SQL Triggers

.NET

A Trigger is SQL code that automatically runs instead of or after an INSERT, UPDATE, or DELETE action is performed on a table.

https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?

SQL – Triggers

https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver15

Code that automatically runs instead of (or after) an insert, update, or delete to a particular table.

```
GO
CREATE TRIGGER Poke.PokemonDateModified ON Poke.Pokemon
AFTER UPDATE
AS
BEGIN
    -- in a trigger, you have access to two special table-valued variables
    -- called Inserted and Deleted.
    UPDATE Poke.Pokemon SET DateModified = GETDATE()
    WHERE PokemonId IN (SELECT PokemonId FROM Inserted);
    -- recursion in triggers is off by default
END

SELECT * FROM Poke.Pokemon;
UPDATE Poke.Pokemon SET Name = 'Charmander' WHERE PokemonId = 1001;
```

```
CREATE [ OR ALTER ] TRIGGER
[ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
{ sql_statement  [ ; ] [ ,...n ] |
EXTERNAL NAME [nameOfMethod] }

<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

Triggers - Overview

https://docs.microsoft.com/en-us/sql/relational-databases/triggers/logon-triggers?view=sql-server-ver15

A ***Trigger*** is a special type of ***Stored Procedure***. It automatically runs when an event occurs in the database. There are three types of ***Triggers***. DML, DDL, and Logon Triggers.

DML triggers – DML events are INSERT, UPDATE, or DELETE statements on a table or view. These triggers fire when any valid event fires, whether table rows are affected or not.

DDL triggers – These fire in response to a variety of Data Definition Language (DDL) events. These events primarily correspond to CREATE, ALTER, and DROP statements, and certain Stored Procedures that perform DDL-like operations.

Logon triggers - These fire in response to the LOGON event that's raised when a user's session is being established. You can create ***Triggers*** directly from SQL statements. SQL Server lets you create multiple triggers for any specific statement.

We will focus on DML triggers.

Trigger Limitations

https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver15#trigger-limitations

- CREATE TRIGGER must be the first statement in the batch and can apply to only one table.
- A ***Trigger*** is created only in the current database.
- The same ***Trigger*** action can be defined for more than one user action (INSERT and UPDATE) in the same CREATE TRIGGER statement.
- INSTEAD OF, DELETE, and UPDATE ***Triggers*** can't be defined on a table that has a Foreign Key with a cascade on DELETE/UPDATE action defined.
- When a ***Trigger*** fires, results are returned to the calling application, just like with ***Stored Procedures***.
- Most DDL statements are not allowed in a DML ***Trigger***.

DML Triggers - Overview

https://docs.microsoft.com/en-us/sql/relational-databases/triggers/dml-triggers?view=sql-server-ver15
https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver15

A DML ***Trigger*** is a special type of **Stored Procedure** that automatically takes effect when a INSERT, UPDATE, or DELETE takes place that affects the table or view defined in the trigger.

The ***Trigger*** and the statement that fires it are treated as a single transaction, which can be "rolled back" from within the trigger. If an error is detected, the entire transaction automatically rolls back.

DML triggers are frequently used for enforcing business rules and data integrity. If constraints exist on a trigger table, they're checked after the INSTEAD OF trigger runs and before the AFTER trigger runs. If the constraints are violated, the INSTEAD OF trigger actions are rolled back and the AFTER trigger isn't fired.

An AFTER trigger is run only after the triggering SQL statement has run successfully.

INSTEAD OF triggers will not run recursively. This means that if it is defined on a table and runs a statement against the same table, the trigger is prevented from being called. Instead, the statement processes as if the table had no INSTEAD OF trigger and starts the chain of constraint operations and AFTER trigger executions.

DML Trigger Benefits

https://docs.microsoft.com/en-us/sql/relational-databases/triggers/dml-triggers?view=sql-server-ver15#dml-trigger-benefits

DML triggers:

- can cascade changes through related tables in the database.
- can guard against malicious or incorrect INSERT, UPDATE, and DELETE operations and enforce other restrictions that are more complex than those defined with CHECK constraints.
- can reference columns in other tables
- can evaluate the state of a table before and after a data modification and take actions based on that difference.
- can disallow or roll back changes that violate ***referential integrity***.

DML Triggers Types

https://docs.microsoft.com/en-us/sql/relational-databases/triggers/dml-triggers?view=sql-server-ver15#types-of-dml-triggers

AFTER Trigger	INSTEAD OF Trigger
• are executed after the action of a INSERT, UPDATE, or DELETE statement. • are never executed if a constraint violation occurs.	• overrides the standard actions of the triggering statement. • can be used to perform error or value checking on one or more columns and perform additional actions before INSERT, UPDATE or DELETE. • enables views that would not be updatable to support updates. • enables you to reject parts of a batch while letting other parts of a batch succeed.

DML Trigger Types Syntax

https://docs.microsoft.com/en-us/sql/relational-databases/triggers/dml-triggers?view=sql-server-ver15#types-of-dml-triggers

AFTER and INSTEAD OF Triggers

```
CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement | EXTERNAL NAME <method specifier [ ; ] > }

<dml_trigger_option> ::=
   [ ENCRYPTION ]  ,  [ EXECUTE AS Clause ]
<method_specifier> ::=   assembly_name.class_name.method_name
```

CREATE an 'AFTER' Trigger (1/3)

https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver15
https://www.sqlservertutorial.net/sql-server-triggers/sql-server-create-trigger/

IMPORTANT: When an INSERT or DELETE operation is completed, there are two internal tables populated. They are called "INSERTED" and "DELETED".

Below, we'll create a table that allows a record of things deleted or inserted into the Customers table to be maintained.

Create a special table (Customer_audits) to record INSERT and DELETE actions, then use the UNION ALL operator to grab that latest data from the appropriate internal ('inserted' or 'deleted') table and copy it to the 'audit' table.

We will place a trigger on the ***Customers*** table.

