# A. Title Page

Lewis University  
CPSC 50900: Database Systems   
Spring 2025 Term Project

Student Performance Tracking System

naseerahmad@lewisu.edu

Work products stored in the Github repository <https://github.com/ahmadnaseer143/Student-Tracking-Database-Course-Project>

Table of Contents

[A. Title Page 1](#_Toc190509060)

[Schedule of Milestones 2](#_Toc190509061)

[B. Initial Proposal 2](#_Toc190509062)

[C. Data Sources 3](#_Toc190509063)

[D. Alternative Ways to Store the Data 5](#_Toc190509064)

[E. Conceptual and Logical Models 5](#_Toc190509065)

[F. Physical Model 11](#_Toc190509066)

[G. Populate the database with data 17](#_Toc190509078)

[H. Data Manipulation Language (DML) Scripts 22](#_Toc190509079)

[I. Indexes 23](#_Toc190509080)

[J. Views 23](#_Toc190509081)

[K. Stored Programs (Stored Procedures, Stored Functions, Triggers) 23](#_Toc190509082)

[L. Transactions 23](#_Toc190509083)

[M. Database Security 24](#_Toc190509084)

[N. Locking and Concurrent Access 24](#_Toc190509085)

[O. Backing Up Your Database 24](#_Toc190509086)

[P. Programming 25](#_Toc190509087)

[Q. Suggested Future Work 25](#_Toc190509088)

[R. Activity Log 25](#_Toc190509089)

# Schedule of Milestones

Here is a schedule that shows when each milestone is due and what sections comprise it.

|  |  |
| --- | --- |
| Deadline | Sections for which you must demonstrate significant progress |
| February 4 at 11:59pm | a. Title page  b. Initial proposal  c. Data sources  d. Alternative ways to store the data  r. Activity Log – at least six entries covering the first two weeks |
| February 18  at 11:59pm | e. Conceptual and logical models  f. Physical model  g. Populate the database with data  r. Activity Log – at least six entries covering the past two weeks |
| March 4 at 11:59pm | h. Data manipulation language (DML) scripts  i. Indexes  j. Views  l. Transactions  m. Security  r. Activity Log – at least six entries covering the past two weeks |

The remaining sections – Triggers, Locking and Concurrency, Backup, and Programming, will be turned in with the final report, which is due March 16 at 11:59pm.

# B. Initial Proposal

I will describe my proposal by answering the following questions:

**1. What data will we store?**

Our database will store:

* **Students** (Name, ID, age, grade level, contact info)
* **Courses** (Course name, code, instructor, credits)
* **Grades** (Student ID, course code, assignments, grades, GPA)
* **Attendance** (Student ID, course, attendance percentage)
* **Teachers** (Name, employee ID, contact info, assigned courses)
* **Assignments** (Course ID, assignment title, due date, max score)
* **Departments** (Name, head, courses offered)
* **Parents** (Name, contact info, relation to student)

**2. Why are we interested in this data?**

Tracking student performance is crucial for improving education outcomes and providing personalized feedback.

**3. Why is it important?**

This system will help schools and educators identify struggling students, measure success, and allocate resources effectively.

**4. Where will the data come from?**

We will generate simulated student records, use mock data for courses and assignments, and anonymize data from public datasets if available.

**5. Who will use this data?**

* Teachers monitor and analyze performance trends.
* School administrators make informed decisions.
* Parents stay informed about their child’s progress.

**6. What kind of application do we plan to build?**

A **web-based platform** for schools, teachers, and parents to view and manage student academic data, attendance, and reports.

# C. Data Sources

Our student performance tracking system utilizes structured data from various sources, including CSV files, JSON files, and relational database tables. Below are the details of these data sources, including field structures and data organization.

**Data Formats & Sources:**

* **CSV Files:** Used for structured, tabular data such as student records, course details, and attendance.
* **JSON Files:** Used for hierarchical, semi-structured data like grades and assignment submissions.
* **SQL Database:** Stores relational data for optimized querying and indexing.

**Detailed Data Structure:**

1. **Students.csv** (Stores student demographic and academic details)
   * Student\_ID (Integer, Primary Key)
   * Name (String)
   * Age (Integer)
   * Grade\_Level (String)
   * Parent\_Contact\_Info (String)
2. **Courses.csv** (Lists courses offered)
   * Course\_Code (String, Primary Key)
   * Course\_Name (String)
   * Credits (Integer)
   * Instructor (String)
3. **Grades.json** (Tracks student performance in assignments)
   * Student\_ID (Integer, Foreign Key referencing Students)
   * Course\_ID (String, Foreign Key referencing Courses)
   * Assignment\_Title (String)
   * Score (Integer)
4. **Attendance.csv** (Stores attendance records)
   * Student\_ID (Integer, Foreign Key referencing Students)
   * Course\_Code (String, Foreign Key referencing Courses)
   * Attendance\_Percentage (Float)
5. **Assignments.csv** (Details on assignments)
   * Assignment\_ID (Integer, Primary Key)
   * Course\_ID (String, Foreign Key referencing Courses)
   * Title (String)
   * Due\_Date (Date)
   * Max\_Score (Integer)
6. **Teachers.csv** (Information on teachers)
   * Employee\_ID (Integer, Primary Key)
   * Name (String)
   * Contact\_Info (String)
   * Assigned\_Courses (String - List of course codes)

