ELEVATE LABS

DEPARTMENT OF DATA SCIENCE & ENGINEERING

A
Internship Project Report On

"STUDENT RESULT PROCESSING SYSTEM (SQL-BASED)"

SUBMITTED IN FULFILLMENT OF THE REQUIREMENT OF
DATA SCIENCE INTERNSHIP @ ELEVATE LABS

GUIDED BY:	**PREPARED BY:**
Elevate Labs Mentor	**MANSURI MO. UMER**

**JULY 2025**

ELEVATE LABS
www.elevatelabs.in

ABSTRACT

This project titled **"Student Result Processing System"** is developed as a part of the internship program at *Elevate Labs*. The goal of this system is to efficiently manage student information, course registration, and academic performance using SQL and relational database design principles.

The system allows administrators to register students and courses, record marks, and automatically compute grades and GPAs. It includes features like ranking students using window functions, handling grade logic through stored procedures, and generating performance views for subjects and students.

This project provides a practical solution for educational institutions to automate the evaluation process and generate detailed academic insights. Tools like **MySQL Workbench** and **GitHub** were used for implementation and version control.

The project enhanced our understanding of SQL procedures, normalization, ER modeling, and query optimization techniques—offering a strong foundation in real-world database systems.
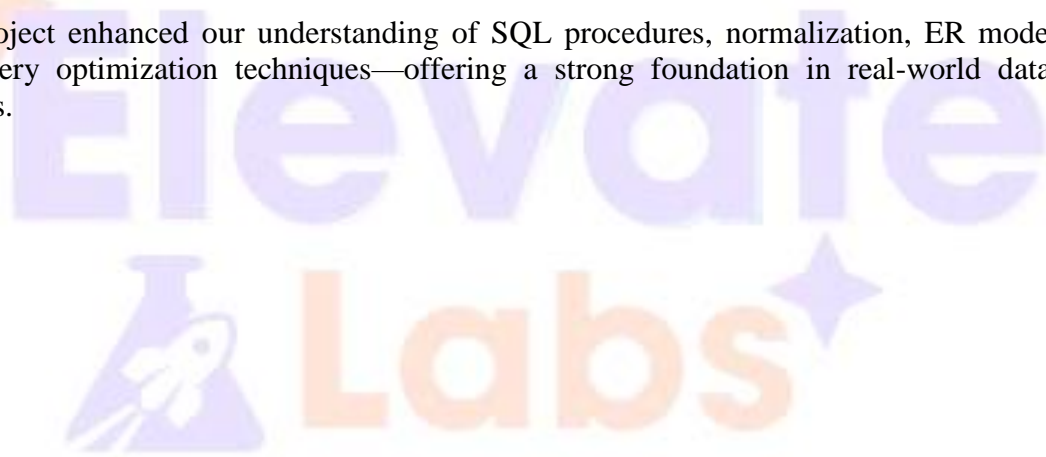
INTRODUCTION

The **Student Result System** is designed to simplify and automate the process of managing student academic data. It acts as a centralized platform that maintains student profiles, course enrollments, exam results, grade calculations, and performance analysis.

In traditional systems, result processing often involves manual effort, spreadsheets, and scattered records, which can lead to errors and inefficiencies. This project introduces a robust, SQL-based relational database that ensures accurate data handling, faster report generation, and easier academic tracking.

The project is built using **MySQLWorkBetch**, employing concepts such as:

- **ER Diagrams** for database modelling,
- **Primary and Foreign Keys** for relational integrity,
- **Stored Procedures** to automate grade logic,
- **Views** to represent complex queries clearly, and
- **Joins** and **Window Functions** for reporting GPA and student rankings.

This system not only reduces human error but also provides a scalable solution that can be integrated into real-world school or university portals.

OBJECTIVES

The main objective of the **Student Result System** is to build a structured, error-free academic result processing system that is reliable, fast, and easy to maintain. This system is aimed at helping educational institutions manage and analyze student performance efficiently.

Key Objectives:

- To **design a relational database** using **MySQL** that accurately stores student, course, and marks data.
- To **automate result calculations**, including GPA/CGPA and grade assignments, using stored procedures.
- To **maintain student-course relationships** via foreign keys and mapping tables.
- To provide **quick access to performance data** through SQL **views** and **joins**.
- To implement **ranking functionality** using **window functions** for academic analysis.
- To ensure **data integrity and normalization** by following best database design practices.
- To present user-friendly outputs for administrators through **SQL queries and reports**.

SYSTEM DESIGN:

The **system design** phase involves creating a structured model of the database that outlines how data is stored, related, and accessed. This system follows the **Entity-Relationship (ER)** model and is implemented using **MySQL** relational database principles.

1. Entity Relationship Diagram (ERD):

The ER diagram represents the core components of the system and their relationships:

Entities:

- Students
- Courses
- StudentCourses (mapping table)
- Results
- Grades
- Gpa

Relationships:

- A student can enroll in multiple courses (*many-to-many*, resolved by StudentCourses).
- A course can have many students enrolled.
- Each result links a student to a course with marks and grade.
- Grades are mapped to score ranges.

