INSERT Operation on Trigger

PRODUCTS TABLE

```
CREATE TABLE products (
   product_id INT NOT NULL AUTO_INCREMENT,
   product_name VARCHAR(50),
   price DECIMAL(10, 2),
   created_date TIMESTAMP,
   PRIMARY KEY (product_id)
);
```

PRODUCTS_LOG TABLE

```
CREATE TABLE products_log (
   log_id INT NOT NULL AUTO_INCREMENT,
   product_id INT,
   log_date TIMESTAMP,
   PRIMARY KEY (log_id)
);
```

BEFORE INSERT Trigger (`before_insert_product`)

```
DELIMITER //
CREATE TRIGGER before_insert_product
BEFORE INSERT ON products
FOR EACH ROW
BEGIN
   SET NEW.created_date = CURRENT_TIMESTAMP;
END //
DELIMITER ;
```

AFTER INSERT Trigger (`after_insert_product`)

```
DELIMITER //

CREATE TRIGGER after_insert_product

AFTER INSERT ON products

FOR EACH ROW

BEGIN

   INSERT INTO products_log (product_id, log_date)

   VALUES (NEW.product_id, CURRENT_TIMESTAMP);

END //

DELIMITER ;
```

Inserting Data to Test the Triggers

Sample Insert into `products`

```
INSERT INTO products (product_name, price)

VALUES ('Laptop', 1200.00);
```

1. **`BEFORE INSERT Trigger`**: The `before_insert_product` trigger sets the `created_date` field in the `products` table to the current timestamp automatically.

2. **`AFTER INSERT Trigger`**: The `after_insert_product` trigger creates a new entry in the `products_log` table, recording the `product_id` and `log_date`.

Delete Operation on Trigger

PRODUCTS TABLE

```
CREATE TABLE products (
   product_id INT NOT NULL AUTO_INCREMENT,
   product_name VARCHAR(50),
   price DECIMAL(10, 2),
   created_date TIMESTAMP,
   PRIMARY KEY (product_id)
);
```

PRODUCTS_LOG TABLE

```
CREATE TABLE products_log (
   log_id INT NOT NULL AUTO_INCREMENT,
   product_id INT,
   log_date TIMESTAMP,
   PRIMARY KEY (log_id)
);
```

BEFORE DELETE Trigger

```
DELIMITER //
CREATE TRIGGER before_product_delete
BEFORE DELETE ON products
FOR EACH ROW
BEGIN
   INSERT INTO products_log (product_id, log_date)
   VALUES (OLD.product_id, NOW());
END;
//
DELIMITER ;
```

AFTER DELETE Trigger

```
DELIMITER //

CREATE TRIGGER after_product_delete

AFTER DELETE ON products

FOR EACH ROW

BEGIN

   INSERT INTO products_log (product_id, log_date)

   VALUES (OLD.product_id, NOW());

END;

//

DELIMITER ;
```

Explanation

- **BEFORE DELETE Trigger**: This trigger records an entry in `products_log` before the product is actually removed from `products`.
- **AFTER DELETE Trigger**: This trigger logs the deletion after the product has been removed

```
INSERT INTO products (product_name, price, created_date)
VALUES
   ('Product A', 10.99, NOW()),
   ('Product B', 20.99, NOW()),
   ('Product C', 30.99, NOW());


DELETE FROM products WHERE product_id = 1;

SELECT * FROM products_log;
```

BEFORE UPDATE Trigger

```
DELIMITER //

CREATE TRIGGER before_product_update
BEFORE UPDATE ON products
FOR EACH ROW
BEGIN
   INSERT INTO products_log (product_id, log_date)
   VALUES (OLD.product_id, NOW());
END;

//

DELIMITER ;
```

AFTER UPDATE Trigger

```
DELIMITER //

CREATE TRIGGER after_product_update
AFTER UPDATE ON products
FOR EACH ROW
BEGIN
   INSERT INTO products_log (product_id, log_date)
   VALUES (NEW.product_id, NOW());
END;

//

DELIMITER ;
```

Explanation

- **BEFORE UPDATE Trigger**: Logs the `product_id` and timestamp before the row is updated, capturing the state of the record prior to the change.
- **AFTER UPDATE Trigger**: Logs the `product_id` and timestamp after the row is updated, capturing the state of the record post-change.

These triggers will create log entries in the `products_log` table for each update on the `products` table, allowing you to track updates both before and after they occur.

```
INSERT INTO products (product_name, price, created_date)
VALUES
   ('Product A', 10.99, NOW()),
   ('Product B', 20.99, NOW()),
   ('Product C', 30.99, NOW());


UPDATE products
SET price = 15.99
WHERE product_id = 1;


SELECT * FROM products_log;
```

Drop Existing Triggers

```
DROP TRIGGER IF EXISTS before_product_update;
DROP TRIGGER IF EXISTS after_product_update;
```

Using `DECLARE variable` in a Trigger

```
DELIMITER //
CREATE TRIGGER before_product_update
BEFORE UPDATE ON products
FOR EACH ROW
BEGIN
   -- Declare a variable to store the price difference
   DECLARE price_difference DECIMAL(10, 2);

   -- Calculate the price difference
   SET price_difference = NEW.price - OLD.price;

   -- Log the update in the products_log table
   INSERT INTO products_log (product_id, log_date)
   VALUES (OLD.product_id, NOW());

   -- Optionally, insert the price difference into the log or use it in conditional logic
   IF price_difference <> 0 THEN
      INSERT INTO products_log (product_id, log_date)
      VALUES (OLD.product_id, NOW());
   END IF;
END;
//
DELIMITER ;
```