Film Production Management System

Triggers

A **trigger** is a predefined piece of code (usually a SQL script) that automatically executes when a specific event occurs in a database table. Triggers are stored in the database and act as a mechanism to enforce business rules, maintain data integrity, and automate specific database tasks.

Why Use Triggers:

1. **Automation**: Automatically perform tasks such as validations, logging, or calculations.
2. **Data Integrity**: Enforce rules like non-negative values or cascading updates/deletes.
3. **Audit Trails**: Record changes made to data for accountability and analysis.
4. **Consistency**: Ensure data transformations or checks are applied consistently without relying on application logic.

Advantages of Triggers

- Automation of repetitive database tasks.
- Ensures database-level consistency and integrity.
- Minimizes reliance on application code for validations or business logic.
- Tracks changes for compliance and monitoring.

Disadvantages of Triggers

- Can increase complexity and make debugging harder.
- May impact performance if overused or poorly designed.
- Hidden logic might confuse developers unfamiliar with the database.

```
USE production_db;

CREATE TABLE finance_update (
   before_update_assigned_budget INT,
   updated_update_assigned_budget INT,
   before_fi_phno INT,
   updated_fi_phno INT
);
```

BEFORE_INSERT Trigger

Trigger Name: finance_team_BEFORE_INSERT

Purpose: Ensures data integrity by validating input before a new record is added to the table.

Why Use It:

- To enforce business rules and prevent invalid data from being inserted into the database.
- In this case, it ensures that the assigned_budget is never negative. If a user tries to insert a negative value, the trigger automatically sets it to 0.

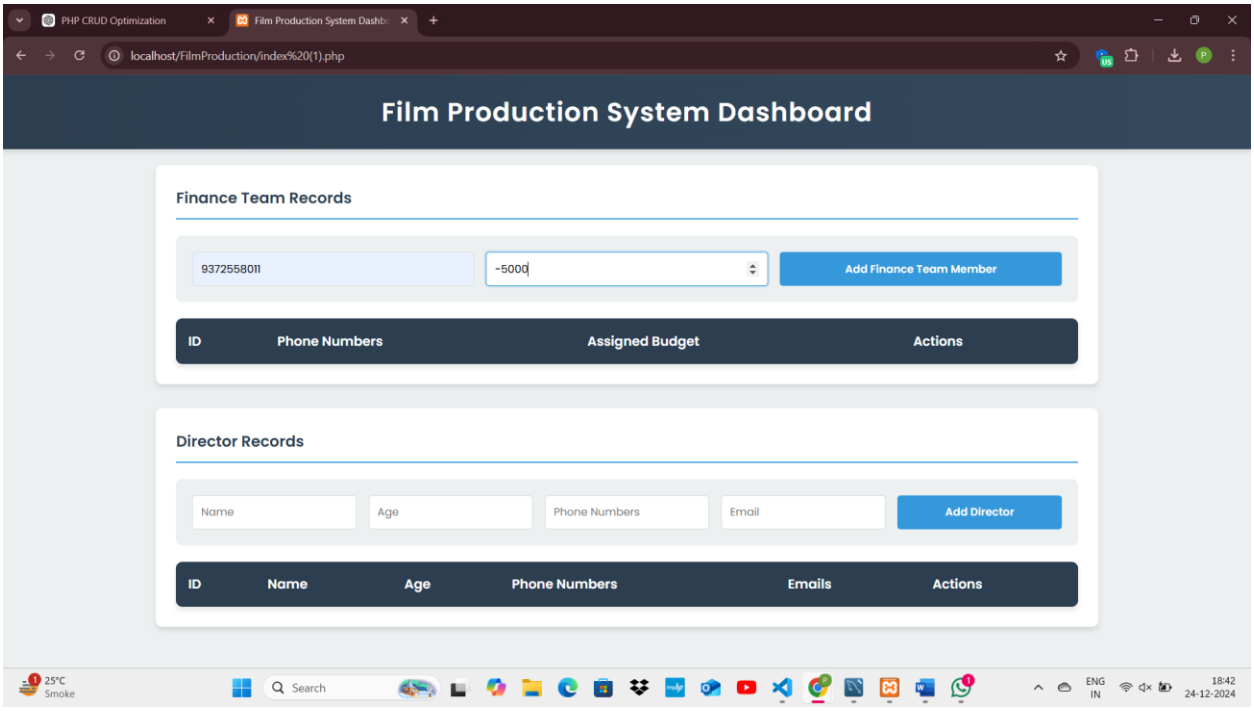
How It Works:

