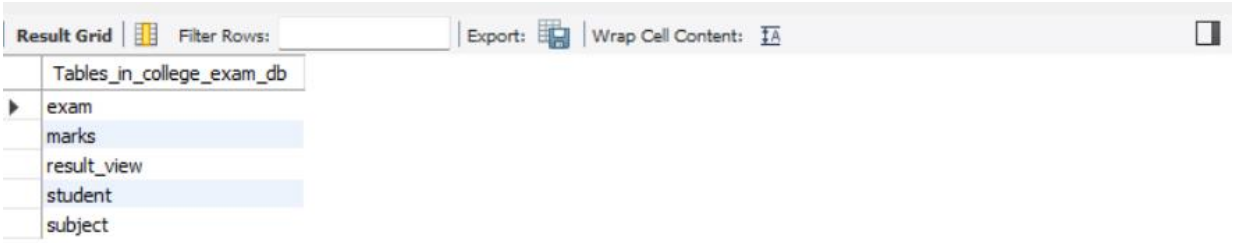
10.1 Query Correctness Testing

All SQL queries including CREATE DATABASE, CREATE TABLE, INSERT, SELECT, and JOIN were tested successfully. The queries were executed step by step to ensure proper database functionality.

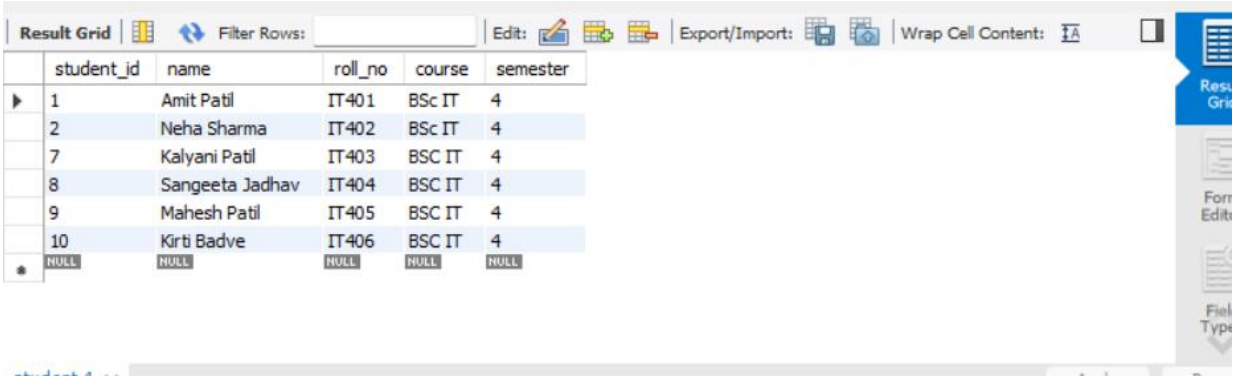
The following operations were verified:

- Database was created successfully using CREATE DATABASE college_exam_db;
- Tables student, subject, exam, and marks were created correctly
- Records were inserted successfully using INSERT statements
- Relationships between tables worked correctly using FOREIGN KEY constraints
- Stored procedure generate_result() executed successfully
- View result_view displayed correct student marks information

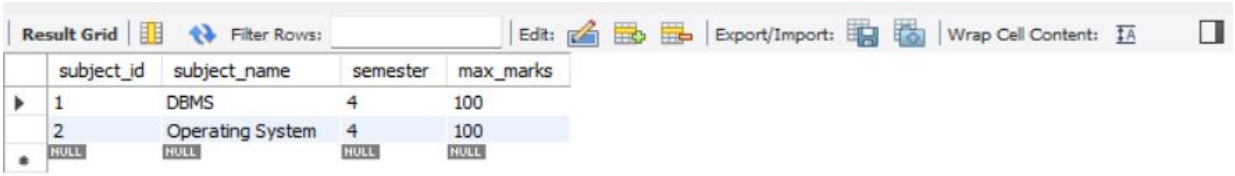
Show tables:



Student table:



Subject table:

