Dipankar Raman

SQL TRIGGERS AND EVENTS

Introduction to Triggers and Events

Triggers and events are two types of database objects in SQL that automate and manage database operations.

- Triggers are automatic actions executed in response to specific events (INSERT, UPDATE, DELETE) on a table or view.
- Events are scheduled tasks that run at predetermined times or intervals.

Purpose

- Triggers are used for tasks such as enforcing business rules, maintaining audit logs, and validating data.
- Events are employed for scheduling and automating tasks like data archiving, regular updates, and maintenance activities.

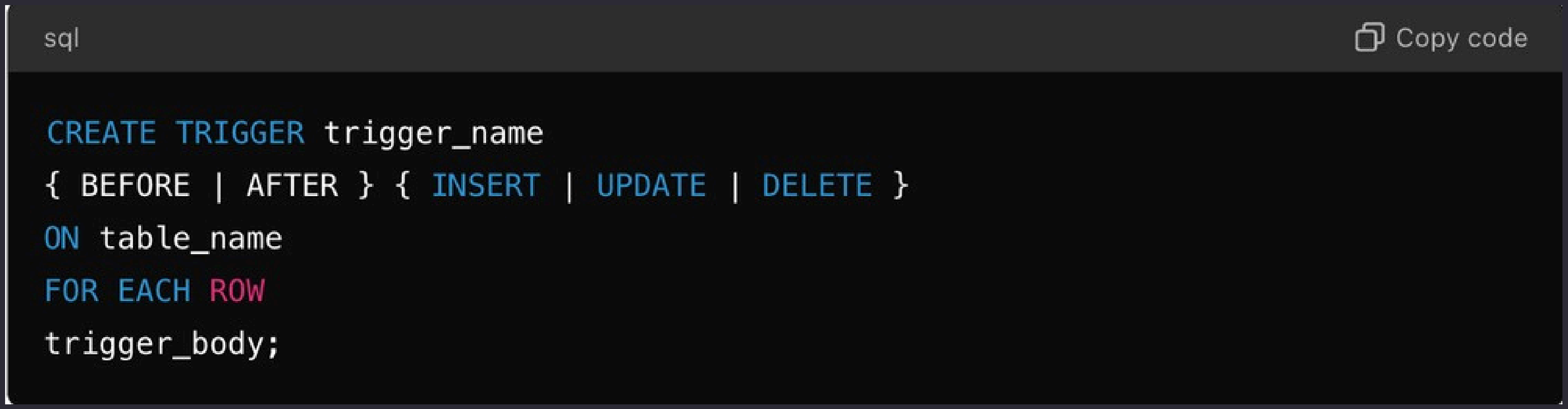
Triggers

A trigger is a stored procedure that is automatically executed in response to certain events on a specific table or view. Triggers help in managing data integrity and consistency without requiring explicit calls from application code.

Types of Triggers

- **BEFORE Trigger:** Executes before an insert, update, or delete operation. This allows modifications to be made to data before the actual operation takes place.
- **AFTER Trigger:** Executes after an insert, update, or delete operation. It is useful for tasks that need to occur only after the data has been modified, such as logging changes.

Syntax of Trigger



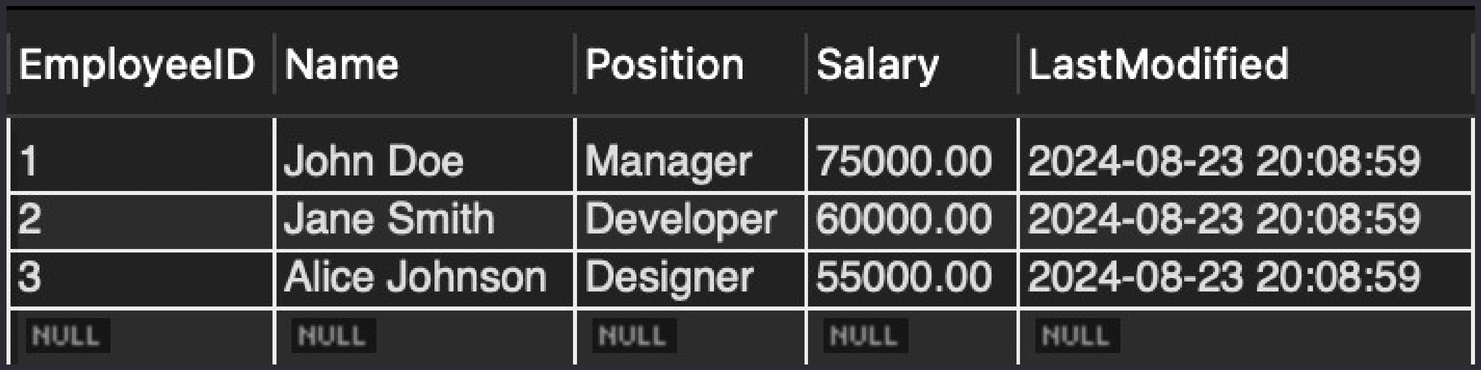
```sql
CREATE TRIGGER trigger_name
{ BEFORE | AFTER } { INSERT | UPDATE | DELETE }
ON table_name
FOR EACH ROW
trigger_body;
```

- trigger_name: The name of the trigger.
- BEFORE | AFTER: Defines when the trigger should be executed (before or after the operation).
- INSERT | UPDATE | DELETE: The type of operation that causes the trigger to fire.
- table_name: The table to which the trigger is attached.
- trigger_body: The SQL statements that are executed when the trigger fires.

Demo Dataset

Let's walk through the explanation of triggers and events using two demo datasets: ‘Employee’ and ‘Sales’ .

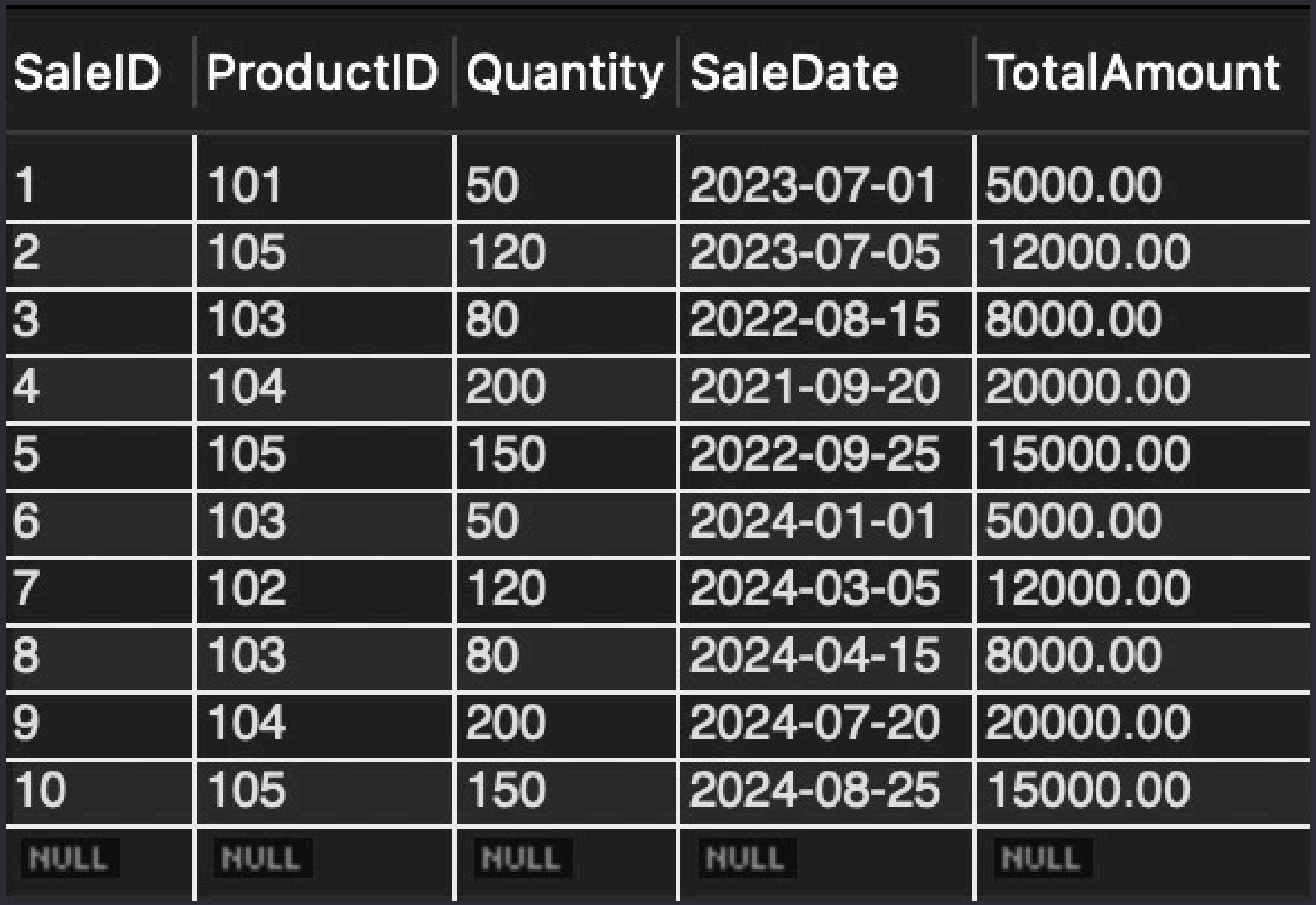
1.Employee Table



EmployeeID	Name	Position	Salary	LastModified
1	John Doe	Manager	75000.00	2024-08-23 20:08:59
2	Jane Smith	Developer	60000.00	2024-08-23 20:08:59
3	Alice Johnson	Designer	55000.00	2024-08-23 20:08:59
NULL	NULL	NULL	NULL	NULL

- **EmployeeID:** Unique identifier for each employee.
- **Name:** Name of the employee.
- **Position:** Job position of the employee.
- **Salary:** Salary of the employee.
- **LastModified:** Timestamp indicating the last modification time.

