8

What is a Database Trigger?

What is a Database Trigger?

A database trigger is special stored procedure that is run when specific actions occur within a database. Most triggers are defined to run when changes are made to a table's data. Triggers can be defined to run *instead of* or *after* DML (Data Manipulation Language) actions such as INSERT, UPDATE, and DELETE.

Triggers help the database designer ensure certain actions, such as maintaining an audit file, are completed regardless of which program or user makes changes to the data.

The programs are called triggers since an event, such as adding a record to a table, fires their execution.

Triggers and their implementations are specific to database vendors. In this article we'll focus on Microsoft SQL server; however, the concepts are the same or similar in Oracle and MySQL.

Note: All the examples for this lesson are based on Microsoft SQL Server Management Studio and the AdventureWorks2012 database. You can get started using these free tools using my *Guide* *Getting Started Using SQL Server*.

Events

The triggers can occur AFTER or INSTEAD OF a DML action. Triggers are associated with the database DML actions INSERT, UPDATE, and DELETE. Triggers are defined to run when these actions are executed on a specific table.

AFTER triggers

Once the DML actions, such as an INSERT completes, the AFTER trigger executes. Here are some key characteristics of AFTER triggers:

- After triggers are run after a DML action, such as an INSERT statement and any ensuing referential cascade actions and constraint checks have run.
- You can't cancel the database action using an AFTER trigger. This is because the action has already completed.
- One or more AFTER triggers per action can be defined on a table, but to keep things simple I recommend only defining one.
- You can't define AFTER triggers on views.

INSTEAD OF triggers

INSTEAD OF triggers, as their name implies, run in place of the DML action which caused them to fire. Items to consider when using INSTEAD OF triggers include:

- An INSTEAD OF trigger overrides the triggering action. If an INSTEAD OF trigger is defined to execute on an INSERT statement, then once the INSERT statement attempt to run, control is immediately passed to the INSTEAD OF trigger.
- At most, one INSTEAD OF trigger can be defined per action for a table. This makes sense, as if you had to "INSTEAD OF" triggers for an insert, which one should run?

Special Database Objects

Triggers use two special database objects, INSERTED and DELETED, to access rows affected by the database actions. Within the scope of a trigger the INSERTED and DELETE objects have the same columns as the trigger's table.

The INSERTED table contains all the new values; whereas, the DELETED table contains old values. Here is how the tables are used:

- INSERT – Use the INSERTED table to determine which rows were added to the table.
- DELETE – Use the DELETED table to see which rows were removed from the table.

- UPDATE – Use the INSERTED table to inspect the new or updated values and the DELETED table to see the values prior to update.

Definition

A trigger is defined for a specific table and one or more events. In most database management systems you can only define one trigger per table.

Below is an example trigger from the AdventureWorks2012 database.

```
BEGIN TRY
    INSERT INTO [Production].[TransactionHistory](
        [ProductID]
        ,[ReferenceOrderID]
        ,[TransactionType]
        ,[TransactionDate]
        ,[Quantity]
        ,[ActualCost])
    SELECT
        inserted.[ProductID]
        ,inserted.[WorkOrderID]
        ,'W'
        ,GETDATE()
        ,inserted.[OrderQty]
        ,0
    FROM inserted;
END TRY
BEGIN CATCH
    EXECUTE [dbo].[uspPrintError];

    -- Rollback any active or uncommittable transactions before
    -- inserting information in the ErrorLog
    IF @@TRANCOUNT > 0
    BEGIN
        ROLLBACK TRANSACTION;
    END

    EXECUTE [dbo].[uspLogError];
END CATCH;
```


You'll notice the syntax for a trigger is very similar to that of a stored procedure. In fact, the trigger uses the same language to implement its logic as do stored procedures. In MS SQL, this is T-SQL; whereas in Oracle it is PL/SQL.

Here are some important parts to a trigger:

1. The CREATE Statement – It defines which table is associated with the trigger. In addition this statement is used to specify when the trigger executes (e.g. after insert).
2. The actual program. In the example, this program runs whenever one or more rows are inserted into the WorkOrder table.
3. Special database objects – Triggers use specially defined databases objects such as INSERTED, or DELETED to access records affected by the database action.
4. In this example the trigger is using the INSERTED object to gain access to the newly created rows. The INSERT statement is used to table those rows and add them to a history table.

Uses for Triggers

Here are some common uses for triggers:

Complex Auditing

You can use triggers to track changes made to tables. In our example above, changes made to the WorkOrder table are recorded a TransactionHistory table.

Typically when creating audit trails, you'll use AFTER triggers.

You may think this is redundant, as many changes are logged in the databases journals, but the logs are meant for database recovery and aren't easily accessible by user programs. The

the logs are meant for database recovery and aren't easily accessible by user programs. The TransactionHistory table is easily referenced and can be incorporated into end user reports.

Enforce Business Rules

Triggers can be used to inspect all data before a DML action is performed. You can use INSTEAD OF triggers to "intercept" the pending DML operation, apply any business rules, and ultimately complete the transaction.

An example business rule may be that a customer status is defined as:

- Gold – Purchases over $1,000,000 in the past 12 months.
- Silver – Purchase of $500,000 to $1,000,000 in the past 12 months.
- Bronze – All other purchase levels.

An INSTEAD OF trigger could be defined to check the customer status each time a customer record is added or modified. The status check would involve creating a sum of all the customers' purchases and ensuring the new status corresponds with the sum of the last 12 months of purchases.

Derive Column Values

Triggers can be used to calculate column values. For instance, for each customer you may wish to maintain a TotalSales column on the customer record. Of course, for this to remain accurate, it would have to be update every time a sales was made.

This could be done using an AFTER trigger on INSERT, UPDATE, and DELETE statements for the Sales table.

Triggers Are Tricky!

In general, my advice is to avoid using triggers unless absolutely necessary.

You should avoid using triggers in place of built in features. For instance, rather than rely on triggers to enforce referential integrity, you're better off using relationships.

Here are some reasons why I shy away from them:

1. They can be hard to troubleshoot.
2. Triggers can cause other triggers to fire. Two Tables, A and B, both have an AFTER UPDATE trigger. If the AFTER UPDATE trigger on Table A updates Table B, then updating Table A causes it's trigger and then B's trigger to Fire.
3. You have to be sure you don't create a trigger storm! Can you imagine if Table B, for some reason, updated Table A? Now you have a circular reference... Boom!
4. I try to move as much logic into Stored Procedures and have applications make changes to the database through them rather than straight up SQL statements.

Kris Wenzel
Kris Wenzel has been working with databases over the past 28 years as a developer, analyst, and DBA. He has a BSE in Computer Engineering from the University of Michigan and a MBA from the University of Notre Dame. Kris has written hundreds of blog articles and many online courses. He loves helping others learn SQL.

aw

December 30, 2019

you ass fucking slave..mind ur own bsiness with bdsm databases