-- Employee Audit Log using SQL Triggers

```
CREATE DATABASE SQLTriggers;

USE SQLTriggers;


-- Step 1: Create the employees table


CREATE TABLE employees (
   emp_id INT PRIMARY KEY,
   name VARCHAR(100),
   department VARCHAR(100),
   salary DECIMAL(10,2)
);


-- Step 2: Create the employee_audit_log table


CREATE TABLE employee_audit_log (
   audit_id INT AUTO_INCREMENT PRIMARY KEY,
   emp_id INT,
   action_type VARCHAR(10),      -- INSERT, UPDATE, DELETE
   old_name VARCHAR(100),
   new_name VARCHAR(100),
   old_department VARCHAR(100),
   new_department VARCHAR(100),
   old_salary DECIMAL(10,2),
   new_salary DECIMAL(10,2),
   action_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

-- Step 3: Create Triggers for INSERT, UPDATE, DELETE

-- AFTER INSERT Trigger

```
DELIMITER $$

CREATE TRIGGER after_insert_employee
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
  INSERT INTO employee_audit_log (
    emp_id, action_type, new_name, new_department, new_salary
  ) VALUES (
    NEW.emp_id, 'INSERT', NEW.name, NEW.department, NEW.salary
  );
END$$

DELIMITER ;
```

-- AFTER UPDATE Trigger

```
DELIMITER $$

CREATE TRIGGER after_update_employee
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO employee_audit_log (
    emp_id, action_type, old_name, new_name,
    old_department, new_department,
    old_salary, new_salary
  ) VALUES (
    OLD.emp_id, 'UPDATE', OLD.name, NEW.name,
    OLD.department, NEW.department,
    OLD.salary, NEW.salary
  );
END$$

DELIMITER ;
```

-- AFTER DELETE

```
DELIMITER $$

CREATE TRIGGER after_delete_employee
AFTER DELETE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO employee_audit_log (
    emp_id, action_type, old_name, old_department, old_salary
  ) VALUES (
    OLD.emp_id, 'DELETE', OLD.name, OLD.department, OLD.salary
  );
END$$

DELIMITER ;
```

Test the Project

-- Insert a record

```
INSERT INTO employees VALUES (1, 'John Doe', 'IT', 50000);
```

-- Update the record

```
UPDATE employees SET salary = 55000 WHERE emp_id = 1;
```

-- Delete the record

```
DELETE FROM employees WHERE emp_id = 1;
```

```
-- View the log

SELECT * FROM employee_audit_log;

SELECT * FROM employees;



DROP TRIGGER IF EXISTS after_insert_employee;
DROP TRIGGER IF EXISTS after_update_employee;
DROP TRIGGER IF EXISTS after_delete_employee;

SHOW TRIGGERS;

-- BEFORE INSERT Trigger

DELIMITER $$

CREATE TRIGGER before_insert_employee
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
  IF NEW.salary < 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Salary cannot be negative';
  END IF;
END$$

DELIMITER ;
```

```
-- 2. BEFORE UPDATE Trigger (Block Salary Decrease)

DELIMITER $$

CREATE TRIGGER before_update_employee
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
  IF NEW.salary < OLD.salary THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Salary cannot be decreased';
  END IF;
END$$

DELIMITER ;

-- 3. BEFORE DELETE Trigger (Block delete for HR employees)

DELIMITER $$

CREATE TRIGGER before_delete_employee
BEFORE DELETE ON employees
FOR EACH ROW
BEGIN
  IF OLD.department = 'HR' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Cannot delete HR department employees';
  END IF;
END$$

DELIMITER ;
```

-- This will fail if salary is negative

```
INSERT INTO employees VALUES (10, 'Test User', 'Sales', -2000);
```

-- This will fail if trying to decrease salary

```
UPDATE employees SET salary = 30000 WHERE emp_id = 1;
```

-- This will fail if trying to delete an HR employee

```
DELETE FROM employees WHERE emp_id = 2;   -- assuming emp_id=2 is in HR
```

```
SELECT * FROM employee_audit_log;
```

```
SHOW triggers;
```

```
DROP TRIGGER IF EXISTS before_insert_employee;
DROP TRIGGER IF EXISTS before_update_employee;
DROP TRIGGER IF EXISTS before_delete_employee;
```

Updated AFTER INSERT Trigger Using CASE

