Triggers

Triggers

- It is a special type of stored procedure that is invoked automatically in response to an event. Each trigger is associated with a table, which is activated on any DML statement such as INSERT, UPDATE, or DELETE.

- A trigger is called a special procedure because it cannot be called directly like a stored procedure. The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.

Types:

1. Before Insert: It is activated before the insertion of data into the table.
2. After Insert: It is activated after the insertion of data into the table.
3. Before Update: It is activated before the update of data in the table.
4. After Update: It is activated after the update of the data in the table.
5. Before Delete: It is activated before the data is removed from the table.
6. After Delete: It is activated after the deletion of data from the table.

- When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.

Triggers

Syntax:

```
DELIMITER //
CREATE TRIGGER trigger_name
(AFTER | BEFORE) (INSERT UPDATE |
DELETE)  ON table_name
FOR EACH ROW
BEGIN
        --variable declarations
        --trigger code
END;
```

trigger_event: It is the type of operation name that activates the trigger. It can be either INSERT, UPDATE, or DELETE operation.

table_name: It is the name of the table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.

trigger_name:
It is the name of the trigger that we want to create. It must be written after the CREATE TRIGGER statement.

trigger_time:
It is the trigger action time, which should be either BEFORE or AFTER. It is the required parameter while defining a trigger. It indicates that the trigger will be invoked before or after each row modification occurs on the table.

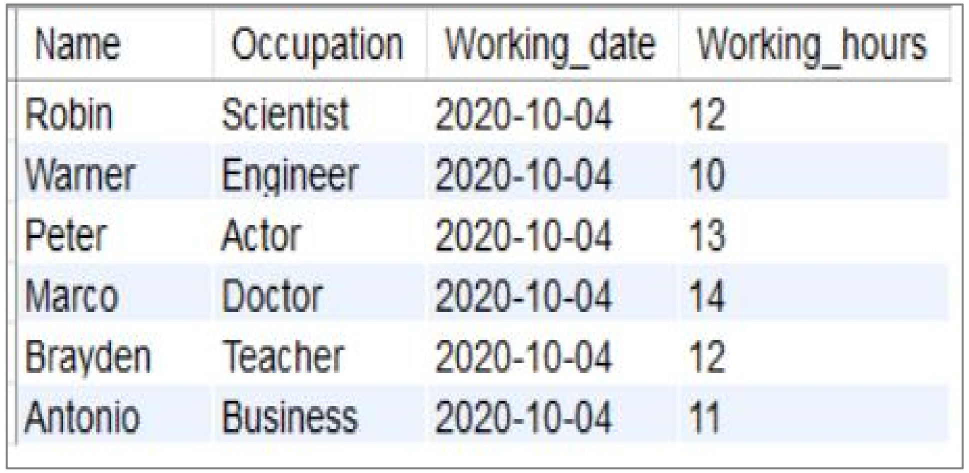
- ✓ The NEW and OLD modifiers are used to distinguish the column values BEFORE and AFTER the execution of the DML statement.
- ✓ We can use the column name with NEW and OLD modifiers as OLD.col_name and NEW.col_name.
- ✓ The OLD.column_name indicates the column of an existing row before the updation or deletion occurs.
- ✓ NEW.col_name indicates the column of a new row that will be inserted or an existing row after it is updated.

Example – Before Insert Trigger

In employee table, using this trigger, we make sure not to insert any negative values in Working_hours column. If any such value appears then trigger should change it to zero.

```
DELIMITER &&
CREATE TRIGGER before_insert_empworkinghours
BEFORE INSERT ON employee FOR EACH ROW
BEGIN
IF NEW.working_hours < 0 THEN SET NEW.working_hours = 0;
END IF;
END &&
```

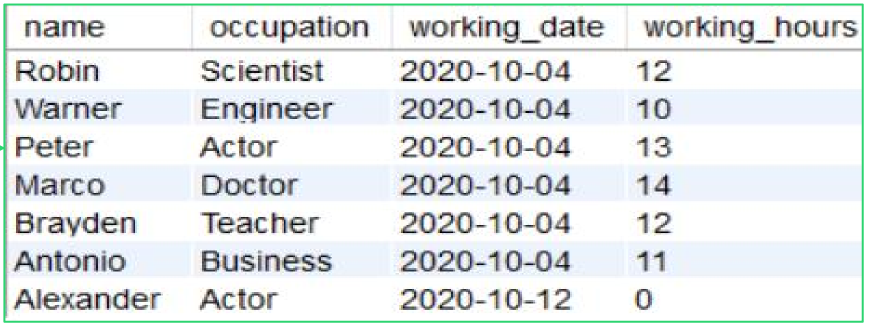
Employee (Original)



