Database Security

CT069-3-3

Database Triggers

Learning Outcomes

A·P·U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

By the end of this lesson, you should be able to:

- Explain what is a SQL Trigger
- Implement Triggers for automation and auditing

Topic & Structure of Lesson

- What Are Triggers?
- Syntax for Creating and Altering Triggers
- How a Trigger Works
- Syntax for Disabling, Enabling and Dropping Triggers

Triggers

- A trigger is a special type of stored procedure that automatically runs when an event occurs in the database server.
- A stored procedure is a set of Structured Query Language (SQL) statements with an assigned name, which are stored in a relational database management system (RDBMS).
- Triggers can evaluate the state of a table before and after a data modification and take actions based on that difference.
- Triggers are a powerful tool that enables you to enforce domain, entity, referential data integrity, enterprise constraints or business rules.
- Triggers can also be used for auditing

Triggers Categories

Three categories of triggers:-

- **Data Manipulation Language (DML) triggers**
  - DML triggers run when a user tries to modify data through a data manipulation language (DML) event.
  - DML events are INSERT, UPDATE, or DELETE statements on a table or view.
- **Data Definition Language (DDL) triggers**
  - DDL triggers run in response to a variety of data definition language (DDL) events.
  - These events primarily correspond to CREATE, ALTER, and DROP statements.
- **Logon triggers.**
  - Logon triggers fire in response to the LOGON event that's raised when a user's session is being established.

DML Triggers Actions

- **INSERT**
  - When an INSERT command is executed on a table, an INSERT trigger on the table fires. For example, when an order is placed (inserting data into the Orders table), you might like to update the on-hand inventory column in the Products table.

- **UPDATE**
  - When an UPDATE command is executed on a table, an UPDATE trigger on the table fires. For example, management might want to know if anyone ever modifies data (such as the Salary column) in the Employees table.

- **DELETE**
  - When a DELETE command is executed on a table, a DELETE trigger on the table fires. For example, the company you work at might not want to ever delete a customer from the Customers table. Instead, customers should be marked as inactive.