**Fields Selected for the Database:**

Based on these sources, the following fields will be included in our final SQL database:

* **Students Table:** Student\_ID, Name, Age, Grade\_Level, Parent\_Contact\_Info
* **Courses Table:** Course\_Code, Course\_Name, Credits, Instructor
* **Grades Table:** Student\_ID, Course\_ID, Assignment\_Title, Score
* **Attendance Table:** Student\_ID, Course\_Code, Attendance\_Percentage
* **Assignments Table:** Assignment\_ID, Course\_ID, Title, Due\_Date, Max\_Score
* **Teachers Table:** Employee\_ID, Name, Contact\_Info

These structured datasets will ensure accurate student performance tracking while allowing for efficient queries and reports.

# D. Alternative Ways to Store the Data

**1. NoSQL (Document Store – MongoDB)**

* **Implementation:** Store each student’s performance and attendance data as JSON documents.
* **Advantages:** Flexible schema, fast access for large datasets.
* **Disadvantages:** Less optimized for structured queries and relationships.

**2. Graph Database (Neo4j)**

* **Implementation:** Represent students, courses, teachers, and grades as nodes, with relationships as edges.
* **Advantages:** Best for tracking relationships (e.g., student-course-teacher connections).
* **Disadvantages:** More complex to implement for tabular queries.

# E. Conceptual and Logical Models

**Conceptual Model**

The **conceptual model** defines the main entities and their relationships in the **Student Performance Tracking System**. Below are the identified entities and their **connectivity and participation constraints**:

1. Students enroll in multiple Courses (M:N, mandatory for Students, optional for Courses).
   * Courses have multiple enrolled Students (M:N, optional for Courses, mandatory for Students).
2. Courses are taught by Teachers (1:M, mandatory for Courses, optional for Teachers).
   * Teachers teach multiple Courses (1:M, optional for Teachers, mandatory for Courses).
3. Students receive Grades for Assignments in Courses (1:M, mandatory).
   * Grades are assigned to Students for Assignments in Courses (M:1, mandatory).
4. Students have Attendance records for each Course (1:M, mandatory).
   * Attendance records exist for Students in Courses (M:1, mandatory).
5. Courses have multiple Assignments (1:M, mandatory).
   * Assignments belong to a single Course (M:1, mandatory).
6. Departments oversee multiple Courses (1:M, mandatory).
   * Courses are managed by one Department (M:1, mandatory).
7. Parents serve as contacts for multiple Students (1:M, mandatory for Parents, optional for Students).

* Each Student has one Parent who is the primary contact (M:1, optional for Students, mandatory for Parents).

**Entity-Relationship (ER) Diagram**

A diagram of a student

AI-generated content may be incorrect.

**Logical Model**

The logical model expands on the conceptual model by specifying attributes, primary keys, and relationships.

**Entities and Attributes**

* 1. Students
* Student\_ID (PK)
* Name
* Age
* Grade\_Level
* Parent\_ID (FK →Parents, Parent\_ID)
  1. Courses
* Course\_Code (PK)
* Course\_Name
* Credits
* Instructor\_ID (FK -> Teachers,Employee\_ID)
  1. Teachers
* **Employee\_ID** (PK)
* Name
* Contact\_Info
  1. Grades
* **Grade\_ID** (PK)
* Student\_ID (FK → Students.Student\_ID)
* Course\_Code (FK → Courses.Course\_Code)
* Assignment\_Title
* Score
  1. Attendance
* **Attendance\_ID** (PK)
* Student\_ID (FK → Students.Student\_ID)
* Course\_Code (FK → Courses.Course\_Code)
* Attendance\_Percentage
  1. Assignments
* Assignment\_ID *(PK)*
* Course\_Code *(FK → Courses.Course\_Code)*
* Title
* Due\_Date
* Max\_Score
  1. Departments
* Department\_ID *(PK)*
* Name
* Head
  1. Parents
* **Parent\_ID** (PK)
* Name
* Contact\_Info
  1. Student\_Course\_Enrollment (Resolving M:N Relationship)
* **Enrollment\_ID** (PK)
* Student\_ID (FK → Students.Student\_ID)
* Course\_Code (FK → Courses.Course\_Code)
* Enrollment\_Date

**Functional Dependencies:**

1. Student\_ID → Name, Age, Grade\_Level, Parent\_ID
2. Course\_Code → Course\_Name, Credits, Instructor
3. Employee\_ID → Name, Contact\_Info
4. Student\_ID, Course\_ID, Assignment\_Title → Score
5. Student\_ID, Course\_Code → Attendance\_Percentage
6. Assignment\_ID → Course\_ID, Title, Due\_Date, Max\_Score
7. Department\_ID → Name, Head
8. Parent\_ID → Name, Contact\_Info
9. Enrollment\_ID → Student\_ID, Course\_Code, Enrollment\_Date

**Normalization Steps:**

1. First Normal Form (1NF) - Ensured that all attributes are atomic (no multi-valued or repeating groups)
2. Second Normal Form (2NF) - Removed partial dependencies by ensuring all attributes depend on the whole primary key (particularly in composite keys).
3. Third Normal Form (3NF) - Eliminated transitive dependencies by splitting related data into separate entities.

