## Lab Experiment 12

### Demonstration & Implementation of SQL Triggers

### 

### Objectives

* Demonstrating the implementation of event driven triggers for database and server domain events

### Introduction

#### SQL Triggers

SQL triggers are powerful tools in database management that automate tasks in response to specific events. By understanding and implementing SQL triggers, you can ensure data integrity, automate repetitive tasks, and enhance overall database performance.

SQL triggers are stored procedures that automatically execute in response to certain events in a specific table or view in a database. They are used to maintain the integrity of the data, enforce business rules, and automate tasks. We can set triggers to fire before or after an INSERT, UPDATE, or DELETE operation. Understanding and implementing SQL triggers can significantly enhance your database management skills

Note: SQL statement and its associated triggers are treated as a single transaction

#### Types of SQL Triggers

There are several main types of SQL triggers. These trigger types are grouped according to the specific events they respond to and the operations they perform.

* **DML Triggers (Data Manipulation Language)**: DML Triggers include AFTER Triggers, which execute after an operation, BEFORE Triggers, which execute before an operation, and INSTEAD OF Triggers, which replace the operation with the trigger’s code.
* **DDL Triggers (Data Definition Language)**: DDL triggers are fired in response to DDL events such as CREATE, ALTER, and DROP statements. They are useful for controlling schema changes, auditing database modifications, and enforcing security policies.
* **Logon Triggers**: Logon triggers are usually executed in response to a LOGON event. They are typically used to control or monitor user sessions, enforce logon policies, or log user activity. For example, a logon trigger can limit access to certain hours or log each user's login time and IP address.

#### Syntax to create a trigger

CREATE or ALTER TRIGGER *trg\_name*

ON *table/view*

FOR/AFTER/INSTEAD OF *DML\_Op*

AS

--declare variables if required here

BEGIN

--Logic here

END

#### Example

Consider the table: ORDERS(orderID,OrderDate,OrderQuantity,Customer\_ID)

Let’s say that we want to create a trigger for every insert statement executed to insert orders to enforce the following business rule:

“The minimum order quantity must be 10USD”

Any orders that violate this rule, won’t be added to the database.

#### Trigger for the above example:

create trigger trg\_insert

on orders

for insert

as

declare @o\_price int

begin

select @o\_price = i.orderPrice from inserted i;

if @o\_price<10

begin

print 'minimum order price is 10'

rollback

end

else

begin

print 'inserted successfully'

end

end

When you execute the following insert statement, this trigger will be executed automatically.

insert into orders values (15, '12-12-2024', 4, 'Wajeeh')

As the order price is less than the specified amount, data will not be inserted, instead there will be a prompt and the attempted changes will be rolled back.

#### Syntax to drop trigger

*DROP TRIGGER IF EXISTS trg\_name*

#### How to see all the triggers?

*select \* from sys.triggers*

#### Delete Trigger example

For the following examples, consider the table:

Employee(emp\_id int identity(1,1), emp\_name varchar(100),emp\_exp int, emp\_dob datetime)

Lets say we’d like to keep record of the deleted employees in our database as a backup. To keep backup records of employees, we’ll need a backup table as well. Backup table is as follows:

Backup\_employee(emp\_id int, emp\_name varchar(100),emp\_exp int, emp\_dob datetime)

Whenever a record from main employee table is deleted, we’ll have to insert that deleted data into the backup\_employee table. For this purpose we’ll create a trigger associated with delete command on employee table as follows:

create or alter trigger trg\_del

on employee

for delete

as

declare @deleted\_emp\_id int;

declare @deleted\_emp\_name varchar(100);

declare @deleted\_emp\_exp int;

declare @deleted\_emp\_dob datetime;

begin

select @deleted\_emp\_id=d.emp\_id,@deleted\_emp\_name=d.emp\_name,@deleted\_emp\_exp=d.emp\_exp,@deleted\_emp\_dob=emp\_dob from deleted d;

insert into backup\_employee values (@deleted\_emp\_id,@deleted\_emp\_name,@deleted\_emp\_exp,@deleted\_emp\_dob);

print 'deleted successfully'

end

When we delete a record for the employee from employee table, trigger will be executed automatically and deleted data will be inserted into back\_employee for future usage. This is a use case of data logging.

Similarly, trigger can be implemented if we need to log data changes that are done by UPDATE command.

Look at the example below:

We have the employee table that contains employee id, his name, experience and date of birth. Using update command, we may want to change any of these attributes. What if there was a requirement to log these changes in your database.

As an example, lets say we want to change the name of an Employee due to a spelling error. We want to keep a log that which attribute was changed, what was its old value and what is the new value of that attribute. Lets say that we also want to keep a log that which database user made this specific change and at what time. To keep such kind of a log, we may need to create a table where these entries are made. See below for such a table:

create table employee\_changelog (logID int identity(1,1), attribute varchar(100),old\_value varchar(100),new\_value varchar(100), whodidthis varchar(100), dateofchange datetime)

#### Update Trigger:

create or alter trigger trg\_update

on employee

for update

as

declare @emp\_name varchar(100);

declare @old\_emp\_name varchar(100);

declare @emp\_exp int;

declare @old\_emp\_exp int;

declare @emp\_dob datetime;

declare @old\_emp\_dob datetime;

begin

select @old\_emp\_name=d.emp\_name,@old\_emp\_exp=d.emp\_exp,@old\_emp\_dob=d.emp\_dob from deleted d;

select @emp\_name=i.emp\_name,@emp\_exp=i.emp\_exp,@emp\_dob=i.emp\_dob from inserted i;

if UPDATE(emp\_name)

begin

insert into employee\_changelog(attribute,old\_value,new\_value,whodidthis,dateofchange)

values

('emp\_name',@old\_emp\_name,@emp\_name,ORIGINAL\_LOGIN(),GETDATE())

end

else if UPDATE(emp\_exp)

begin

insert into employee\_changelog(attribute,old\_value,new\_value,whodidthis,dateofchange)

values

('emp\_exp',@old\_emp\_exp,@emp\_exp,ORIGINAL\_LOGIN(),GETDATE())

end

else if UPDATE(emp\_dob)

begin

insert into employee\_changelog(attribute,old\_value,new\_value,whodidthis,dateofchange)

values

('emp\_dob',@old\_emp\_dob,@emp\_dob,ORIGINAL\_LOGIN(),GETDATE())

end

end

The above example shows how to keep track of changes for audit purposes.

