**Trigger Types**

1. **DML (Data Manipulation Language) triggers:**
2. **DDL (Data Definition Language) triggers**  :
3. **DML**

***triggers are fired in response to data modification events, such as INSERT, UPDATE, and DELETE operations on tables or views. DML triggers can be used to enforce business rules, perform auditing, or automate complex tasks that involve multiple tables. DML triggers can be further classified into the following subtypes:***

* **AFTER Triggers:** These triggers are fired after the data modification event has occurred. They are commonly used to update other tables, perform calculations, or send notifications.

**Example : 1**

Suppose you have two tables named "Orders" and "OrderDetails". The "Orders" table stores the general information about each order, and the "OrderDetails" table stores the specific details of each order, such as the product ID, quantity, and price. You want to ensure that the total order amount in the "Orders" table is updated automatically whenever a new order detail is added, updated, or deleted in the "OrderDetails" table.

-- Create the Orders table

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

OrderDate DATE,

CustomerID INT,

OrderAmount DECIMAL(10,2)

)

-- Create the OrderDetails table

CREATE TABLE OrderDetails (

OrderDetailID INT PRIMARY KEY,

OrderID INT,

ProductID INT,

Quantity INT,

Price DECIMAL(10,2)

)

-- Insert some sample records into the Orders table

INSERT INTO Orders (OrderID, OrderDate, CustomerID, OrderAmount)

VALUES

(1, '2022-01-01', 1001, 0.00),

(2, '2022-02-01', 1002, 0.00),

(3, '2022-03-01', 1003, 0.00)

-- Insert some sample records into the OrderDetails table

INSERT INTO OrderDetails (OrderDetailID, OrderID, ProductID, Quantity, Price)

VALUES

(1, 1, 101, 2, 10.00),

(2, 1, 102, 3, 15.00),

(3, 2, 101, 1, 10.00),

(4, 3, 103, 4, 20.00)

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Create Trigger \_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TRIGGER UpdateOrderAmount

ON OrderDetails

AFTER INSERT, UPDATE, DELETE

AS

BEGIN

UPDATE Orders

SET OrderAmount = (SELECT SUM(Quantity \* Price) FROM OrderDetails WHERE OrderID = Orders.OrderID)

WHERE OrderID IN (SELECT OrderID FROM inserted UNION SELECT OrderID FROM deleted)

END

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Test upper Trigger \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

insert into OrderDetails values (5,4,1,50,5000)

-- Check the updated OrderAmount in the Orders table

SELECT \* FROM Orders WHERE OrderID = 2

-- Update an existing record in the OrderDetails table

UPDATE OrderDetails SET Quantity = 2 WHERE OrderDetailID = 1

-- Check the updated OrderAmount in the Orders table

SELECT \* FROM Orders WHERE OrderID = 1

-- Delete a record from the OrderDetails table

DELETE FROM OrderDetails WHERE OrderDetailID = 3

-- Check the updated OrderAmount in the Orders table

SELECT \* FROM Orders WHERE OrderID = 2

* **INSTEAD OF Triggers:** These triggers are fired instead of the data modification event, and they allow the trigger logic to modify or suppress the original data modification. They are commonly used to implement complex data validation rules or cascading updates.

**Example : 1**

Suppose you have a table named "Employees" with columns "EmployeeID", "FirstName", "LastName", "Email", and "Salary". You want to prevent any changes to the "Email" column for employees with a salary greater than $100,000. Instead of raising an error when such changes are attempted, you want to silently ignore them and keep the original value of the "Email" column.

* Ager me 🡪 higher Salary>1000000$ wala record ke Email update kron ga 🡪 update nhin kraa gaa.
* Else update kr da ga

-- Create the Employee table

CREATE TABLE Employees(

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

Salary DECIMAL(10,2)

)

-- Insert some sample records into the Employee table

INSERT INTO Employees (EmployeeID, FirstName, LastName, Email, Salary)

VALUES

(1, 'John', 'Doe', 'doe@example.com', 50000.00),

(2, 'Jane', 'Doe', 'janedoe@example.com', 60000.00),

(3, 'Bob', 'Smith', 'bobsmith@example.com', 80000.00),

(4, 'jahaz', '23423', 'xyz@example.com', 1000000.00),

(5, 'Rahmanad', 'sdfsdf', 'abc@example.com', 10000000.00),

(6, 'Khalaeel', 'Basit', 'lmn@example.com', 80000.00)

CREATE TRIGGER PreventEmailChange

ON Employees

INSTEAD OF UPDATE

AS

BEGIN

UPDATE Employees

SET FirstName = inserted.FirstName,

LastName = inserted.LastName,

Email = CASE WHEN Employees.Salary > 100000 THEN deleted.Email ELSE inserted.Email END,

Salary = inserted.Salary

FROM Employees INNER JOIN inserted ON Employees.EmployeeID = inserted.EmployeeID

INNER JOIN deleted ON Employees.EmployeeID = deleted.EmployeeID

END

drop trigger PreventEmailChange

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Test The Trigger \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-- updated Successfuly

