1. Write a program in MySQL using CASE statement where if grade is 'A', then performance will be set to 'Excellent'. If grade is 'B', then performance will be set to 'Good'. Otherwise, performance will be set to 'Needs Improvement'.

```
mysql> create database StudentGrades;
Query OK, 1 row affected (0.02 sec)

mysql> use StudentGrades;
Database changed
```

```
mysql> CREATE TABLE StudentGrades (student_id INT PRIMARY KEY AUTO_INCREMENT,student_name VARCHAR(50), grade CHAR(1),performance VARCHAR(20));
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO StudentGrades (student_name, grade) VALUES('Alice', 'A'),('Bob', 'B'),('Charlie', 'C'),('Diana', 'A'),('Eve', 'D');
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

```
mysql> UPDATE StudentGrades
    -> SET performance = CASE
    -> WHEN grade = 'A' THEN 'Excellent'
    ->  WHEN grade = 'B' THEN 'Good'
    -> ELSE 'Needs Improvement'
    -> END;
Query OK, 5 rows affected (0.02 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM StudentGrades;
+------------+--------------+-------+-------------------+
| student_id | student_name | grade | performance       |
+------------+--------------+-------+-------------------+
|          1 | Alice        | A     | Excellent         |
|          2 | Bob          | B     | Good              |
|          3 | Charlie      | C     | Needs Improvement |
|          4 | Diana        | A     | Excellent         |
|          5 | Eve          | D     | Needs Improvement |
+------------+--------------+-------+-------------------+
```

2. Write a program in MySQL using CASE statement where if order_amount is greater than 1000, then discount will be set to '15%'. If order_amount is between 500 and 1000, then discount will be set to '10%'. Otherwise, discount will be set to '5%'.

```
mysql> create database order1;
Query OK, 1 row affected (0.00 sec)

mysql> use order1;
Database changed
mysql> CREATE TABLE Order1 (order_id INT PRIMARY KEY AUTO_INCREMENT,customer_name VARCHAR(50),order_amount DECIMAL(10, 2),discount VARCHAR(10));
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO Order1 (customer_name, order_amount) VALUES('Anto', 1200.00),('chandhana', 800.00),('manju', 450.00),('sriram', 1000.00),('fleix', 300.00),('tim', 400.00
);
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> UPDATE Order1
    -> SET discount = CASE
    -> WHEN order_amount > 1000 THEN '15%'
    -> WHEN order_amount BETWEEN 500 AND 1000 THEN '10%'
    -> ELSE '5%'
    -> END;
Query OK, 6 rows affected (0.00 sec)
Rows matched: 6  Changed: 6  Warnings: 0

mysql> SELECT * FROM Order1;
+----------+---------------+--------------+----------+
| order_id | customer_name | order_amount | discount |
+----------+---------------+--------------+----------+
|        1 | Anto          |      1200.00 | 15%      |
|        2 | chandhana     |       800.00 | 10%      |
|        3 | manju         |       450.00 | 5%       |
|        4 | sriram        |      1000.00 | 10%      |
|        5 | fleix         |       300.00 | 5%       |
|        6 | tim           |       400.00 | 5%       |
+----------+---------------+--------------+----------+
6 rows in set (0.00 sec)
```

3. Write a program in MySQL using CASE statement where if hours_worked is greater than 40, then overtime_status will be set to 'Overtime'. If hours_worked is exactly 40, then overtime_status will be set to 'Full-time'. Otherwise, overtime_status will be set to 'Part-time'.

```
mysql> create database employee_work_hours;
Query OK, 1 row affected (0.00 sec)

mysql> use employee_work_hours;
Database changed
mysql> CREATE TABLE employee_work_hours ( employee_id INT PRIMARY KEY,employee_name VARCHAR(50),hours_worked INT,overtime_status VARCHAR(20));
Query OK, 0 rows affected (0.04 sec)

