To implement triggers on the `clientmstr` table in MySQL that keep track of records being updated or deleted, and insert the old values into an `audit\_trade` table, we can create two types of triggers: \*\*row-level triggers\*\* and \*\*statement-level triggers\*\*. Below is a simple implementation for both types of triggers.

### Assumptions:

- We have a `clientmstr` table where client data is stored.

- We will create an `audit\_trade` table to store the old values when records are updated or deleted.

- For simplicity, the `clientmstr` table has columns: `client\_id`, `client\_name`, `client\_email`, `client\_phone`.

- The `audit\_trade` table will store the following columns: `audit\_id`, `client\_id`, `old\_client\_name`, `old\_client\_email`, `old\_client\_phone`, `action\_type`, `timestamp`.

### 1. \*\*Create the `clientmstr` and `audit\_trade` Tables\*\*

```sql

-- Create the clientmstr table

CREATE TABLE clientmstr (

client\_id INT PRIMARY KEY,

client\_name VARCHAR(100),

client\_email VARCHAR(100),

client\_phone VARCHAR(20)

);

-- Create the audit\_trade table

CREATE TABLE audit\_trade (

audit\_id INT AUTO\_INCREMENT PRIMARY KEY,

client\_id INT,

old\_client\_name VARCHAR(100),

old\_client\_email VARCHAR(100),

old\_client\_phone VARCHAR(20),

action\_type VARCHAR(10), -- 'UPDATE' or 'DELETE'

timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP

);

```

### 2. \*\*Row-Level Trigger for `UPDATE`\*\*

A row-level trigger executes once for each row affected by the `UPDATE` statement. It will insert the old values into the `audit\_trade` table whenever a record in `clientmstr` is updated.

```sql

DELIMITER $$

CREATE TRIGGER track\_client\_update\_row

AFTER UPDATE ON clientmstr

FOR EACH ROW

BEGIN

-- Insert old values into audit\_trade table for updated record

INSERT INTO audit\_trade (client\_id, old\_client\_name, old\_client\_email, old\_client\_phone, action\_type)

VALUES (OLD.client\_id, OLD.client\_name, OLD.client\_email, OLD.client\_phone, 'UPDATE');

END $$

DELIMITER ;

```

### Explanation:

- `AFTER UPDATE ON clientmstr`: This trigger fires \*\*after\*\* an update operation on the `clientmstr` table.

- `FOR EACH ROW`: This indicates that the trigger will execute for each updated row.

- `OLD.client\_id, OLD.client\_name, etc.`: The `OLD` keyword is used to refer to the values before the update. These values are inserted into the `audit\_trade` table to track the old data.

- The action type is labeled as `'UPDATE'` to specify that this trigger is for an update operation.

### 3. \*\*Row-Level Trigger for `DELETE`\*\*

A row-level trigger also works for `DELETE` statements, where it stores the old values of deleted rows in the `audit\_trade` table.

```sql

DELIMITER $$

CREATE TRIGGER track\_client\_delete\_row

AFTER DELETE ON clientmstr

FOR EACH ROW

BEGIN

-- Insert old values into audit\_trade table for deleted record

INSERT INTO audit\_trade (client\_id, old\_client\_name, old\_client\_email, old\_client\_phone, action\_type)

VALUES (OLD.client\_id, OLD.client\_name, OLD.client\_email, OLD.client\_phone, 'DELETE');

END $$

DELIMITER ;

```

### Explanation:

- `AFTER DELETE ON clientmstr`: This trigger fires \*\*after\*\* a delete operation on the `clientmstr` table.

- `FOR EACH ROW`: Similar to the update trigger, this executes once for each row that is deleted.

- `OLD.client\_id, OLD.client\_name, etc.`: These are the old values of the row that was deleted, which will be inserted into the `audit\_trade` table.

- The action type is labeled as `'DELETE'` to specify that this trigger is for a delete operation.

### 4. \*\*Statement-Level Trigger for `UPDATE`\*\*

A statement-level trigger executes only once for the entire statement (regardless of how many rows are affected). This trigger will log an action when any `UPDATE` operation occurs on the `clientmstr` table.

```sql

DELIMITER $$

CREATE TRIGGER track\_client\_update\_statement

AFTER UPDATE ON clientmstr

FOR EACH STATEMENT

BEGIN

-- Insert a record into audit\_trade for any update action

INSERT INTO audit\_trade (action\_type)

VALUES ('UPDATE');

END $$

DELIMITER ;

```

### Explanation:

- `AFTER UPDATE ON clientmstr`: This fires \*\*after\*\* an update operation on the table.

- `FOR EACH STATEMENT`: The trigger executes only once for the entire statement, regardless of how many rows are updated.

- This version does not store the old values, only that an update action occurred.

### 5. \*\*Statement-Level Trigger for `DELETE`\*\*

Similarly, a statement-level trigger will execute once for a delete operation on the `clientmstr` table.

```sql

DELIMITER $$

CREATE TRIGGER track\_client\_delete\_statement

AFTER DELETE ON clientmstr

FOR EACH STATEMENT

BEGIN

-- Insert a record into audit\_trade for any delete action

INSERT INTO audit\_trade (action\_type)

VALUES ('DELETE');

END $$

DELIMITER ;

```

### Explanation:

- `AFTER DELETE ON clientmstr`: This triggers \*\*after\*\* a delete operation on the `clientmstr` table.

- `FOR EACH STATEMENT`: This ensures the trigger only executes once for the entire delete statement.

- It only logs that a delete operation occurred but does not log the old values.

### Summary of Trigger Types:

- \*\*Row-Level Triggers\*\*:

- These are useful when you want to log specific details about each row affected by the `UPDATE` or `DELETE` operation. It gives you granular control over the actions being performed, such as capturing the old values of updated or deleted records.

- \*\*Statement-Level Triggers\*\*:

- These are useful when you just need to track that an action (like `UPDATE` or `DELETE`) has occurred, without the need to track specific details of each individual row affected.

### Testing the Triggers:

1. \*\*Test the Row-Level Update Trigger\*\*:

Execute an update statement on the `clientmstr` table:

```sql

UPDATE clientmstr

SET client\_name = 'Jane Doe'

WHERE client\_id = 1;

```

After running this update, check the `audit\_trade` table to see the old values for the updated record.

```sql

SELECT \* FROM audit\_trade;

```

2. \*\*Test the Row-Level Delete Trigger\*\*:

Execute a delete statement on the `clientmstr` table:

```sql

DELETE FROM clientmstr

WHERE client\_id = 1;

```

After running this delete, check the `audit\_trade` table to see the old values for the deleted record.

```sql

SELECT \* FROM audit\_trade;

```

3. \*\*Test the Statement-Level Update Trigger\*\*:

Execute an update on multiple records:

```sql

UPDATE clientmstr

SET client\_name = 'Updated Name';

```

Check the `audit\_trade` table to see the action type (`UPDATE`) but without the old values.

```sql

SELECT \* FROM audit\_trade;

```

4. \*\*Test the Statement-Level Delete Trigger\*\*:

Execute a delete statement on multiple records:

```sql

DELETE FROM clientmstr;

```

Check the `audit\_trade` table to see the action type (`DELETE`).

```sql

SELECT \* FROM audit\_trade;

```

### Conclusion:

- \*\*Row-level triggers\*\* allow you to capture and store detailed information about each record that is updated or deleted.

- \*\*Statement-level triggers\*\* are simpler and only track whether an action (such as update or delete) was performed, without storing the old values of the affected rows.

By using these triggers effectively, you can maintain an audit trail of changes in your database for tracking and security purposes.