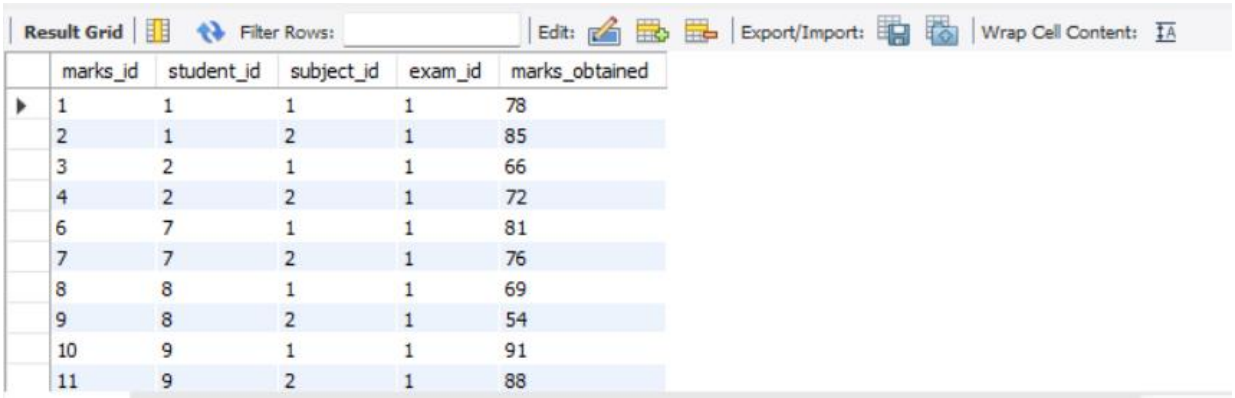


subject_id	subject_name	semester	max_marks
1	DBMS	4	100
2	Operating System	4	100
NULL	NULL	NULL	NULL

Exam Table:

exam_id	exam_type	exam_year
1	End Semester	2025
NULL	NULL	NULL

Marks Table:



marks_id	student_id	subject_id	exam_id	marks_obtained
1	1	1	1	78
2	1	2	1	85
3	2	1	1	66
4	2	2	1	72
6	7	1	1	81
7	7	2	1	76
8	8	1	1	69
9	8	2	1	54
10	9	1	1	91
11	9	2	1	88

Procedure statement:

The stored procedure **generate_result()** is used to calculate the examination result of each student automatically. It retrieves student details and marks from the database, then calculates total marks and percentage using aggregate functions. Based on the percentage, it assigns the result classification such as Distinction, First Class, Pass, or Fail and displays the result.

- **CREATE PROCEDURE generate_result()**
  Creates a stored procedure named `generate_result` to calculate student results.
- **BEGIN … END**
  Defines the start and end of the procedure.
- **SELECT s.roll_no, s.name**
  Retrieves student roll number and name from the `student` table.
- **SUM(m.marks_obtained) AS total_marks**
  Calculates total marks obtained by each student.
- **ROUND(SUM(m.marks_obtained)/COUNT(m.subject_id),2) AS percentage**
  Calculates the percentage and rounds it to 2 decimal places.
- **CASE statement**
  Assigns result based on percentage:
  • ≥ 75 → Distinction
  • ≥ 60 → First Class
  • ≥ 40 → Pass
  • < 40 → Fail
- **FROM student s JOIN marks m ON s.student_id = m.student_id**
  Combines student and marks tables using student ID.
- **GROUP BY s.student_id**
  Groups data so result is calculated separately for each student.
- **CALL generate_result();**
  Executes the stored procedure and displays the result.

```
CALL generate_result();
```

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

student_id	roll_no	name	total_marks	percentage	result	class
1	IT401	Amit Patil	163	81.50	PASS	DISTINCTION
2	IT402	Neha Sharma	138	69.00	PASS	FIRST CLASS
7	IT403	Kalyani Patil	157	78.50	PASS	DISTINCTION
8	IT404	Sangeeta Jadhav	123	61.50	PASS	FIRST CLASS
9	IT405	Mahesh Patil	179	89.50	PASS	DISTINCTION
10	IT406	Kirti Badve	142	71.00	PASS	FIRST CLASS