- The BEFORE_INSERT trigger activates just before a new record is inserted into the finance_team table.
- It checks the value of the NEW.assigned_budget field.
- If the value is less than 0, the trigger modifies the value of NEW.assigned_budget to 0.
- The modified value is then inserted into the table.
- This ensures that the data being added adheres to the business rule without requiring additional checks in the application layer.

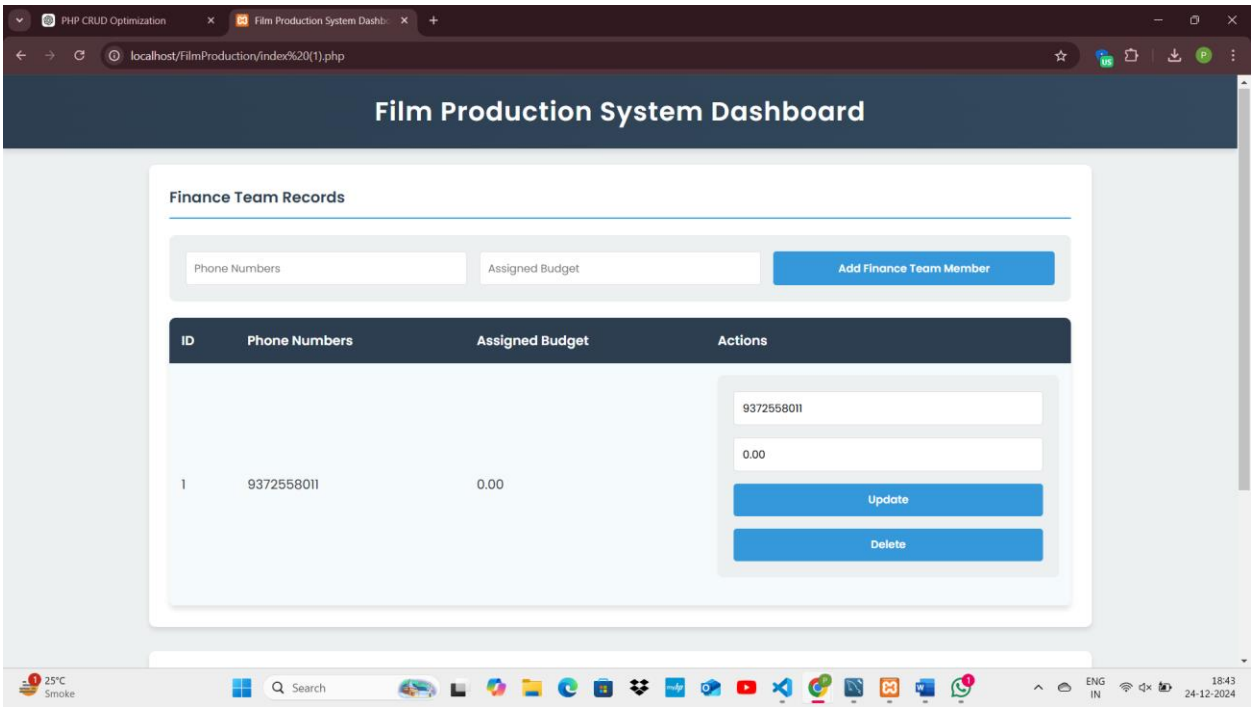
Create the BEFORE_INSERT Trigger

```
DELIMITER $$

CREATE DEFINER=`root`@`localhost` TRIGGER `finance_team_BEFORE_INSERT`
BEFORE INSERT ON `finance_team`
FOR EACH ROW
BEGIN
   IF NEW.assigned_budget < 0 THEN
      SET NEW.assigned_budget = 0;
   END IF;
END $$
DELIMITER ;
```

Insert a sample record into the finance_team table to test the BEFORE_INSERT trigger.