E-R Diagram for Student Result Processing System:
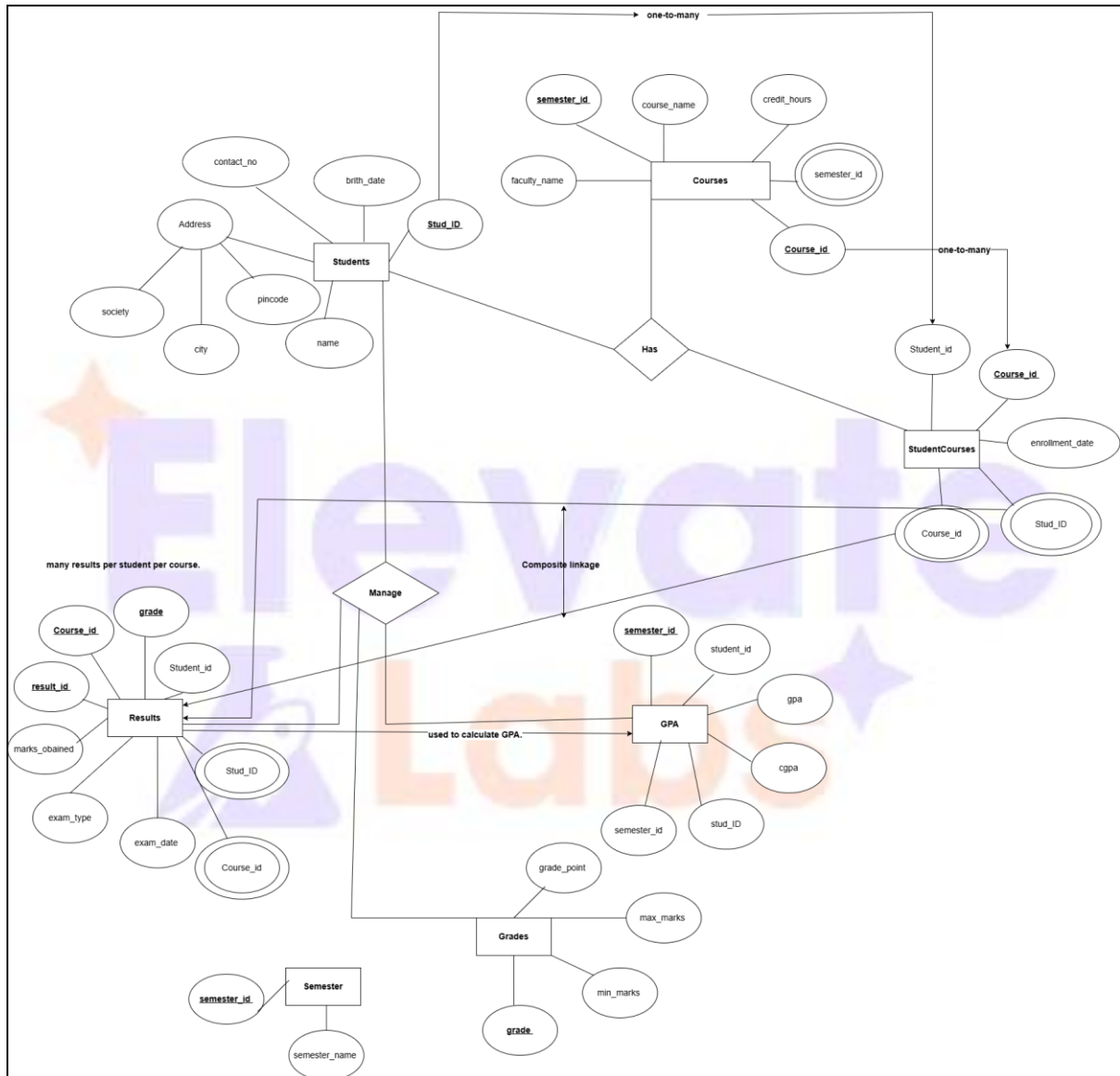
2. Schema Creation (Table Creation):

1. Students Table:

Column Name	Data Type	Key Type	Description
stud_ID	INT	Primary Key	Unique student ID
name	VARCHAR(100)	–	Full name of the student
birth_date	DATE	–	Date of birth
contact_no	VARCHAR(10)	–	Mobile number
society	VARCHAR(100)	–	Society or area name
city	VARCHAR(50)	–	City
pincode	VARCHAR(10)	–	Postal code

2. Courses Table:

Column Name	Data Type	Key Type	Description
course_id	INT	Primary Key	Unique ID for each course
course_name	VARCHAR(100)	–	Course title
credit_hours	INT	–	Number of credit hours
faculty_name	VARCHAR(100)	–	Name of faculty/instructor
semester_id	INT	Foreign Key	Links to Semester.semester_id

3. Semester Table:

Column Name	Data Type	Key Type	Description
semester_id	INT	Primary Key	Unique semester ID
semester_name	VARCHAR(50)	–	Name of semester (e.g., "Sem 1")

4. StudentCourses Table:

Column Name	Data Type	Key Type	Description
student_id	INT	Foreign Key	References Students.stud_ID
course_id	INT	Foreign Key	References Courses.course_id
enrollment_date	DATE	–	Date student enrolled in this course
PRIMARY KEY	–	Composite Key	(student_id, course_id)

5. Results Table:

Column Name	Data Type	Key Type	Description
result_id	INT	Primary Key	Unique result ID
student_id	INT	Foreign Key	References Students.stud_ID
course_id	INT	Foreign Key	References Courses.course_id
marks_obtained	INT	–	Marks scored by the student
grade	VARCHAR(2)	–	Grade (e.g., A, B+, etc.)
exam_type	VARCHAR(50)	–	Midterm / Final
exam_date	DATE	–	Date of the exam

6. Grades Table:

Column Name	Data Type	Key Type	Description
grade	VARCHAR(2)	Primary Key	Grade code (A, B+, etc.)
min_marks	INT	–	Minimum marks for the grade
max_marks	INT	–	Maximum marks for the grade
grade_point	FLOAT	–	Numeric grade point (for GPA)

7. GPA Table:

Column Name	Data Type	Key Type	Description
student_id	INT	Foreign Key	References Students.stud_ID
semester_id	INT	Foreign Key	References Semester.semester_id
gpa	DECIMAL(4,2)	–	GPA for that semester
cgpa	DECIMAL(4,2)	–	Cumulative GPA
PRIMARY KEY	–	Composite Key	(student_id, semester_id)