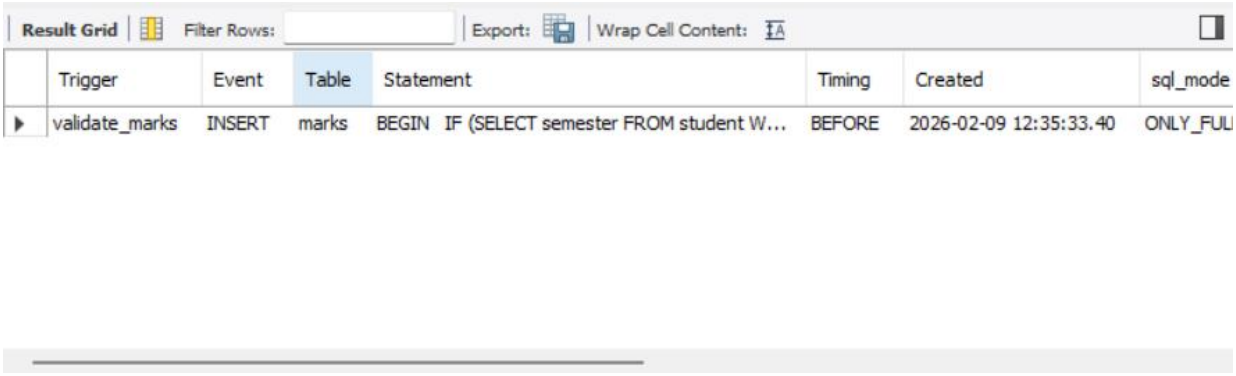
Trigger statement:

The trigger validate_marks is used to validate data before inserting marks into the marks table. It checks whether the student's semester matches the subject's semester. If the semesters do not match, the trigger prevents insertion and displays an error message, ensuring data integrity.

- **CREATE TRIGGER validate_marks**
  Creates a trigger named `validate_marks` on the `marks` table.
- **BEFORE INSERT ON marks**
  The trigger runs automatically before inserting any new record into the marks table.
- **FOR EACH ROW**
  The trigger checks every row being inserted.
- **IF condition**
  It compares the student's semester with the subject's semester.
- **SIGNAL SQLSTATE '45000'**
  If semesters do not match, it shows an error message and stops the insertion.

- **MESSAGE_TEXT**
  Displays the message: *"Semester mismatch between student and subject"*.

Show triggers();

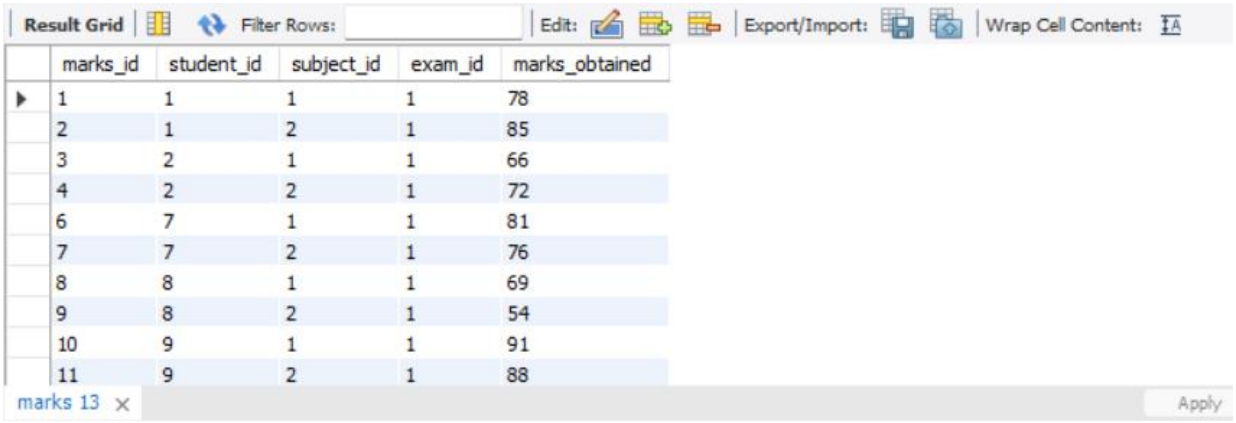


Trigger	Event	Table	Statement	Timing	Created	sql_mode
validate_marks	INSERT	marks	BEGIN IF (SELECT semester FROM student W...	BEFORE	2026-02-09 12:35:33.40	ONLY_FUL

Transaction statement:

Transaction is used to ensure safe and reliable data insertion. START TRANSACTION begins the transaction, INSERT adds the data, and COMMIT permanently saves the changes. If an error occurs, ROLLBACK can be used to undo the changes.

```
START TRANSACTION;
INSERT INTO marks (student_id, subject_id, exam_id, marks_obtained)
VALUES (1, 1, 1, 78);
COMMIT;
```



marks_id	student_id	subject_id	exam_id	marks_obtained
1	1	1	1	78
2	1	2	1	85
3	2	1	1	66
4	2	2	1	72
6	7	1	1	81
7	7	2	1	76
8	8	1	1	69
9	8	2	1	54
10	9	1	1	91
11	9	2	1	88