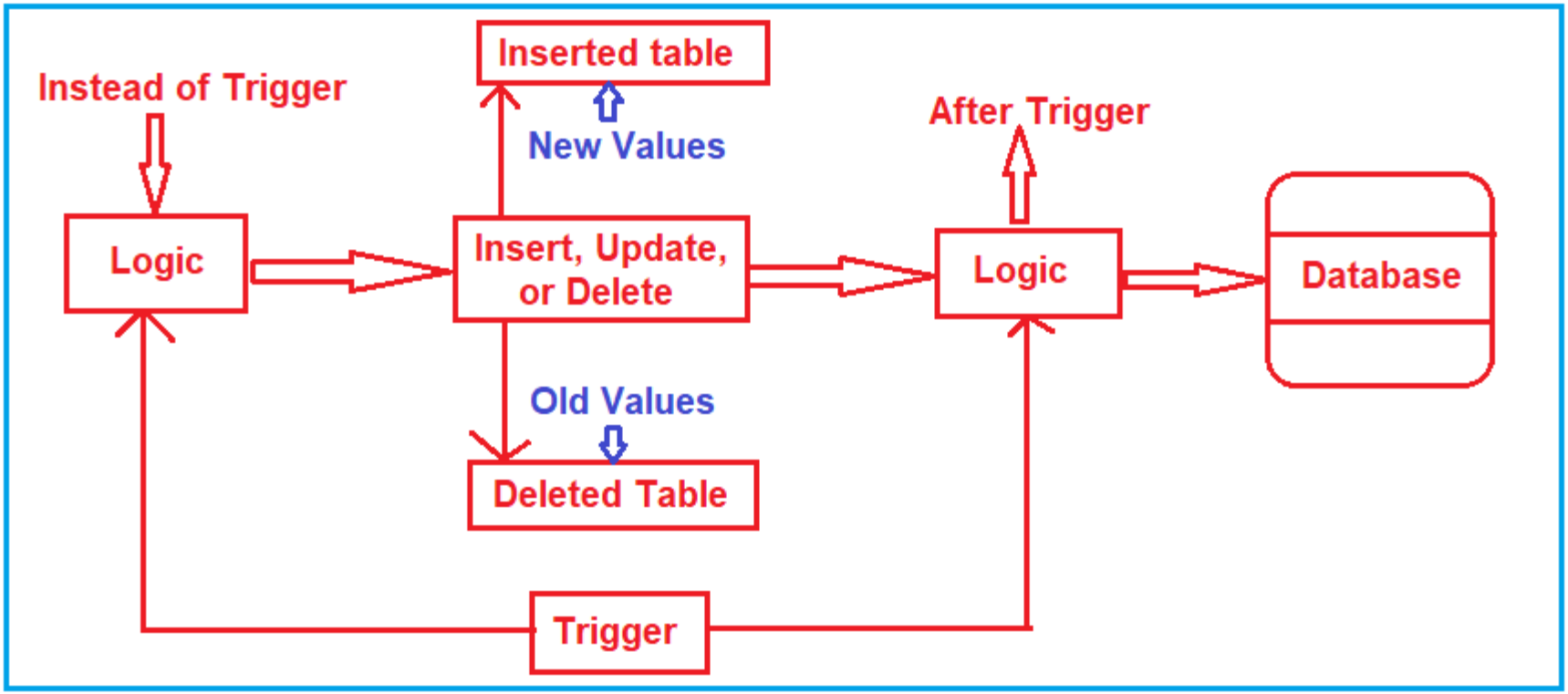
DML Trigger Types

- DML triggers can be programmed to perform either **after** the data modification or **instead of** the data modification. When a trigger is created, one of these choices is specified:

- **AFTER**: An AFTER trigger fires **after the data modification**. In other words, the data modification occurs, and then the trigger fires in response to the data modification. Therefore, an after trigger that rolls back a data modification becomes an expensive operation because the data is changed twice when it shouldn't be changed at all.

- **INSTEAD OF**: An INSTEAD OF trigger fires **before the data modification.** In other words, the data modification is prevented from occurring, and instead, the trigger is executed.

DML Trigger


Instead of Trigger
Inserted table
New Values
After Trigger
Logic
Insert, Update, or Delete
Logic
Database
Old Values
Deleted Table
Trigger

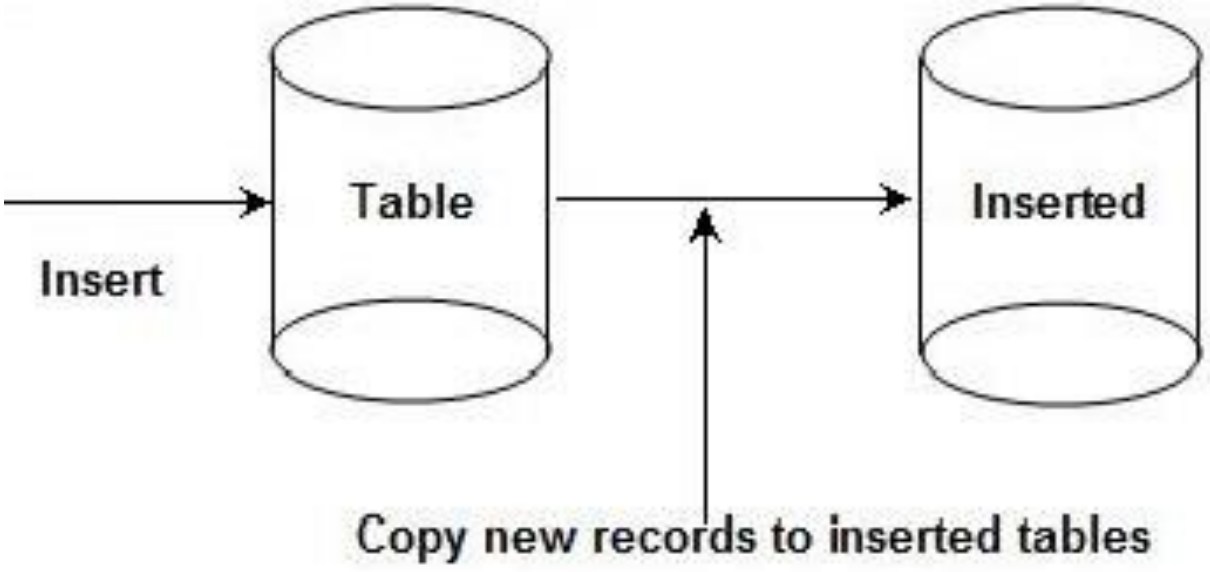
Inserted and **Deleted** tables

- Triggers use two special tables that are only accessible to triggers. The data in these tables is available immediately after an INSERT, UPDATE or DELETE statement.

- **Inserted** table: The inserted table holds data from the last INSERT or UPDATE statement.