```
CREATE TABLE Customer_audits
(
--record a unique id of the change
ChangeId INT IDENTITY PRIMARY KEY,
CustomerId INT NOT NULL,
FirstName VARCHAR(255),
LastName VARCHAR(255),
AddressID INT,
LastOrderDate DATE,
Remarks VARCHAR(255),
--record when the operation happened
UpdatedAt DATETIME NOT NULL,
--Record what type of operation it is.
Operation CHAR(3) NOT NULL,
CHECK(operation = 'INS' or operation = 'DEL')
);
```

CREATE an 'AFTER' Trigger (2/3)

https://www.sqlservertutorial.net/sql-server-triggers/sql-server-create-trigger/

Create a ***Trigger*** called '***WhenCustomerAdded***'.

When an INSERT or DELETE action occurs on the ***Customers*** table, the ***Trigger*** will copy the data from the deleted or inserted internal tables into the ***Customer_audits*** table using a UNION ALL statement.

```
CREATE TRIGGER WhenCustomerAdded
ON Customers
AFTER INSERT, DELETE
AS
BEGIN
    -- to suppress the number of rows affected
    -- messages from being returned (@@ROWCOUNT)
    SET NOCOUNT ON;
    INSERT INTO Customer_audits
    (
        CustomerId,FirstName,LastName,AddressID,
        LastOrderDate,Remarks,UpdatedAt,Operation
    )
    SELECT
        CustomerId,FirstName,LastName,AddressID,
        LastOrderDate,Remarks,GETDATE(),'INS'
    FROM
        inserted
    UNION ALL
    SELECT
        CustomerId,FirstName,LastName,AddressID,
        LastOrderDate,Remarks,GETDATE(),'DEL'
    FROM
        deleted
END
```

CREATE an 'AFTER' Trigger (3/3)

https://www.sqlservertutorial.net/sql-server-triggers/sql-server-create-trigger/

Now, test that the ***Trigger*** works by inserting and/or deleting something from the ***Customers*** table and then checking the ***Customer_audits*** table for any new rows.

```
--INSERT a new Customer
INSERT INTO Customers
(FirstName, LastName, AddressID,
LastOrderDate, Remarks)
VALUES
('Test', 'Testerson', 6, '0999-12-31',
'Testing for the first millennium');

--Look at the Customer_audits table
SELECT * FROM Customer_audits;
```

Create an 'INSTEAD OF' Trigger (1/3)

https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver15
https://www.sqlservertutorial.net/sql-server-triggers/sql-server-instead-of-trigger/

IMPORTANT: When an INSERT or DELETE operation is completed, there are two internal tables populated. They are called "INSERTED" and "DELETED".

An INSTEAD OF ***Trigger*** maintain a record of changes people make and allow those changes to be approved by others.

Create a special table (Customers_pending) to record pending INSERT and DELETE actions, then use the UNION ALL operator to grab that latest data from the appropriate internal ('inserted' or 'deleted') table and copy it to the 'audit' table.

We will place a ***Trigger*** on the ***Customers*** table.

```
CREATE TABLE Customers_pending
(
    --record a unique id of the change
    PendingChangeId INT IDENTITY PRIMARY KEY,
    FirstName VARCHAR(255),
    LastName VARCHAR(255),
    AddressID INT,
    LastOrderDate DATE,
    Remarks VARCHAR(255),
);
```

CREATE an 'INSTEAD OF' Trigger (2/3)

https://www.sqlservertutorial.net/sql-server-triggers/sql-server-create-trigger/

This statement Creates a view, ***NewCustomerAdded***.

When there is an INSERT action on the ***Customers*** table, it will copy the data from the inserted internal table into the ***Customers_pending*** table.

```
CREATE TRIGGER NewCustomerAdded
ON Customers
INSTEAD OF INSERT
AS
BEGIN
    -- to suppress the number of rows
    affected
    -- messages from being returned
    (@@ROWCOUNT)
    SET NOCOUNT ON;
    INSERT INTO Customers_pending
    (
        FirstName,LastName,AddressID,
        LastOrderDate,Remarks
    )
    SELECT
        FirstName,LastName,AddressID,
        LastOrderDate,Remarks
    FROM
        inserted
END
```

CREATE an 'INSTEAD OF' Trigger (3/3)

https://www.sqlservertutorial.net/sql-server-triggers/sql-server-create-trigger/

Now test that the ***Trigger*** works by inserting something into the ***Customers*** table and then checking the ***Customers_pending*** table for new rows.

```
--INSERT a new Customer
INSERT INTO Customers
(FirstName, LastName, AddressID,
LastOrderDate, Remarks)
VALUES
('Testy 'McTesterson',6, '0999-12-31',
'Testing the Test of a test');

--Look at the Customers_pending table
SELECT * FROM Customers_pending;
```