2. Insert Dummy Data into All Tables: (Screenshot of the data)

1. Student table:

2. Semester table:

3. Courses table:

```
48 •   INSERT INTO Courses (course_id, course_name, credit_hours, semester_id, faculty_name) VALUES
49     (1, 'Mathematics', 4, 1, 'Dr. Mehta'),
50     (2, 'Computer Programming', 5, 1, 'Mr. Patel'),
51     (3, 'Database Systems', 8, 4, 'Ms. Shah'),
52     (4, 'Cloud Computing', 5, 2, 'Mr. Nikunj'),
53     (5, 'Python', 6, 3, 'Ms. Shain'),
54     (6, 'Basic Of Infromation Tech', 4, 2, 'Ms. Anjali'),
55     (7, 'Intern net of Thinges', 4, 2, 'Mr. Pandor'),
56     (8, 'MySQL', 4, 2, 'Mr. SakirDenaths'),
57     (9, 'BDE', 4, 2, 'Mr. mohsin');
58
59     -- StudentCourses Table (Enrollment)
60 •   INSERT INTO StudentCourses (student_id, course_id, enrollment_date) VALUES
```

Output

Action Output

#	Time	Action	Message
✔ 1	14:55:59	INSERT INTO Courses (course_id, course_name, credit_hours, semester_id, faculty_name) VALUES (1, 'Mathematics', 4, ...	9 row(s) affected Records: 9 Duplicates: 0 Warnings: 0

4. StudentCourses table & Grade Table:

```
59     -- StudentCourses Table (Enrollment)
60 •   INSERT INTO StudentCourses (student_id, course_id, enrollment_date) VALUES
61     (101, 1, '2025-06-01'),
62     (101, 2, '2025-06-01'),
63     (102, 1, '2025-06-01'),
64     (103, 3, '2025-06-15');
65
66     -- Grades Table
67 •   INSERT INTO Grades (grade, min_marks, max_marks, grade_point) VALUES
68     ('A+', 90, 100, 10.0),
69     ('A', 80, 89, 9.0),
70     ('B+', 70, 79, 8.0),
71     ('B', 60, 69, 7.0),
72     ('C', 50, 59, 6.0),
73     ('F', 0, 49, 0.0);
74
75         Results Table
```

Output

Action Output

#	Time	Action	Message
✔ 1	14:55:59	INSERT INTO Courses (course_id, course_name, credit_hours, semester_id, faculty_name) VALUES (1, 'Mathematics', 4, ...	9 row(s) affected Records: 9 Duplicates: 0 Warnings: 0
✔ 2	14:56:04	INSERT INTO StudentCourses (student_id, course_id, enrollment_date) VALUES (101, 1, '2025-06-01'), (101, 2, '2025-06-...	4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0
✔ 3	14:56:10	INSERT INTO Grades (grade, min_marks, max_marks, grade_point) VALUES ('A+', 90, 100, 10.0), ('A', 80, 89, 9.0), ('B+', ...	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0

3. ALL ABOUT THE SELECT QUERIES AND SOME OPERATIONS:
1. SELECT Queries & Views next. (Student performance par Subject)
Purpose:

This query is used to **analyze individual student performance** in each course they are enrolled in.

What the Query Does:

- Retrieves **student name**, **course name**, **marks obtained**, **grade**, **exam type**, and **exam date**.
- Joins the `Results`, `Students`, and `Courses` tables to gather complete details.
- Shows results **per subject per student**, useful for performance tracking and reports.

Tables Used:

- Results – stores exam results (marks, grade, type, date).
- Students – student basic details.
- Courses – subject/course details.

2. Course-Wise Marks:

Purpose:

This query helps analyze **how students have performed in each course** by calculating:

- Average marks
- Highest marks
- Lowest marks

What the Query Does:

It performs an aggregation using:

- AVG() to find the average marks,
- MIN() to find the lowest marks,
- MAX() to find the highest marks
  Grouped by each course to produce summarized performance statistics.

Tables Used:

- results – contains student marks
- courses – contains course details (course name, ID, etc.)

```
15 •   SELECT
16         c.course_name,
17             AVG(r.marks_obained) AS avg_marks,
18             MIN(r.marks_obained) AS lowest_marks,
19             MAX(r.marks_obained) AS highest_marks
20         FROM results r
21         JOIN courses c ON r.Course_id = c.Course_id
22         GROUP BY c.course_name;
```

course_name	avg_marks	lowest_marks	highest_marks
Mathematics	70.0000	55	85
Computer Programming	78.0000	78	78
Database Systems	90.4286	76	98

Result 2 ✕

Output

Action Output

#	Time	Action	Message
✓ 1	16:49:33	SELECT s.name AS Student_Name, c.Course_name, r.marks_obained, r.grade, r.exam_type, r.exam_date FR...	10 row(s) returned
✓ 2	16:53:58	SELECT c.course_name, AVG(r.marks_obained) AS avg_marks, MIN(r.marks_obained) AS lowest_marks, MAX(r.marks_o...	3 row(s) returned