2.Sales Table



SaleID	ProductID	Quantity	SaleDate	TotalAmount
1	101	50	2023-07-01	5000.00
2	105	120	2023-07-05	12000.00
3	103	80	2022-08-15	8000.00
4	104	200	2021-09-20	20000.00
5	105	150	2022-09-25	15000.00
6	103	50	2024-01-01	5000.00
7	102	120	2024-03-05	12000.00
8	103	80	2024-04-15	8000.00
9	104	200	2024-07-20	20000.00
10	105	150	2024-08-25	15000.00
NULL	NULL	NULL	NULL	NULL

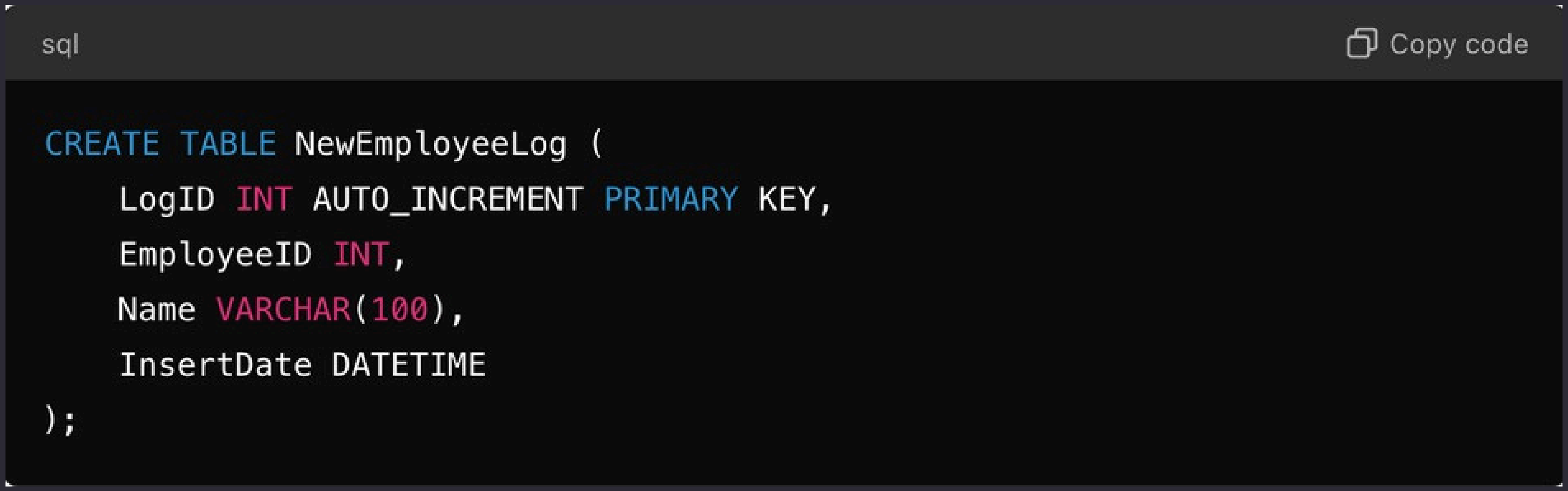
- **SaleID:** Unique identifier for each sale.
- **ProductID:** Identifier for the product being sold.
- **Quantity:** Quantity of the product sold.
- **SaleDate:** Date of the sale.
- **TotalAmount:** Total amount of the sale.

Example of Triggers:

1.INSERT Trigger:

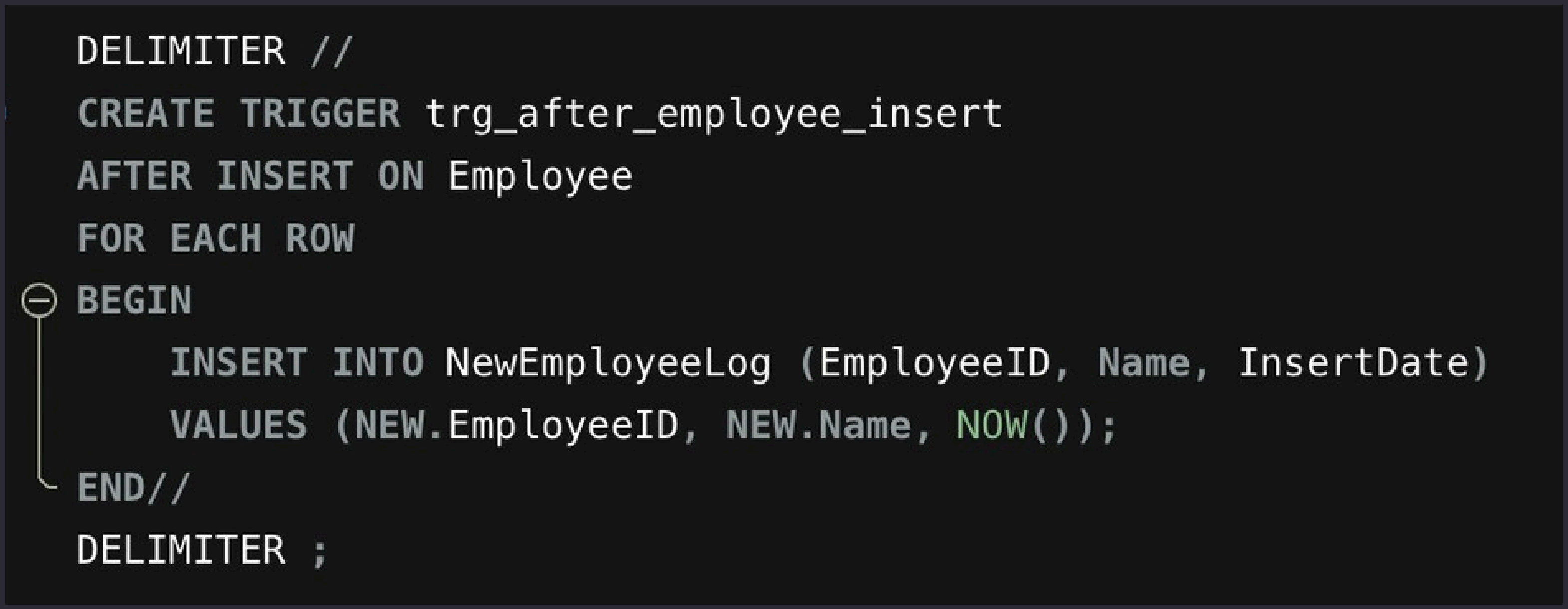
Problem Statement : Automatically Log Employee Insertions
Whenever a new employee is added to the Employee table, a log entry should be created in a NewEmployeeLog table that records the EmployeeID, Name, and InsertDate (the date when the record was inserted).
Solution: Create a Trigger for INSERT on Employee

1.Create the 'NewEmployeeLog' Table



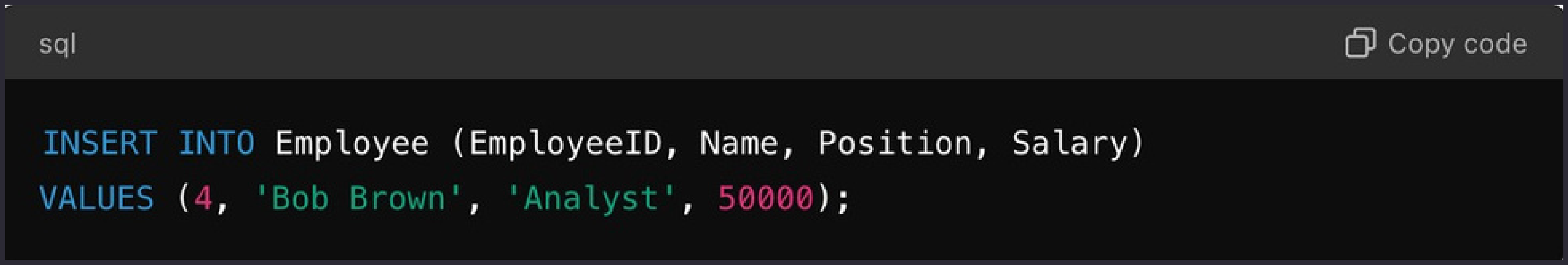
```sql
CREATE TABLE NewEmployeeLog (
    LogID INT AUTO_INCREMENT PRIMARY KEY,
    EmployeeID INT,
    Name VARCHAR(100),
    InsertDate DATETIME
);
```

