Data Management Using Microsoft SQL Server

Session: 12

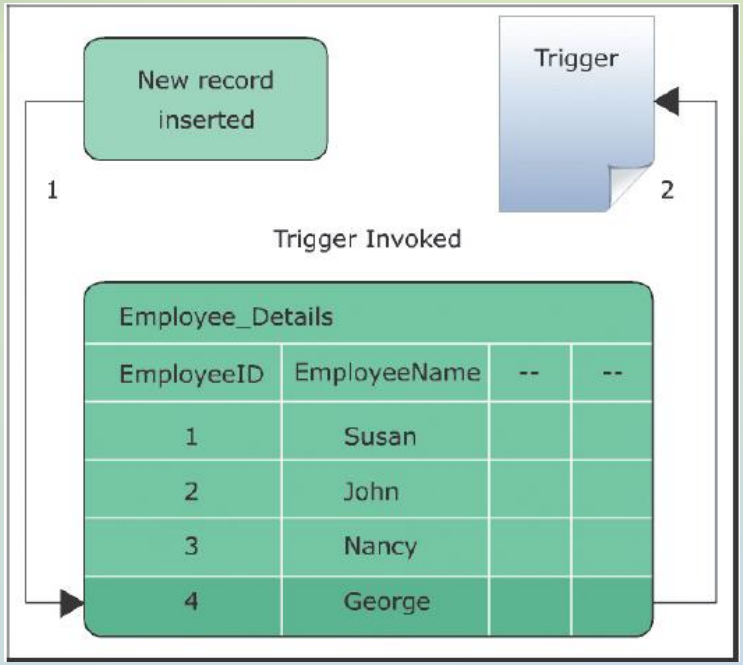
Triggers

Objectives

- Explain triggers
- Explain the different types of triggers
- Explain the procedure to create DML triggers
- Explain the procedure to alter DML triggers
- Describe nested triggers
- Describe update functions
- Explain the handling of multiple rows in a session
- Explain the performance implication of triggers

Introduction

- A trigger:
  - is a stored procedure that is executed when an attempt is made to modify data in a table protected by the trigger.
  - cannot be executed directly, nor do they pass or receive parameters.
  - is defined on specific tables and these tables are referred to as trigger tables.
  - is defined on the `INSERT`, `UPDATE`, or `DELETE` action on a table, it fires automatically when these actions are attempted.
  - is created using the `CREATE TRIGGER` statement.



Uses of Triggers

Triggers can contain complex processing logic and are generally used for maintaining low-level data integrity. Primary uses of triggers can be classified as follows:

Cascading changes through related tables

- Users can use a trigger to cascade changes through related tables.

Enforcing complex data integrity than CHECK constraints

- Unlike `CHECK` constraints, triggers can reference the columns in other tables.
- Can be used to apply complex data integrity checks by,
  - Checking constraints before cascading updates or deletes
  - Creating multi-row triggers for actions executed on multiple rows
  - Enforcing referential integrity between databases

Defining custom error messages

- Are used for providing more suitable or detailed explanations in certain error situations.

Transact-SQL Programming Elements

Transact-SQL programming elements enable to perform various operations that cannot be done in a single statement.

Maintaining denormalized data

- Low-level data integrity can be maintained in denormalized database environments using triggers.
- Denormalized data generally refers to redundant or derived data. Here, triggers are used for checks that do not require exact matches.

Comparing before and after states of data being modified

- Triggers provide the option to reference changes that are made to data by `INSERT`, `UPDATE`, and `DELETE` statements.

Types of Triggers

A trigger can be set to automatically execute an action when a language event occurs in a table or a view. Triggers in SQL Server 2012 can be classified into three basic types:

DML Triggers

- Execute when data is inserted, modified, or deleted in a table or a view using the `INSERT`, `UPDATE`, or `DELETE` statements.

DDL Triggers

- Execute when a table or a view is created, modified, or deleted using the `CREATE`, `ALTER`, or `DROP` statements.

Logon Triggers

- Execute stored procedures when a session is established with a `LOGON` event.

DDL Triggers versus DML Triggers

DDL and DML triggers have different uses and are executed with different database events.

- Following table lists some of the Transact-SQL control-of-flow language keywords:

DDL Triggers	DML Triggers
DDL triggers execute stored procedures on CREATE, ALTER, and DROP statements.	DML triggers execute on INSERT, UPDATE, and DELETE statements.
DDL triggers are used to check and control database operations.	DML triggers are used to enforce business rules when data is modified in tables or views.
DDL triggers operate only after the table or a view is modified.	DML triggers execute either while modifying the data or after the data is modified.
DDL triggers are defined at either the database or the server level.	DML triggers are defined at the database level.