3. Filtered Results by Exam Type:
Purpose:

To retrieve student performance details filtered by a specific exam type (e.g., **Midterm**, **Final**). This helps in analysing marks based on different exams.
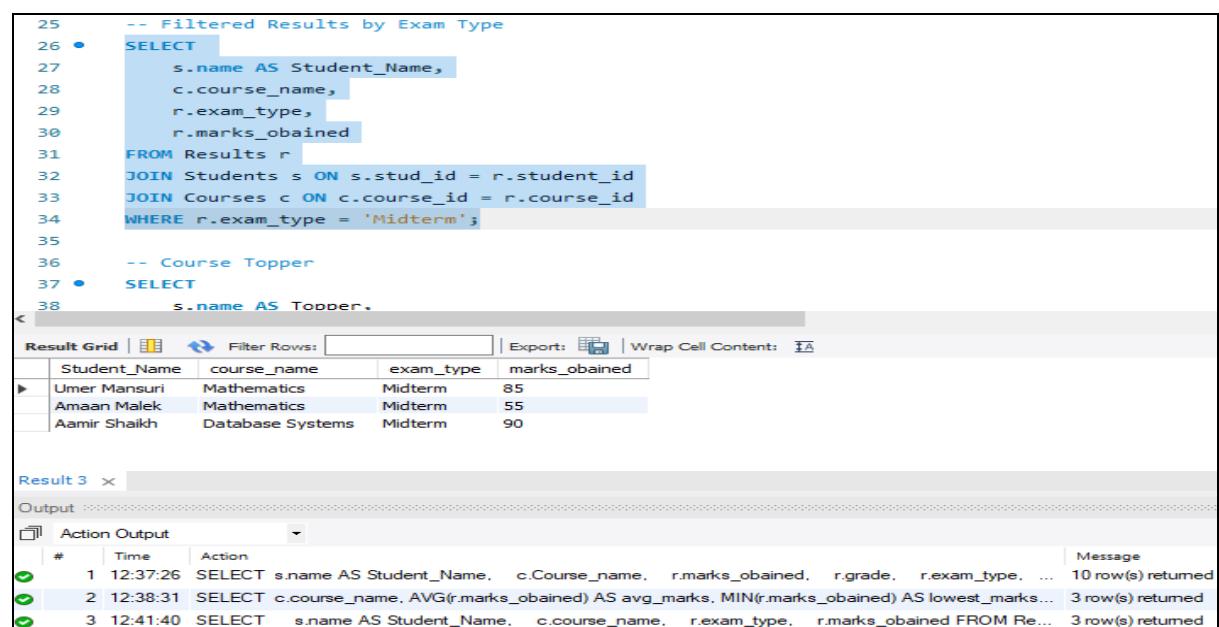
What the Query Does:

- Selects:
  - Student Name
  - Course Name
  - Exam Type
  - Marks Obtained
- Filters records **only** where the exam type = 'Midterm'.
- Uses **JOIN** between:
  - Results (main exam data),
  - Students (to get student name),
  - Courses (to get course name).

Tables Used:

Table	Purpose
Results	Stores exam data, marks, exam type
Students	Provides student details (name, ID)
Courses	Provides course names (linked via ID)

```
25      -- Filtered Results by Exam Type
26  •   SELECT
27          s.name AS Student_Name,
28          c.course_name,
29          r.exam_type,
30          r.marks_obained
31      FROM Results r
32      JOIN Students s ON s.stud_id = r.student_id
33      JOIN Courses c ON c.course_id = r.course_id
34      WHERE r.exam_type = 'Midterm';
35
36      -- Course Topper
37  •   SELECT
38          s.name AS Topper,
```

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

Student_Name	course_name	exam_type	marks_obained
Umer Mansuri	Mathematics	Midterm	85
Amaan Malek	Mathematics	Midterm	55
Aamir Shaikh	Database Systems	Midterm	90

Result 3 ×

Output

Action Output

#	Time	Action	Message
1	12:37:26	SELECT s.name AS Student_Name, c.Course_name, r.marks_obained, r.grade, r.exam_type, ...	10 row(s) returned
2	12:38:31	SELECT c.course_name, AVG(r.marks_obained) AS avg_marks, MIN(r.marks_obained) AS lowest_marks...	3 row(s) returned
3	12:41:40	SELECT s.name AS Student_Name, c.course_name, r.exam_type, r.marks_obained FROM Re...	3 row(s) returned

4. Course Topper:
Purpose:

To identify the top-performing student(s) in each course based on the **highest marks obtained**.

What the Query Does:

- Joins the Results, Students, and Courses tables.
- Uses a sub query to get the **maximum marks per course**.
- Filters the main results to show only the student(s) who scored those highest marks for each course.
- Returns the name of the student, the course name, and their marks.

Tables Used:

Table Name	Description
Results	Stores marks, grades, exam types, etc.
Students	Contains student personal data
Courses	Details of courses (name, credits, etc.)

```
36      -- Course Topper
37  •   SELECT
38          s.name AS Topper,
39          c.course_name,
40          r.marks_obained
41      FROM Results r
42      JOIN Students s ON s.stud_id = r.student_id
43      JOIN Courses c ON c.course_id = r.course_id
44      WHERE r.marks_obained = (
45          SELECT MAX(marks_obained)
46          FROM Results r2
47          WHERE r2.course_id = r.course_id
48      );
49
50      -- Students Who Failed (Marks < 40)
```

Topper	course_name	marks_obained
Umer Mansuri	Mathematics	85
Umer Mansuri	Computer Programming	78
Tufail Shaikh	Database Systems	98

Result 4 ✕

Output

#	Time	Action	Message
2	12:38:31	SELECT c.course_name, AVG(r.marks_obained) AS avg_marks, MIN(r.marks_obained) AS lowest_mar...	3 row(s) returned
3	12:41:40	SELECT s.name AS Student_Name, c.course_name, r.exam_type, r.marks_obained FROM R...	3 row(s) returned
4	12:43:44	SELECT s.name AS Topper, c.course_name, r.marks_obained FROM Results r JOIN Students ...	3 row(s) returned