mysql> INSERT INTO employee_work_hours (employee_id, employee_name, hours_worked)VALUES (1,'Alex', 45),(2,'kannan',60),(3, 'kumar', 20),(4,'kavin',70),(5,'kanish',50);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> UPDATE employee_work_hours
    -> SET overtime_status = CASE
    -> WHEN hours_worked > 40 THEN 'Overtime'
    -> WHEN hours_worked = 40 THEN 'Full-time'
    ->  ELSE 'Part-time'
    -> END;
Query OK, 5 rows affected (0.00 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM employee_work_hours;
+-------------+---------------+-------------+-----------------+
| employee_id | employee_name | hours_worked | overtime_status |
+-------------+---------------+-------------+-----------------+
|           1 | Alex          |           45 | Overtime        |
|           2 | kannan        |           60 | Overtime        |
|           3 | kumar         |           20 | Part-time       |
|           4 | kavin         |           70 | Overtime        |
|           5 | kanish        |           50 | Overtime        |
+-------------+---------------+-------------+-----------------+
```

4. Write a program in MySQL using CASE statement where if temperature is below 0, then weather_condition will be set to 'Freezing'. If temperature is between 0 and 20, then weather_condition will be set to 'Cold'. Otherwise, weather_condition will be set to 'Warm'.

```
mysql> create database weathers;
Query OK, 1 row affected (0.04 sec)

mysql> use weathers;
Database changed
mysql> CREATE TABLE Weather (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     temperature INT,
    ->     weather_condition VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.17 sec)

mysql> INSERT INTO Weather (temperature) VALUES (-5), (10), (25), (0), (20);
Query OK, 5 rows affected (0.05 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> UPDATE Weather
    -> SET weather_condition =
    ->     CASE
    ->         WHEN temperature < 0 THEN 'Freezing'
    ->         WHEN temperature BETWEEN 0 AND 20 THEN 'Cold'
    ->         ELSE 'Warm'
    ->     END;
Query OK, 5 rows affected (0.02 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM Weather;
+----+-------------+-------------------+
| id | temperature | weather_condition |
+----+-------------+-------------------+
|  1 |          -5 | Freezing          |
|  2 |          10 | Cold              |
|  3 |          25 | Warm              |
|  4 |           0 | Cold              |
|  5 |          20 | Cold              |
+----+-------------+-------------------+
5 rows in set (0.01 sec)
```

5. Write a program in MySQL using CASE statement where if score is 100, then rating will be set to 'Perfect'. If score is between 90 and 99, then rating will be set to 'Excellent'. Otherwise, rating will be set to 'Good'.

```
Query OK, 1 row affected (0.03 sec)

mysql> use score;
Database changed
mysql> CREATE TABLE Scores (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     score INT,
    ->     rating VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO Scores (score) VALUES (100), (95), (88), (90), (70);
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> UPDATE Scores
    -> SET rating =
    ->     CASE
    ->         WHEN score = 100 THEN 'Perfect'
    ->         WHEN score BETWEEN 90 AND 99 THEN 'Excellent'
    ->         ELSE 'Good'
    ->     END;
Query OK, 5 rows affected (0.02 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM Scores;
+----+-------+-----------+
| id | score | rating    |
+----+-------+-----------+
|  1 |   100 | Perfect   |
|  2 |    95 | Excellent |
|  3 |    88 | Good      |
|  4 |    90 | Excellent |
|  5 |    70 | Good      |
+----+-------+-----------+
5 rows in set (0.00 sec)
```

6. Write a program in MySQL using CASE statement where if age is less than 18, then category will be set to 'Minor'. If age is between 18 and 65, then category will be set to 'Adult'. Otherwise, category will be set to 'Senior'.

```
mysql> create database persons;
Query OK, 1 row affected (0.03 sec)

mysql> use persons;
Database changed
mysql> create table persons(
    -> person_id int auto_increment primary key,age int,category varchar(10));
Query OK, 0 rows affected (0.03 sec)

mysql> insert into persons(age) values
    -> (15),
    -> (25),
    -> (65),
    -> (70);
Query OK, 4 rows affected (0.02 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> update persons
    -> set category =
    -> case
    -> when age<18 then 'minor'
    -> when age between 18 and 65 then 'adult'
    -> else 'senior'
    -> end;
Query OK, 4 rows affected (0.01 sec)
Rows matched: 4  Changed: 4  Warnings: 0

mysql> select * from persons;
+-----------+------+----------+
| person_id | age  | category |
+-----------+------+----------+
|         1 |   15 | minor    |
|         2 |   25 | adult    |
|         3 |   65 | adult    |
|         4 |   70 | senior   |
+-----------+------+----------+
4 rows in set (0.00 sec)
```

7. Write a program in MySQL using CASE statement where if salary is less than 30000, then tax_bracket will be set to 'Low'. If salary is between 30000 and 70000, then tax_bracket will be set to 'Medium'. Otherwise, tax_bracket will be set to 'High'.