Let us see some examples:

**1st Normal Form (1NF) - Eliminate Repeating Groups**

**Problem:**  
A Students table might store multiple courses in a single row:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student\_ID** | **Name** | **Age** | **Courses\_Enrolled** | **Parent\_Contact\_Info** |
| 101 | Alice | 20 | Math, Science | alice\_parent@email.com |
| 102 | Bob | 19 | English, History | bob\_parent@email.com |

**Issues:**

* Multivalued attribute (Courses\_Enrolled contains multiple values in a single column).
* Difficult to query (e.g., finding all students enrolled in "Math" is challenging).

**Solution (1NF Applied):**  
Create a separate Student\_Course\_Enrollment table to store course enrollments as separate rows.

|  |  |  |  |
| --- | --- | --- | --- |
| **Enrollment\_ID** | **Student\_ID** | **Course\_Code** | **Enrollment\_Date** |
| 1 | 101 | MATH101 | 2024-01-10 |
| 2 | 101 | SCI102 | 2024-01-10 |
| 3 | 102 | ENG103 | 2024-01-12 |
| 4 | 102 | HIST104 | 2024-01-12 |

Now, each piece of data is atomic, and no columns contain multiple values.

**2nd Normal Form (2NF) - Remove Partial Dependencies**

**Problem:**  
A table with a composite primary key (like Student\_ID, Course\_Code) may have attributes that depend on only part of the key instead of the whole key.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student\_ID** | **Course\_Code** | **Course\_Name** | **Credits** | **Enrollment\_Date** |
| 101 | MATH101 | Mathematics | 3 | 2024-01-10 |
| 101 | SCI102 | Science | 4 | 2024-01-10 |
| 102 | ENG103 | English | 3 | 2024-01-12 |

**Issue:**

* Course\_Name and Credits only depend on Course\_Code, not Student\_ID.
* This is a partial dependency (i.e., some attributes do not fully depend on the entire composite key).

**Solution (2NF Applied):**

* Separate Courses into its own table so Course\_Name and Credits depend only on Course\_Code.
* Keep Student\_Course\_Enrollment only for enrollment records.

**Courses Table:**

|  |  |  |
| --- | --- | --- |
| **Course\_Code** | **Course\_Name** | **Credits** |
| MATH101 | Mathematics | 3 |
| SCI102 | Science | 4 |
| ENG103 | English | 3 |

**Student\_Course\_Enrollment Table (2NF-compliant):**

|  |  |  |  |
| --- | --- | --- | --- |
| **Enrollment\_ID** | **Student\_ID** | **Course\_Code** | **Enrollment\_Date** |
| 1 | 101 | MATH101 | 2024-01-10 |
| 2 | 101 | SCI102 | 2024-01-10 |
| 3 | 102 | ENG103 | 2024-01-12 |

Now, each non-key attribute fully depends on the entire primary key of its respective table.

**3rd Normal Form (3NF) - Remove Transitive Dependencies**

**Problem:**  
A Grades table might contain an indirect (transitive) dependency where Assignment\_Title determines Max\_Score.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Grade\_ID | Student\_ID | Course\_Code | Assignment\_Title | Score | Max\_Score |
| 1 | 101 | MATH101 | Homework 1 | 85 | 100 |
| 2 | 101 | MATH101 | Quiz 1 | 90 | 50 |

**Issue:**

* Max\_Score depends on Assignment\_Title, not Grade\_ID.
* This is a transitive dependency because Grade\_ID → Assignment\_Title → Max\_Score.

**Solution (3NF Applied):**

* Create an Assignments table to store assignment-specific information separately.
* Keep Grades only for student scores.

**Assignments Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Assignment\_ID** | **Course\_Code** | **Title** | **Max\_Score** | **Due\_Date** |
| 1 | MATH101 | Homework 1 | 100 | 2024-02-01 |
| 2 | MATH101 | Quiz 1 | 50 | 2024-02-05 |

**Grades Table (3NF-compliant):**

|  |  |  |  |
| --- | --- | --- | --- |
| **Grade\_ID** | **Student\_ID** | **Assignment\_ID** | **Score** |
| 1 | 101 | 1 | 85 |
| 2 | 101 | 2 | 90 |

Now, Max\_Score depends only on Assignment\_ID, and we have removed transitive dependencies.

**Final Logical ERD Diagram**

A computer screen shot of a diagram

AI-generated content may be incorrect.

# F. Physical Model

**Final Physical Model ERD**

A diagram of a student course

AI-generated content may be incorrect.