2.Create the Trigger



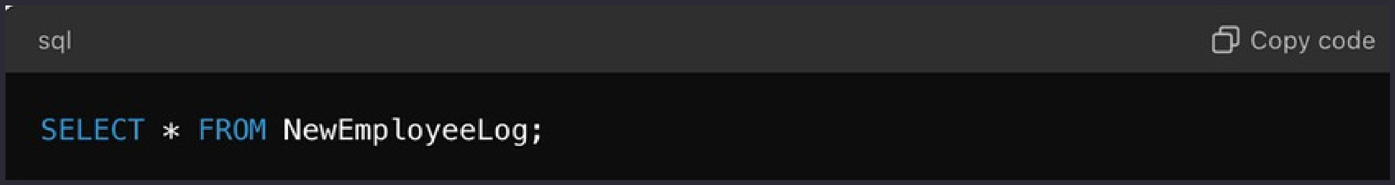
```sql
DELIMITER //
CREATE TRIGGER trg_after_employee_insert
AFTER INSERT ON Employee
FOR EACH ROW
BEGIN
    INSERT INTO NewEmployeeLog (EmployeeID, Name, InsertDate)
    VALUES (NEW.EmployeeID, NEW.Name, NOW());
END//
DELIMITER ;
```

3.Test the Trigger by Inserting a New Employee

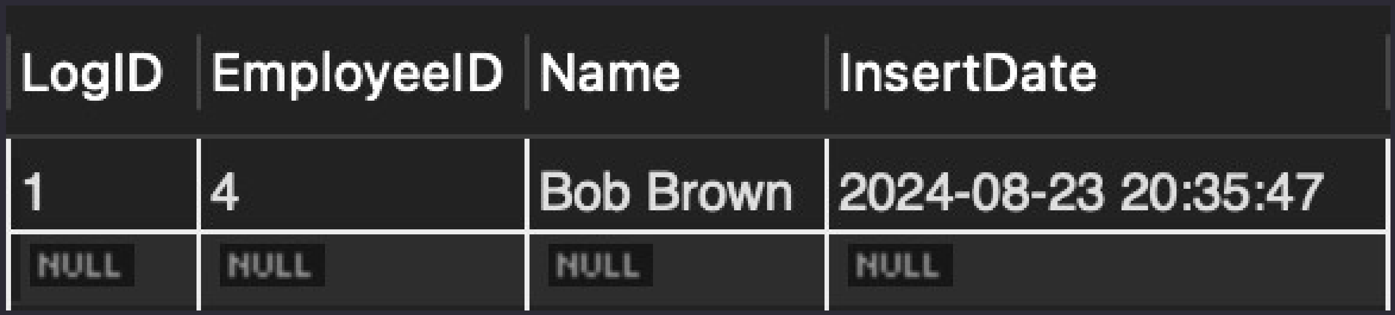


```sql
INSERT INTO Employee (EmployeeID, Name, Position, Salary)
VALUES (4, 'Bob Brown', 'Analyst', 50000);
```

4.Output of NewEmployeeLog Table After the Insert



```sql
SELECT * FROM NewEmployeeLog;
```



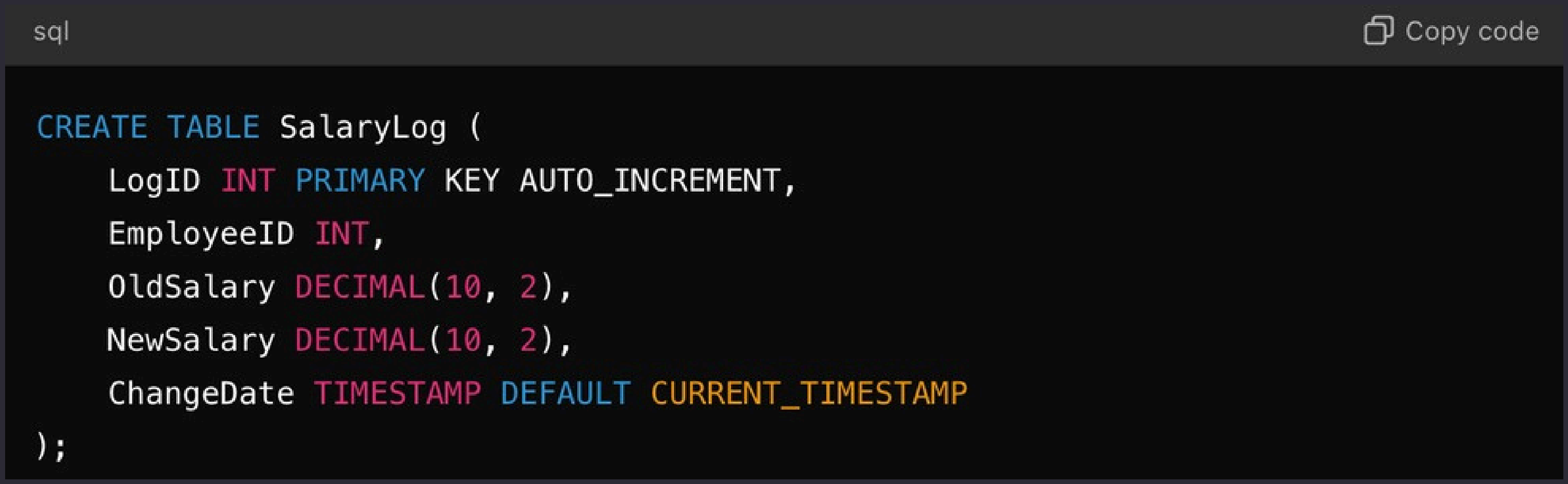
LogID	EmployeeID	Name	InsertDate
1	4	Bob Brown	2024-08-23 20:35:47
NULL	NULL	NULL	NULL

Explanation: After inserting a new employee, the NewEmployeeLog table is automatically populated with the employee's details.

2.UPDATE Trigger:

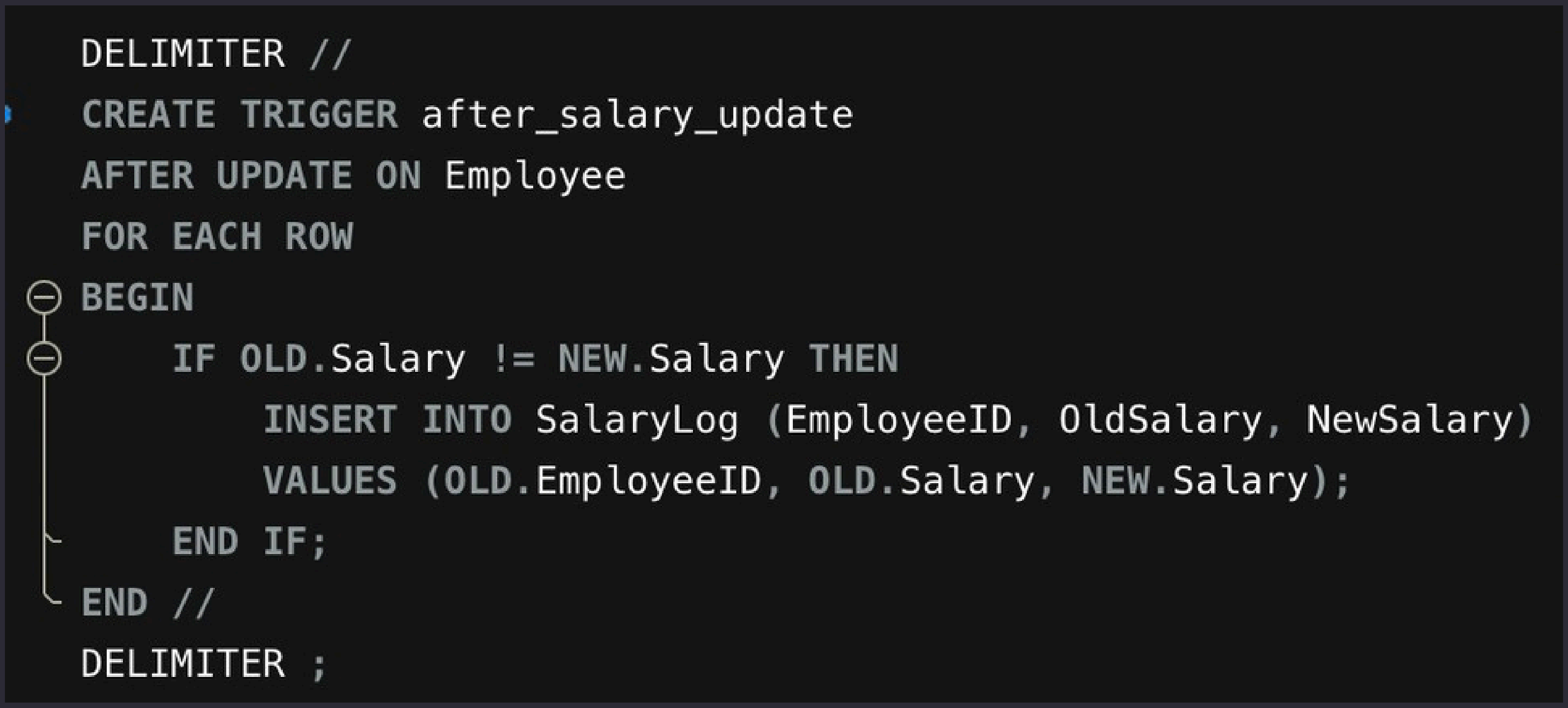
Problem Statement : Suppose you want to keep a log of any changes made to the Salary field in the Employee table.
Solution: Logging Salary Changes in the Employee Table