Creating DML Triggers

DML triggers are executed when DML events occur in tables or views. These DML events include the `INSERT`, `UPDATE`, and `DELETE` statements.

DML triggers can execute either on completion of the DML events or in place of the DML events.

DML triggers enforce referential integrity by cascading changes to related tables when a row is modified.

DML triggers can perform multiple actions for each modification statement.

DML triggers are of three main types namely, `INSERT` trigger, `UPDATE` trigger, and `DELETE` trigger.

Introduction to Inserted and Deleted Tables

SQL statements in DML triggers use two special types of tables to modify data in the database. These tables are as follows:

Inserted Table

- Contains copies of records that are modified with the `INSERT` and `UPDATE` operations on the trigger table.
- The `INSERT` and `UPDATE` operations insert new records into the Inserted and Trigger tables.

Deleted Table

- Contains copies of records that are modified with the `DELETE` and `UPDATE` operations on the trigger table.
- These operations delete the records from the trigger table and insert them in the Deleted table.

The Inserted and Deleted tables do not physically remain present in the database and are created and dropped whenever any triggering events occur.

Insert Triggers 1-4

Are executed when a new record is inserted in a table.

Ensure that the value being entered conforms to the constraints defined on that table.

Save a copy of that record in the Inserted table and checks whether the new value in the Inserted table conforms to the specified constraints.

Insert the row in the trigger table if the record is valid otherwise, it displays an error message.

Are created using the `INSERT` keyword in the `CREATE TRIGGER` and `ALTER TRIGGER` statements.

Insert Triggers 2-4

Syntax:

```
CREATE TRIGGER [schema_name.] trigger_name
ON [schema_name.] table_name [WITH ENCRYPTION]
{FOR INSERT} AS
[IF UPDATE (column_name)...]
[{AND | OR} UPDATE (column_name)...]
<sql_statements>
```

where,

`schema_name`: specifies the name of the schema to which the table/trigger belongs.

`trigger_name`: specifies the name of the trigger.

`table_name`: specifies the table on which the DML trigger is created.

`WITH ENCRYPTION`: encrypts the text of the `CREATE TRIGGER` statement.

`FOR`: specifies that the DML trigger executes after the modification operations are complete.

`INSERT`: specifies that this DML trigger will be invoked by insert operations.

`UPDATE`: Returns a Boolean value that indicates whether an `INSERT` or `UPDATE` attempt was made on a specified column.

Insert Triggers 3-4

`column_name`: Is the name of the column to test for the `UPDATE` action.

`AND`: Combines two Boolean expressions and returns TRUE when both expressions are TRUE.

`OR`: Combines two Boolean expressions and returns TRUE if at least one expression is TRUE.

`sql_statement`: specifies the SQL statements that are executed in the DML trigger.

- Following code snippet shows how to create an `INSERT` trigger on a table named **`Account_Transactions`**:

```
CREATE TRIGGER CheckWithdrawal_Amount
ON Account_Transactions
FOR INSERT
AS
IF (SELECT Withdrawal From inserted) > 80000
BEGIN
PRINT 'Withdrawal amount cannot exceed 80000'
ROLLBACK TRANSACTION
END
```

Insert Triggers 4-4

- Following code snippet inserts a record and displays an error message when the Withdrawal amount exceeds 80000:

```
INSERT INTO Account_Transactions
(TransactionID, EmployeeID, CustomerID,TransactionTypeID,TransactionDate,
TransactionNumber,Deposit,Withdrawal)
VALUES
(1008,'E08','C08','T08','05/02/12','TN08', 300000,90000)
```

Output:

```
Withdrawal amount cannot exceed 80000.
```

Update Triggers 1-5

Copies the original record in the Deleted table and the new record into the Inserted table when a record is updated.

Evaluates the new record to determine if the values conform to the constraints specified in the trigger table.

Copies the record from the Inserted table to the trigger table provided the record is valid.

Displays an error message if the new values are invalid and copies the record from the Deleted table back into the trigger table.

Is created using the `UPDATE` keyword in the `CREATE TRIGGER` and `ALTER TRIGGER` statements.

Update Triggers 2-5

Syntax:

```
CREATE TRIGGER [schema_name.] trigger_name
ON [schema_name.] table_name [WITH ENCRYPTION]
{FOR UPDATE} AS
[IF UPDATE (column_name)...]
[{AND | OR} UPDATE (column_name)...]
<sql_statements>
```

where,

`schema_name`: specifies the name of the schema to which the table/trigger belongs.