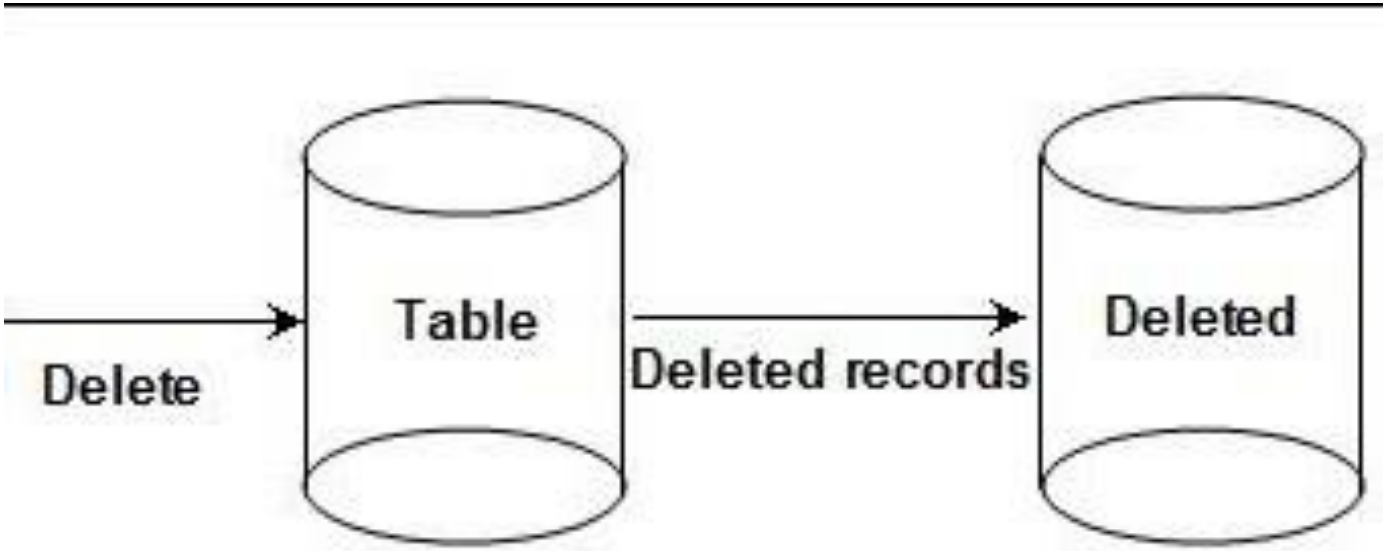
- **Deleted** table: The deleted table holds data from the last DELETE or UPDATE statement.

Insert Operation



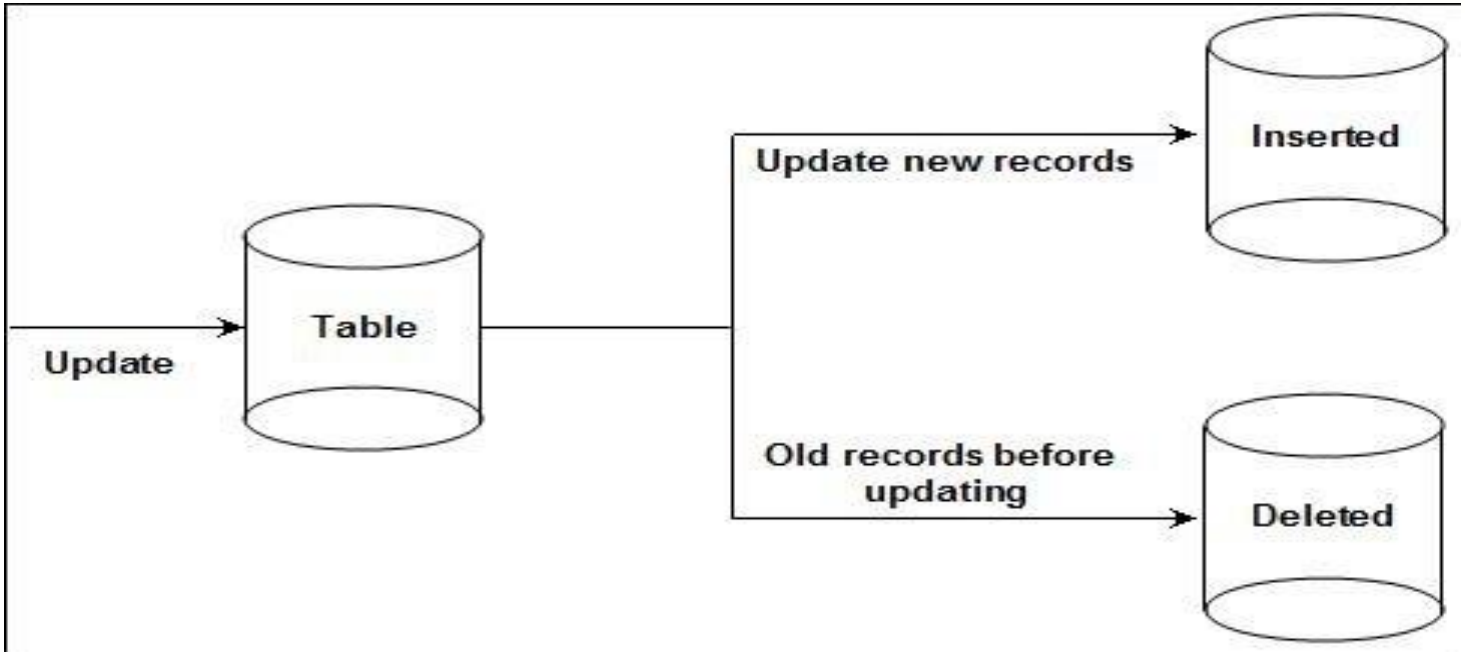
- If one or more rows are added with an INSERT statement, the row(s) are held in the inserted table.