```
mysql> create database salary;
Query OK, 1 row affected (0.01 sec)

mysql> use salary;
Database changed
mysql> CREATE TABLE Employees (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     salary DECIMAL(10, 2),
    ->     tax_bracket VARCHAR(10)
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO Employees (salary) VALUES (25000), (45000), (75000), (30000), (70000);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> UPDATE Employees
    -> SET tax_bracket =
    ->     CASE
    ->         WHEN salary < 30000 THEN 'Low'
    ->         WHEN salary BETWEEN 30000 AND 70000 THEN 'Medium'
    ->         ELSE 'High'
    ->     END;
Query OK, 5 rows affected (0.01 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM Employees;
+----+----------+-------------+
| id | salary   | tax_bracket |
+----+----------+-------------+
|  1 | 25000.00 | Low         |
|  2 | 45000.00 | Medium      |
|  3 | 75000.00 | High        |
|  4 | 30000.00 | Medium      |
|  5 | 70000.00 | Medium      |
+----+----------+-------------+
5 rows in set (0.00 sec)
```

8. Write a program in MySQL using CASE statement where if experience is less than 1 year, then job_level will be set to 'Junior'. If experience is between 1 and 5 years, then job_level will be set to 'Mid'. Otherwise, job_level will be set to 'Senior'.

```
mysql> create database experience;
Query OK, 1 row affected (0.02 sec)

mysql> use experience;
Database changed
mysql> CREATE TABLE EmployeesExperience (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     experience DECIMAL(5, 2),
    ->     job_level VARCHAR(10)
    -> );
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO EmployeesExperience (experience) VALUES (0.5), (3), (6), (1), (5);
Query OK, 5 rows affected (0.03 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> UPDATE EmployeesExperience
    -> SET job_level =
    ->     CASE
    ->         WHEN experience < 1 THEN 'Junior'
    ->         WHEN experience BETWEEN 1 AND 5 THEN 'Mid'
    ->         ELSE 'Senior'
    ->     END;
Query OK, 5 rows affected (0.01 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM EmployeesExperience;
+----+------------+-----------+
| id | experience | job_level |
+----+------------+-----------+
|  1 |       0.50 | Junior    |
|  2 |       3.00 | Mid       |
|  3 |       6.00 | Senior    |
|  4 |       1.00 | Mid       |
|  5 |       5.00 | Mid       |
+----+------------+-----------+
5 rows in set (0.00 sec)
```

9. Write a program in MySQL using CASE statement where if balance is less than 1000, then account_status will be set to 'Inactive'. If balance is between 1000 and 5000, then account_status will be set to 'Active'. Otherwise, account_status will be set to 'Premium'.

```
mysql> create database balances;
Query OK, 1 row affected (0.02 sec)

mysql> use balances;
Database changed
mysql> CREATE TABLE BankAccounts (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     balance DECIMAL(10, 2),
    ->     account_status VARCHAR(10)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO BankAccounts (balance) VALUES (500), (2000), (7000), (1000), (4500);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> UPDATE BankAccounts
    -> SET account_status =
    ->     CASE
    ->         WHEN balance < 1000 THEN 'Inactive'
    ->         WHEN balance BETWEEN 1000 AND 5000 THEN 'Active'
    ->         ELSE 'Premium'
    ->     END;
Query OK, 5 rows affected (0.01 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM BankAccounts;
+----+---------+----------------+
| id | balance | account_status |
+----+---------+----------------+
|  1 |  500.00 | Inactive       |
|  2 | 2000.00 | Active         |
|  3 | 7000.00 | Premium        |
|  4 | 1000.00 | Active         |
|  5 | 4500.00 | Active         |
+----+---------+----------------+
5 rows in set (0.00 sec)
```

10. Write a program in MySQL using CASE statement where if attendance is 100%, then bonus will be set to 'Full'. If attendance is between 90% and 99%, then bonus will be set to 'Partial'. Otherwise, bonus will be set to 'None'.