1.Create the 'SalaryLog' Table



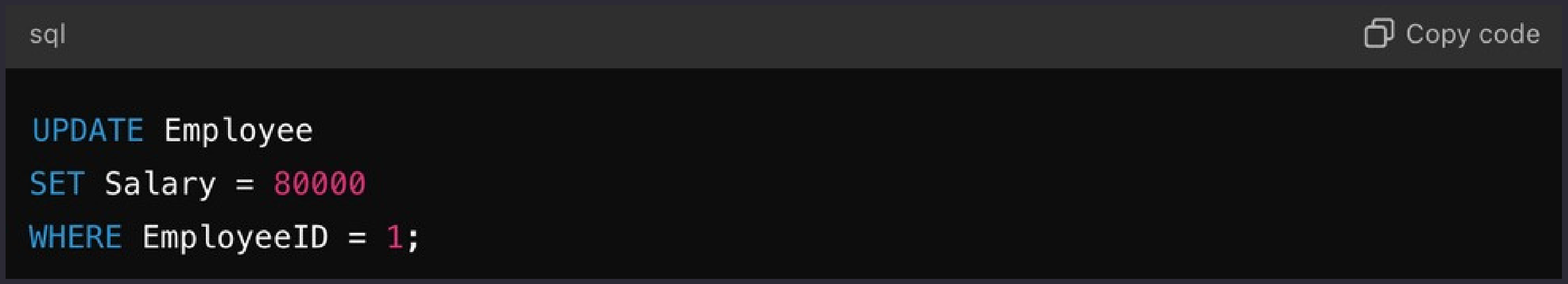
```sql
CREATE TABLE SalaryLog (
    LogID INT PRIMARY KEY AUTO_INCREMENT,
    EmployeeID INT,
    OldSalary DECIMAL(10, 2),
    NewSalary DECIMAL(10, 2),
    ChangeDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2.Create the Trigger



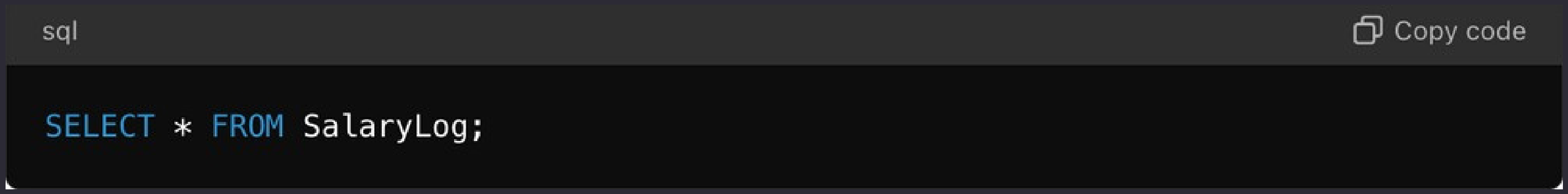
```sql
DELIMITER //
CREATE TRIGGER after_salary_update
AFTER UPDATE ON Employee
FOR EACH ROW
BEGIN
    IF OLD.Salary != NEW.Salary THEN
        INSERT INTO SalaryLog (EmployeeID, OldSalary, NewSalary)
        VALUES (OLD.EmployeeID, OLD.Salary, NEW.Salary);
    END IF;
END //
DELIMITER ;
```

3.Test the Trigger by Updating Employee Salary

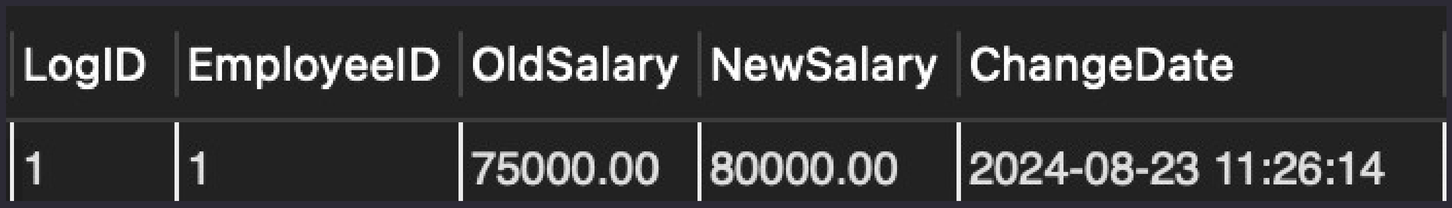


```sql
UPDATE Employee
SET Salary = 80000
WHERE EmployeeID = 1;
```

4.Output of NewEmployeeLog Table After the Insert



```sql
SELECT * FROM SalaryLog;
```



LogID	EmployeeID	OldSalary	NewSalary	ChangeDate
1	1	75000.00	80000.00	2024-08-23 11:26:14

Explanation:

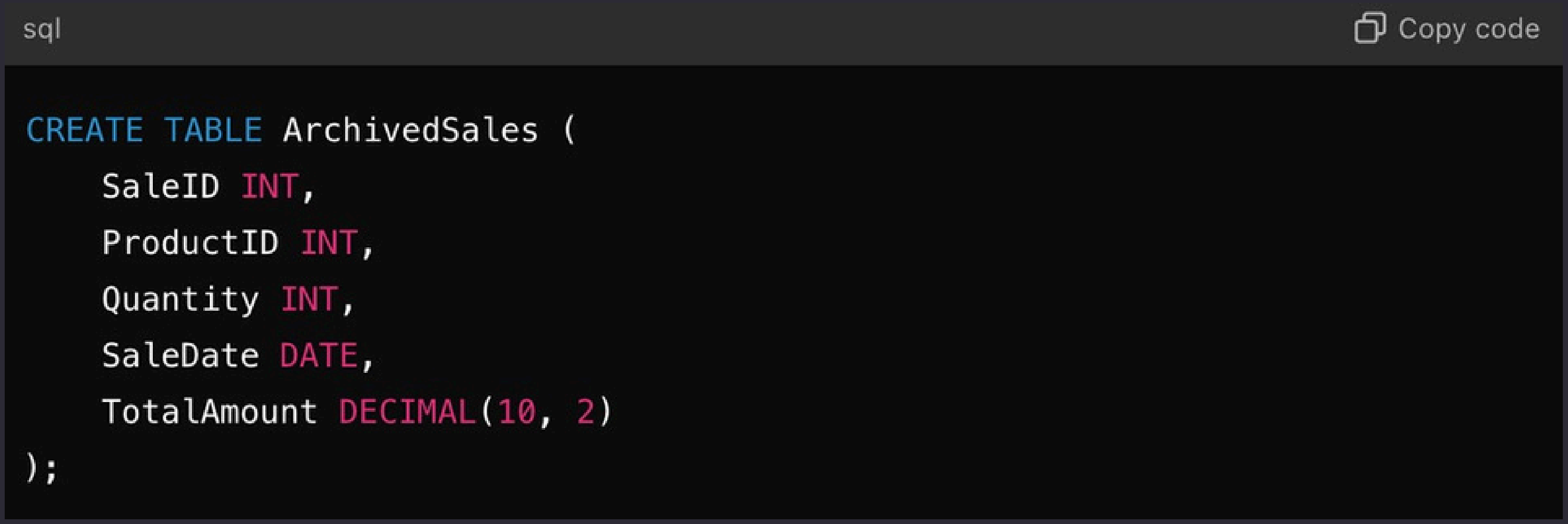
- **SalaryLog Table: This table will store the logs of salary changes.**
- **Trigger: The after_salary_update trigger will activate after any update on the Employee table. If the Salary field is modified, it inserts the old and new salary values into the SalaryLog table.**

3.DELETE Trigger:

Problem Statement : Whenever a record is deleted from the Sales table, it should automatically be archived into the ArchivedSales table. This helps keep a backup of all sales records even after they are deleted.

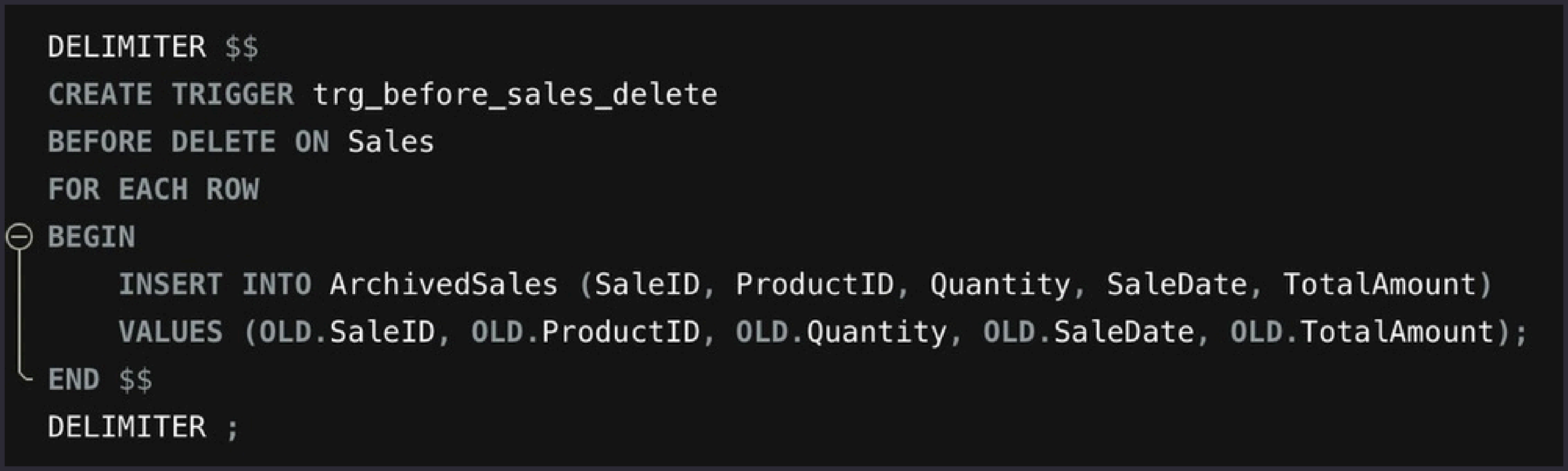
Solution: Create a Trigger for DELETE on Sales

1.Create the ‘ArchivedSales’ Table (If Not Already Created)