UPDATE Employees SET Email = 'newSaiqb@example.com' WHERE EmployeeID = 1

-- but salary 10000000 > not update

UPDATE Employees SET Email = 'newSaiqb@example.com' WHERE EmployeeID = 4

-- Check the updated record in the Employee table

SELECT \* FROM Employees

1. **DDL**

***Triggers are fired in response to changes in the database schema, such as table or view (creation, alteration, or deletion).***

***DDL triggers can be used to enforce***

* 1. ***naming conventions,***
  2. ***prevent unauthorized changes,***
  3. ***Audit database schema changes (logon … other information)***

***DDL triggers can be further classified into the following subtypes:***

* **AFTER Triggers:** These triggers are fired after the DDL event has occurred. They are commonly used to audit schema changes or enforce naming conventions.

CREATE TABLE AuditLog (

EventID int IDENTITY(1,1) PRIMARY KEY,

EventType nvarchar(100) NOT NULL,

ObjectName nvarchar(100) NOT NULL,

Timestamp datetime NOT NULL

);

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Create DDL Trigger \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

create TRIGGER ddl\_audit

ON DATABASE

AFTER CREATE\_TABLE, ALTER\_TABLE, DROP\_TABLE

AS

BEGIN

INSERT INTO AuditLog (EventType, ObjectName, Timestamp)

VALUES (EVENTDATA().value('(/EVENT\_INSTANCE/EventType)[1]', 'nvarchar(100)'),

EVENTDATA().value('(/EVENT\_INSTANCE/ObjectName)[1]', 'nvarchar(100)'),

GETDATE())

END

--\_\_=========================================

---------------- NOte -----------------

--f Any Issue Comes in Trriger you you can not Drop this Trriger

--you have to below Steps

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1. check Trigger Exist in Database \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SELECT \* FROM sys.triggers WHERE name = 'ddl\_audit';

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2. check Alter permission \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SELECT \* FROM sys.database\_permissions WHERE grantee\_principal\_id = USER\_ID() AND permission\_name = 'ALTER'

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3. Give permision \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

GRANT ALTER ON DATABASE::practice2 TO "SAQIB\m4357";

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 4. Disable trigger \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DISABLE TRIGGER ddl\_audit ON DATABASE;

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 5. Drop trigger \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DROP TRIGGER ddl\_audit ON database

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Testing Trigging \_\_\_\_\_\_\_\_\_\_\_\_

create table xyz(

id int primary key,

name varchar(25)

)

-- check Value

select \* from AuditLog

* **INSTEAD OF Triggers:** are used for DML (ins,up,delet)

--No for DDL --

* **INSTEAD OF Triggers:** are used for DML (ins,up,delet)

***--\_\_\_\_\_\_\_\_ Server Side \_\_\_\_\_\_\_***

**\_\_\_\_\_\_\_\_\_\_ DDL Trigger on Login to MSSQL \_\_\_\_\_**

1. Trigger for Auditing Logins: The following DDL trigger logs all successful and failed login attempts to the "LoginAudit" table:

CREATE TABLE LoginAudit (

LoginAuditID INT PRIMARY KEY IDENTITY(1,1),

EventType NVARCHAR(128),

LoginName NVARCHAR(128) ,

HostName NVARCHAR(128),

ApplicationName NVARCHAR(128),

LogonTime DATETIME

);

drop table LoginAudit

CREATE TRIGGER AuditLogins

ON ALL SERVER

FOR LOGON

AS

BEGIN

DECLARE @EventData xml

SET @EventData = EVENTDATA()

INSERT INTO LoginAudit (EventType, LoginName, HostName, ApplicationName, LogonTime)

VALUES (

@EventData.value('(/EVENT\_INSTANCE/EventType)[1]', 'nvarchar(128)'),

@EventData.value('(/EVENT\_INSTANCE/LoginName)[1]', 'nvarchar(128)'),

@EventData.value('(/EVENT\_INSTANCE/HostName)[1]', 'nvarchar(128)'),

@EventData.value('(/EVENT\_INSTANCE/ApplicationName)[1]', 'nvarchar(128)'),

GETDATE()

)

END;

GO

select \* from LoginAudit

use master

go

select \* from sys.server\_triggers order by create\_date desc

--\_\_\_\_\_\_\_\_ Disable Trigger \_\_\_\_\_\_\_\_\_\_\_

disable trigger [AuditLogins] on all server

--\_\_\_\_\_\_\_\_ Disable Trigger \_\_\_\_\_\_\_\_\_\_\_

enable trigger [AuditLogins] on all server

--\_\_\_\_\_\_\_\_ Drop Trigger \_\_\_\_\_\_\_\_\_\_\_

DROP TRIGGER AuditLogins ON ALL SERVER

--\_\_\_\_\_\_\_\_\_\_\_ check UserName \_\_\_\_\_\_\_\_\_\_

SELECT SUSER\_SNAME()