Sample Result Explanation:

- Two students, Amit Patil (IT401) and Neha Sharma (IT402), were inserted into the student table with course BSc IT and semester 4.
- Two subjects, DBMS and Operating System, were inserted for semester 4 in the subject table.
- One exam record, End Semester 2025, was inserted into the exam table.
- Marks were inserted for both students, where Amit Patil scored 78 and 85 marks, and Neha Sharma scored 66 and 72 marks.
- These records were used to calculate total marks, percentage, and result using the stored procedure and view.

11. SECURITY, BACKUP AND RECOVERY

11.1 Security

Security is important to protect the database from unauthorized access and ensure data safety. In MySQL, security is managed using user accounts, privileges, and access control.

User Privileges

User privileges define what operations a user can perform on the database. The database administrator can grant permissions such as SELECT, INSERT, UPDATE, and DELETE. For example, a user may be allowed to view student results but not modify them. This ensures controlled access to sensitive data.

Example:

GRANT SELECT ON college_exam_db.* TO 'user1'@'localhost';

Access Control

Access control ensures that only authorized users can access the database. Each user must log in using a username and password. The administrator can restrict access based on user roles. For example, only the admin can insert or modify marks, while other users can only view results. This prevents unauthorized data changes.

11.2 Backup

Backup is the process of creating a copy of the database to prevent data loss due to system failure, errors, or accidental deletion.

mysqldump Explanation

MySQL provides the mysqldump utility to create a backup of the database. It saves the database structure and data into a SQL file, which can be used later for recovery.

Example command:

mysqldump -u root -p college_exam_db > backup.sql

This command creates a backup file named backup.sql containing all tables and data of the database.

11.3 Recovery

Recovery is the process of restoring the database from a backup file when data is lost or corrupted.

Restore Concept

The backup file created using mysqldump can be used to restore the database using the MySQL command.

Example command:

mysql -u root -p college_exam_db < backup.sql

This command restores the database structure and data from the backup file.

Recovery ensures that the database can be restored to its original state after data loss or system failure.

12. FUTURE SCOPE AND CONCLUSION

12.1 Future Scope

The College Examination Result Processing Database can be further improved by adding more advanced features and integrating it with modern technologies.

- **Web Integration**

The database can be integrated with a web application so that students and faculty can access results online. Students can log in using their credentials and view their marks, percentage, and result securely. This will make the system more user-friendly and accessible from anywhere.

- **Advanced Reporting**

Advanced reporting features can be added to generate detailed reports such as subject-wise performance, semester-wise results, and overall student performance. These reports can help teachers and administrators analyze student progress more effectively.

- **Analytics Features**

Analytics features can be implemented to analyze student performance trends. The system can identify top-performing students, weak subjects, and overall class performance. This will help institutions improve teaching methods and academic planning.

12.2 Conclusion

The College Examination Result Processing Database was successfully designed and implemented using MySQL. The system was able to store student, subject, exam, and marks data efficiently. SQL queries, stored procedures, triggers, views, and transactions were used to manage and process examination results accurately.

This project helped in understanding database design, table creation, data insertion, and result generation using SQL. It also provided practical knowledge of advanced MySQL features such as stored procedures, triggers, views, and transaction management.

MySQL proved to be an effective and reliable database management system for storing and processing examination data. It ensures data accuracy, security, and efficient result generation, making it suitable for educational institutions.

13. REFERENCES

- **MySQL Official Documentation** – For detailed explanations of MySQL syntax, commands, and best practices.
- **Prescribed Textbooks** – Recommended books provided by your course or instructor.
- **Online Learning Sources** – Tutorials, articles, and guides from reputable websites.

14. GLOSSARY

DBMS (Database Management System): Software that manages databases and allows users to store, retrieve, and manipulate data efficiently.

SQL (Structured Query Language): A standard programming language used to manage and manipulate relational databases.

Primary Key: A unique identifier for a record in a database table, ensuring no duplicate entries exist.

Foreign Key: A field in one table that links to the primary key of another table, establishing a relationship between the two tables.

MySQL: An open-source relational database management system that uses SQL for database operations.

Table: A collection of related data organized in rows (records) and columns (fields) in a database.