```
mysql> create database attendance;
Query OK, 1 row affected (0.02 sec)

mysql> use attendance;
Database changed
mysql> CREATE TABLE EmployeeAttendance (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     attendance DECIMAL(5, 2),
    ->     bonus VARCHAR(10)
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO EmployeeAttendance (attendance) VALUES (100), (95), (80), (99.5), (70);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> UPDATE EmployeeAttendance
    -> SET bonus =
    ->     CASE
    ->         WHEN attendance = 100 THEN 'Full'
    ->         WHEN attendance BETWEEN 90 AND 99.99 THEN 'Partial'
    ->         ELSE 'None'
    ->     END;
Query OK, 5 rows affected (0.03 sec)
Rows matched: 5  Changed: 5  Warnings: 0

mysql> SELECT * FROM EmployeeAttendance;
+----+------------+---------+
| id | attendance | bonus   |
+----+------------+---------+
|  1 |     100.00 | Full    |
|  2 |      95.00 | Partial |
|  3 |      80.00 | None    |
|  4 |      99.50 | Partial |
|  5 |      70.00 | None    |
+----+------------+---------+
5 rows in set (0.00 sec)
```

11. Create a trigger in MySQL to log the insertions into the ORDERS table with fields order_id, customer_name, and order_date. Also create a new table named ORDERS_AUDIT to keep the changes. Create an AFTER INSERT trigger that is invoked after a record is inserted into the ORDERS table.

```
mysql> create database orders;
Query OK, 1 row affected (0.01 sec)

mysql> use orders;
Database changed
mysql> CREATE TABLE ORDERS_AUDIT (
    ->     audit_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     order_id INT,
    ->     customer_name VARCHAR(255),
    ->     order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.06 sec)

mysql> CREATE TABLE ORDERS (
    ->     order_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     customer_name VARCHAR(255),
    ->     order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.06 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER after_order_insert
    -> AFTER INSERT ON ORDERS
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO ORDERS_AUDIT (order_id, customer_name, order_date)
    ->     VALUES (NEW.order_id, NEW.customer_name, NEW.order_date);
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.05 sec)

mysql> DELIMITER ;
```

```
mysql> INSERT INTO ORDERS (customer_name) VALUES ('John Doe');
Query OK, 1 row affected (0.04 sec)

mysql>
mysql> SELECT * FROM ORDERS_AUDIT;
+----------+----------+---------------+---------------------+
| audit_id | order_id | customer_name | order_date          |
+----------+----------+---------------+---------------------+
|        1 |        1 | John Doe      | 2024-11-21 09:23:26 |
+----------+----------+---------------+---------------------+
1 row in set (0.00 sec)

mysql>
```

12. Create a trigger in MySQL to log the updates to the CUSTOMERS table with fields customer_id, old_address, and new_address. Also create a new table named CUSTOMERS_AUDIT to keep the changes. Create a BEFORE UPDATE trigger that is invoked before a record is updated in the CUSTOMERS table.

```
mysql> create database customers;
Query OK, 1 row affected (0.02 sec)

mysql> use customers;
Database changed
mysql> CREATE TABLE CUSTOMERS_AUDIT (
    ->     audit_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     customer_id INT,
    ->     old_address VARCHAR(255),
    ->     new_address VARCHAR(255),
    ->     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> CREATE TABLE CUSTOMERS (
    ->     customer_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     customer_name VARCHAR(255),
    ->     address VARCHAR(255)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER before_customer_update
    -> BEFORE UPDATE ON CUSTOMERS
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO CUSTOMERS_AUDIT (customer_id, old_address, new_address)
    ->     VALUES (OLD.customer_id, OLD.address, NEW.address);
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> INSERT INTO CUSTOMERS (customer_name, address) VALUES ('Alice Smith', '123 Main St');
Query OK, 1 row affected (0.01 sec)

mysql> UPDATE CUSTOMERS
    -> SET address = '456 Oak St'
    -> WHERE customer_name = 'Alice Smith';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM CUSTOMERS_AUDIT;
+----------+-------------+-------------+-------------+---------------------+
| audit_id | customer_id | old_address | new_address | updated_at          |
+----------+-------------+-------------+-------------+---------------------+
|        1 |           1 | 123 Main St | 456 Oak St  | 2024-11-21 09:25:40 |
+----------+-------------+-------------+-------------+---------------------+
1 row in set (0.00 sec)
```

13. Create a trigger in MySQL to log the deletions from the INVENTORY table with fields item_id, item_name, and deleted_at. Also create a new table named INVENTORY_AUDIT to

keep the changes. Create an AFTER DELETE trigger that is invoked after a record is deleted from the INVENTORY table.