Delete Operation



- If one or more rows are deleted with a DELETE statement, the row(s) are held in the deleted table.

Update Operation



- When an UPDATE statement is executed, the database engine deletes the existing row (and holds it in the deleted table) and then adds a new row (and holds it in the inserted table).

Example scenarios for DML Triggers

- Make changes to other columns (in the same table or other tables within the database) in response to a data modification.
- Perform validation before data is inserted or updated into table
- Provide specialized error messages.
- Perform auditing by recording who made a data modification and when.

DML Trigger Syntax


```
-- SQL Server Syntax
-- Trigger on an INSERT, UPDATE, or DELETE statement to a table
or view (DML Trigger)

CREATE OR ALTER TRIGGER [schema_name].[trigger_name]
ON { table | view }
{ AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS
{
    sql_statements  (business logic)
}
```

How an INSERT Trigger Works

- When an INSERT trigger fires, new rows are added to both the **trigger** table and the **inserted** table.
- The **inserted** table is a temporary table that holds a copy of the rows that have been inserted.
- The **inserted** table allows you to reference logged data from the initiating INSERT statement.
- The trigger can examine the **inserted** table to determine whether, or how, the trigger actions should be executed.

Sample Tables


```
USE [HospitalInfoSys]
GO

Create Table Medicine(
MedicineID varchar(10) primary key,
MedicineName varchar(100),
QuantityInStock int
)
Create Table Prescription(
PresID integer identity (1000000,1) primary key,
MedicineID varchar(10) references Medicine(MedicineID),
Quantity int
)
```

Example of INSERT Trigger with AFTER clause


```
CREATE OR ALTER TRIGGER dbo.PrescribeMedicine_Inserted
ON Prescription
AFTER
INSERT
AS
Begin
Declare @quantity int, @medicineid varchar(10)
Select @medicineid=MedicineID, @quantity = Quantity
From Inserted

Update Medicine
Set QuantityInStock = QuantityInStock - @quantity
Where MedicineID = @medicineid
End
Go
```

```
select * from sys.triggers
```

How an DELETE Trigger Works

- When a DELETE trigger is fired, deleted rows from the affected table are placed in a special **deleted** table.
- The **deleted** table is a logical table that holds a copy of the rows that have been deleted.
- The **deleted** table enables you to reference logged data from the initiating DELETE statement.

- Consider the following facts when you use the DELETE trigger:
  1. When a row is appended to the **deleted table, it no longer exists in the database table;** therefore, the **deleted table and the database tables have no rows in common.**
  2. Space is allocated from memory to create the **deleted table. The deleted table is** always in the cache.