Record (Row): A single entry in a table representing one object, person, or event.

Field (Column): A single attribute or property of a record in a database table.

Index: A database structure that improves the speed of data retrieval operations on a table.

Query: A request for data or information from a database using SQL commands.

Normalization: The process of organizing database data to reduce redundancy and improve data integrity.

Constraint: A rule applied to a table column to enforce data integrity, such as NOT NULL or UNIQUE.

Appendix :

A.Source Code Implementation (SQL):

```
CREATE DATABASE college_exam_db;
USE college_exam_db;
CREATE TABLE student (
  student_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  roll_no VARCHAR(20) UNIQUE,
  course VARCHAR(30),
  semester INT CHECK (semester BETWEEN 1 AND 6)
);
CREATE TABLE subject (
  subject_id INT AUTO_INCREMENT PRIMARY KEY,
  subject_name VARCHAR(50),
  semester INT,
  max_marks INT DEFAULT 100
);
CREATE TABLE exam (
  exam_id INT AUTO_INCREMENT PRIMARY KEY,
  exam_type VARCHAR(20),
  exam_year YEAR
);
CREATE TABLE marks (
  marks_id INT AUTO_INCREMENT PRIMARY KEY,
  student_id INT,
  subject_id INT,
  exam_id INT,
  marks_obtained INT CHECK (marks_obtained BETWEEN 0 AND 100),
  FOREIGN KEY (student_id) REFERENCES student(student_id),
  FOREIGN KEY (subject_id) REFERENCES subject(subject_id),
  FOREIGN KEY (exam_id) REFERENCES exam(exam_id)
);
INSERT INTO student (name, roll_no, course, semester)
VALUES
('Amit Patil','IT401','BSc IT',4),
('Neha Sharma','IT402','BSc IT',4);
INSERT INTO subject (subject_name, semester)
VALUES
('DBMS',4),
('Operating System',4);
INSERT INTO exam (exam_type, exam_year)
VALUES ('End Semester',2025);
INSERT INTO marks (student_id, subject_id, exam_id, marks_obtained)
VALUES
(1,1,1,78),
(1,2,1,85),
(2,1,1,66),
(2,2,1,72);
DELIMITER //
```

```
CREATE PROCEDURE generate_result()
BEGIN
  SELECT
    s.roll_no,
    s.name,
    SUM(m.marks_obtained) AS total_marks,
    ROUND(SUM(m.marks_obtained)/COUNT(m.subject_id),2) AS percentage,
    CASE
      WHEN SUM(m.marks_obtained)/COUNT(m.subject_id) >= 75 THEN
'Distinction'
      WHEN SUM(m.marks_obtained)/COUNT(m.subject_id) >= 60 THEN
'First Class'
      WHEN SUM(m.marks_obtained)/COUNT(m.subject_id) >= 40 THEN
'Pass'
      ELSE 'Fail'
    END AS result
  FROM student s
  JOIN marks m ON s.student_id = m.student_id
  GROUP BY s.student_id;
END //

DELIMITER ;
CALL generate_result();
DELIMITER //

CREATE TRIGGER validate_marks
BEFORE INSERT ON marks
FOR EACH ROW
BEGIN
  IF (SELECT semester FROM student WHERE student_id = NEW.student_id)
     !=
     (SELECT semester FROM subject WHERE subject_id = NEW.subject_id)
  THEN
     SIGNAL SQLSTATE '45000'
     SET MESSAGE_TEXT = 'Semester mismatch between student and
subject';
  END IF;
END //

DELIMITER ;
CREATE VIEW result_view AS
SELECT
  s.roll_no,
  s.name,
  sub.subject_name,
  m.marks_obtained
FROM student s
JOIN marks m ON s.student_id = m.student_id
JOIN subject sub ON m.subject_id = sub.subject_id;
SELECT * FROM result_view;
```

```
START TRANSACTION;

INSERT INTO marks (student_id, subject_id, exam_id, marks_obtained)
VALUES (1, 1, 1, 78);

COMMIT;

show tables;
SHOW PROCEDURE STATUS WHERE Db = 'college_exam_db';

SHOW TRIGGERS;
SHOW FULL TABLES WHERE Table_type = 'VIEW';
SELECT * FROM student;
SELECT * FROM subject;
SELECT * FROM exam;
SELECT * FROM marks;

USE college_exam_db;
SELECT * FROM student;
DESCRIBE student;
ALTER TABLE student
MODIFY roll_no VARCHAR(20) NOT NULL;
ALTER TABLE student AUTO_INCREMENT = 3;
SELECT * FROM marks
WHERE marks_id = 5;
CREATE OR REPLACE VIEW result_view AS
SELECT
    s.student_id,
    s.roll_no,
    s.name,

    SUM(m.marks_obtained) AS total_marks,

    ROUND(AVG(m.marks_obtained), 2) AS percentage,

    CASE
        WHEN AVG(m.marks_obtained) >= 40 THEN 'PASS'
        ELSE 'FAIL'
    END AS result,

    CASE
        WHEN AVG(m.marks_obtained) >= 75 THEN 'DISTINCTION'
        WHEN AVG(m.marks_obtained) >= 60 THEN 'FIRST CLASS'
        WHEN AVG(m.marks_obtained) >= 50 THEN 'SECOND CLASS'
        WHEN AVG(m.marks_obtained) >= 40 THEN 'PASS CLASS'
        ELSE 'FAIL'
    END AS class

FROM student s
JOIN marks m ON s.student_id = m.student_id
```

```
GROUP BY s.student_id, s.roll_no, s.name;

ALTER TABLE student DROP PRIMARY KEY;
SELECT
    ROW_NUMBER() OVER (ORDER BY student_id) AS Sr_No,
    student_id,
    name,
    roll_no,
    course,
    semester
FROM student;
show tables;
select * from student;
select * from subject;
select * from exam;
select * from marks;

CALL generate_result();
SHOW TRIGGERS;
```