```
CREATE TRIGGER after_insert_employee
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
  INSERT INTO employee_audit_log (
    emp_id, action_type, new_name, new_department, new_salary, salary_level
  ) VALUES (
    NEW.emp_id,
    'INSERT',
    NEW.name,
    NEW.department,
    NEW.salary,
    CASE
      WHEN NEW.salary < 3000 THEN 'Low'
      WHEN NEW.salary BETWEEN 3000 AND 7000 THEN 'Medium'
      ELSE 'High'
    END
  );
END;
```

Updated AFTER UPDATE Trigger Using CASE

```
CREATE TRIGGER after_update_employee
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO employee_audit_log (
    emp_id, action_type, old_salary, new_salary, salary_change, salary_level
  ) VALUES (
    OLD.emp_id,
    'UPDATE',
    OLD.salary,
    NEW.salary,
    NEW.salary - OLD.salary,
    CASE
      WHEN NEW.salary < 3000 THEN 'Low'
      WHEN NEW.salary BETWEEN 3000 AND 7000 THEN 'Medium'
      ELSE 'High'
    END
  );
END;
```

employee_audit_log Table Must Include These New Columns:

```
ALTER TABLE employee_audit_log
ADD COLUMN salary_level VARCHAR(10),
ADD COLUMN salary_change INT;
```

```
SELECT
    OrderID,
    CustomerName,
    CustomerLocation,
    ProductLine,
    UnitPrice,
    Quantity,
    Total,

    -- 1. Discount Flag
    CASE
        WHEN Total > 500 THEN 'Eligible'
        ELSE 'Not Eligible'
    END AS DiscountStatus,

    -- 2. Product Category Grouping
    CASE
        WHEN ProductLine = 'Electronics' THEN 'Tech'
        WHEN ProductLine = 'Furniture' THEN 'Home'
        WHEN ProductLine = 'Clothing' THEN 'Apparel'
        ELSE 'Other'
    END AS ProductCategory,

    -- 3. Quantity Tier
    CASE
        WHEN Quantity >= 5 THEN 'Bulk'
        WHEN Quantity = 1 THEN 'Single'
        ELSE 'Standard'
    END AS QuantityType,
```

```
-- 4. Region Code
CASE
   WHEN CustomerLocation = 'Tokyo' THEN 'JP'
   WHEN CustomerLocation = 'Mexico City' THEN 'MX'
   WHEN CustomerLocation = 'Toronto' THEN 'CA'
   WHEN CustomerLocation = 'Vancouver' THEN 'CA'
   WHEN CustomerLocation = 'San Francisco' THEN 'US'
   ELSE 'Other'
END AS RegionCode,

-- 5. High Price Product
CASE
   WHEN UnitPrice >= 300 THEN 'High-End'
   WHEN UnitPrice >= 200 THEN 'Mid-Range'
   ELSE 'Budget'
END AS PriceCategory,

-- 6. Even or Odd Order
CASE
   WHEN MOD(OrderID, 2) = 0 THEN 'Even'
   ELSE 'Odd'
END AS OrderType,

-- 7. Free Shipping Eligibility
CASE
   WHEN Total >= 400 THEN 'Free Shipping'
   ELSE 'Shipping Charges Apply'
END AS ShippingStatus,
```

```
    -- 8. Customer Segment Based on Purchase
    CASE
        WHEN Total >= 500 THEN 'Premium'
        WHEN Total >= 300 THEN 'Gold'
        ELSE 'Regular'
    END AS CustomerSegment,

    -- 9. Clothing Sale Tag
    CASE
        WHEN ProductLine = 'Clothing' AND Quantity >= 4 THEN 'Clothing Sale'
        ELSE 'No Sale'
    END AS ClothingSaleStatus,

    -- 10. Location Group
    CASE
        WHEN CustomerLocation IN ('Toronto', 'Vancouver') THEN 'Canada'
        WHEN CustomerLocation = 'San Francisco' THEN 'USA'
        WHEN CustomerLocation = 'Tokyo' THEN 'Japan'
        WHEN CustomerLocation = 'Mexico City' THEN 'Mexico'
        ELSE 'Other'
    END AS LocationGroup

FROM data;
```