Example of DELETE Trigger with AFTER clause


```
CREATE OR ALTER TRIGGER dbo.PrescribeMedicine_Deleted
ON Prescription
AFTER
DELETE
AS
Begin
Declare @quantity int, @medicineid varchar(10)
Select @medicineid=MedicineID, @quantity = Quantity
From Deleted

Update Medicine
Set QuantityInStock = QuantityInStock + @quantity
Where MedicineID = @medicineid
End
Go
```

How an UPDATE Trigger Works

- When an UPDATE statement is executed on a table that has a trigger defined on it, the original rows are moved into the deleted table, and the updated rows are inserted into the inserted table.
- The trigger can examine the deleted and inserted tables, as well as the updated table, to determine whether multiple rows have been updated and how the trigger actions should be carried out.
- You can define a trigger to monitor data updates on a specific column by using the IF UPDATE statement.

How an INSTEAD OF Trigger Works

- This trigger executes instead of the original triggering action.
- INSTEAD OF triggers increase the variety of types of updates that you can perform on the data
- Each table or view is limited to one INSTEAD OF trigger for each triggering action (INSERT, UPDATE, or DELETE).
- For example, lets say if we want to check the quantity in stock first before allowing user to insert a record in the Prescription table, we can do so using this trigger

Example of DELETE Trigger with INSTEAD OF clause


```
CREATE OR ALTER TRIGGER dbo.PrescribeMedicine_InsteadOfInsert
ON Prescription
INSTEAD OF
INSERT
AS
Begin
Declare @quantity_in_stock int, @quantity_ins int, @medicineid varchar(10)
Select @medicineid=MedicineID, @quantity_ins = Quantity
From Inserted
Select @quantity_in_stock = QuantityInStock
From Medicine
Where MedicineID = @medicineid
If @quantity_in_stock > @quantity_ins
Begin
INSERT INTO [dbo].[Prescription] ([MedicineID],[Quantity])
VALUES   (@medicineid, @quantity_ins)
End
Else
Begin
Print 'Not enough stock for this medicine. Prescription Rejected.'
End
End
Go
```

DDL Triggers

- DDL triggers fire in response to a variety of Data Definition Language (DDL) events which correspond to SQL statements that start with the keywords CREATE, ALTER, DROP, GRANT, DENY or REVOKE

- Use DDL triggers when you want to do the following:
  - Prevent certain changes to your database schema.
  - Have something occur in the database in response to a change in your database schema.
  - Record changes or events in the database schema.

DDL Triggers

- DDL triggers do not create the special inserted and deleted tables
- The information about an event that fires a DDL trigger is captured by using the **EVENTDATA** function.

```
EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1
]', 'nvarchar(max)')
```

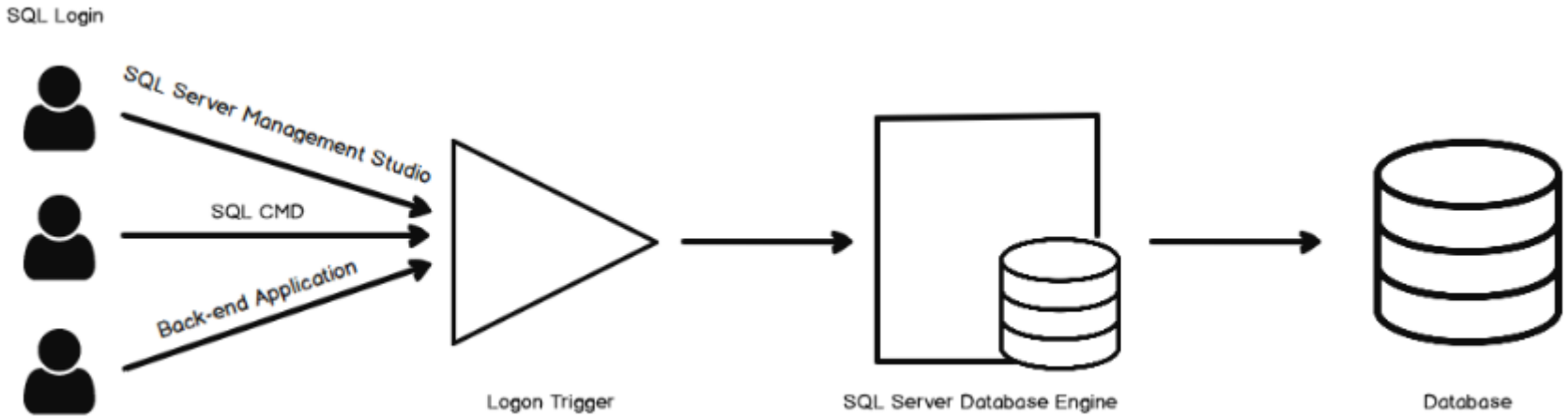
DDL Trigger Example


```
CREATE OR ALTER TRIGGER [JustWannaBeSafe]
ON DATABASE
FOR DROP_TABLE
AS
PRINT 'You must disable Trigger [JustWannaBeSafe] to drop
tables!'
ROLLBACK;
```