B.GUI Implementation (Python – Tkinter) :

```
import tkinter as tk
from tkinter import ttk, messagebox
import mysql.connector

# ---------------- DATABASE CONNECTION ----------------
def get_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="root123",
        database="college_exam_db"
    )

# ---------------- MAIN WINDOW ----------------
root = tk.Tk()
root.title("College Exam Management System")
root.geometry("1200x700")
root.resizable(False, False)

bg_main = "#1f2937"
bg_sidebar = "#111827"
accent = "#2563eb"

root.configure(bg=bg_main)

# ---------------- HEADER ----------------
header = tk.Frame(root, bg=accent, height=60)
header.pack(fill="x")

tk.Label(header,
         text="College Exam Management System",
         bg=accent,
         fg="white",
         font=("Segoe UI", 18, "bold")).pack(pady=10)

# ---------------- SIDEBAR ----------------
sidebar = tk.Frame(root, bg=bg_sidebar, width=220)
sidebar.pack(side="left", fill="y")

# ---------------- MAIN AREA ----------------
main_area = tk.Frame(root, bg=bg_main)
main_area.pack(side="right", fill="both", expand=True)

# ---------------- TREEVIEW ----------------
tree = ttk.Treeview(main_area)
tree.pack(fill="both", expand=True, padx=20, pady=20)

style = ttk.Style()
style.theme_use("clam")
```

```
style.configure("Treeview",
                background="white",
                foreground="black",
                rowheight=28,
                fieldbackground="white")

# ---------------- CLEAR TREE ----------------
def clear_tree():
    for item in tree.get_children():
        tree.delete(item)

# ---------------- SHOW DATA ----------------
def show_data(query):

    try:
        conn = get_connection()
        cursor = conn.cursor(buffered=True)

        cursor.execute(query)

        rows = cursor.fetchall()
        columns = [desc[0] for desc in cursor.description]

        clear_tree()

        tree["columns"] = columns
        tree["show"] = "headings"

        for col in columns:
            tree.heading(col, text=col)
            tree.column(col, width=150)

        for row in rows:
            tree.insert("", tk.END, values=row)

        cursor.close()
        conn.close()

    except Exception as e:
        messagebox.showerror("Error", str(e))


# ---------------- FORM AREA ----------------
form_frame = tk.Frame(main_area, bg=bg_main)
form_frame.pack(pady=10)

entries = {}

def create_form(labels):
```

```
    for widget in form_frame.winfo_children():
        widget.destroy()

    entries.clear()

    for i, label in enumerate(labels):

        tk.Label(form_frame,
                 text=label,
                 bg=bg_main,
                 fg="white",
                 font=("Segoe UI", 10)).grid(row=0, column=i, padx=5)

        entry = tk.Entry(form_frame)
        entry.grid(row=1, column=i, padx=5)

        entries[label] = entry


# ---------------- STUDENT MODULE ----------------
def student_module():

    create_form(["ID", "Name", "Roll No", "Course", "Semester"])

    # ADD
    def add():

        try:
            conn = get_connection()
            cursor = conn.cursor()

            cursor.execute(
                "INSERT INTO student (name, roll_no, course, semester)
VALUES (%s,%s,%s,%s)",
                (
                    entries["Name"].get(),
                    entries["Roll No"].get(),
                    entries["Course"].get(),
                    entries["Semester"].get()
                )
            )

            conn.commit()

            cursor.close()
            conn.close()

            show_data("""
SELECT
ROW_NUMBER() OVER (ORDER BY student_id) AS Sr_No,
student_id,
```

```
name,
roll_no,
course,
semester
FROM student
""")

        except Exception as e:
            messagebox.showerror("Error", str(e))


    # UPDATE
    def update():

        if entries["ID"].get() == "":
            messagebox.showwarning("Warning", "Enter ID to update")
            return

        try:
            conn = get_connection()
            cursor = conn.cursor()

            cursor.execute(
                """UPDATE student
                   SET name=%s, roll_no=%s, course=%s, semester=%s
                   WHERE student_id=%s""",