**Database Schema and Table Definitions**

*-- Table: Students  
CREATE TABLE Students (  
 Student\_ID INT PRIMARY KEY AUTO\_INCREMENT,  
 Name VARCHAR(100) NOT NULL,  
 Age INT CHECK (Age >= 5 AND Age <= 25),  
 Grade\_Level VARCHAR(50) NOT NULL,  
 Parent\_ID INT,  
 CONSTRAINT Students\_Parents FOREIGN KEY (Parent\_ID) REFERENCES Parents(Parent\_ID)  
);  
  
-- Table: Parents  
CREATE TABLE Parents (  
 Parent\_ID INT PRIMARY KEY AUTO\_INCREMENT,  
 Name VARCHAR(100) NOT NULL,  
 Contact\_Info VARCHAR(255)  
);  
  
-- Table: Teachers  
CREATE TABLE Teachers (  
 Employee\_ID INT PRIMARY KEY AUTO\_INCREMENT,  
 Name VARCHAR(100) NOT NULL,  
 Contact\_Info VARCHAR(255)  
);  
  
-- Table: Departments  
CREATE TABLE Departments (  
 Department\_ID INT PRIMARY KEY AUTO\_INCREMENT,  
 Name VARCHAR(100) UNIQUE NOT NULL,  
 Head VARCHAR(100)  
);  
  
-- Table: Courses  
CREATE TABLE Courses (  
 Course\_Code VARCHAR(10) PRIMARY KEY,  
 Course\_Name VARCHAR(255) NOT NULL,  
 Credits INT CHECK (Credits > 0),  
 Teacher\_ID INT,  
 Department\_ID INT,  
 CONSTRAINT Courses\_Teachers FOREIGN KEY (Teacher\_ID) REFERENCES Teachers(Employee\_ID),  
 CONSTRAINT Courses\_Departments FOREIGN KEY (Department\_ID) REFERENCES Departments(Department\_ID)  
);  
  
-- Table: Assignments  
CREATE TABLE Assignments (  
 Assignment\_ID INT PRIMARY KEY AUTO\_INCREMENT,  
 Course\_Code VARCHAR(10) NOT NULL,  
 Title VARCHAR(255) NOT NULL,  
 Due\_Date DATE NOT NULL,  
 Max\_Score INT CHECK (Max\_Score > 0),  
 CONSTRAINT Assignments\_Courses FOREIGN KEY (Course\_Code) REFERENCES Courses(Course\_Code)  
);  
  
-- Table: Grades  
CREATE TABLE Grades (  
 Grade\_ID INT PRIMARY KEY AUTO\_INCREMENT,  
 Student\_ID INT NOT NULL,  
 Assignment\_ID INT NOT NULL,  
 Score DECIMAL(5,2) CHECK (Score BETWEEN 0 AND 100),  
 CONSTRAINT Grades\_Students FOREIGN KEY (Student\_ID) REFERENCES Students(Student\_ID),  
 CONSTRAINT Grades\_Assignments FOREIGN KEY (Assignment\_ID) REFERENCES Assignments(Assignment\_ID)  
);  
  
-- Table: Attendance  
CREATE TABLE Attendance (  
 Attendance\_ID INT PRIMARY KEY AUTO\_INCREMENT,  
 Student\_ID INT NOT NULL,  
 Course\_Code VARCHAR(10) NOT NULL,  
 Attendance\_Percentage DECIMAL(5,2) CHECK (Attendance\_Percentage BETWEEN 0 AND 100),  
 CONSTRAINT Attendance\_Students FOREIGN KEY (Student\_ID) REFERENCES Students(Student\_ID),  
 CONSTRAINT Attendance\_Courses FOREIGN KEY (Course\_Code) REFERENCES Courses(Course\_Code)  
);  
  
-- Table: Student\_Course\_Enrollment  
CREATE TABLE Student\_Course\_Enrollment (  
 Enrollment\_ID INT PRIMARY KEY AUTO\_INCREMENT,  
 Student\_ID INT NOT NULL,  
 Course\_Code VARCHAR(10) NOT NULL,  
 Enrollment\_Date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 CONSTRAINT Student\_Course\_Enrollment\_Students FOREIGN KEY (Student\_ID) REFERENCES Students(Student\_ID),  
 CONSTRAINT Student\_Course\_Enrollment\_Courses FOREIGN KEY (Course\_Code) REFERENCES Courses(Course\_Code)  
);*

## SQL Execution Proof

### Creating the Database:

CREATE DATABASE StudentPerformanceDB;

A screenshot of a computer program

AI-generated content may be incorrect.

USE StudentPerformanceDB;

A screenshot of a computer program

AI-generated content may be incorrect.

Running the SHOW DATABASES; command to verify database creation

A screenshot of a computer

AI-generated content may be incorrect.

Running the SHOW TABLES; command to confirm table creation

A screenshot of a computer

AI-generated content may be incorrect.

