Complete Guide on SQL Triggers

In SQL Server, a trigger is a special type of stored procedure that automatically executes in response to certain events on a table or view, such as data insertions, updates, or deletions.

Triggers help enforce business rules, maintain audit trails, and manage complex data integrity constraints.

Why Use Triggers?

Triggers are used to:

1. Automatically enforce rules and constraints without requiring manual intervention.
2. Track changes in data (audit logging).
3. Maintain related records and ensure consistency.

Types of Triggers

SQL Server supports the following types of triggers:

DML Triggers (Data Manipulation Language)

- AFTER Trigger: Executes after an INSERT, UPDATE, or DELETE operation on a table.
- INSTEAD OF Trigger: Executes in place of the INSERT, UPDATE, or DELETE operation, often used in views where direct modification isn't allowed.

Example:

```
-- AFTER INSERT trigger example
CREATE TRIGGER trgAfterInsert
ON Employees
AFTER INSERT
AS
BEGIN
    INSERT INTO AuditLog (EmpID, Action)
    SELECT ID, 'INSERT' FROM INSERTED;
END;
```

DDL Triggers (Data Definition Language)

- Fired in response to schema-related changes like CREATE, ALTER, or DROP.

Example:

```
CREATE TRIGGER trgOnDDLChange
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
    PRINT 'A table modification was attempted!';
END;
```

Logon Triggers

- Fired when a user session is established, used to enforce login rules or restrict access.

Example:

```
CREATE TRIGGER trgLogon
ON ALL SERVER
FOR LOGON
AS
BEGIN
    IF ORIGINAL_LOGIN () = 'guest'
    ROLLBACK;
END;
```

Where Not to Use Triggers

Triggers can cause performance issues if they are overused, as they add overhead. They are not ideal when:

1. A high volume of data changes is expected, as they can slow down operations.
2. Complex business logic is required, as triggers can make debugging and maintenance harder.
3. Simple constraints (like FOREIGN KEY or CHECK) can enforce rules more efficiently.

DML Triggers

DML Triggers are specialized triggers that respond to data manipulation actions like INSERT, UPDATE, and DELETE. Here's a breakdown of the two main types of DML triggers, AFTER Trigger and INSTEAD OF Trigger, along with examples for each.

Examples of AFTER Trigger

Example 1: Logging an Insert Operation

```
CREATE TRIGGER trgAfterInsert
ON Employees
AFTER INSERT
AS
BEGIN
  INSERT INTO AuditLog (EmpID, Action, ActionDate)
  SELECT ID, 'INSERT', GETDATE()
  FROM INSERTED;
END;
```

Explanation: After a new record is inserted into Employees, an entry is logged in AuditLog.

Example 2: Preventing Deletion of Important Records

```
CREATE TRIGGER trgAfterDelete
ON Orders
AFTER DELETE
```

```
AS
BEGIN
   IF EXISTS (SELECT * FROM DELETED WHERE Status = 'Important')
   BEGIN
      ROLLBACK;
      RAISERROR ('Important records cannot be deleted.', 16, 1);
   END
END;
```

Explanation: If an attempt is made to delete an Important order, the trigger rolls back the transaction.

Example 3: Updating Stock Quantity

```
CREATE TRIGGER trgAfterOrder
ON Sales
AFTER INSERT
AS
BEGIN

   UPDATE Products
   SET StockQuantity = StockQuantity - i.Quantity
   FROM INSERTED i
   WHERE Products.ProductID = i.ProductID;
END;
```

Explanation: After a sale is made, the stock quantity in the Products table is updated.

Example 4: Updating Audit Log After Update

```
CREATE TRIGGER trgAfterUpdate
ON Employees
AFTER UPDATE
AS
BEGIN
   INSERT INTO AuditLog (EmpID, Action, ActionDate)
   SELECT ID, 'UPDATE', GETDATE()
```

```
 FROM INSERTED;

END;
```

Explanation: After an update on Employees, an audit log entry is created.

Example 5: Sending Notification After New Entry

```
CREATE TRIGGER trgAfterNewEmployee

ON Employees

AFTER INSERT

AS

BEGIN

    DECLARE @EmpName NVARCHAR(50);

    SELECT @EmpName = Name FROM INSERTED;

    PRINT 'New Employee Added: ' + @EmpName;

END;
```

2. INSTEAD OF Trigger

An INSTEAD OF Trigger is fired instead of the INSERT, UPDATE, or DELETE operation. It allows custom actions to be taken in place of the usual operation, often used in views where direct modification is not allowed.

Examples of INSTEAD OF Trigger

Example 1: INSTEAD OF INSERT on View

```
CREATE TRIGGER trgInsteadOfInsert

ON EmployeeView

INSTEAD OF INSERT

AS

BEGIN

    INSERT INTO Employees (ID, Name, Department)

    SELECT ID, Name, Department

    FROM INSERTED;

END;
```

Explanation: Allows insertion into EmployeeView by directing it to the underlying Employees table.