```
mysql> create database inventory;
Query OK, 1 row affected (0.05 sec)

mysql> use inventory;
Database changed
mysql> CREATE TABLE INVENTORY_AUDIT (
    ->     audit_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     item_id INT,
    ->     item_name VARCHAR(255),
    ->     deleted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.38 sec)

mysql> CREATE TABLE INVENTORY (
    ->     item_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     item_name VARCHAR(255),
    ->     quantity INT,
    ->     price DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER after_inventory_delete
    -> AFTER DELETE ON INVENTORY
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO INVENTORY_AUDIT (item_id, item_name, deleted_at)
    ->     VALUES (OLD.item_id, OLD.item_name, NOW());
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.04 sec)

mysql> DELIMITER ;
```

```
mysql> INSERT INTO INVENTORY (item_name, quantity, price)
    -> VALUES ('Laptop', 10, 999.99),
    ->        ('Smartphone', 25, 499.49),
    ->        ('Tablet', 15, 299.99);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> DELETE FROM INVENTORY WHERE item_name = 'Smartphone';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM INVENTORY_AUDIT;
+----------+---------+------------+---------------------+
| audit_id | item_id | item_name  | deleted_at          |
+----------+---------+------------+---------------------+
|        1 |       2 | Smartphone | 2024-11-21 09:27:53 |
+----------+---------+------------+---------------------+
1 row in set (0.00 sec)
```

14. Create a trigger in MySQL to log the changes to the SUPPLIERS table with fields supplier_id, old_contact, and new_contact. Also create a new table named SUPPLIERS_AUDIT to keep the changes. Create a BEFORE UPDATE trigger that is invoked before a change is made to the SUPPLIERS table.

```
mysql> create database supplier;
Query OK, 1 row affected (0.03 sec)

mysql> use supplier;
Database changed
mysql> CREATE TABLE SUPPLIERS_AUDIT (
    ->     audit_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     supplier_id INT,
    ->     old_contact VARCHAR(255),
    ->     new_contact VARCHAR(255),
    ->     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.24 sec)

mysql> CREATE TABLE SUPPLIERS (
    ->     supplier_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     supplier_name VARCHAR(255),
    ->     contact VARCHAR(255)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER before_supplier_update
    -> BEFORE UPDATE ON SUPPLIERS
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO SUPPLIERS_AUDIT (supplier_id, old_contact, new_contact, updated_at)
    ->     VALUES (OLD.supplier_id, OLD.contact, NEW.contact, NOW());
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.03 sec)

mysql> DELIMITER ;
```

```
mysql> INSERT INTO SUPPLIERS (supplier_name, contact)
    -> VALUES ('Supplier A', '123-456-7890'),
    ->        ('Supplier B', '987-654-3210'),
    ->        ('Supplier C', '555-555-5555');
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> UPDATE SUPPLIERS
    -> SET contact = '111-222-3333'
    -> WHERE supplier_name = 'Supplier A';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM SUPPLIERS_AUDIT;
+----------+-------------+--------------+--------------+---------------------+
| audit_id | supplier_id | old_contact  | new_contact  | updated_at          |
+----------+-------------+--------------+--------------+---------------------+
|        1 |           1 | 123-456-7890 | 111-222-3333 | 2024-11-21 09:33:43 |
+----------+-------------+--------------+--------------+---------------------+
1 row in set (0.00 sec)
```

15. Create a trigger in MySQL to log the insertions into the SALES table with fields sale_id, product_name, and sale_amount. Also create a new table named SALES_AUDIT to keep the changes. Create an AFTER INSERT trigger that is invoked after a record is inserted into the SALES table.

```
mysql> create database sales;
Query OK, 1 row affected (0.03 sec)

mysql> use sales;
Database changed
mysql> CREATE TABLE SALES_AUDIT (
    ->     audit_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     sale_id INT,
    ->     product_name VARCHAR(255),
    ->     sale_amount DECIMAL(10, 2),
    ->     inserted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.68 sec)

mysql> CREATE TABLE SALES (
    ->     sale_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     product_name VARCHAR(255),
    ->     sale_amount DECIMAL(10, 2),
    ->     sale_date DATE
    -> );
Query OK, 0 rows affected (0.23 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER after_sales_insert
    -> AFTER INSERT ON SALES
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO SALES_AUDIT (sale_id, product_name, sale_amount, inserted_at)
    ->     VALUES (NEW.sale_id, NEW.product_name, NEW.sale_amount, NOW());
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.04 sec)