```sql
CREATE TABLE ArchivedSales (
    SaleID INT,
    ProductID INT,
    Quantity INT,
    SaleDate DATE,
    TotalAmount DECIMAL(10, 2)
);
```

2.Create the Trigger



```sql
DELIMITER $$
CREATE TRIGGER trg_before_sales_delete
BEFORE DELETE ON Sales
FOR EACH ROW
BEGIN
    INSERT INTO ArchivedSales (SaleID, ProductID, Quantity, SaleDate, TotalAmount)
    VALUES (OLD.SaleID, OLD.ProductID, OLD.Quantity, OLD.SaleDate, OLD.TotalAmount);
END $$
DELIMITER ;
```

3.Test the Trigger by Deleting a Sale Record

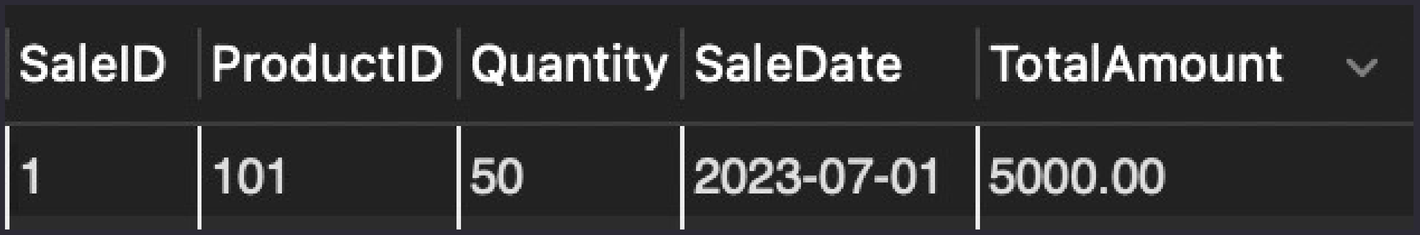
sql Copy code

```sql
DELETE FROM Sales WHERE SaleID = 1;
```

4.Output of NewEmployeeLog Table After the Insert

sql Copy code

```sql
SELECT * FROM ArchivedSales;
```

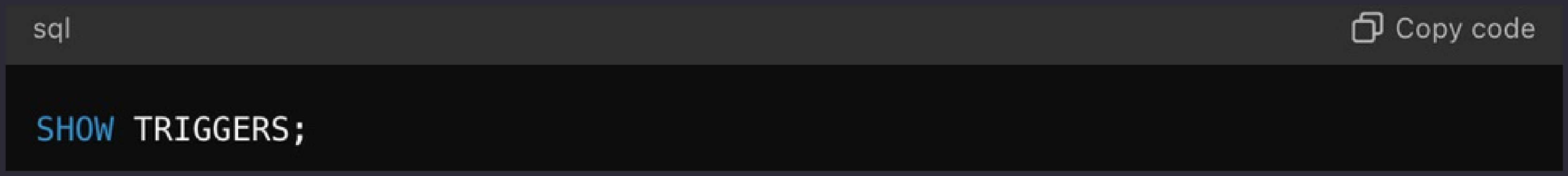


SaleID	ProductID	Quantity	SaleDate	TotalAmount
1	101	50	2023-07-01	5000.00

Explanation: When a sale record is deleted, it is automatically moved to the ArchivedSales table, ensuring that the data is not lost.

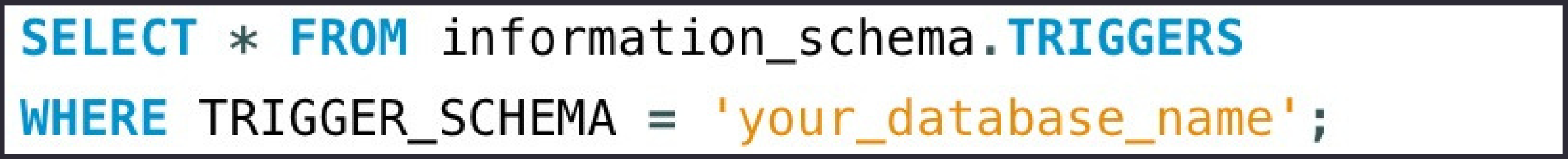
Managing Triggers

- **Viewing Existing Triggers:** You can list all triggers in your database using the SHOW TRIGGERS command.



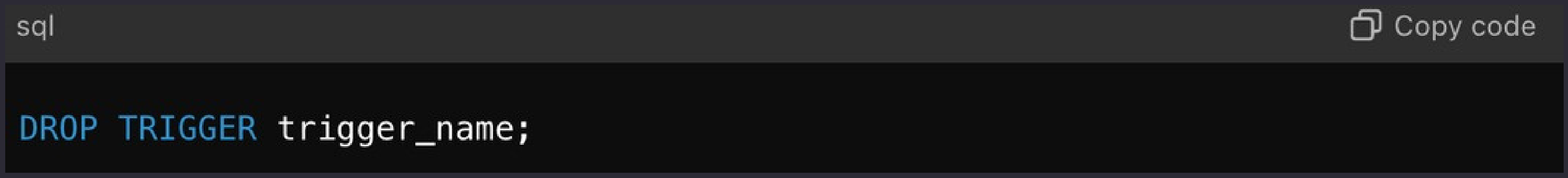
```sql
SHOW TRIGGERS;
```

You can also query the information_schema.TRIGGERS table for more detailed information about the triggers.



```sql
SELECT * FROM information_schema.TRIGGERS
WHERE TRIGGER_SCHEMA = 'your_database_name';
```

- **Dropping or Modifying Triggers:** To remove a trigger, use the DROP TRIGGER command.



