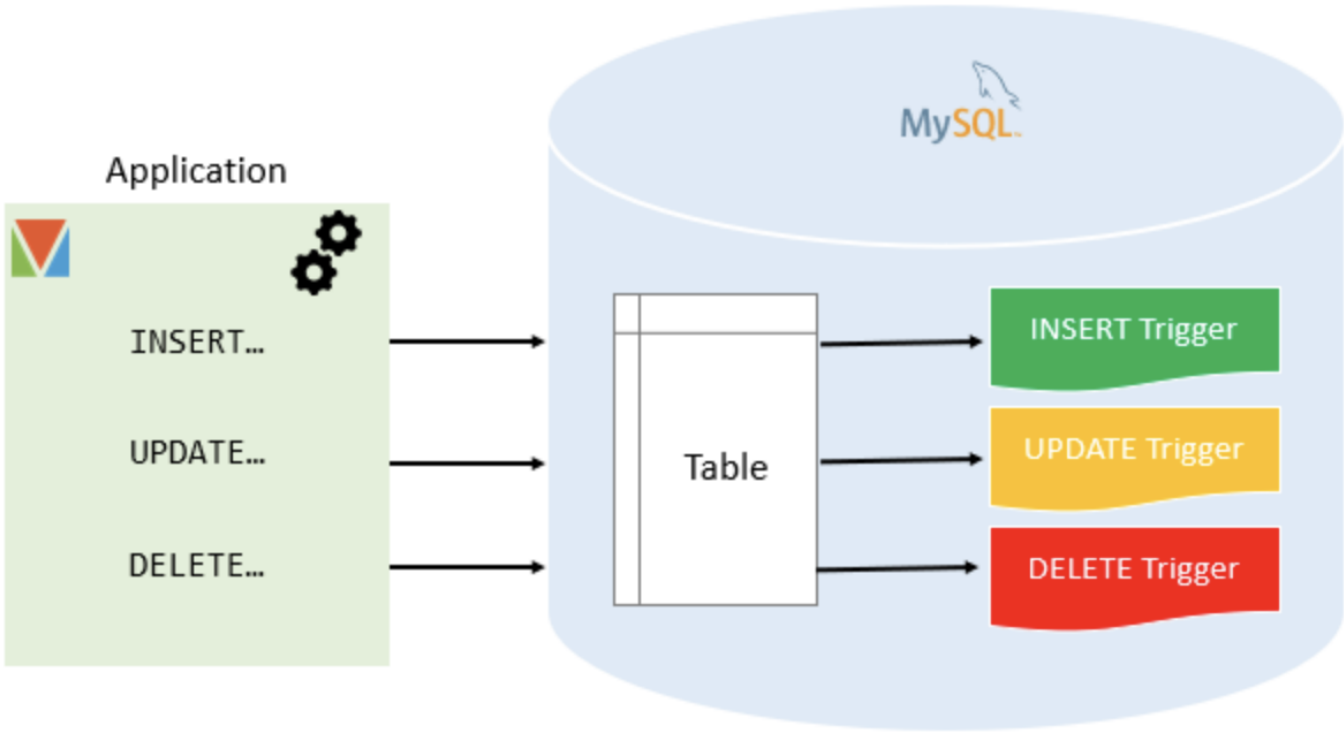
16-Table Trigger

Author: Vincent Lau

Learning Objectives

- Understand the purpose of creating table trigger
- Understand the ways of implementing table trigger
- Understand the advantages and disadvantages of using table trigger

Introduction



A table trigger is a database object that is associated with a specific table and automatically executed in response to certain events or actions on that table. Triggers are often used to enforce data integrity, perform logging, or automate specific actions when data is inserted, updated, or deleted in the table.

Creating a Table Trigger

```
CREATE TRIGGER trigger_name
[AFTER | BEFORE] [INSERT | UPDATE | DELETE]
ON table_name FOR EACH ROW
BEGIN
    -- Trigger logic here
END;
```

- `AFTER` or `BEFORE` : Specifies whether the trigger should fire after the event (`AFTER`) or before the event (`BEFORE`) occurs.
- `INSERT` , `UPDATE` , `DELETE` : Specifies the event that triggers the execution of the trigger.
- `table_name` : The name of the table to which the trigger is associated.
- `FOR EACH ROW` : Indicates that the trigger will be executed for each affected row.