#### DDL Triggers

DDL triggers respond to DDL events like CREATE, ALTER, DROP, GRANT, DENY, REVOKE, or UPDATE STATISTICS. For example, you can define a DDL trigger that records CREATE or ALTER TABLE operations. DDL trigger fires only after the events that fired them are executed successfully. They cannot be used as INSTEAD OF triggers.You can create a DDL trigger to:

* Log changes made to the database schema;
* Prevent certain changes to the schema;
* To respond to any change in the database schema.

CREATE TRIGGER trigger\_name

ON { DATABASE | ALL SERVER}

[WITH ddl\_trigger\_option]

FOR { event\_type | event\_group }

AS

{sql\_statement}

In the above syntax:

* trigger\_name is the name of the new trigger being created.
* ON DATABASE specifies that the trigger is fired for DATABASE or ALL SERVER scoped events.
* ddl\_trigger\_option specifies the ENCRYPTION or EXECUTE AS clause. Encryption encrypts the trigger definition. EXECUTE AS defines the security context under which the trigger is executed.
* event\_type specifies the event that causes the trigger to fire e.g., CREATE\_TABLE, ALTER\_TABLE, etc. The event\_group is a group of event\_type such as DDL\_TABLE\_EVENTS.

Let's create a DDL trigger that logs changes whenever a DB user creates, alters, or deletes tables.

First, create a database table TableLog to capture the logs, as shown below.

CREATE TABLE dbo.TableLog(

LogID int IDENTITY(1,1) PRIMARY KEY,

EventVal xml NOT NULL,

EventDate datetime NOT NULL,

ChangedBy SYSNAME NOT NULL);

Now, create a trigger which will be fired every time a CREATE, ALTER, OR DROP table event occurs. The trigger will capture and log the event values into the TableLog table.

CREATE TRIGGER trgTablechanges

ON DATABASE

FOR

CREATE\_TABLE,

ALTER\_TABLE,

DROP\_TABLE

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO TableLog

(

EventVal,

DateChanged,

ChangedBy

)

VALUES (

EVENTDATA(),

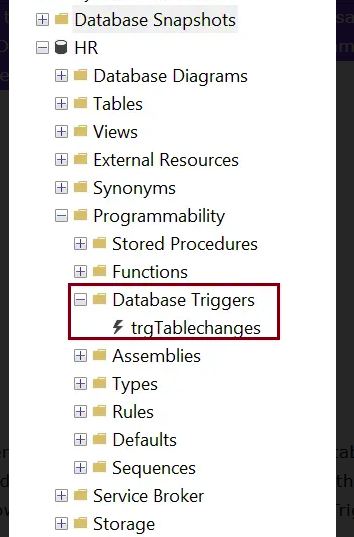
GETDATE(),

USER

);

END;

The above trgTablechanges trigger is fired whenever a table in the database is created, altered, or dropped. The EVENTDATA() value inserted into the EventVal field is an inbuilt function of the DDL trigger. It returns the transaction event details in XML format. The DDL triggers are created under Programmability -> Database Triggers, as shown below.



You can test the above trigger by creating a new table in the database. If the trigger is working as expected, then a new row is inserted into the TableLog table for the CREATE event. Now, create a new table called TestDDLTrigger.

CREATE TABLE dbo.TestDDLTrigger(

LogID int IDENTITY(1,1) PRIMARY KEY,

TestedBy SYSNAME NOT NULL

);

After creating the above trigger, select rows from the TableLog table and you will find a new entry for the TestDDLTrigger table

### Tasks

1. Create a database students with the following relational schema:

StudentDetails(rollno int, s\_name, semester int, age int, dob datetime)

Contact Details(rollno int, c\_type,c\_number)

Address(rollno int, houseno int, streetno int, block varchar(10), city varchar(50))

Note: Set all foreign key constraint such that no action is taken when a primary key is deleted in the parent table.

1. Create a trigger such that when a record of student is inserted, the operation log is saved in another table
2. In the above trigger, enforce that the student whose record is being inserted must not be over 20 years old and he should be a 1st year student. Failing any of this business rule, must enforce rollback of the whole operation.
3. Create a trigger such that when a record of a student is deleted, all associated contact numbers and addresses are deleted likewise from relevant tables and all the deleted data is back up in separate tables along with the details of user who performed the action and the time at which the operation was performed.
4. Create a trigger such that when a record of a student is updated, the changelog is maintained which specifies which value was changed and should also maintain the old value, new value, user and time.
5. Create a trigger to restrict executing drop and alter commands on the database.

**Rubric for Lab Assessment**

|  |  |  |  |
| --- | --- | --- | --- |
| **The student performance for the assigned task during the lab session was:** | | | |
| Excellent | The student completed assigned tasks without any help from the instructor and showed the results appropriately. | 4 |  |
| Good | The student completed assigned tasks with minimal help from the instructor and showed the results appropriately. | 3 |  |
| Average | The student could not complete all assigned tasks and showed partial results. | 2 |  |
| Worst | The student did not complete assigned tasks. | 1 |  |

**Instructor Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**