**3.TRIGGERS AND PROCEDURE**

**A) TRIGGERS:**

1) Calculate Order Amount and Update Inventory:

Code:

CREATE TRIGGER calculate\_order\_amount\_and\_update\_inventory BEFORE INSERT ON Orders

FOR EACH ROW

BEGIN

    DECLARE item\_rate DECIMAL(10, 2);

    SELECT rate INTO item\_rate FROM Inventory WHERE identifier = NEW.identifier

    SET NEW.amount = NEW.quantity \* item\_rate;

    -- Decrease the quantity of inventory

    UPDATE Inventory

    SET quantity = quantity - NEW.quantity

    WHERE identifier = NEW.identifier;

END;

Functionality: This trigger fires before inserting a new record into the "Orders" table. It calculates the total amount for the order based on the quantity and rate of the items ordered and updates the inventory by decreasing the quantity of the ordered items.

Actions:

* Retrieves the rate of the item from the Inventory table.
* Calculates the order amount by multiplying the quantity with the item rate.
* Updates the inventory by reducing the quantity of the ordered items.

2) Update Inventory after Production Insert:

Code:

CREATE TRIGGER update\_inventory\_after\_production\_insert AFTER INSERT ON Produced

FOR EACH ROW

BEGIN

    -- Update existing identifiers

    INSERT INTO Inventory (identifier, quantity, rate)

    SELECT

        NEW.identifier,

        SUM(NEW.quantity) AS total\_quantity,

        MAX(NEW.rate) \* 1.4 AS rate

    FROM Produced

    WHERE identifier = NEW.identifier

    GROUP BY identifier

    ON DUPLICATE KEY UPDATE

        quantity = quantity + NEW.quantity;

    -- Insert new identifier

    IF NOT EXISTS (SELECT 1 FROM Inventory WHERE identifier = NEW.identifier) THEN

        INSERT INTO Inventory (identifier, quantity, rate) VALUES (NEW.identifier, NEW.quantity, NEW.rate \* 1.4);

    END IF;

END;

Functionality: This trigger fires after inserting a new record into the "Produced" table. It updates the inventory based on the production entries by either inserting new identifiers or updating existing ones. It also increases the quantity of existing identifiers or inserts new ones if they don't exist.

Actions:

* Updates existing inventory items by increasing the quantity.
* Inserts new inventory items if they don't exist.
* Adjusts the rate of the items in the inventory based on a multiplier.

3) Update Salary After Production Insert:

Code:

CREATE TRIGGER update\_salary\_after\_production\_insert AFTER INSERT ON Produced

FOR EACH ROW

BEGIN

    DECLARE month\_year DATE;

    SET month\_year = DATE\_FORMAT(NEW.production\_date, '%Y-%m-01');

    INSERT INTO Salary (month, worker\_id, amount)

    VALUES (month\_year, NEW.worker\_id, NEW.quantity \* NEW.rate)

    ON DUPLICATE KEY UPDATE

    amount = amount + NEW.quantity \* NEW.rate;

END;

Functionality: This trigger fires after inserting a new record into the "Produced" table. It updates the salary table by adding the salary amount for the worker involved in the production. It either inserts a new entry for the month and worker if it doesn't exist or updates the existing entry.

Actions:

* Calculates the salary amount based on the quantity produced and the rate.
* Inserts a new entry into the salary table for the corresponding month and worker.
* Updates the existing entry if it already exists.

4) Update Profit After Salary Insert:

Code:

CREATE TRIGGER update\_profit\_after\_salary\_insert AFTER INSERT ON Produced

FOR EACH ROW

BEGIN

    DECLARE month\_year DATE;

    SET month\_year = DATE\_FORMAT(NEW.production\_date, '%Y-%m-01');

    INSERT INTO Profit (month, total\_salary\_expenses)

    VALUES (month\_year, NEW.quantity \* NEW.rate)

    ON DUPLICATE KEY UPDATE

    total\_salary\_expenses = total\_salary\_expenses + NEW.quantity \* NEW.rate;

END;

Functionality: This trigger fires after inserting a new record into the "Produced" table. It updates the profit table by adding the total salary expenses incurred due to the production. It either inserts a new entry for the month if it doesn't exist or updates the existing entry.