Example of Creating a Table Trigger

Suppose you have a database with a table named "orders" and you want to create a trigger that automatically updates the "order_count" column in the "customers" table whenever a new order is inserted.

```
DELIMITER //

CREATE TRIGGER update_order_count
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
    UPDATE customers
    SET order_count = order_count + 1
    WHERE customer_id = NEW.customer_id;
END;
//

DELIMITER ;
```

In this example, the trigger is created to execute after an `INSERT` operation on the "orders" table. For each inserted row, the trigger logic updates the "order_count" column in the "customers" table for the corresponding customer.

Trigger Event Types

Triggers can be associated with different events (`INSERT` , `UPDATE` , `DELETE`) and can fire either before or after those events. Here's a breakdown:

- `BEFORE INSERT` , `AFTER INSERT` : Triggered when data is **inserted** into the table.
- `BEFORE UPDATE` , `AFTER UPDATE` : Triggered when data in the table is **updated**.
- `BEFORE DELETE` , `AFTER DELETE` : Triggered when data is **deleted** from the table.

Why Table Triggers

Benefits of Table Triggers

1. Data Integrity

Triggers can enforce complex integrity constraints by automatically validating and enforcing rules when data changes occur.

2. Audit and Logging

Triggers can be used to log changes to a separate audit table, allowing you to track historical data modifications.

3. Automated Actions

Triggers automate actions based on data changes, reducing the need for manual intervention.

4. Maintain Consistency

Triggers ensure that related data remains consistent across different tables.

Downsides of Table Triggers

1. Complexity

Overuse of triggers can lead to complex interactions that are difficult to debug and maintain.

2. Performance Impact

Poorly designed triggers can impact database performance, especially if they involve resource-intensive operations.

3. Implicit Logic

Triggers hide some logic from the application code, which can make it harder to understand data flow.

Triggers can be a powerful tool for automating tasks and enforcing data integrity, but they should be used judiciously and with careful consideration of their potential impact on database performance and complexity.

Tables Triggers vs Procedures

Table triggers and procedures are both database objects used to perform automated actions in response to certain events. However, they have distinct purposes, characteristics, and use cases.

Table Triggers

1. ### Event-Driven
    - Triggers are event-driven and automatically execute in response to specific events (INSERT, UPDATE, DELETE) on a table.
    - They are tied to the table and are executed whenever the corresponding event occurs on that table.

2. ### Implicit Execution

- Triggers execute implicitly; you don't need to call them explicitly in your SQL statements.
- They are often used to enforce data integrity rules, audit changes, or automate actions tied to specific data changes.

3. Row-Level Logic

- Triggers operate at the row level. They can access and modify the data of the affected rows using the `NEW` and `OLD` keywords (for `INSERT` and `UPDATE` triggers).

4. Tightly Coupled to Data

- Triggers are closely tied to the table they are associated with. They automatically respond to changes in that table's data.

5. Real-Time

- Triggers execute in real-time as part of the data manipulation operation that triggered them.

Procedures

1. User-Invoked

- Procedures are not event-driven. They are user-defined code blocks that need to be explicitly invoked using a `CALL` statement or other SQL statements.

2. Explicit Execution

- Procedures are explicitly called when needed, providing more control over when and how they execute.

3. Business Logic Encapsulation

- Procedures are often used to encapsulate business logic, complex calculations, and reusable operations. They can involve conditional statements, loops, and more.

4. Flexible Scope

- Procedures can operate across multiple tables and perform various actions, making them more versatile for business logic that spans different parts of the application.

5. Manual or Scheduled Execution

- Procedures can be executed manually by developers or scheduled to run at specific intervals, depending on application requirements.

When to Use

- Triggers: Use triggers when you need to enforce **data integrity rules, automatically audit changes**, or perform specific actions immediately when data changes occur in a specific table. Triggers are suitable for **real-time** actions tied to data manipulation operations.
- Procedures: Use procedures when you need to encapsulate complex logic, **automate business processes**, perform calculations, or **reuse code** across different parts of your application. Procedures offer more flexibility in terms of when and how they are executed.

In many cases, a combination of both triggers and procedures is used in a database-driven application. Triggers handle data integrity and immediate actions tied to data changes, while procedures encapsulate reusable business logic and more complex operations. The choice depends on the specific requirements of your application and the nature of the tasks you need to perform.