Name	Occupation	Working_date	Working_hours
Robin	Scientist	2020-10-04	12
Warner	Engineer	2020-10-04	10
Peter	Actor	2020-10-04	13
Marco	Doctor	2020-10-04	14
Brayden	Teacher	2020-10-04	12
Antonio	Business	2020-10-04	11

```
INSERT INTO employee VALUES
('Alexander', 'Actor', '2020-10-12', -13);
```

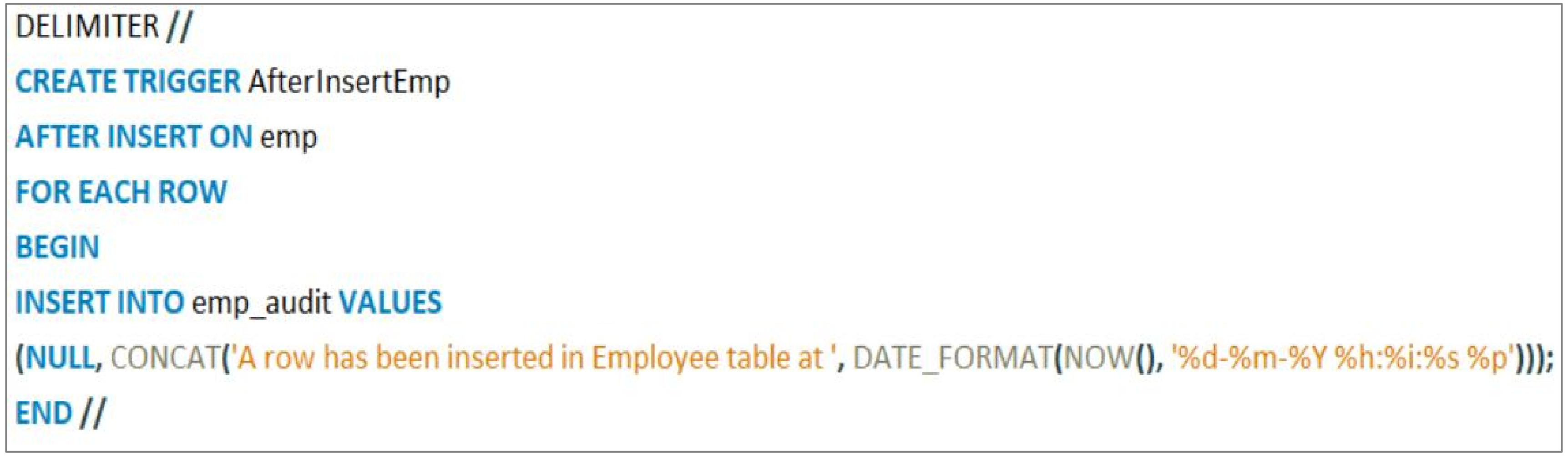
Employee (Output)



name	occupation	working_date	working_hours
Robin	Scientist	2020-10-04	12
Warner	Engineer	2020-10-04	10
Peter	Actor	2020-10-04	13
Marco	Doctor	2020-10-04	14
Brayden	Teacher	2020-10-04	12
Antonio	Business	2020-10-04	11
Alexander	Actor	2020-10-12	0

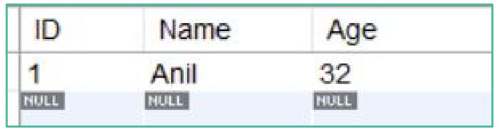
Example – After Insert Trigger

If any employee information is inserted in emp table then trigger is inserting the row in emp_audit table automatically.