                (
                    entries["Name"].get(),
                    entries["Roll No"].get(),
                    entries["Course"].get(),
                    entries["Semester"].get(),
                    int(entries["ID"].get())
                )
            )

            conn.commit()

            cursor.close()
            conn.close()

            show_data("SELECT * FROM student")

        except Exception as e:
            messagebox.showerror("Error", str(e))


    # DELETE (FIXED)
    def delete():

        if entries["ID"].get() == "":
            messagebox.showwarning("Warning", "Enter ID to delete")
```

```
            return

        try:
            conn = get_connection()
            cursor = conn.cursor()

            cursor.execute(
                "DELETE FROM student WHERE student_id=%s",
                (int(entries["ID"].get()),)
            )

            conn.commit()

            cursor.close()
            conn.close()

            messagebox.showinfo("Success", "Deleted Successfully")

            show_data("SELECT * FROM student")

        except Exception as e:
            messagebox.showerror("Error", str(e))


    tk.Button(form_frame,
              text="Add",
              bg="#16a34a",
              fg="white",
              command=add).grid(row=2, column=0, pady=10)

    tk.Button(form_frame,
              text="Update",
              bg="#f59e0b",
              fg="white",
              command=update).grid(row=2, column=1)

    tk.Button(form_frame,
              text="Delete",
              bg="#dc2626",
              fg="white",
              command=delete).grid(row=2, column=2)

    show_data("SELECT * FROM student")


# ---------------- SUBJECT MODULE ----------------
def subject_module():

    create_form(["Subject Name", "Semester"])

    def add():
```

```
        conn = get_connection()
        cursor = conn.cursor()

        cursor.execute(
            "INSERT INTO  subject (subject_name, semester) VALUES
(%s,%s)",
            (
                entries["Subject Name"].get(),
                entries["Semester"].get()
            )
        )

        conn.commit()

        cursor.close()
        conn.close()

        show_data("SELECT * FROM subject")

    tk.Button(form_frame,
              text="Add Subject",
              bg="#16a34a",
              fg="white",
              command=add).grid(row=2, column=0, pady=10)

    show_data("SELECT * FROM subject")


# ---------------- EXAM MODULE ----------------
def exam_module():

    create_form(["Exam Type", "Exam Year"])

    def add():

        conn = get_connection()
        cursor = conn.cursor()

        cursor.execute(
            "INSERT INTO exam (exam_type, exam_year) VALUES (%s,%s)",
            (
                entries["Exam Type"].get(),
                entries["Exam Year"].get()
            )
        )

        conn.commit()

        cursor.close()
        conn.close()
```

```
        show_data("SELECT * FROM exam")

    tk.Button(form_frame,
              text="Add Exam",
              bg="#16a34a",
              fg="white",
              command=add).grid(row=2, column=0, pady=10)

    show_data("SELECT * FROM exam")


# ---------------- MARKS MODULE ----------------

def marks_module():

    create_form(["Marks ID", "Student ID", "Subject ID", "Exam ID",
"Marks"])

    # ---------------- FILL FORM WHEN ROW CLICKED ----------------
    def fill_form(event):

        selected = tree.focus()

        if not selected:
            return

        values = tree.item(selected, "values")

        entries["Marks ID"].delete(0, tk.END)
        entries["Student ID"].delete(0, tk.END)
        entries["Subject ID"].delete(0, tk.END)
        entries["Exam ID"].delete(0, tk.END)
        entries["Marks"].delete(0, tk.END)

        entries["Marks ID"].insert(0, values[0])
        entries["Student ID"].insert(0, values[1])
        entries["Subject ID"].insert(0, values[2])
        entries["Exam ID"].insert(0, values[3])
        entries["Marks"].insert(0, values[4])


    tree.bind("<ButtonRelease-1>", fill_form)


    # ---------------- ADD MARKS ----------------
    def add():