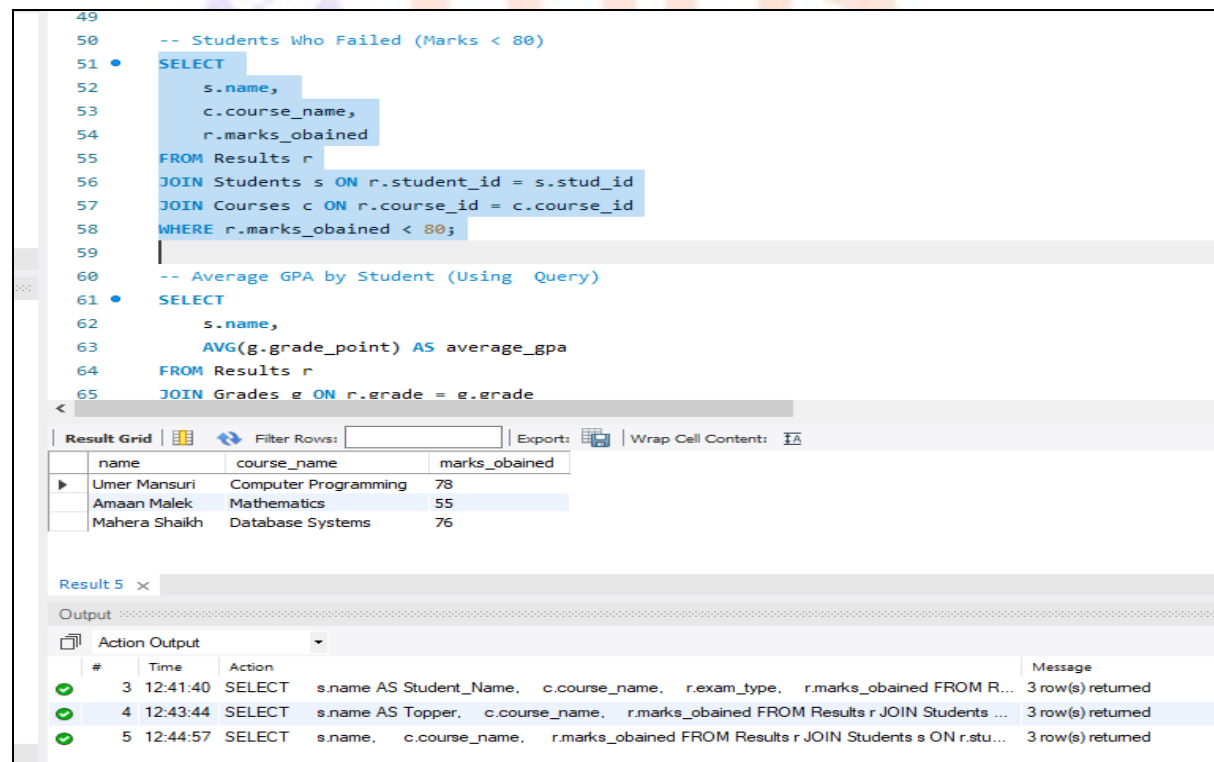5. Students Who Failed (Marks < 80):

Purpose:

This query is used to identify all students who scored less than 80 marks in any subject. It helps in generating a fail list or tracking students who are underperforming and may need academic support or intervention.

What the Query Does:

- Joins the `Results` table with the `Students` and `Courses` tables to fetch student names and course names.
- Filters the dataset using a `WHERE` clause to only show records where `marks_obtained < 80`.
- Returns a list of:
  - Student names
  - Course names
  - Marks obtained (less than 80)

Tables Used:

1. **Students**: Contains student details (e.g., `Stud_ID`, `name`)
2. **Results**: Contains marks and exam results (`Student_id`, `Course_id`, `marks_obtained`)
3. **Courses**: Contains course details (`Course_id`, `course_name`)

```
49
50        -- Students Who Failed (Marks < 80)
51 •      SELECT
52            s.name,
53            c.course_name,
54            r.marks_obained
55        FROM Results r
56        JOIN Students s ON r.student_id = s.stud_id
57        JOIN Courses c ON r.course_id = c.course_id
58        WHERE r.marks_obained < 80;
59    |
60        -- Average GPA by Student (Using  Query)
61 •      SELECT
62            s.name,
63            AVG(g.grade_point) AS average_gpa
64        FROM Results r
65        JOIN Grades g ON r.grade = g.grade
```

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

name	course_name	marks_obained
Umer Mansuri	Computer Programming	78
Amaan Malek	Mathematics	55
Mahera Shaikh	Database Systems	76

Result 5 ×

Output

Action Output

#	Time	Action		Message
✓	3 12:41:40	SELECT	s.name AS Student_Name, c.course_name, r.exam_type, r.marks_obained FROM R...	3 row(s) returned
✓	4 12:43:44	SELECT	s.name AS Topper, c.course_name, r.marks_obained FROM Results r JOIN Students ...	3 row(s) returned
✓	5 12:44:57	SELECT	s.name, c.course_name, r.marks_obained FROM Results r JOIN Students s ON r.stu...	3 row(s) returned

6. Average GPA by Student (Using Query):

Purpose:

To calculate the **average GPA** of each student based on their grades in various subjects. This helps track the overall academic performance of a student.

What the Query Does:

- It joins the `Results`, `Grades`, and `Students` tables.
- Converts grades like 'A+', 'A', etc. into their respective **grade points** using the `Grades` table.
- Averages the grade points for each student using the `AVG()` function.
- Groups the results by student name to return one GPA per student.

  Tables Used:

- **Students** – to fetch student names.
- **Results** – to access the student's grades.
- **Grades** – to map grades to grade points.

7. Create Views:

1. Create View: Students with Distinction:

Purpose:

To create a view that shows students who scored **75 or more marks** in any subject — i.e., students achieving distinction.

What the Query Does:

- Joins the `Results` table with the `Students` table based on `student_id`.
- Filters students whose `marks_obtained >= 75`.
- Displays their `name`, `marks_obtained`, and `grade`.
- Creates a virtual table (view) named `view_distinction_students`.