Example 2: Prevent Updates on Specific Column

```
CREATE TRIGGER trgInsteadOfUpdate

ON Orders

INSTEAD OF UPDATE

AS

BEGIN

   IF UPDATE(OrderDate)

   BEGIN

      RAISERROR ('Order Date cannot be modified.', 16, 1);

      RETURN;

   END

   UPDATE Orders

   SET Quantity = i.Quantity, Status = i.Status

   FROM INSERTED i

   WHERE Orders.OrderID = i.OrderID;

END;
```

Explanation: Prevents modification of the OrderDate column and updates only other columns.

Example 3: Redirect DELETE Operation

```
CREATE TRIGGER trgInsteadOfDelete

ON EmployeeView

INSTEAD OF DELETE

AS

BEGIN

   DELETE FROM Employees

   WHERE ID IN (SELECT ID FROM DELETED);

END;
```

Explanation: Allows deletions on EmployeeView by deleting records in the Employees table.

Example 4: Handle Multi-Table Updates via View

```
CREATE TRIGGER trgInsteadOfUpdate
ON ProductView
INSTEAD OF UPDATE
AS
BEGIN
    UPDATE Products
    SET Name = i.Name
    FROM INSERTED i
    WHERE Products.ProductID = i.ProductID;

    UPDATE Inventory
    SET Quantity = i.Quantity
    FROM INSERTED i
    WHERE Inventory.ProductID = i.ProductID;
END;
```

Explanation: Handles updates on a view that joins Products and Inventory, updating both tables accordingly.

Example 5: INSTEAD OF INSERT with Validation

```
CREATE TRIGGER trgInsteadOfInsertCheck
ON Orders
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (SELECT * FROM INSERTED WHERE Quantity <= 0)
    BEGIN
        RAISERROR ('Quantity must be greater than zero.', 16, 1);
    END
    ELSE
    BEGIN
```

```
        INSERT INTO Orders (OrderID, ProductID, Quantity)
        SELECT OrderID, ProductID, Quantity
        FROM INSERTED;
    END
END;
```

Explanation: Checks the Quantity during an insert and raises an error if it's zero or negative.

DDL Triggers (Data Definition Language Triggers) in SQL Server are used to respond to schema-related changes in the database, such as when tables or other objects are created, altered, or dropped.

Unlike DML triggers, which respond to data changes, DDL triggers are fired for changes at the structural level, affecting the schema of the database.

Purpose of DDL Triggers

DDL triggers help:

- Enforce rules on structural changes (e.g., preventing certain objects from being dropped or modified).
- Track and log schema changes for auditing purposes.
- Provide alerts when critical schema operations are attempted.

When DDL Triggers are Fired

DDL triggers are activated for:

- CREATE statements: When new database objects like tables, views, or stored procedures are created.
- ALTER statements: When existing objects are modified, such as changing a column in a table.
- DROP statements: When objects are removed from the database, such as dropping a table or view.

Example Uses and Scenarios

1. Preventing Object Deletion
   Suppose you have critical tables that should not be deleted. A DDL trigger can help block this.

```
CREATE TRIGGER trgPreventTableDrop
ON DATABASE
```

```
FOR DROP_TABLE
AS
BEGIN
   PRINT 'Table deletion is not allowed in this database.';
   ROLLBACK;
END;
```

Explanation: This trigger stops any attempt to drop a table and provides a message. If someone tries to run a DROP TABLE command, it rolls back the operation.

2. Logging Schema Changes
   It’s useful to track who is making changes to the database schema.

```
CREATE TRIGGER trgLogSchemaChanges
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
   DECLARE @EventData XML = EVENTDATA();
   INSERT INTO SchemaChangeLog (EventType, ObjectName, DateChanged, ChangedBy)
   VALUES (
      @EventData.value('(/EVENT_INSTANCE/EventType)[1]', 'NVARCHAR(100)'),
      @EventData.value('(/EVENT_INSTANCE/ObjectName)[1]', 'NVARCHAR(100)'),
      GETDATE(),
      SYSTEM_USER
   );
END;
```

Explanation: This trigger logs details (event type, object name, date, user) every time a table is created, altered, or dropped.

3. Restricting ALTER Operations

Sometimes, you may want to restrict certain modifications, such as changing specific tables or columns.

```
CREATE TRIGGER trgPreventColumnAlter
ON DATABASE
```

```
FOR ALTER_TABLE
AS
BEGIN
   DECLARE @ObjectName NVARCHAR(100) =
EVENTDATA().value('(/EVENT_INSTANCE/ObjectName)[1]', 'NVARCHAR(100)');
   IF @ObjectName = 'ImportantTable'
   BEGIN
      PRINT 'Modifications to ImportantTable are not allowed.';
      ROLLBACK;
   END
END;
```

Explanation: This trigger prevents alterations on a specific table, ImportantTable. It checks the table name and rolls back any modification attempts.