mysql> DELIMITER ;
```

```
mysql> INSERT INTO SALES (product_name, sale_amount, sale_date)
    -> VALUES ('Laptop', 1200.99, '2024-11-21'),
    ->        ('Smartphone', 799.49, '2024-11-21'),
    ->        ('Tablet', 399.99, '2024-11-21');
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM SALES_AUDIT;
+----------+---------+--------------+-------------+---------------------+
| audit_id | sale_id | product_name | sale_amount | inserted_at
|
+----------+---------+--------------+-------------+---------------------+
|        1 |       1 | Laptop       |     1200.99 | 2024-11-21 09:36:11 |
|        2 |       2 | Smartphone   |      799.49 | 2024-11-21 09:36:11 |
|        3 |       3 | Tablet       |      399.99 | 2024-11-21 09:36:11 |
+----------+---------+--------------+-------------+---------------------+
3 rows in set (0.00 sec)
```

16. Create a trigger in MySQL to log the updates to the INVENTORY table with fields item_id, old_quantity, and new_quantity. Also create a new table named INVENTORY_AUDIT to keep the changes. Create a BEFORE UPDATE trigger that is invoked before a record is updated in the INVENTORY table.

```
mysql> create database inventorys;
Query OK, 1 row affected (0.02 sec)

mysql> use inventorys;
Database changed
mysql> ^C
mysql> CREATE TABLE INVENTORY_AUDIT (
    ->     audit_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     item_id INT,
    ->     old_quantity INT,
    ->     new_quantity INT,
    ->     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql> CREATE TABLE INVENTORY (
    ->     item_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     item_name VARCHAR(255),
    ->     quantity INT
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER before_inventory_update
    -> BEFORE UPDATE ON INVENTORY
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO INVENTORY_AUDIT (item_id, old_quantity, new_quantity, updated_at)
    ->     VALUES (OLD.item_id, OLD.quantity, NEW.quantity, NOW());
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.04 sec)

mysql> DELIMITER ;
```

```
mysql> INSERT INTO INVENTORY (item_name, quantity)
    -> VALUES ('Laptop', 10),
    ->        ('Smartphone', 25),
    ->        ('Tablet', 15);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> UPDATE INVENTORY
    -> SET quantity = 12
    -> WHERE item_name = 'Laptop';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM INVENTORY_AUDIT;
+----------+---------+--------------+--------------+---------------------+
| audit_id | item_id | old_quantity | new_quantity | updated_at
 |
+----------+---------+--------------+--------------+---------------------+
|        1 |       1 |           10 |           12 | 2024-11-21 09:45:50 |
+----------+---------+--------------+--------------+---------------------+
1 row in set (0.00 sec)

mysql>
```

17. Create a trigger in MySQL to log the changes to the PROJECTS table with fields project_id, old_budget, and new_budget. Also create a new table named PROJECTS_AUDIT to keep the changes. Create a BEFORE UPDATE trigger that is invoked before a change is made to the PROJECTS table.

```
mysql> create database project;
Query OK, 1 row affected (0.02 sec)

mysql> use project;
Database changed
mysql> CREATE TABLE PROJECTS_AUDIT (
    ->     audit_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     project_id INT,
    ->     old_budget DECIMAL(10, 2),
    ->     new_budget DECIMAL(10, 2),
    ->     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.06 sec)

mysql> CREATE TABLE PROJECTS (
    ->     project_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     project_name VARCHAR(255),
    ->     budget DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER before_project_update
    -> BEFORE UPDATE ON PROJECTS
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO PROJECTS_AUDIT (project_id, old_budget, new_budget, updated_at)
    ->     VALUES (OLD.project_id, OLD.budget, NEW.budget, NOW());
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.06 sec)

mysql> DELIMITER ;
```

```
mysql> INSERT INTO PROJECTS (project_name, budget)
    -> VALUES ('Project A', 50000.00),
    ->        ('Project B', 75000.00),
    ->        ('Project C', 100000.00);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> UPDATE PROJECTS
    -> SET budget = 55000.00
    -> WHERE project_name = 'Project A';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM PROJECTS_AUDIT;
+----------+------------+------------+------------+---------------------+
| audit_id | project_id | old_budget | new_budget | updated_at
|
+----------+------------+------------+------------+---------------------+
|        1 |          1 |   50000.00 |   55000.00 | 2024-11-21 09:53:13 |
+----------+------------+------------+------------+---------------------+
1 row in set (0.00 sec)
```

18. Create a trigger in MySQL to log the deletions from the REGISTRATIONS table with fields registration_id, student_name, and deleted_at. Also create a new table named REGISTRATIONS_AUDIT to keep the changes. Create an AFTER DELETE trigger that is invoked after a record is deleted from the REGISTRATIONS table.

