**Triggers and Procedures**

**Objective**

To automate database actions using triggers and stored procedures for efficient and maintainable student record management.

**Database Setup**

Database Creation

CREATE DATABASE student\_management;

USE student\_management;

**Explanation:**  
 Creates a new database named student\_management and sets it as the active context for all subsequent operations.

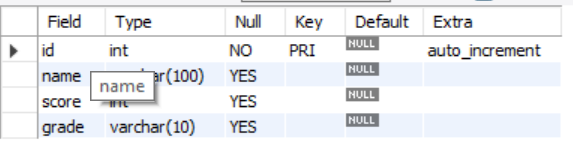
Table Creation

1.students Table

CREATE TABLE students ( id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(100), score INT, grade VARCHAR(10) );

**Explanation:**  
 Stores student details. id is the unique identifier, score represents performance, and grade reflects academic standing.

**Output:**

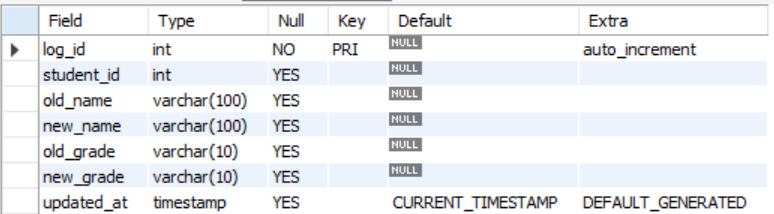


2. student\_update\_log Table

CREATE TABLE student\_update\_log ( log\_id INT AUTO\_INCREMENT PRIMARY KEY, student\_id INT, old\_name VARCHAR(100), new\_name VARCHAR(100), old\_grade VARCHAR(10), new\_grade VARCHAR(10), updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP );

**Explanation:**  
 Captures audit logs whenever a student record is updated. It stores both old and new values for name and grade, along with a timestamp.

**Output:**

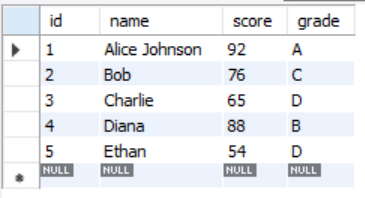


Sample Data Insertion

INSERT INTO students (name, score, grade) VALUES ('Alice', 92, 'B'), ('Bob', 76, 'C'), ('Charlie', 65, 'D'), ('Diana', 88, 'B'), ('Ethan', 54, 'D');

**Explanation:**  
 Populates the students table with sample data for testing triggers and procedures.

**Output:**



**Trigger Definition**

DELIMITER //

CREATE TRIGGER log\_student\_update

AFTER UPDATE ON students

FOR EACH ROW

BEGIN

INSERT INTO student\_update\_log ( student\_id, old\_name, new\_name, old\_grade, new\_grade ) VALUES ( OLD.id, OLD.name, NEW.name, OLD.grade, NEW.grade );

END; //

DELIMITER ;

**Explanation:**  
 This trigger activates automatically after any update to the students table. It logs the previous and new values into student\_update\_log.

**Procedure Definition**

DELIMITER //

CREATE PROCEDURE update\_student\_grades( IN min\_score INT, IN max\_score INT, IN new\_grade VARCHAR(10) ) BEGIN

UPDATE students SET grade = new\_grade WHERE score BETWEEN min\_score AND max\_score;

END; //

DELIMITER ;

**Explanation:**  
 Encapsulates reusable logic to update student grades based on score ranges. Accepts parameters for flexible grading criteria.

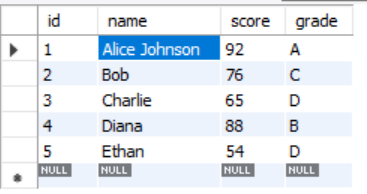
**Sample Executions**

**Trigger Test**

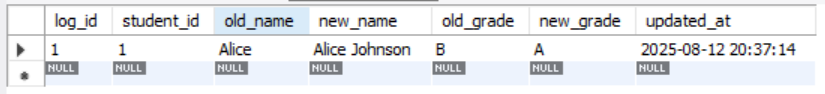
UPDATE students SET name = 'Alice Johnson', grade = 'A' WHERE name = 'Alice';

**Output:**

**Student table**



Log table



**Output Explanation:**  
 The student\_update\_log table should contain a new entry showing:

student\_id of Alice

old\_name as "Alice"

new\_name as "Alice Johnson"

old\_grade as "B"

new\_grade as "A"

This confirms the trigger is functioning correctly.

**Procedure Test**

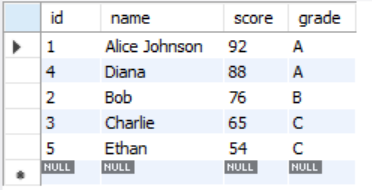
CALL update\_student\_grades(85, 100, 'A');

CALL update\_student\_grades(70, 84, 'B');

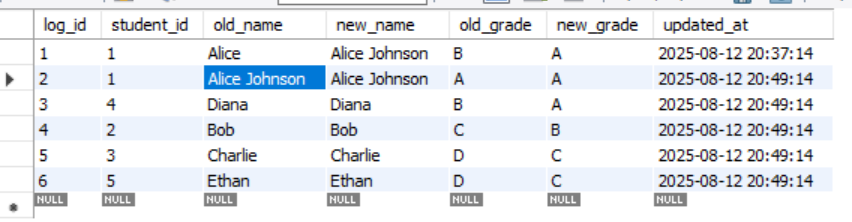
CALL update\_student\_grades(50, 69, 'C');

**Output:**

**Student table**

****

**Log table**

****

**Output Explanation:**  
 The students table should reflect updated grades:

Scores ≥ 85 → Grade 'A'

Scores between 70–84 → Grade 'B'

Scores between 50–69 → Grade 'C'

This confirms the procedure correctly applied the grading logic.

**Summary of Findings**

The implementation successfully automated student record updates and grading logic. The trigger accurately logged changes to student names and grades in the audit table, ensuring traceability. The stored procedure provided a flexible and reusable way to update grades based on performance criteria, simplifying bulk operations. Both mechanisms were validated through sample executions and produced the expected results. This approach enhances database maintainability and ensures consistent data handling.