4. Auditing Database Schema Creation

You can monitor all new object creations within the database.

```
CREATE TRIGGER trgAuditObjectCreation
ON DATABASE
FOR CREATE_PROCEDURE, CREATE_VIEW, CREATE_FUNCTION
AS
BEGIN
   DECLARE @ObjectType NVARCHAR(100) =
EVENTDATA().value('(/EVENT_INSTANCE/ObjectType)[1]', 'NVARCHAR(100)');
   DECLARE @ObjectName NVARCHAR(100) =
EVENTDATA().value('(/EVENT_INSTANCE/ObjectName)[1]', 'NVARCHAR(100)');
   INSERT INTO ObjectCreationAudit (ObjectType, ObjectName, CreatedBy, CreationDate)
   VALUES (@ObjectType, @ObjectName, SYSTEM_USER, GETDATE());
END;
```

Explanation: This trigger logs new procedures, views, and functions created, recording the type, name, user, and date.

5. Preventing Unauthorized Database Modifications

DDL triggers can be configured to restrict any unauthorized schema modifications.

```
CREATE TRIGGER trgRestrictSchemaChange

ON DATABASE

FOR ALTER_DATABASE

AS

BEGIN

   IF SYSTEM_USER <> 'AdminUser'

   BEGIN

      PRINT 'Only AdminUser can alter the database schema.';

      ROLLBACK;

   END

END;
```

Explanation: Only the user AdminUser can alter the database schema; any other attempts will be blocked.

Logon Triggers in SQL Server are special types of triggers that activate w

session is established with the SQL Server instance.
These triggers allow you to control and monitor user logins, enforce specific login policies, or restrict access based on various conditions.
Purpose of Logon Triggers

Logon triggers help:

- Enforce custom security policies, like restricting specific users or enforcing time-based access rules.
- Track login activity for security auditing.
- Block unauthorized access based on conditions like login location, time, or IP address.

When Logon Triggers Are Fired

Logon triggers are fired when a new login session is initiated (after authentication). They work at the instance level and are typically used for controlling access to the SQL Server environment.

Example Uses and Scenarios

1. Restricting Login Times
   Suppose you want to restrict user logins to only certain hours.

```
CREATE TRIGGER trgRestrictLoginTimes
ON ALL SERVER
FOR LOGON
AS
BEGIN
   DECLARE @hour INT = DATEPART(HOUR, GETDATE());
   IF @hour < 9 OR @hour > 18
   BEGIN
      PRINT 'Logins are only allowed between 9 AM and 6 PM.';
      ROLLBACK;
   END
END;
```

Explanation: This trigger prevents users from logging in outside business hours (9 AM to 6 PM). If a login attempt is made during restricted hours, the session is terminated.

2. Blocking Specific User Logins
   Block logins for a specific user (e.g., during maintenance or if an account is compromised).

```
CREATE TRIGGER trgBlockSpecificUser
ON ALL SERVER
FOR LOGON
AS
BEGIN
   IF ORIGINAL_LOGIN() = 'UnauthorizedUser'
   BEGIN
      PRINT 'Access denied for UnauthorizedUser.';
      ROLLBACK;
   END
END;
```

Explanation: This trigger prevents UnauthorizedUser from logging into the server by rolling back the session. It’s useful for temporarily blocking specific users.

3. Restricting Access Based on IP Address

Allow login only from specific IP addresses.

```
CREATE TRIGGER trgRestrictIP

ON ALL SERVER

FOR LOGON

AS

BEGIN

  DECLARE @IP NVARCHAR(100) = (SELECT client_net_address FROM
sys.dm_exec_connections WHERE session_id = @@SPID);

  IF @IP NOT IN ('192.168.1.10', '192.168.1.20')

  BEGIN

    PRINT 'Access restricted to specific IP addresses.';

    ROLLBACK;

  END

END;
```

Explanation: This trigger allows only certain IP addresses to access the server. If the login originates from any other IP, it’s blocked.

4. Logging User Login Attempts

Record login attempts for auditing purposes.

```
CREATE TRIGGER trgLogLoginAttempts

ON ALL SERVER

FOR LOGON

AS

BEGIN

  DECLARE @LoginName NVARCHAR(100) = ORIGINAL_LOGIN();

  DECLARE @LoginTime DATETIME = GETDATE();

  INSERT INTO LoginAudit (LoginName, LoginTime)

  VALUES (@LoginName, @LoginTime);

END;
```

Explanation: This trigger logs each login attempt with the user’s login name and login time, which can be used to monitor and audit access to the server.

5. Restricting Login During Maintenance

Deny all logins except for specific admin users during maintenance windows.

```
CREATE TRIGGER trgRestrictDuringMaintenance
ON ALL SERVER
FOR LOGON
AS
BEGIN
   DECLARE @LoginName NVARCHAR(100) = ORIGINAL_LOGIN();
   IF @LoginName NOT IN ('AdminUser1', 'AdminUser2')
   BEGIN
      PRINT 'Server is under maintenance. Only admins can log in.';
      ROLLBACK;
   END
END;
```

Explanation: This trigger restricts access to only specified admin users during maintenance. Other users are prevented from logging in to avoid interference.