```sql
DROP TRIGGER trigger_name;
```

Uses of Triggers

Triggers are used in a variety of scenarios, including but not limited to:

- Enforcing Business Rules: Ensure that certain business logic is applied automatically, such as preventing negative stock levels.
- Maintaining Audit Trails: Automatically log changes to critical data, such as tracking salary changes or recording who modified a record.
- Validating Data: Automatically enforce data validation rules, such as ensuring that an email address is unique before inserting a new customer.
- Synchronizing Tables: Automatically update related tables when changes occur, such as cascading updates to maintain referential integrity.
- Automating System Tasks: Perform system maintenance tasks automatically, like archiving old records or generating summaries.

Events

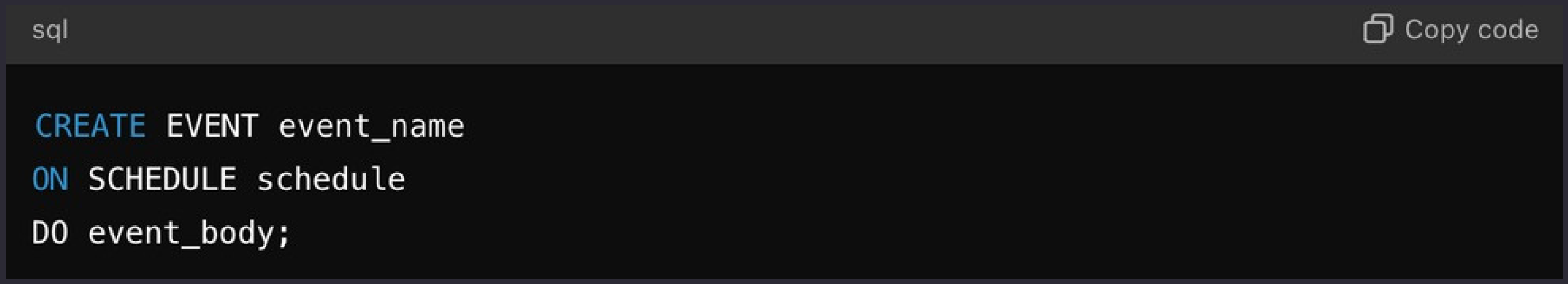
An event is a scheduled task that runs automatically at specified intervals or times, managed by the MySQL event scheduler.

Types of Events

- **One-time Event:** A one-time event is executed once at a specified date and time.
  Example: Automatically archiving records at midnight on a specific date.

- **Recurring Event:** A recurring event is executed repeatedly at a specified interval, such as every day, week, or month.
  Example: Deleting old log entries every day at midnight.

Syntax of Event



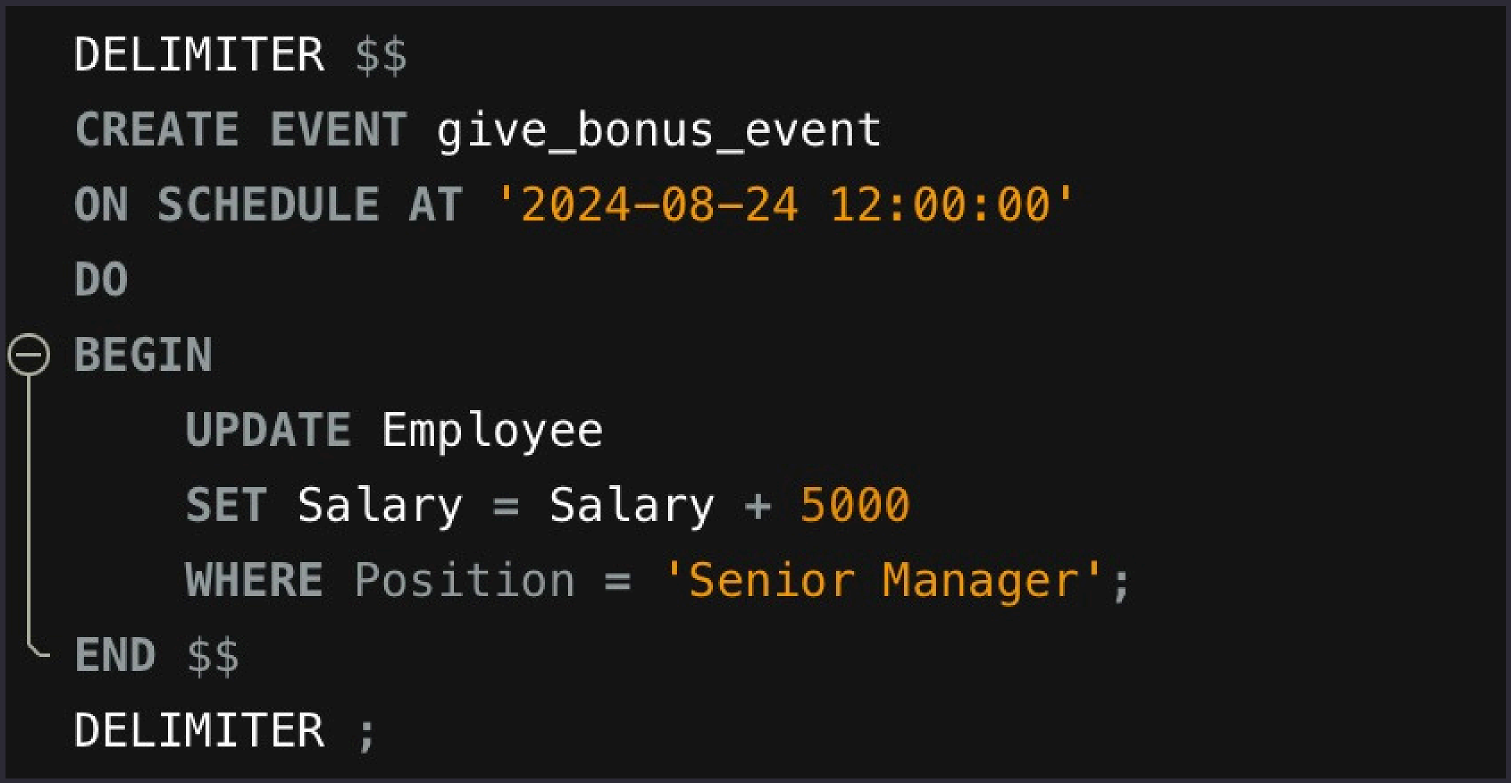
```sql
CREATE EVENT event_name
ON SCHEDULE schedule
DO event_body;
```

Example of Events:

1.One-Time Event to Give a Bonus

Problem Statement : The company wants to give a one-time bonus of $5,000 to all employees currently in the "Senior Manager" position.
Solution:

1.Create the One-Time Event

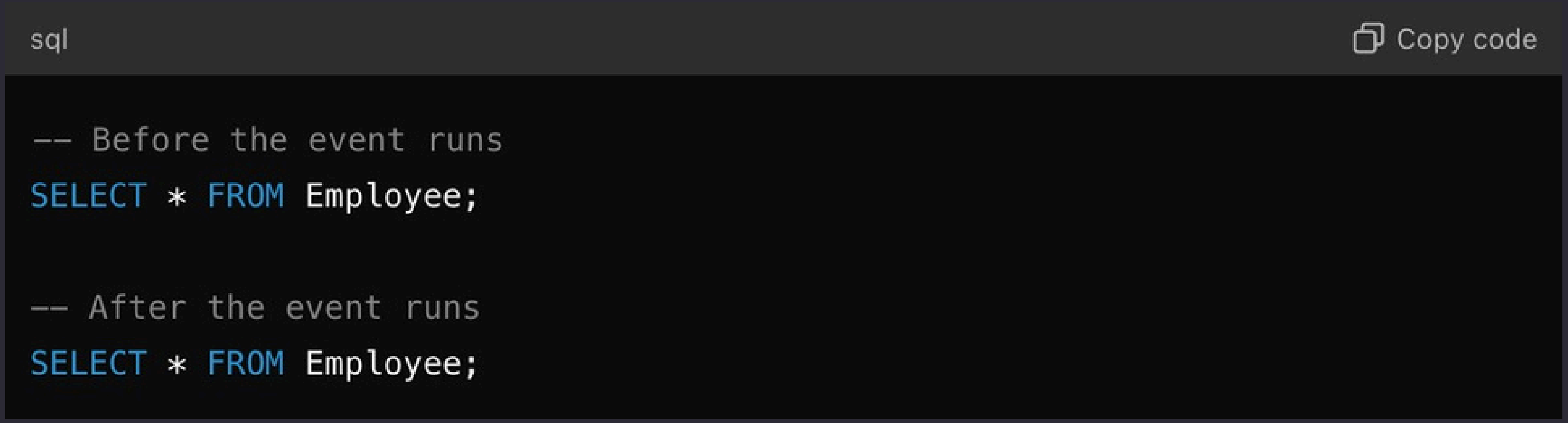


```sql
DELIMITER $$
CREATE EVENT give_bonus_event
ON SCHEDULE AT '2024-08-24 12:00:00'
DO
BEGIN
    UPDATE Employee
    SET Salary = Salary + 5000
    WHERE Position = 'Senior Manager';
END $$
DELIMITER ;
```

Explanation:

- The event give_bonus_event is scheduled to run at noon on August 24, 2024.
- It updates the salary of all employees who are currently holding the position of "Senior Manager" by adding $5,000.

2.Query to Check the Employee Table Before and After the Event Runs



```sql
-- Before the event runs
SELECT * FROM Employee;