Tables Used:

- `Students`: To fetch student names.
- `Results`: To fetch their marks and grades.

```
69     -- Create Views
70     -- A.Create View: Students with Distinction
71  •  CREATE OR REPLACE VIEW view_distinction_students AS
72     SELECT
73         s.name,
74         r.marks_obained,
75         r.grade
76     FROM Results r
77     JOIN Students s ON s.stud_id = r.student_id
78     WHERE r.marks_obained >= 75;
79
80  •  SELECT * FROM view_distinction_students;
81
```

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

name	marks_obained	grade
Umer Mansuri	85	A
Umer Mansuri	78	B+
Mahera Shaikh	76	B+
Aamir Shaikh	90	A
Assad Shaikh	95	A+
Assad Shaikh	92	A+
Fizvan Savaid	07	A+

view_distinction_students 7 ✕

Output

Action Output ▾

#	Time	Action			Message
✓ 1	12:48:14	CREATE OR REPLACE VIEW view_distinction_students AS SELECT	s.name,	r.marks_obained, ...	0 row(s) affected
✓ 2	12:48:20	SELECT * FROM view_distinction_students			9 row(s) returned

2. View: view_student_results:

Purpose:

To **display consolidated student performance** by combining data from multiple tables such as students, results, and courses. This view simplifies querying individual student scores, grades, and subjects.
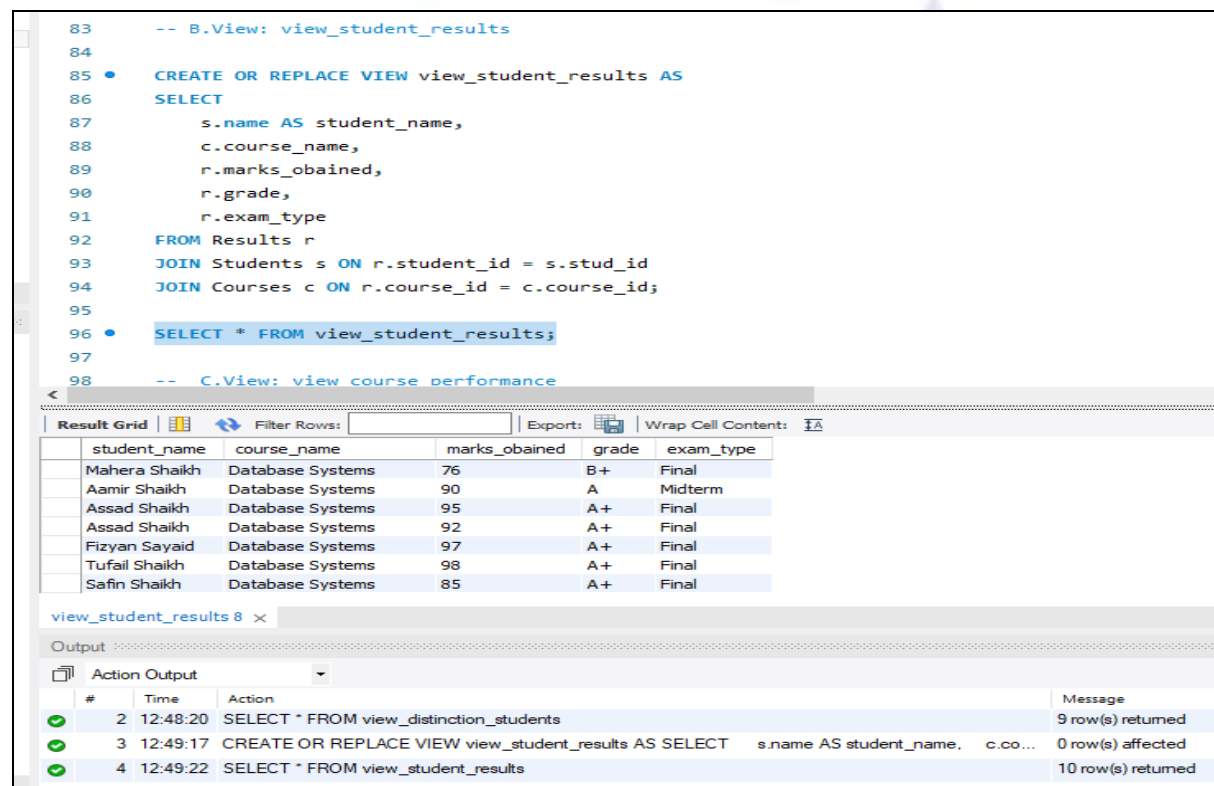
What the Query Does:

- Joins the Students, Results, and Courses tables.
- Retrieves:
  - Student name
  - Course name
  - Marks obtained
  - Grade
  - Exam type
- Saves this result as a **view**, making it reusable in other reports or queries.

Tables Used:

- Students
- Courses
- Results

```
83        -- B.View: view_student_results
84
85 •    CREATE OR REPLACE VIEW view_student_results AS
86      SELECT
87          s.name AS student_name,
88          c.course_name,
89          r.marks_obained,
90          r.grade,
91          r.exam_type
92      FROM Results r
93      JOIN Students s ON r.student_id = s.stud_id
94      JOIN Courses c ON r.course_id = c.course_id;
95
96 •    SELECT * FROM view_student_results;
97
98        -- C.View: view course performance
```

student_name	course_name	marks_obained	grade	exam_type
Mahera Shaikh	Database Systems	76	B+	Final
Aamir Shaikh	Database Systems	90	A	Midterm
Assad Shaikh	Database Systems	95	A+	Final
Assad Shaikh	Database Systems	92	A+	Final
Fizyan Sayaid	Database Systems	97	A+	Final
Tufail Shaikh	Database Systems	98	A+	Final
Safin Shaikh	Database Systems	85	A+	Final

view_student_results 8 ×

Output

Action Output

#	Time	Action	Message
● 2	12:48:20	SELECT * FROM view_distinction_students	9 row(s) returned
● 3	12:49:17	CREATE OR REPLACE VIEW view_student_results AS SELECT s.name AS student_name, c.co...	0 row(s) affected
● 4	12:49:22	SELECT * FROM view_student_results	10 row(s) returned

3. View: view_course_performance:

Purpose:

To summarize how each course is performing based on how many students attempted it and what the average score is.