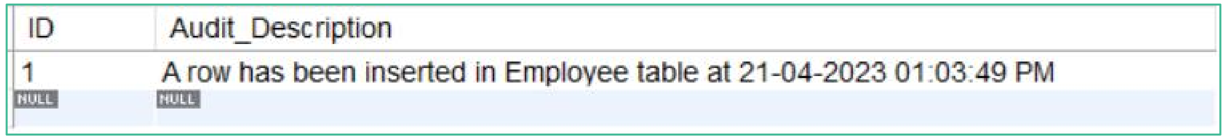
```
DELIMITER //
CREATE TRIGGER AfterInsertEmp
AFTER INSERT ON emp
FOR EACH ROW
BEGIN
INSERT INTO emp_audit VALUES
(NULL, CONCAT('A row has been inserted in Employee table at ', DATE_FORMAT(NOW(), '%d-%m-%Y %h:%i:%s %p')));
END //
```

Emp



ID	Name	Age
1	Anil	32
NULL	NULL	NULL

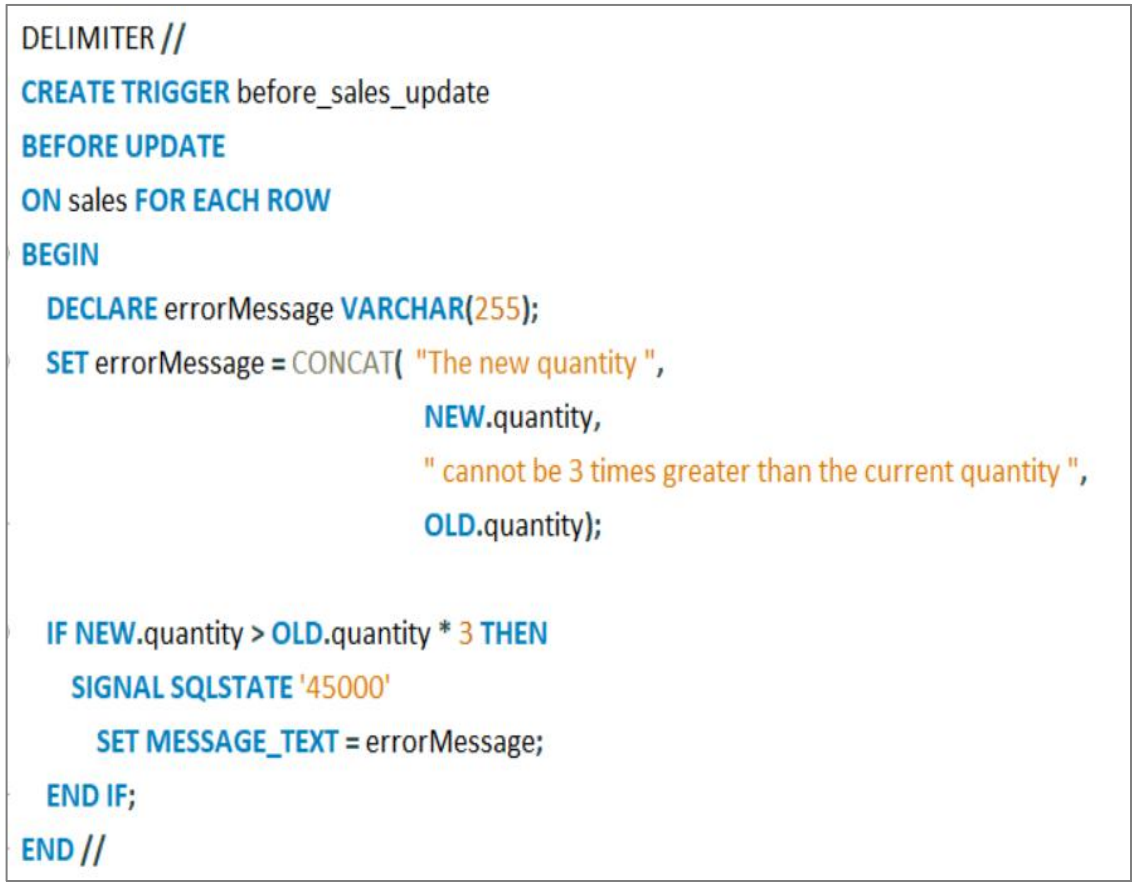
Emp_audit



ID	Audit_Description
1	A row has been inserted in Employee table at 21-04-2023 01:03:49 PM
NULL	NULL