-- After the event runs
SELECT * FROM Employee;
```

Output Before Event:

EmployeeID	Name	Position	Salary
1	John Doe	Senior Manager	80000
2	Jane Smith	Developer	60000
3	Alice Johnson	Designer	55000
4	Bob Brown	Analyst	50000

Output After Event:

EmployeeID	Name	Position	Salary
1	John Doe	Senior Manager	85000
2	Jane Smith	Developer	60000
3	Alice Johnson	Designer	55000
4	Bob Brown	Analyst	50000

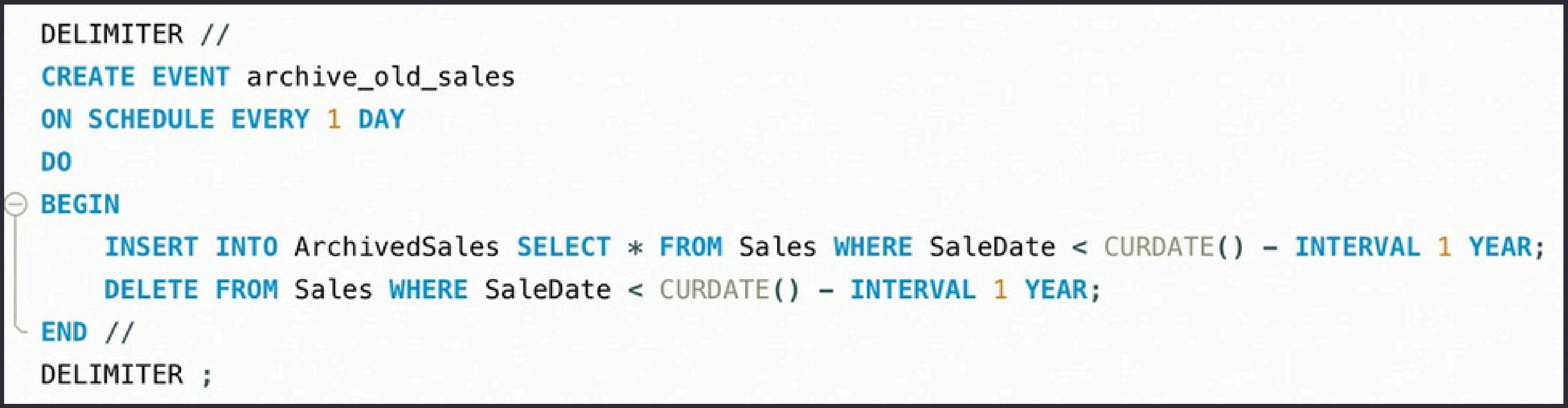
Explanation: The event increases the salary of the "Senior Manager" by $5,000.

2.Recurring Event to Archive Old Sales

Problem Statement : Suppose you want to automatically move sales records older than one year to an ArchivedSales table every day.

Solution:

1.Create the Recurring Event

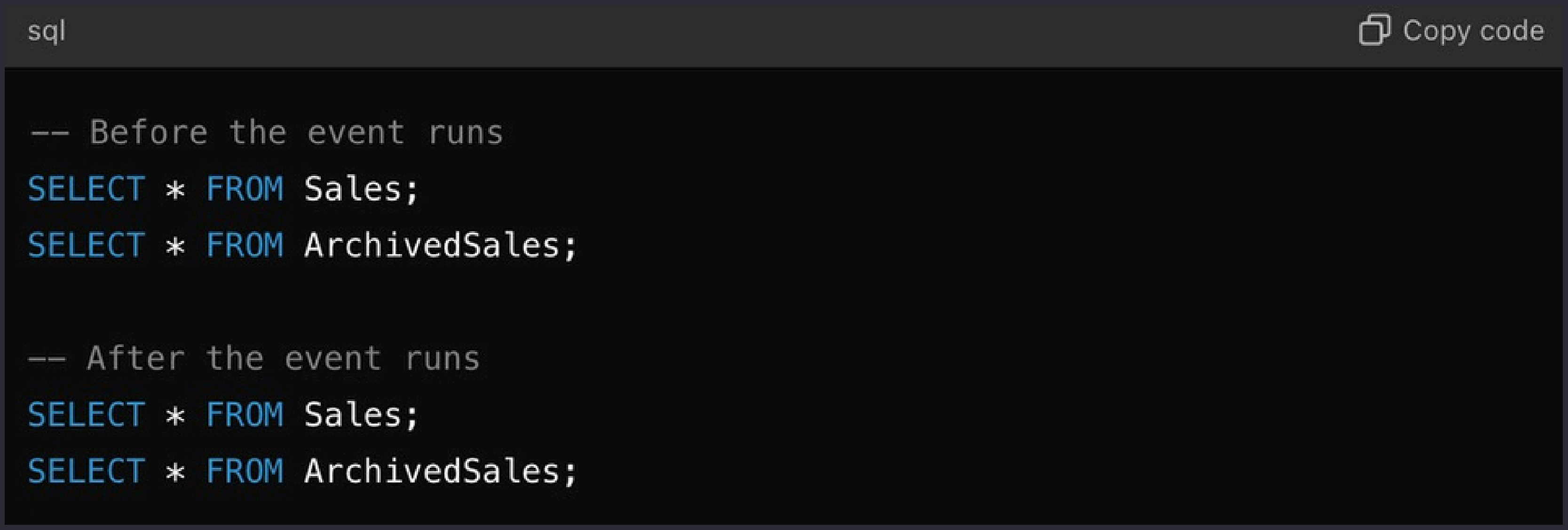


```sql
DELIMITER //
CREATE EVENT archive_old_sales
ON SCHEDULE EVERY 1 DAY
DO
BEGIN
    INSERT INTO ArchivedSales SELECT * FROM Sales WHERE SaleDate < CURDATE() - INTERVAL 1 YEAR;
    DELETE FROM Sales WHERE SaleDate < CURDATE() - INTERVAL 1 YEAR;
END //
DELIMITER ;
```

Explanation:

- The event archive_old_sales is scheduled to run every day.
- It archives all sales records that are older than 1 year and removes them from the Sales table.

2.Query to Check the Sales and ArchivedSales Tables Before and After the Event Runs

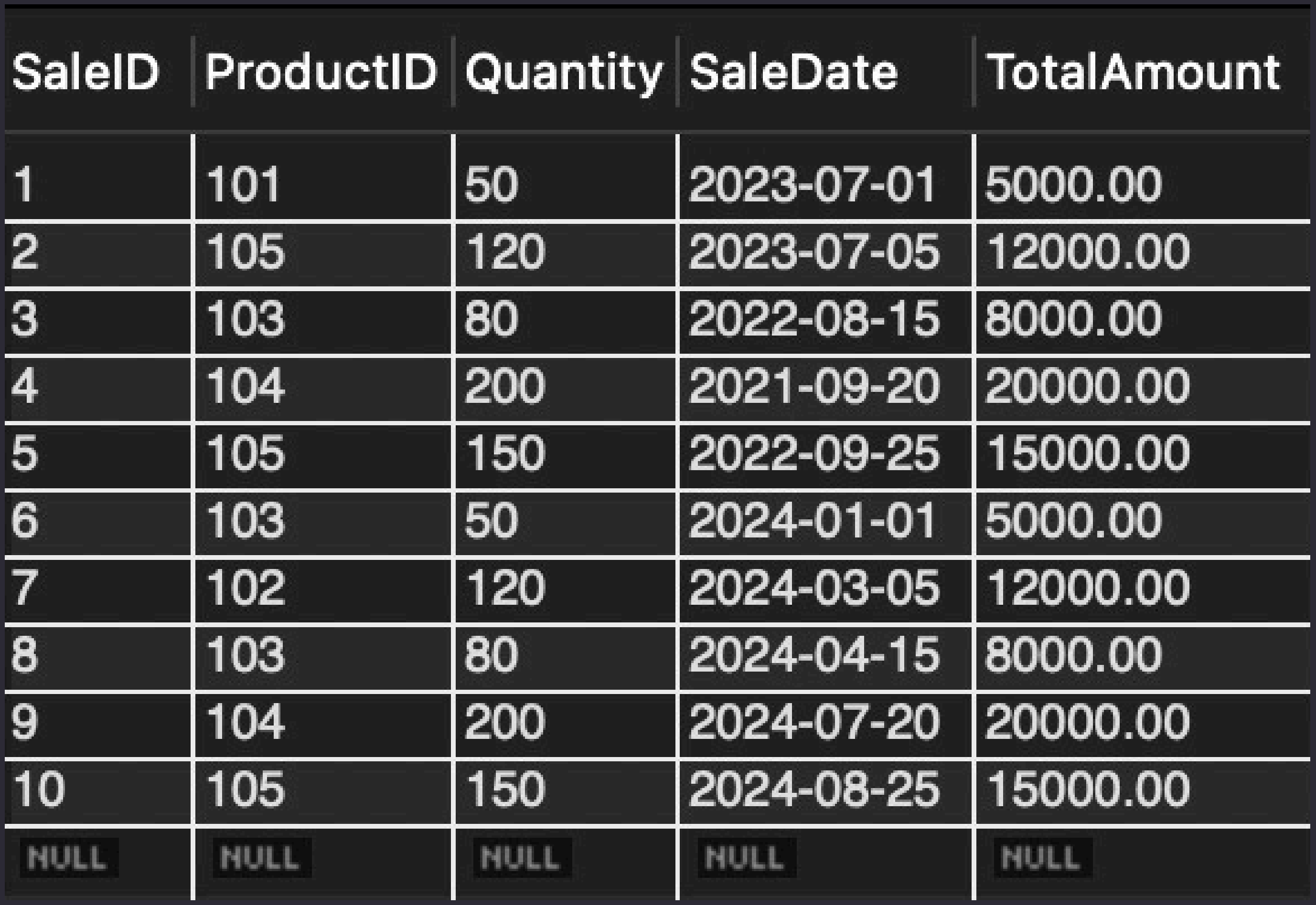


```sql
-- Before the event runs
SELECT * FROM Sales;
SELECT * FROM ArchivedSales;