This DDL trigger will prevent any accidental deletion of tables

Logon Triggers

- SQL Server automatically executes the logon triggers once a logon event occurs.
- It gets executed before a user session is established and authentication is successful. If any user is unable to authenticate to SQL Server (wrong credentials), SQL Server does not execute the logon triggers. SQL



Logon Triggers Usage

- Can be used for tracking logon activity
- Can also be used for restricting login activities such as :-
  - limiting the number of sessions for a specific login.
  - limiting the hours sql server can be accessed

Logon Trigger Example 1 - limiting who can login


```
CREATE TRIGGER Prevent_login
ON ALL SERVER
FOR LOGON
AS
BEGIN
  DECLARE @LoginName sysname

  SET @LoginName = ORIGINAL_LOGIN()

  IF(@LoginName NOT IN ('sa', 'EAD\kpalasundram',’Test1’))
  BEGIN
    ROLLBACK; --Disconnect the session
  END
END
```

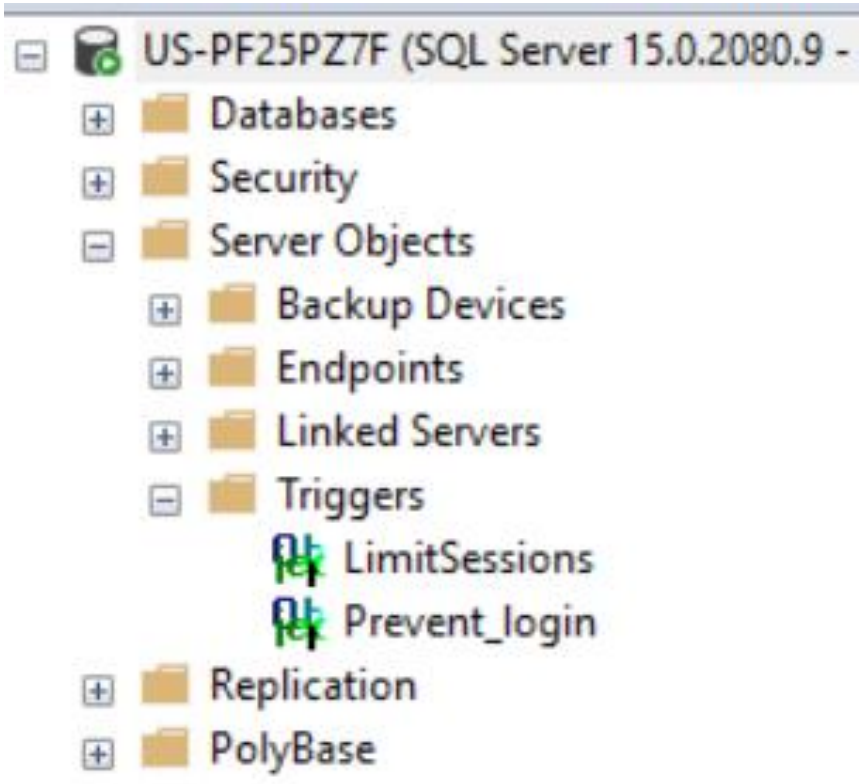
Logon Trigger Example 2 - limiting the number of sessions for login Test1.

```
CREATE OR ALTER TRIGGER [LimitSessions]
ON ALL SERVER FOR LOGON
AS   BEGIN
DECLARE @Login nvarchar(100)
Set @Login = ORIGINAL_LOGIN()
IF (SELECT COUNT(*) FROM sys.dm_exec_sessions
  WHERE is_user_process = 1
  AND original_login_name = @Login
  AND @Login = 'Test1') > 5
  BEGIN
      PRINT 'Maximum connection allowed per user is 5 only'
      ROLLBACK
  END
END;
```

Note: Test1 user must be granted assigned **View Server State** permission →
GRANT VIEW SERVER STATE to [Test1]

Checking Existing Server Level Triggers

```
select * from sys.server_triggers
```



Disabling or Dropping Triggers

Triggers can be disabled, enabled back or dropped

DISABLE TRIGGER <trigger name>
ON <table name>

ENABLE TRIGGER <trigger name>
ON <table name>

DROP TRIGGER <trigger name>

Q & A