        student_id = entries["Student ID"].get().strip()
        subject_id = entries["Subject ID"].get().strip()
        exam_id = entries["Exam ID"].get().strip()
```

```
        marks = entries["Marks"].get().strip()

        if not student_id or not subject_id or not exam_id or not
marks:
            messagebox.showwarning("Warning", "Fill all fields")
            return

        try:
            conn = get_connection()
            cursor = conn.cursor()

            # check duplicate
            cursor.execute("""
                SELECT COUNT(*)
                FROM marks
                WHERE student_id=%s AND subject_id=%s AND exam_id=%s
            """, (student_id, subject_id, exam_id))

            count = cursor.fetchone()[0]

            if count > 0:
                messagebox.showerror(
                    "Duplicate Error",
                    "Marks already exist for this student in this
subject and exam"
                )
            else:
                cursor.execute("""
                    INSERT INTO marks
                    (student_id, subject_id, exam_id, marks_obtained)
                    VALUES (%s,%s,%s,%s)
                """, (student_id, subject_id, exam_id, marks))

                conn.commit()

                messagebox.showinfo("Success", "Marks added")

                show_data("SELECT * FROM marks")

            cursor.close()
            conn.close()

        except Exception as e:
            messagebox.showerror("Error", str(e))


    # ---------------- DELETE MARKS ----------------
    def delete():

        marks_id = entries["Marks ID"].get().strip()
```

```
        if not marks_id:
            messagebox.showwarning("Warning", "Select a row to
delete")
            return

        try:
            conn = get_connection()
            cursor = conn.cursor()

            cursor.execute(
                "DELETE FROM marks WHERE marks_id=%s",
                (marks_id,)
            )

            conn.commit()

            cursor.close()
            conn.close()

            messagebox.showinfo("Success", "Marks deleted")

            show_data("SELECT * FROM marks")

        except Exception as e:
            messagebox.showerror("Error", str(e))


    # ---------------- BUTTONS ----------------
    tk.Button(form_frame,
              text="Add Marks",
              bg="#16a34a",
              fg="white",
              font=("Segoe UI", 10, "bold"),
              command=add).grid(row=2, column=0, pady=10)

    tk.Button(form_frame,
              text="Delete Marks",
              bg="#dc2626",
              fg="white",
              font=("Segoe UI", 10, "bold"),
              command=delete).grid(row=2, column=1, pady=10)


    show_data("SELECT * FROM marks")
# ---------------- GENERATE RESULT ----------------
def generate_result():

    try:

        conn = get_connection()
        cursor = conn.cursor()
```

```
        cursor.callproc("generate_result")

        conn.commit()

        cursor.close()
        conn.close()

        messagebox.showinfo("Success", "Result Generated
Successfully")

        result_view()

    except Exception as e:
        messagebox.showerror("Error", str(e))


# ---------------- RESULT VIEW (FIXED) ----------------
def result_view():

    show_data("SELECT * FROM result_view")


# ---------------- SIDEBAR BUTTONS ----------------
modules = [

    ("Students", student_module),
    ("Subjects", subject_module),
    ("Exams", exam_module),
    ("Marks", marks_module),
    ("Generate Result", generate_result),
    ("Result View", result_view)

]

for text, command in modules:

    tk.Button(sidebar,
              text=text,
              bg=bg_sidebar,
              fg="white",
              activebackground=accent,
              relief="flat",
              font=("Segoe UI", 11),
              command=command).pack(fill="x", pady=3, ipady=8)
root.mainloop()
```

This GUI was developed using Python Tkinter and connected to MySQL using mysql.connector. It provides modules for managing students, subjects, exams, and marks. SQL queries are executed dynamically and displayed using Treeview. A stored procedure is used to generate final results automatically.