-- After the event runs
SELECT * FROM Sales;
SELECT * FROM ArchivedSales;
```

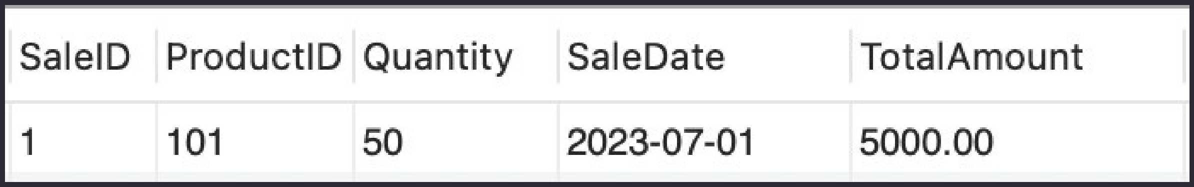
Output Before Event:

‘Sales’ Table:



SaleID	ProductID	Quantity	SaleDate	TotalAmount
1	101	50	2023-07-01	5000.00
2	105	120	2023-07-05	12000.00
3	103	80	2022-08-15	8000.00
4	104	200	2021-09-20	20000.00
5	105	150	2022-09-25	15000.00
6	103	50	2024-01-01	5000.00
7	102	120	2024-03-05	12000.00
8	103	80	2024-04-15	8000.00
9	104	200	2024-07-20	20000.00
10	105	150	2024-08-25	15000.00
NULL	NULL	NULL	NULL	NULL

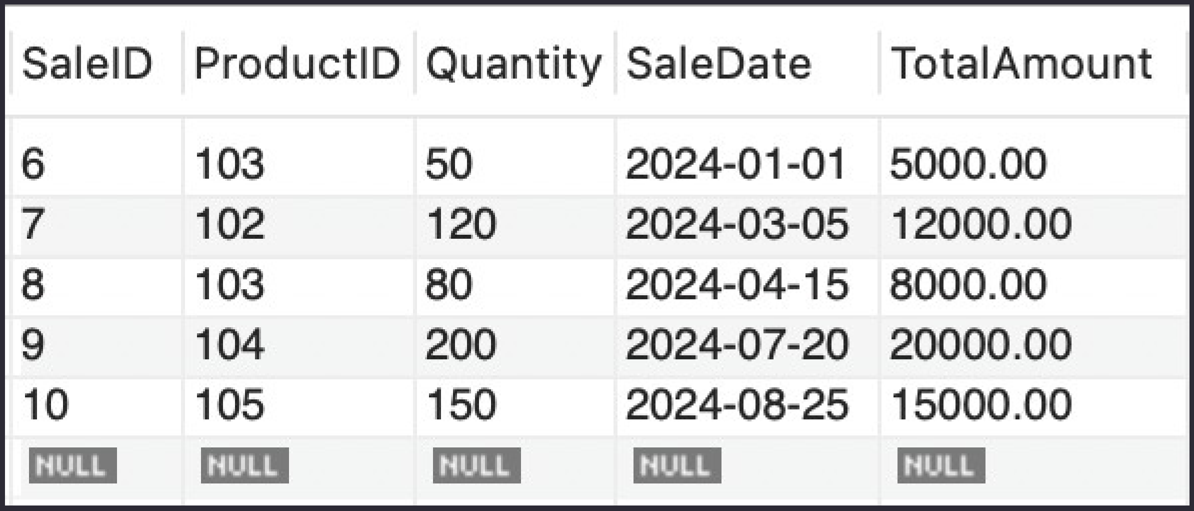
‘ArchivedSales’ Table:



SaleID	ProductID	Quantity	SaleDate	TotalAmount
1	101	50	2023-07-01	5000.00

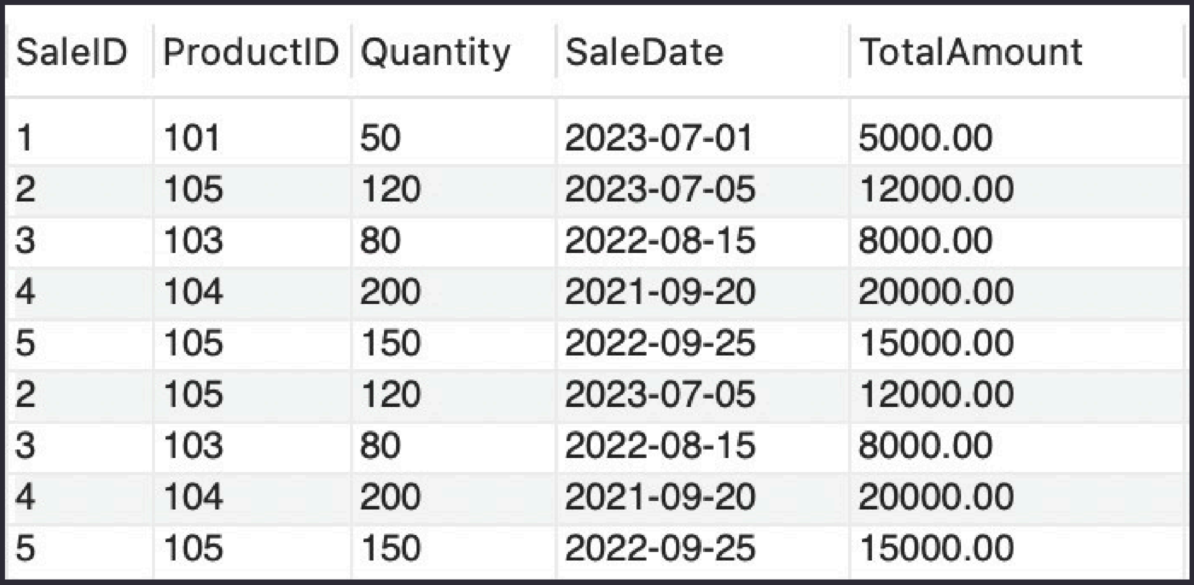
Output After Event

‘Sales’ Table:



SaleID	ProductID	Quantity	SaleDate	TotalAmount
6	103	50	2024-01-01	5000.00
7	102	120	2024-03-05	12000.00
8	103	80	2024-04-15	8000.00
9	104	200	2024-07-20	20000.00
10	105	150	2024-08-25	15000.00
NULL	NULL	NULL	NULL	NULL

‘ArchivedSales’ Table:



SaleID	ProductID	Quantity	SaleDate	TotalAmount
1	101	50	2023-07-01	5000.00
2	105	120	2023-07-05	12000.00
3	103	80	2022-08-15	8000.00
4	104	200	2021-09-20	20000.00
5	105	150	2022-09-25	15000.00
2	105	120	2023-07-05	12000.00
3	103	80	2022-08-15	8000.00
4	104	200	2021-09-20	20000.00
5	105	150	2022-09-25	15000.00

Explanation: The event moves all sales records that are older than 1 year to the ArchivedSales table on the last day of every month and deletes them from the Sales table.

Scheduling Events

Events are scheduled using the CREATE EVENT statement with specific time intervals or dates.

- **Enabling and Disabling Events :**

```sql
ALTER EVENT event_name ENABLE;
ALTER EVENT event_name DISABLE;
```

- **Controlling Scheduler:**

```sql
SET GLOBAL event_scheduler = ON;
SET GLOBAL event_scheduler = OFF;
```

Managing Events

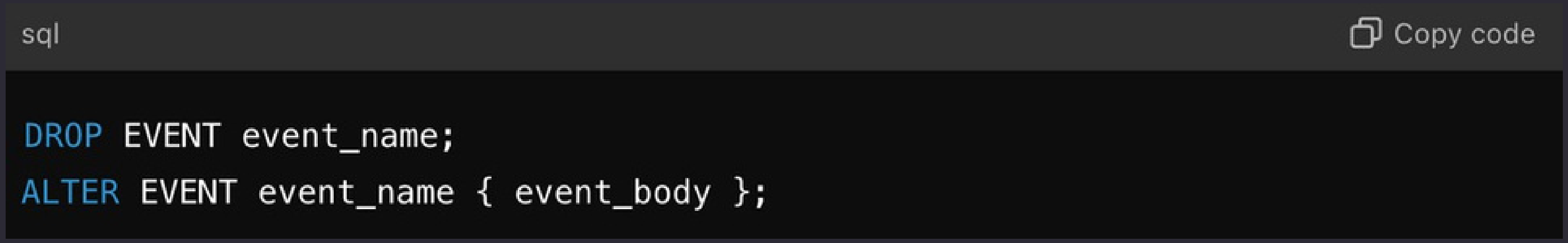
Viewing Existing Events:

To see the list of events in your database:

```sql
SHOW EVENTS;
```

- This query displays details such as event names, schedules, and status.

Dropping or Modifying Events:



```sql
DROP EVENT event_name;
ALTER EVENT event_name { event_body };
```

Uses of Events

Automating Maintenance Tasks:

- Example: Regularly cleaning up old records or logs to optimize database performance.

Archiving Data:

- Example: Periodically moving data from active tables to archive tables to manage table sizes.

Performing Regular Updates:

- Example: Automatically updating summary tables or materialized views at regular intervals.