Example – Before Update Trigger

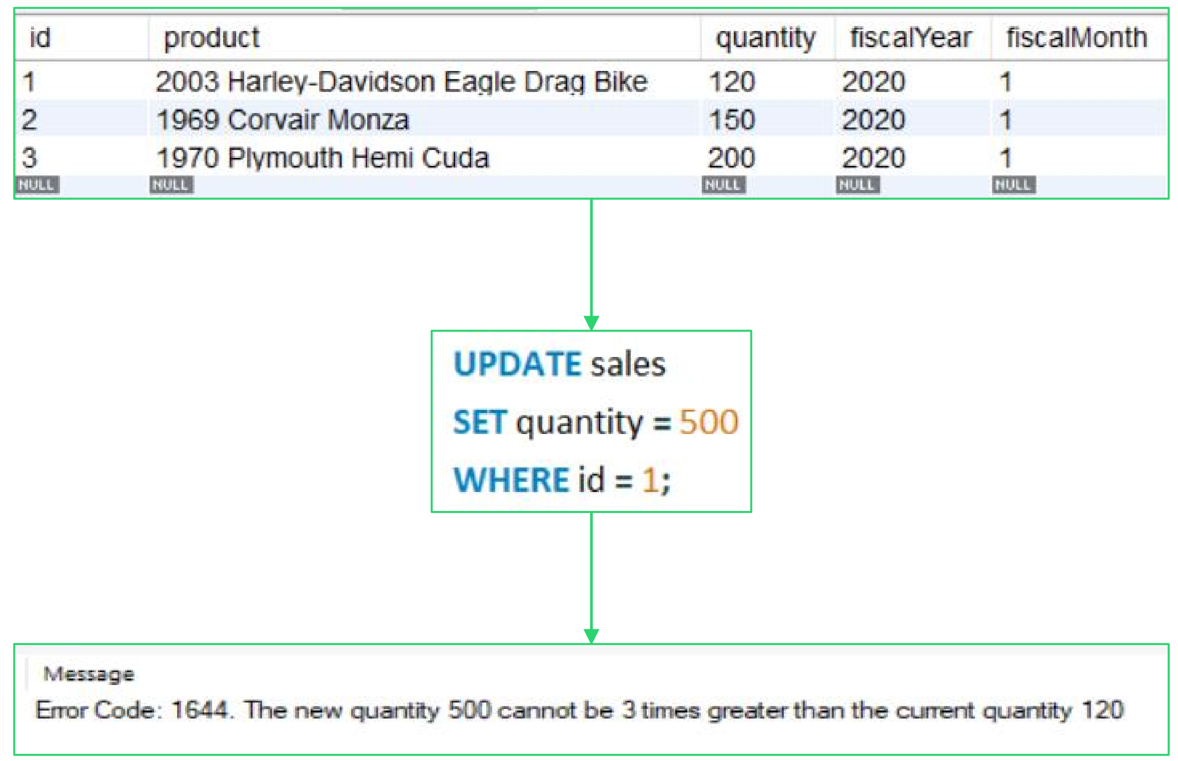
If a new quantity value is more than 3 times the current quantity value for any product then trigger is raising an exception with an error message in our own words.



```
DELIMITER //
CREATE TRIGGER before_sales_update
BEFORE UPDATE
ON sales FOR EACH ROW
BEGIN
  DECLARE errorMessage VARCHAR(255);
  SET errorMessage = CONCAT( "The new quantity ",
                             NEW.quantity,
                             " cannot be 3 times greater than the current quantity ",
                             OLD.quantity);

  IF NEW.quantity > OLD.quantity * 3 THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = errorMessage;
  END IF;
END //
```

Sales



id	product	quantity	fiscalYear	fiscalMonth
1	2003 Harley-Davidson Eagle Drag Bike	120	2020	1
2	1969 Corvair Monza	150	2020	1
3	1970 Plymouth Hemi Cuda	200	2020	1
NULL	NULL	NULL	NULL	NULL

```
UPDATE sales
SET quantity = 500
WHERE id = 1;
```