```
mysql> create database registration;
Query OK, 1 row affected (0.03 sec)

mysql> use registration;
Database changed
mysql> CREATE TABLE REGISTRATIONS_AUDIT (
    ->     audit_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     registration_id INT,
    ->     student_name VARCHAR(255),
    ->     deleted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql> CREATE TABLE REGISTRATIONS (
    ->     registration_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     student_name VARCHAR(255),
    ->     course_name VARCHAR(255),
    ->     registration_date DATE
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER after_registration_delete
    -> AFTER DELETE ON REGISTRATIONS
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO REGISTRATIONS_AUDIT (registration_id, student_name, deleted_at)
    ->     VALUES (OLD.registration_id, OLD.student_name, NOW());
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.03 sec)

mysql> DELIMITER ;
```

```
mysql> INSERT INTO REGISTRATIONS (student_name, course_name, registration_date)
    -> VALUES ('John Doe', 'Math 101', '2024-09-01'),
    ->        ('Jane Smith', 'Science 101', '2024-09-10'),
    ->        ('Bob Brown', 'History 101', '2024-09-15');
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> DELETE FROM REGISTRATIONS
    -> WHERE student_name = 'John Doe';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM REGISTRATIONS_AUDIT;
+----------+-----------------+--------------+---------------------+
| audit_id | registration_id | student_name | deleted_at          |
+----------+-----------------+--------------+---------------------+
|        1 |               1 | John Doe     | 2024-11-21 09:57:02 |
+----------+-----------------+--------------+---------------------+
1 row in set (0.00 sec)
```

19. Create a trigger in MySQL to log the insertions into the ATTENDANCE table with fields attendance_id, student_name, and attendance_date. Also create a new table named ATTENDANCE_AUDIT to keep the changes. Create an AFTER INSERT trigger that is invoked after a record is inserted into the ATTENDANCE table.

```
mysql> create database attendances;
Query OK, 1 row affected (0.02 sec)

mysql> use attendances;
Database changed
mysql> ^C
mysql> CREATE TABLE ATTENDANCE_AUDIT (
    ->     audit_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     attendance_id INT,
    ->     student_name VARCHAR(255),
    ->     attendance_date DATE,
    ->     inserted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql> CREATE TABLE ATTENDANCE (
    ->     attendance_id INT AUTO_INCREMENT PRIMARY KEY,
    ->     student_name VARCHAR(255),
    ->     attendance_date DATE
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> DELIMITER //
mysql>
mysql> CREATE TRIGGER after_attendance_insert
    -> AFTER INSERT ON ATTENDANCE
    -> FOR EACH ROW
    -> BEGIN
    ->     INSERT INTO ATTENDANCE_AUDIT (attendance_id, student_name, attendance_date, inserted_at)
    ->     VALUES (NEW.attendance_id, NEW.student_name, NEW.attendance_date, NOW());
    -> END;
    ->
    -> //
Query OK, 0 rows affected (0.04 sec)

mysql> DELIMITER ;
```

```
mysql> INSERT INTO ATTENDANCE (student_name, attendance_date)
    -> VALUES ('John Doe', '2024-11-21'),
    ->        ('Jane Smith', '2024-11-21'),
    ->        ('Bob Brown', '2024-11-21');
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> ^C
mysql> SELECT * FROM ATTENDANCE_AUDIT;
+----------+---------------+--------------+-----------------+---------------------+
| audit_id | attendance_id | student_name | attendance_date | inserted_at         |
+----------+---------------+--------------+-----------------+---------------------+
|        1 |             1 | John Doe     | 2024-11-21      | 2024-11-21 10:00:15 |
|        2 |             2 | Jane Smith   | 2024-11-21      | 2024-11-21 10:00:15 |
|        3 |             3 | Bob Brown    | 2024-11-21      | 2024-11-21 10:00:15 |
+----------+---------------+--------------+-----------------+---------------------+
3 rows in set (0.01 sec)
```