Running DESCRIBE on key tables to verify structures

DESC Students;

A screen shot of a computer program

AI-generated content may be incorrect.  
DESC Courses;

A black screen with white text

AI-generated content may be incorrect.  
DESC Grades;

A black screen with white text

AI-generated content may be incorrect.

# G. Populate the database with data

After designing and implementing the database schema in MySQL, the next step is to populate it with relevant data. This involved transforming raw datasets into SQL INSERT statements. Below is a step-by-step explanation of how the data was generated, cleaned, and inserted into the database.

**Step 1: Data Acquisition**

The data for this project was sourced and generated in the following ways:  
- Students, Courses, and Grades Data: Downloaded from Kaggle.  
- Data Cleaning: Processed using Python scripts to remove duplicates, handle missing values, and ensure consistency.  
- Other Data (Teachers, Assignments, Attendance, Parents, etc.): Randomly generated using Python scripts.

**Step 2: Data Cleaning and Transformation**

To clean and prepare the data:  
- Used Pandas to load and process CSV files.  
- Handled missing values by either imputing reasonable defaults or removing incomplete records.  
- Converted data types to match the MySQL schema.  
- Ensured referential integrity between tables (e.g., valid Student\_IDs in Grades matched those in Students).

**Step 3: Converting CSV Data to SQL INSERT Statements**

For each table, we used Python scripts to read CSV files and generate INSERT statements.

**Python Script to Convert CSV to SQL INSERT Statements**

For each table, the following general script structure was used:

import pandas as pd

# Load CSV file (update file name for different tables)

df = pd.read\_csv("Processed\_Students.csv")

# Generate SQL INSERT statements

table\_name = "Students"

insert\_statements = []

for \_, row in df.iterrows():

values = tuple(row)

query = f"INSERT INTO {table\_name} VALUES {values};"

insert\_statements.append(query)

# Save SQL queries to a file

with open(f"{table\_name}\_inserts.sql", "w") as file:

file.write("\n".join(insert\_statements))

print(f"SQL INSERT statements saved for {table\_name}!")

**Step 4: Running Sql Scripts to populate the database**

After generating the insert statements, they were executed using the MySQL **source** command. First, we logged into mysql and then selected the required database. After that we ran the source command.

mysql -u root -p

USE studentperformancedb;

**Using mysql source command**

SOURCE path\_to\_file/Students\_inserts.sql;

SOURCE path\_to\_file/Teachers\_inserts.sql;

SOURCE path\_to\_file/Courses\_inserts.sql;

SOURCE path\_to\_file/Grades\_inserts.sql;

SOURCE path\_to\_file/Attendance\_inserts.sql;

SOURCE path\_to\_file/Parents\_inserts.sql;

SOURCE path\_to\_file/Assignments\_inserts.sql;

SOURCE path\_to\_file/Student\_Course\_Enrollment\_inserts.sql;

**Step 5: Verifying Data insertion**

To confirm that the data was correctly inserted, SELECT queries were executed.

SELECT \* FROM Students LIMIT 5;

A black screen with white text

AI-generated content may be incorrect.

SELECT \* FROM Teachers LIMIT 5;

A screen shot of a computer

AI-generated content may be incorrect.

SELECT \* FROM Courses LIMIT 5;

A screen shot of a computer

AI-generated content may be incorrect.

SELECT \* FROM Grades LIMIT 5;

A computer screen with white text

AI-generated content may be incorrect.

SELECT \* FROM Attendance LIMIT 5;

A screen shot of a computer program

AI-generated content may be incorrect.

SELECT \* FROM Parents LIMIT 5;

A screen shot of a computer

AI-generated content may be incorrect.

SELECT \* FROM Assignments LIMIT 5;

A computer screen shot of a black screen

AI-generated content may be incorrect.

SELECT \* FROM Departments;

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \* FROM Student\_Course\_Enrollment LIMIT 5;

A screen shot of a computer

AI-generated content may be incorrect.

# H. Data Manipulation Language (DML) Scripts

1. Insert Statements

INSERT INTO Students (Student\_ID, Name, Age, Grade\_Level, Parent\_ID) VALUES (307, 'Naseer', 25, 'Senior', 101);

A computer screen shot of a black background

AI-generated content may be incorrect.

INSERT INTO Courses (Course\_Code, Course\_Name, Credits, Teachers\_ID, Department\_ID) VALUES ('AI101', 'Introduction to AI', 3, 2, 1);

A black screen with white text

AI-generated content may be incorrect.

1. Update Statements

update students set age=22 where student\_id=307;

update courses set course\_name='Intro to Ai2' where course\_code='Ai101';

A computer screen with white text

AI-generated content may be incorrect.

1. Delete Statement

DELETE FROM Students WHERE Student\_ID = 307;

A black screen with white text

AI-generated content may be incorrect.

1. Simple Select Statement

SELECT Name, Age, Grade\_Level FROM Students WHERE Age > 24;

A computer screen with white text

AI-generated content may be incorrect.