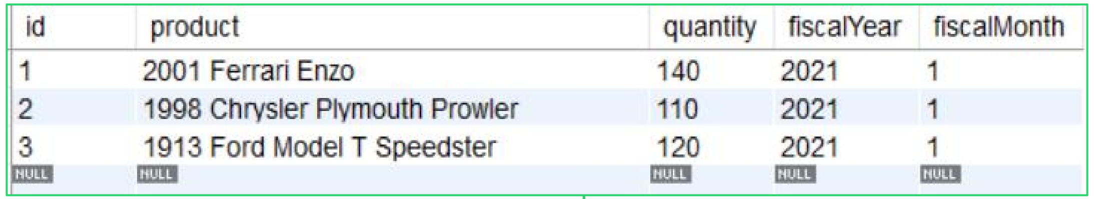
```
Message
Error Code: 1644. The new quantity 500 cannot be 3 times greater than the current quantity 120
```

Example – After Update Trigger

This trigger keeps the history of all the changed quantities and old quantities, being updated over the period of time, in SalesChanges table and updates the values in original table.

```
DELIMITER $$
CREATE TRIGGER after_sales_update
AFTER UPDATE
ON sales FOR EACH ROW
BEGIN
  IF OLD.quantity <> NEW.quantity THEN
    INSERT INTO SalesChanges(salesId,beforeQuantity, afterQuantity)
    VALUES(old.id, old.quantity, new.quantity);
  END IF;
END$$
```

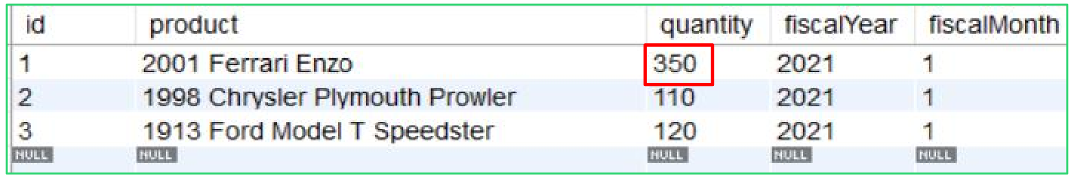
Sales (Before update)



id	product	quantity	fiscalYear	fiscalMonth
1	2001 Ferrari Enzo	140	2021	1
2	1998 Chrysler Plymouth Prowler	110	2021	1
3	1913 Ford Model T Speedster	120	2021	1
NULL	NULL	NULL	NULL	NULL

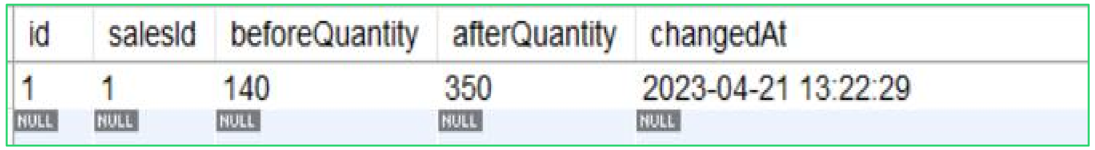
```
UPDATE Sales
SET quantity = 350
WHERE id = 1;
```

Sales (After update)



id	product	quantity	fiscalYear	fiscalMonth
1	2001 Ferrari Enzo	350	2021	1
2	1998 Chrysler Plymouth Prowler	110	2021	1
3	1913 Ford Model T Speedster	120	2021	1
NULL	NULL	NULL	NULL	NULL