Actions:

* Calculates the total salary expenses based on the quantity produced and the rate.
* Inserts a new entry into the profit table for the corresponding month.
* Updates the existing entry if it already exists.

5) Update Profit After Maintenance Insert:

Code:

CREATE TRIGGER update\_profit\_after\_maintenance\_insert AFTER INSERT ON Maintenance

FOR EACH ROW

BEGIN

    DECLARE month\_year DATE;

    SET month\_year = DATE\_FORMAT(NEW.maintenance\_date, '%Y-%m-01');

    INSERT INTO Profit (month, total\_expenses)

    VALUES (month\_year, NEW.cost)

    ON DUPLICATE KEY UPDATE

    total\_expenses = total\_expenses + NEW.cost;

END;

Functionality: This trigger fires after inserting a new record into the "Maintenance" table. It updates the profit table by adding the total maintenance expenses incurred. It either inserts a new entry for the month if it doesn't exist or updates the existing entry.

Actions:

* Calculates the total maintenance expenses based on the cost of maintenance.
* Inserts a new entry into the profit table for the corresponding month.
* Updates the existing entry if it already exists.

6) Update Profit After Material Insert:

Code:

CREATE TRIGGER update\_profit\_after\_material\_insert AFTER INSERT ON Material

FOR EACH ROW

BEGIN

    DECLARE month\_year DATE;

    SET month\_year = DATE\_FORMAT(NEW.purchase\_date, '%Y-%m-01');

    INSERT INTO Profit (month, total\_expenses)

    VALUES (month\_year, NEW.rate \* NEW.quantity)

    ON DUPLICATE KEY UPDATE

    total\_expenses = total\_expenses + NEW.rate \* NEW.quantity;

END;

Functionality: This trigger fires after inserting a new record into the "Material" table. It updates the profit table by adding the total material expenses incurred. It either inserts a new entry for the month if it doesn't exist or updates the existing entry.

Actions:

* Calculates the total material expenses based on the rate and quantity of the material purchased.
* Inserts a new entry into the profit table for the corresponding month.
* Updates the existing entry if it already exists.

7) Update Profit After Orders Insert:

Code:

CREATE TRIGGER update\_profit\_after\_orders\_insert AFTER INSERT ON Orders

FOR EACH ROW

BEGIN

    DECLARE month\_year DATE;

    SET month\_year = DATE\_FORMAT(NEW.order\_date, '%Y-%m-01');

    INSERT INTO Profit (month, total\_income)

    VALUES (month\_year, NEW.amount)

    ON DUPLICATE KEY UPDATE

    total\_income = total\_income + NEW.amount;

END;

Functionality: This trigger fires after inserting a new record into the "Orders" table. It updates the profit table by adding the total income generated from the orders. It either inserts a new entry for the month if it doesn't exist or updates the existing entry.

Actions:

* Inserts a new entry into the profit table for the corresponding month with the total income from the order.
* Updates the existing entry if it already exists.

SQL Queries for fetching/processing triggers and procedures details:

-- Query to fetch trigger details

SHOW TRIGGERS;

-- Query to fetch procedure details

SHOW PROCEDURE STATUS;

**B) PROCEDURE:**

1) Procedure to calculate total profit for a given month:

Name: calculate\_total\_profit

Input Parameter: month\_year (DATE) - Represents the month for which the total profit is to be calculated.

Functionality: This procedure calculates the total profit for a specified month by subtracting the total expenses from the total income recorded in the Profit table.

Code:

CREATE PROCEDURE calculate\_total\_profit (IN month\_year DATE)

BEGIN

    SELECT

        SUM(total\_income) - SUM(total\_expenses) AS total\_profit

    FROM Profit

    WHERE month = month\_year;

END;

Ex: CALL calculate\_total\_profit('2024-01-01');

2) Procedure to find the average salary of workers:

Name: calculate\_average\_salary

Functionality: This procedure calculates the average salary of all workers based on the data stored in the Salary table.

CREATE PROCEDURE calculate\_average\_salary ()

BEGIN

    SELECT AVG(amount) AS average\_salary FROM Salary;

END;

Ex: CALL calculate\_average\_salary();

SQL Queries for fetching/processing triggers and procedures details:

SHOW TRIGGERS;

SHOW PROCEDURE STATUS;