What the Query Does:

- Displays each **course name**.
- Calculates:
    - Total number of **students who attempted** the course.
    - **Average marks** obtained in that course.
- Aggregates results **per course** using GROUP BY.

Tables Used:

- Courses – for the course name.
- Results – for marks and counting student attempts.

```
98       --  C.View: view_course_performance
99
100 •    CREATE OR REPLACE VIEW view_course_performance AS
101      SELECT
102          c.course_name,
103          COUNT(r.result_id) AS total_attempts,
104          AVG(r.marks_obained) AS average_marks
105      FROM Results r
106      JOIN Courses c ON r.course_id = c.course_id
107      GROUP BY c.course_name;
108
109 •    SELECT * FROM view_course_performance;
110
```

course_name	total_attempts	average_marks
Mathematics	2	70.0000
Computer Programming	1	78.0000
Database Systems	7	90.4286

view_course_performance 9 ×

Output

Action Output

#	Time	Action			Message
✓	4 12:49:22	SELECT * FROM view_student_results			10 row(s) returned
✓	5 12:50:35	CREATE OR REPLACE VIEW view_course_performance AS SELECT	c.course_name,	COUNT(r....	0 row(s) affected
✓	6 12:50:40	SELECT * FROM view_course_performance			3 row(s) returned

8. Stored Procedure to Calculate Grades and GPA:

Purpose:

This stored procedure is designed to **automatically assign grades and calculate GPA** for each student based on their marks in the Results table. It eliminates the need for manual grade entry and GPA computation by mapping marks to grade ranges and assigning corresponding grade points.

What the Query Does:

1. Iterates through each student's marks in the Results table.
2. Determines the corresponding grade using predefined grade ranges from the Grades table.
3. Assigns the grade into the Results table.
4. Calculates GPA based on grade points using the Grades mapping.
5. Inserts or updates the GPA into the GPA table per student and semester.

Tables Used:

Table Name	Purpose
Results	Stores marks for each student's course, which are used for grade mapping.
Grades	Contains mapping of mark ranges to grades and their grade points.
GPA	Stores final GPA and CGPA per student per semester.
Students	Reference for student details (used in JOINs or updates if required).

SQL Logic Used:

- CASE conditions to assign grades.
- AVG() aggregation to compute GPA.
- UPDATE or INSERT statements to reflect GPA in the GPA table.
- Optionally includes JOIN between Results and Grades to fetch grade points.

Student Result System ✕

File   Edit   View   Query   Database   Server   Tools   Scripting   Help

Navigator:

SCHEMAS

Filter objects

- ▶ ecommerce
- ▶ sakila
- ▼ student_result_pro...
  - ▼ Tables
    - ▶ course...
    - ▶ gpa
    - ▶ grades
    - ▶ results
    - ▶ semester
    - ▶ studentcourse
    - ▶ students
  - ▶ Views
  - Stored Procedures
  - Functions
- ▶ sys

Administration   Schemas

Information

No object selected

Table_Schema   data   queries_day2   queries_day3* ✕   queries_day4

Don't Limit

```sql
1    -- Stored Procedure to Calculate Grades and GPA
2    DELIMITER //
3
4 ●  CREATE PROCEDURE Calculate_Grades()
5    BEGIN
6        UPDATE results
7        SET grades = (
8            CASE
9                WHEN marks_obained >=90 THEN 'A+'
10               WHEN marks_obained >=80 THEN 'A'
11               WHEN marks_obained >=70 THEN 'B+'
12               WHEN marks_obained >=60 THEN 'B'
13               WHEN marks_obained >=50 THEN 'C'
14               WHEN marks_obained >=40 THEN
15               ELSE 'F'
16           END
17       );
18   END //
19
20   DELIMITER ;
21
22
23
24 ● -- GPA Logic using CASE
25   CREATE VIEW GPA_View AS
26   SELECT
```

Output

Action Output

#	Time	Action	Message
✓ 1	13:58:36	CREATE PROCEDURE Calculate_Grades() BEGIN UPDATE results SET grades = (CASE WHEN marks_obained >=9...	0 row(s) affected

---

```sql
21
22 ● SHOW CREATE PROCEDURE Calculate_Grades;
23
24
25
```

Result Grid | 📊 Filter Rows: | Export: 📊 | Wrap Cell Content: 🅣Ａ

Procedure	sql_mode	Create Procedure	character_set_client	collation_...
▶ Calculate_Grades	ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLE...	CREATE DEFINER=`root`@`localhost` PROCE...	utf8mb4	utf8mb4_0

Result 1 ✕

Output

Action Output

#	Time	Action	Message
✓ 1	13:58:36	CREATE PROCEDURE Calculate_Grades() BEGIN UPDATE results SET grades = (CASE WHEN marks_obained >=9...	0 row(s) affected
✓ 2	14:00:49	SHOW CREATE PROCEDURE Calculate_Grades	1 row(s) returned

10. GPA Logic using CASE:

Purpose:

To compute GPA for each student based on their grades by assigning numeric grade points using CASE. This helps in converting letter grades (e.g., A+, B, etc.) to numerical GPA values for further analysis like ranking.

What the Query Does:

- Uses a CASE statement inside SELECT to assign grade points:
  - 'A+' → 10, 'A' → 9, 'B+' → 8, etc.
- Averages these grade points per student using AVG(...).
- Groups results by student to get individual GPA.
- Uses ROUND() for formatting the GPA to 2 decimal places.

Tables Used:

- Students: for fetching student names.
- Results: contains marks and grades.

10. Create result_summary View:

Purpose:

To create a consolidated view showing each student's performance in all enrolled subjects, including the subject name, marks, grade, and exam type. This view simplifies reporting and analysis by providing a unified result set for external queries or display.