1. Join Statements

SELECT Students.Name, Courses.Course\_Name FROM Students JOIN Student\_Course\_Enrollment ON Students.Student\_ID = Student\_Course\_Enrollment.Student\_ID JOIN Courses ON Student\_Course\_Enrollment.Course\_Code = Courses.Course\_Code;

A computer screen shot of a black screen

AI-generated content may be incorrect.

SELECT Assignments.Title, Grades.Score FROM Assignments JOIN Grades ON Assignments.Assignment\_ID = Grades.Assignment\_ID;

A screen shot of a computer

AI-generated content may be incorrect.

1. Summary Functions

SELECT AVG(Score) AS Average\_Score FROM Grades;

A screen shot of a computer

AI-generated content may be incorrect.

SELECT COUNT(Student\_ID) AS Total\_Students FROM Students;

A computer screen shot of a black screen

AI-generated content may be incorrect.

1. Multi-Table Query

SELECT Students.Name, Courses.Course\_Name, Teachers.Name AS Teacher\_Name FROM Students JOIN Student\_Course\_Enrollment ON Students.Student\_ID = Student\_Course\_Enrollment.Student\_ID JOIN Courses ON Student\_Course\_Enrollment.Course\_Code = Courses.Course\_Code JOIN Teachers ON Courses.Teachers\_ID = Teachers.Employee\_ID;

A screenshot of a computer screen

AI-generated content may be incorrect.

1. Custom Query

SELECT Departments.Name AS Department\_Name, COUNT(Courses.Course\_Code) AS Number\_of\_Courses FROM Departments LEFT JOIN Courses ON Departments.Department\_ID = Courses.Department\_ID GROUP BY Departments.Name;

A computer screen with white text

AI-generated content may be incorrect.

# I. Indexes

Indexes are a critical component in database optimization, as it will enhance query performance by allowing faster data retrieval. Below are three indexes added to our database schema, along with explanations of why they were chosen.

**Index on Students (Student\_ID)**

Since Student\_ID is frequently used as a foreign key in multiple tables (e.g., Grades, Attendance, Student\_Course\_Enrollment), indexing it improves join performance and lookup efficiency.

SQL Command:

CREATE INDEX idx\_students\_student\_id ON Students(Student\_ID);

**Index on Courses (Course\_Code)**

The Course\_Code is heavily used in joins with the Assignments, Attendance, and Student\_Course\_Enrollment tables. Indexing it will speed up queries filtering by or joining on Course\_Code.

SQL Command:

CREATE INDEX idx\_courses\_course\_code ON Courses(Course\_Code);

**Index on Grades (Assignment\_ID, Student\_ID)**

Queries often filter by both Assignment\_ID and Student\_ID when retrieving grades. A composite index on these columns improves performance by allowing the database to efficiently locate relevant records.

SQL Command:

CREATE INDEX idx\_grades\_assignment\_student ON Grades(Assignment\_ID, Student\_ID);

**Performance Improvement**

To measure the impact of these indexes, we can follow these steps:

* **Run EXPLAIN ANALYZE Before and After**

We will execute queries without indexes and use EXPLAIN ANALYZE to review execution time.  
Apply indexes and run the same queries again, comparing the performance.

* **Measure Query Execution Time**

We will use a SQL profiling tool to log execution times before and after indexing.  
Queries that previously required full table scans should show reduced execution time.

* **Monitor Database Statistics**

We will use MySQL’s SHOW INDEX FROM table\_name to verify index usage.

# J. Views

**Database Views**

1. View 1: Student Grades Overview

SQL to Create the View:

CREATE VIEW Student\_Grades\_View AS

SELECT s.Student\_ID, s.Name AS Student\_Name, c.Course\_Name, a.Title AS Assignment\_Title, g.Score

FROM Students s

JOIN Grades g ON s.Student\_ID = g.Student\_ID

JOIN Assignments a ON g.Assignment\_ID = a.Assignment\_ID

JOIN Courses c ON a.Course\_Code = c.Course\_Code;

Why This View is Valuable:

- This view provides an easy way to retrieve student grades along with assignment and course details.

- Useful for teachers and administrators to analyze student performance efficiently.

- Simplifies complex joins when accessing student grade data.

1. View 2: Course Enrollment Summary

SQL to Create the View:

CREATE VIEW Course\_Enrollment\_View AS

SELECT c.Course\_Code, c.Course\_Name, COUNT(sce.Student\_ID) AS Total\_Students

FROM Courses c

JOIN Student\_Course\_Enrollment sce ON c.Course\_Code = sce.Course\_Code

GROUP BY c.Course\_Code, c.Course\_Name;

Why This View is Valuable:

- Provides a quick summary of student enrollment per course.

- Helps department heads and administrators track course popularity and student distribution.

- Reduces the need for complex aggregate queries when checking course enrollment numbers.

**Screenshots of Views:**

SELECT \* FROM Student\_Grades\_View;

A computer screen shot of a program

AI-generated content may be incorrect.

SELECT \* FROM Course\_Enrollment\_View;

A computer screen shot of a computer program

AI-generated content may be incorrect.