SalesChanges

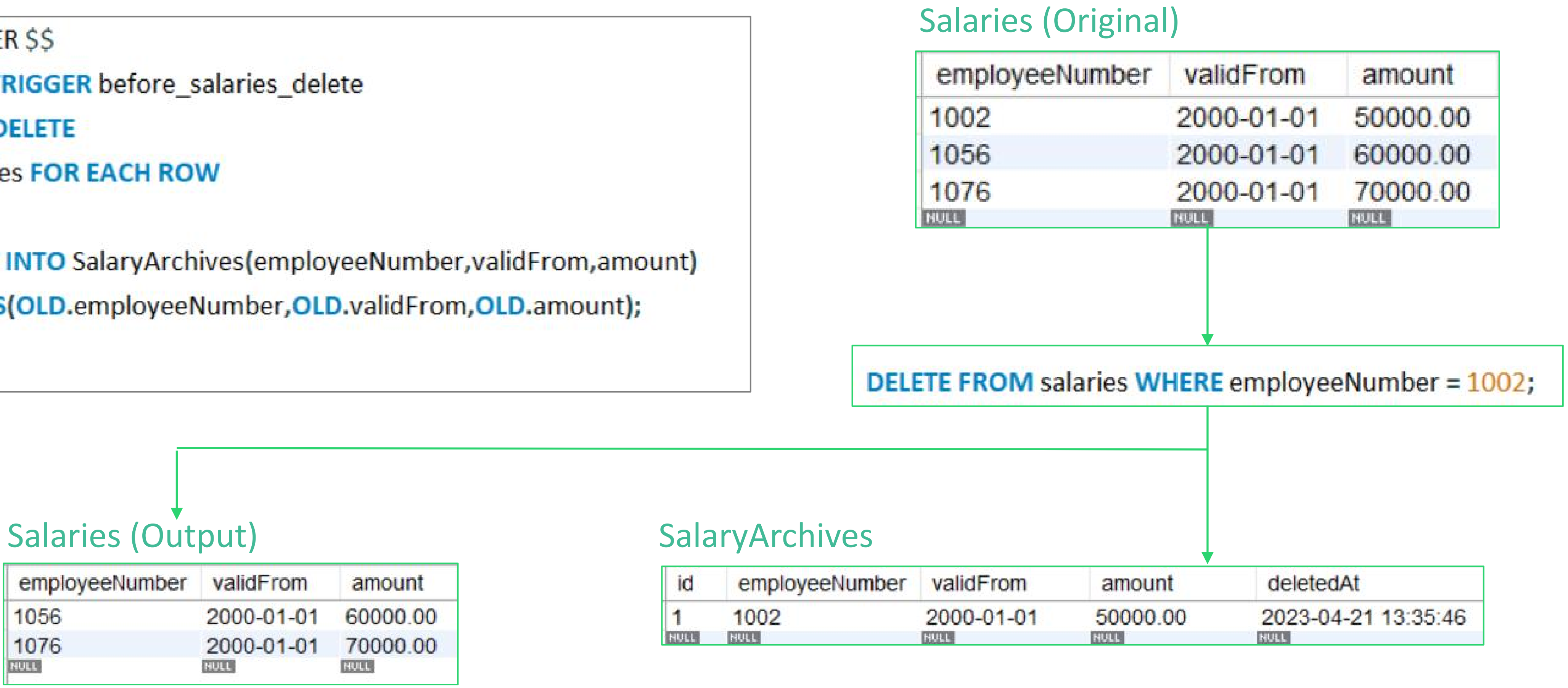


id	salesId	beforeQuantity	afterQuantity	changedAt
1	1	140	350	2023-04-21 13:22:29
NULL	NULL	NULL	NULL	NULL

Example – Before Delete Trigger

This trigger, before removing employee information, takes the back up of same employee in another table and then deletes it from main table.

```
DELIMITER $$
CREATE TRIGGER before_salaries_delete
BEFORE DELETE
ON salaries FOR EACH ROW
BEGIN
   INSERT INTO SalaryArchives(employeeNumber,validFrom,amount)
   VALUES(OLD.employeeNumber,OLD.validFrom,OLD.amount);
END$$
```



Salaries (Original)

employeeNumber	validFrom	amount
1002	2000-01-01	50000.00
1056	2000-01-01	60000.00
1076	2000-01-01	70000.00
NULL	NULL	NULL

```
DELETE FROM salaries WHERE employeeNumber = 1002;
```

Salaries (Output)

employeeNumber	validFrom	amount
1056	2000-01-01	60000.00
1076	2000-01-01	70000.00
NULL	NULL	NULL