What the Query Does:

- Joins the Students, Courses, and Results tables
- Retrieves student name, course name, marks obtained, grade, exam type, and exam date
- Displays a detailed report of each subject-wise result per student
- It is used for reporting or performance dashboards

Tables Used:

Table Name	Purpose
Students	Provides the Stud_ID and name of each student
Courses	Provides Course_id and course_name
Results	Stores marks, grade, and exam data linked via foreign keys

11. GPA Rankings Using RANK():

Description of the Query:

This query calculates the **GPA (Grade Point Average)** for each student based on letter grades and ranks them in descending order of GPA using the SQL RANK() window function.

Purpose of the Query:
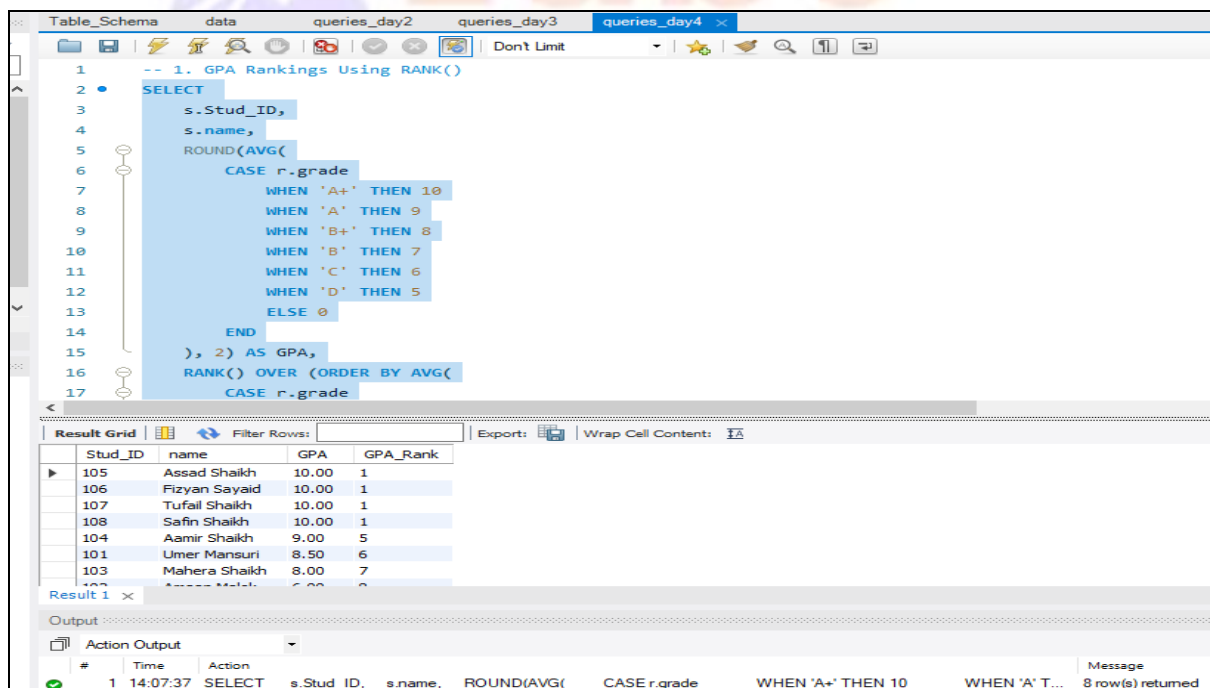
- To compute **GPA scores** from letter grades.
- To generate a **ranking of students** based on their GPA.
- To help in identifying **top-performing students**.
- Useful for academic reporting, awards, scholarships, and performance analysis.

What the Query Does:

1. **Joins** the Students table with the Results table using student IDs.
2. **Converts letter grades to numeric grade points** using a CASE statement.
3. **Calculates the average grade point (GPA)** for each student using AVG().
4. **Rounds the GPA** to 2 decimal places with ROUND().
5. **Ranks students by GPA** in descending order using the RANK() window function.
6. **Groups** results by student ID and name to calculate GPA per student.

Tables Used:

- Students: Contains student data (e.g., Stud_ID, name).
- Results: Contains course results for students (Student_id, grade).

12. Top 3 Students View:

Purpose:

This SQL query creates a **view** named view_top_3_students that shows the **top 3 students based on their GPA**, calculated from their grades. It's useful for displaying academic performance rankings in educational systems.

What the Query Does:

1. **Calculates GPA** for each student based on their grades using a weighted average system where:
   - 'A+' = 10
   - 'A' = 9
   - 'B+' = 8
   - Any other grade = 0
2. **Ranks students** using the RANK() window function, ordered by GPA in **descending order**.
3. **Filters** only the top 3 students using WHERE rank_position <= 3.
4. **Wraps everything** in a view so it can be reused like a table in other queries without writing this logic again.

Tables Used:

1. **Students Table (s):** Fields used: Stud_ID, name,Holds basic student information.
2. **Results Table (r):** Fields used: Student_id, grade,Contains student grades for subjects or exams.

13. Failed Students Report: (NO number of Fail Students)

14. Subject Topper View:

Conclusion

The **Student Result System** project successfully demonstrates the real-world application of relational database design and SQL programming. Throughout this project, we designed and implemented a normalized database structure that efficiently manages student records, course enrollments, marks, grades, and GPA calculations.

Key features such as stored procedures, views, and advanced SQL queries like JOIN, GROUP BY, and RANK() functions were used to automate core academic operations. This not only ensured data integrity but also optimized reporting and performance analysis.

Additionally, the use of ER diagrams and well-structured schemas helped in understanding the relationships among entities such as students, courses, grades, and exam results. Realistic dummy data further validated the functionality and helped simulate real-time use cases.

Overall, this project has enhanced practical skills in database development using MySQL Workbench and strengthened understanding of business logic, automation, and reporting in SQL-driven systems.