# K. Stored Programs (Stored Procedures, Stored Functions, Triggers)

*Description: Add a stored procedure, stored function or trigger to a table and demonstrate using it.*

*Rubric: Your work will be graded as follows:*

* *3 points for including the SQL for the stored program (procedure, function, or trigger in your Word document*
* *3 points for clearly explaining the purpose of the stored program*
* *3 points for a screenshot and explanation that shows the stored program in action.*

*Total points possible: 9*

ENTER YOUR WORK WITH STORED PROGRAMS HERE

# L. Transactions

**Importance of Transactions in Ensuring ACID Behavior**

Transactions are essential in database management as they ensure the **ACID** (Atomicity, Consistency, Isolation, Durability) properties, which maintain data integrity and reliability. Here’s how transactions support each ACID principle:

* **Atomicity**: Ensures that all operations within a transaction are completed successfully. If any part fails, the entire transaction is rolled back, preventing partial updates.
* **Consistency**: Guarantees that a transaction brings the database from one valid state to another, maintaining all integrity constraints.
* **Isolation**: Prevents concurrent transactions from interfering with each other, ensuring that intermediate transaction states are not visible to other operations.
* **Durability**: Once a transaction is committed, the changes are permanently saved to the database, even in the case of a system failure.

By using transactions, databases prevent data corruption, avoid lost updates, and ensure reliable operations.

**Demonstrating a MySQL Transaction**

**MySQL Transaction SQL Example:**

START TRANSACTION;

-- Insert a new student

INSERT INTO Students (Student\_ID, Name, Age, Grade\_Level, Parent\_ID)

VALUES (101, 'John Doe', 15, '10th Grade', 5);

-- Enroll the student in a course

INSERT INTO Student\_Course\_Enrollment (Enrollment\_ID, Enrollment\_Date, Course\_Code, Student\_ID)

VALUES (201, '2025-02-25', 'MATH101', 101);

-- Commit the transaction

COMMIT;

**Explanation:**

1. **START TRANSACTION;** begins the transaction.
2. **INSERT statements** add a student and enroll them in a course.
3. **COMMIT;** finalizes the changes, ensuring they are saved permanently.

If any of these operations fail, a ROLLBACK; command could be used to undo all changes, preserving data integrity.

**Screenshot of MySQL Transaction Execution**

A computer screen with text on it

AI-generated content may be incorrect.

By using transactions, we ensure that student enrollment occurs only if both insertions succeed, preventing inconsistent states where a student is registered without an associated enrollment.

# M. Database Security

*Description: Identify the different kinds of users who will use your database. Write GRANT statements to define the privileges for these different kinds of users.*

*Rubric: Your work will be graded as follows:*

* *4 points for clearly identifying and describing the various kinds of users who will use the databases and identifying and justifying what privileges each should have.*
* *4 points for writing GRANT statements that assign privileges to these different kinds of users.*
* *4 points for demonstrating with screenshots that your GRANT statements do distinguish among different kinds of users in regard to what they can do with the database.*

**User Roles and Privileges**

In the **Student Performance Tracking System**, different users require different levels of access to ensure security and efficient database management. Below are the key user roles and their corresponding privileges:

1. **Administrator** (Full Access)
   * Manages all aspects of the database, including user management, data modification, and system maintenance.
   * **Privileges:** ALL PRIVILEGES on all tables.
2. **Teachers** (Limited Access)
   * Can update grades, assignments, and attendance but cannot modify student personal details.
   * **Privileges:** INSERT, UPDATE, SELECT on **Grades, Assignments, Attendance** tables.
3. **Students** (Read-Only Access)
   * Can only view their grades and attendance records.
   * **Privileges:** SELECT on **Grades, Attendance** (restricted to their own data).
4. **Parents** (Limited Read-Only Access)
   * Can view their child’s academic progress and attendance but cannot modify any data.
   * **Privileges:** SELECT on **Grades, Attendance** (restricted to their child's data).

Now before granting different users different privileges we need to make these users. The following screenshot demonstrates creating different users:

CREATE USER 'teacher\_user'@'localhost' IDENTIFIED BY 'password123';

CREATE USER 'student\_user'@'localhost' IDENTIFIED BY 'password123';

CREATE USER 'parent\_user'@'localhost' IDENTIFIED BY 'password123';

CREATE USER 'admin\_user'@'localhost' IDENTIFIED BY 'password123';

A computer screen with white text

AI-generated content may be incorrect.