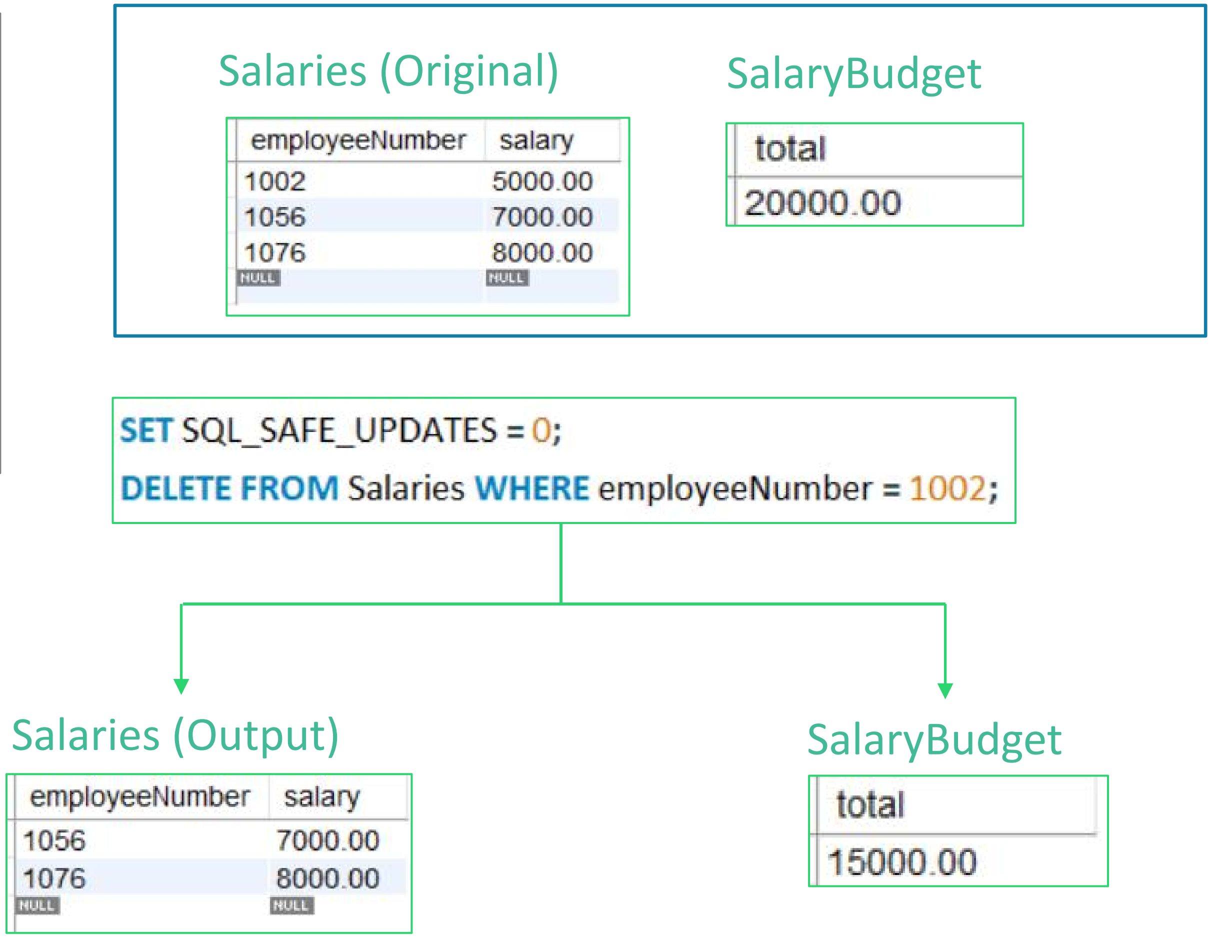
SalaryArchives

id	employeeNumber	validFrom	amount	deletedAt
1	1002	2000-01-01	50000.00	2023-04-21 13:35:46
NULL	NULL	NULL	NULL	NULL

Example – After Delete Trigger

This trigger changes the budget of a company when any employee leaves the company. First Employee is deleted from main table and then budget changes.

```
DELIMITER //
CREATE TRIGGER after_salaries_delete
AFTER DELETE
ON Salaries FOR EACH ROW
BEGIN
UPDATE SalaryBudgets
SET total = total - old.salary;
END //
```



Salaries (Original)

employeeNumber	salary
1002	5000.00
1056	7000.00
1076	8000.00
NULL	NULL

SalaryBudget

total
20000.00

```
SET SQL_SAFE_UPDATES = 0;
DELETE FROM Salaries WHERE employeeNumber = 1002;
```

Salaries (Output)

employeeNumber	salary
1056	7000.00
1076	8000.00
NULL	NULL

SalaryBudget

total
15000.00