This should automatically set assigned_budget to 0.



AFTER_UPDATE Trigger

Trigger Name: finance_team_AFTER_UPDATE

Purpose: Tracks changes made to critical columns and logs the old and new values in a separate table for auditing purposes.

Why Use It:

- To maintain an audit trail of changes made to important data, such as budget updates and phone number changes.
- It allows you to track who made changes, what was changed, and the before-and-after values.
- Useful for troubleshooting, compliance, and monitoring.

How It Works:

- The AFTER_UPDATE trigger activates after a record in the finance_team table is updated.
- It captures the old and new values of the assigned_budget and fi_phno fields.
- It inserts these values into the finance_update table as a new record, maintaining a log of updates.
- This mechanism provides a transparent history of modifications and helps maintain accountability within the system.

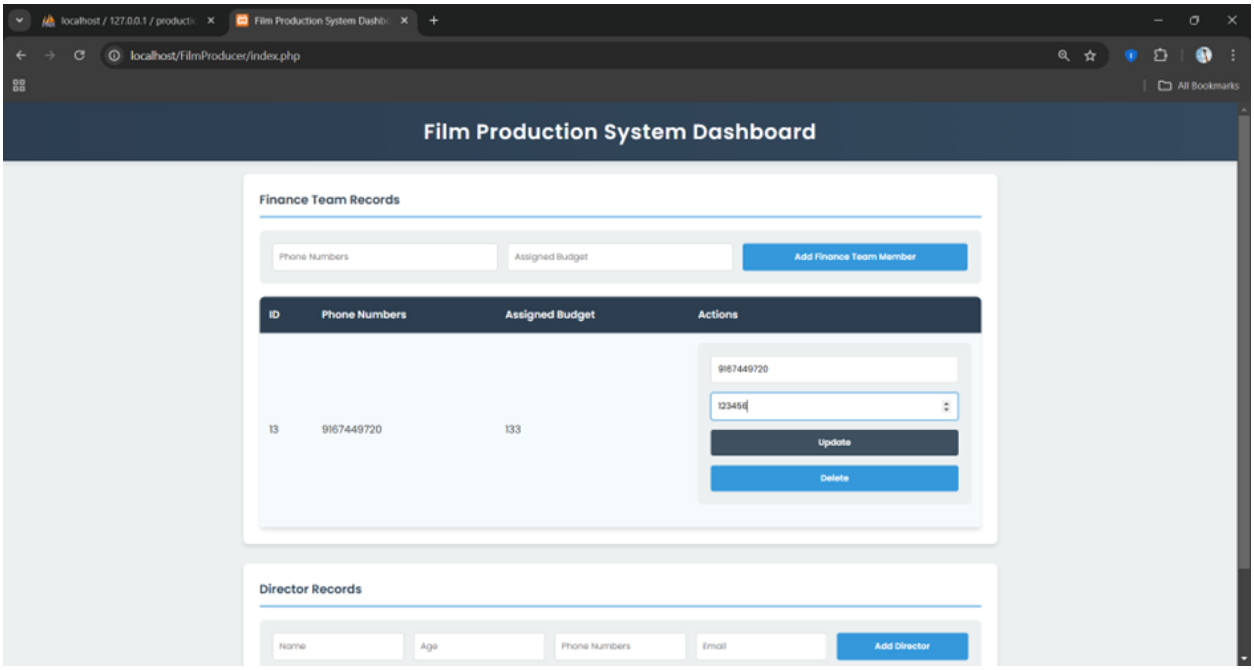
Create the AFTER_UPDATE Trigger

```
DELIMITER $$

CREATE DEFINER=`root`@`localhost` TRIGGER `finance_team_AFTER_UPDATE`
AFTER UPDATE ON `finance_team`
FOR EACH ROW
BEGIN
   INSERT INTO finance_update
   VALUES (OLD.assigned_budget, NEW.assigned_budget, OLD.fi_phno, NEW.fi_phno);
END $$

DELIMITER ;
```