**GRANT Statements**

1. **Granting Full Access to Administrator**

GRANT ALL PRIVILEGES ON studentperformancedb.\* TO 'admin\_user'@'localhost' WITH GRANT OPTION;

1. **Granting Privileges to Teachers**

GRANT INSERT, UPDATE, SELECT ON studentperformancedb.Grades TO 'teacher\_user'@'localhost';

GRANT INSERT, UPDATE, SELECT ON studentperformancedb.Assignments TO 'teacher\_user'@'localhost';

GRANT INSERT, UPDATE, SELECT ON studentperformancedb.Attendance TO 'teacher\_user'@'localhost';

1. **Granting Read-Only Access to Students**

GRANT SELECT ON studentperformancedb.Grades TO 'student\_user'@'localhost';

GRANT SELECT ON studentperformancedb.Attendance TO 'student\_user'@'localhost';

1. **Granting Read-Only Access to Parents**

GRANT SELECT ON studentperformancedb.Grades TO 'parent\_user'@'localhost';

GRANT SELECT ON studentperformancedb.Attendance TO 'parent\_user'@'localhost';

A screenshot of a computer program

AI-generated content may be incorrect.

**Demonstrating Privileges with Screenshots**

SHOW GRANTS FOR 'teacher\_user'@'localhost';

A screen shot of a computer

AI-generated content may be incorrect.

SHOW GRANTS FOR 'student\_user'@'localhost';

A screen shot of a computer

AI-generated content may be incorrect.

SHOW GRANTS FOR 'parent\_user'@'localhost';

A screen shot of a computer

AI-generated content may be incorrect.

SHOW GRANTS FOR 'admin\_user'@'localhost';

A screen shot of a computer program

AI-generated content may be incorrect.

# N. Locking and Concurrent Access

*Description: Explain the purpose of locking tables and show how to do that to prevent inconsistencies that may arise in your data when concurrent transactions take place.*

*Rubric: Your work will be graded as follows:*

* *3 points for clearly explaining an example that shows why you should lock tables to prevent inconsistencies.*
* *3 points for providing a screenshot and accompanying explanation of locking tables.*

*Total points possible: 5*

ENTER YOUR WORK WITH LOCKING AND CONCURRENT ACCESS HERE

# O. Backing Up Your Database

*Description: How you will back up your database. What commands will you issue? How frequently will the commands run? How can they be automated? Where will the backups be stored?*

*Rubric: Your work will be graded as follows:*

* *6 points for clearly explaining and justifying your database backup strategy, including the frequency with which you will back up the database, how you will automate backups, where you will store them, and how you will secure them. You will earn three points for addressing each factor (frequency, location, automation, and security)*
* *2 points for providing a screenshot of the command you would issue to back up the database and for including a portion of the resulting file.*

*Total points possible: 8*

ENTER YOUR WORK ON DATABASE BACKUPS HERE

# P. Programming

*Description: Write a Python, Java, or PHP program that generates a report that contains a subset of the data from your database. Include the code for your Python program in your Word document, and also post the program to your GitHub repository.*

*Rubric: Your work will be graded as follows:*

* *10 points for writing a Python script (and including its code in the Word doc) that will pull data from a database and store it to a text file and present it to the screen. Your code must have comments in it that explain how it works. You will be awarded 3 points for successfully connecting to the database, 3 points for successfully querying it, and 4 points for presenting the data to the screen and to a file. Internal comments count for 2 points.*
* *2 points for posting the code to GitHub*
* *6 points for showing a screenshot of your running the script and showing the results it produces on the screen.*

*Total points possible: 18*

ENTER YOUR PYTHON, PHP, or JAVA DATABASE PROGRAMMING WORK HERE

# Q. Suggested Future Work

*Description: Describe the limitations of your current database and explain how you or someone else could improve the design to address these shortcomings. Also describe how you might take advantage of leverage cloud services to increase the performance and availability of your database. Finally, explain the advantages and disadvantages of storing your data in a NoSQL format instead.*

*Rubric: Your work will be graded as follows:*

* *3 points for clearly describing the limitations of your databases*
* *3 points for explaining how you would address these shortcomings*
* *3 points for explaining how you might migrate the database to the cloud and describing what advantages you might gain from doing that.*
* *3 points for explaining the advantages and disadvantages of storing your data in a document-based NoSQL format instead.*

*Total points possible: 12*

ENTER YOUR SUGGESTED FUTURE WORK IDEAS HERE

# R. Activity Log

**Week 1:**

* Finalized the project topic: **Student Performance Tracking System**.
* Conducted research on student performance tracking systems to understand key database requirements.
* Created a list of data entities such as Students, Courses, Grades, Attendance, Teachers, and Assignments.

**Week 2:**

* Collected sample datasets for Students and Courses to analyze data structure requirements.
* Identified potential data sources for student records, course details, and attendance tracking.
* Updated sections A, B, C, D and R in the project report.

**Week 3:**

* Developed and finalized the Conceptual and Logical Models, including detailed entity relationships and normalization steps.
* Created ER diagrams for conceptual, logical and physical models.
* Designed the MySQL physical schema, defining table structures, keys, and constraints.
* Completed the E and F section in the document.

**Week 4:**

* Populated the database by generating and executing SQL INSERT statements from cleaned CSV data.
* Utilized Python scripts to generate data for Students, Teachers, Courses, Grades, Attendance, Parents, Assignments, and Student-Course Enrollment.
* Verified data integrity by running SELECT queries in MySQL and updated the GitHub repository with the final scripts and documentation.
* Completed the G and update the R section in the document.