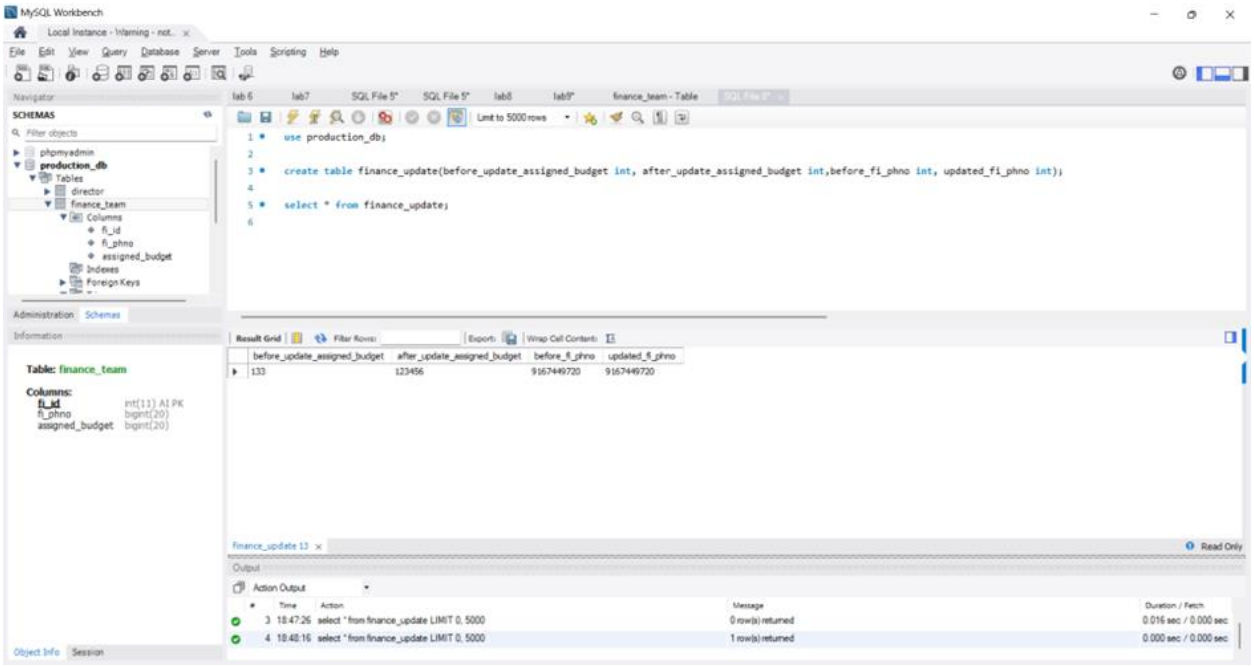
Verify Result:

```
SELECT * FROM finance_update;
```

Update a record in the finance_team table to test the AFTER_UPDATE trigger.



Check the finance_update table for the inserted record.



Observation:

I understand that triggers are a powerful tool in databases for automating tasks, maintaining data integrity, and creating audit trails without relying on application-level code. They help

enforce business rules, such as ensuring non-negative budgets, and provide transparency by logging changes for accountability. However, I find it challenging to debug triggers because the logic is executed automatically and is often hidden, making it harder to trace issues when something goes wrong. Additionally, understanding the flow of execution and testing triggers thoroughly requires a clear grasp of database events and their interactions, which can be complex at times.