`trigger_name`: specifies the name of the trigger.

`table_name`: specifies the table on which the DML trigger is created.

`WITH ENCRYPTION`: encrypts the text of the `CREATE TRIGGER` statement.

`FOR`: specifies that the DML trigger executes after the modification operations are complete.

`INSERT`: specifies that this DML trigger will be invoked after the update operations.

`UPDATE`: Returns a Boolean value that indicates whether an `INSERT` or `UPDATE` attempt was made on a specified column.

Update Triggers 3-5

`column_name`: Is the name of the column to test for the `UPDATE` action.

`AND`: Combines two Boolean expressions and returns TRUE when both expressions are TRUE.

`OR`: Combines two Boolean expressions and returns TRUE if at least one expression is TRUE.

`sql_statement`: specifies the SQL statements that are executed in the DML trigger.

- Following code snippet shows how to create an `UPDATE` trigger at the table level on the **`EmployeeDetails`** table:

```
CREATE TRIGGER CheckBirthDate
ON EmployeeDetails
FOR UPDATE
AS
IF (SELECT BirthDate From inserted) > getDate()
BEGIN
PRINT 'Date of birth cannot be greater than today's date'
ROLLBACK
END
```

Update Triggers 4-5

- Following code snippet updates a record and displays an error message when an invalid date of birth is specified:

```
UPDATE EmployeeDetails
SET BirthDate='2015/06/02'
WHERE EmployeeID='E06')
```

Output:

```
Date of birth cannot be greater than today's date.
```

Creating Update Triggers

- Are created either at the column level or at the table level.
- Triggers at the column level execute when updates are made in the specified column.
- Triggers at the table level execute when updates are made anywhere in the entire table.
- `UPDATE()` function is used to specify the column when creating an `UPDATE` trigger at the column level.

Update Triggers 5-5

- Following code snippet creates an `UPDATE` trigger at the column level on the **`EmployeeID`** column of **`EmployeeDetails`** table:

```
CREATE TRIGGER Check_EmployeeID
ON EmployeeDetails
FOR UPDATE
AS
IF UPDATE(EmployeeID)
BEGIN
PRINT 'You cannot modify the ID of an employee'
ROLLBACK TRANSACTION
END
```

- Following code snippet causes the update trigger to fire:

```
UPDATE EmployeeDetails
SET EmployeeID='E12'
WHERE EmployeeID='E04'
```

Delete Triggers 1-3

Can be created to restrict a user from deleting a particular record in a table.

The following will happen if the user tries to delete the record:

- The record is deleted from the trigger table and inserted in the Deleted table.
- It is checked for constraints against deletion.
- If there is a constraint on the record to prevent deletion, the `DELETE` trigger displays an error message.
- The deleted record stored in the Deleted table is copied back to the trigger table.

Is created using the `DELETE` keyword in the `CREATE TRIGGER` statement.

Delete Triggers 2-3

Syntax:

```
CREATE TRIGGER <trigger_name>
ON <table_name>
[WITH ENCRYPTION]
FOR DELETE
AS <sql_statement>
```

where,

`DELETE`: specifies that this DML trigger will be invoked by delete operations.

Delete Triggers 3-3

- Following code snippet shows how to create a `DELETE` trigger on the **`Account_Transactions`** table:

```
CREATE TRIGGER CheckTransactions
ON Account_Transactions
FOR DELETE
AS
IF 'T01' IN (SELECT TransactionID FROM deleted)
BEGIN
PRINT 'Users cannot delete the transactions.'
ROLLBACK TRANSACTION
END
```

- Following code snippet delete records from the **`Account_Transactions`** table where Deposit is 50000 and displays an error message:

```
DELETE FROM Account_Transactions
WHERE Deposit= 50000
```

Output:

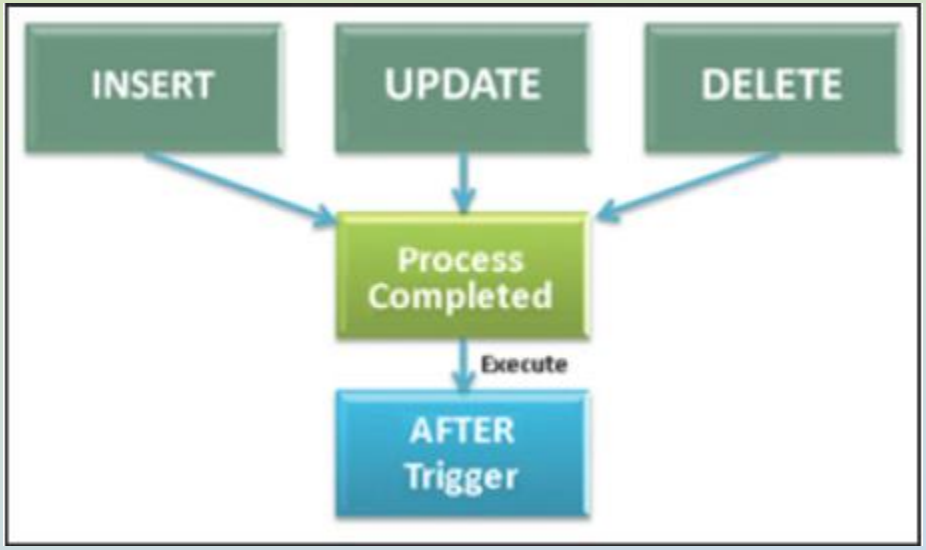
```
Users cannot delete the transactions.
```

AFTER Triggers 1-3

Is executed on completion of `INSERT`, `UPDATE`, or `DELETE` operations and can be created only on tables.

A table can have multiple `AFTER` triggers defined for each `INSERT`, `UPDATE`, and `DELETE` operation and the user must define the order of execution of triggers.

Is executed when the constraint check in the table is completed and also the trigger is executed after the Inserted and Deleted tables are created.



AFTER Triggers 2-3

Syntax:

```
CREATE TRIGGER <trigger_name>
ON <table_name>
[WITH ENCRYPTION]
{ FOR | AFTER }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS <sql_statement>
```

where,

`FOR | AFTER`: specifies that DML trigger executes after the modification operations are complete.

`{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }`: specifies the operations that invoke the DML trigger.

AFTER Triggers 3-3

- Following code snippet shows how to create an `AFTER DELETE` trigger on the **`EmployeeDetails`** table:

```
CREATE TRIGGER Employee_Deletion
ON EmployeeDetails
AFTER DELETE
AS
BEGIN
DECLARE @num nchar;
SELECT @num = COUNT(*) FROM deleted
PRINT 'No. of employees deleted = ' + @num
END
```

- Following code snippet deletes a record from the **`EmployeeDetails`** table and displays an error message:

```
DELETE FROM EmployeeDetails WHERE EmployeeID='E07'
```

Output:

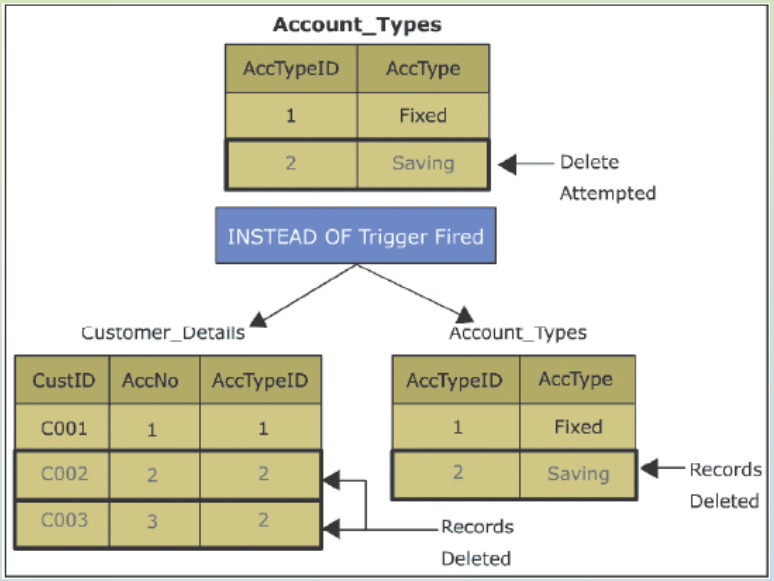
```
No. of employees deleted = 0.
```

INSTEAD OF Triggers 1-3

Is executed in place of the `INSERT`, `UPDATE`, or `DELETE` operations.

Can be created on tables as well as views and there can be only one `INSTEAD OF` trigger defined for each `INSERT`, `UPDATE`, and `DELETE` operation.

Are executed before constraint checks are performed on the table and after the creation of the Inserted and Deleted tables.



INSTEAD OF Triggers 2-3

Syntax:

```
CREATE TRIGGER <trigger_name>
ON { <table_name> | <view_name> }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS <sql_statement>
```

where,

`view_name`: specifies the view on which the DML trigger is created.

`INSTEAD OF`: specifies that the DML trigger executes in place of the modification operations. These triggers are not defined on updatable views using `WITH CHECK OPTION`.

INSTEAD OF Triggers 3-3

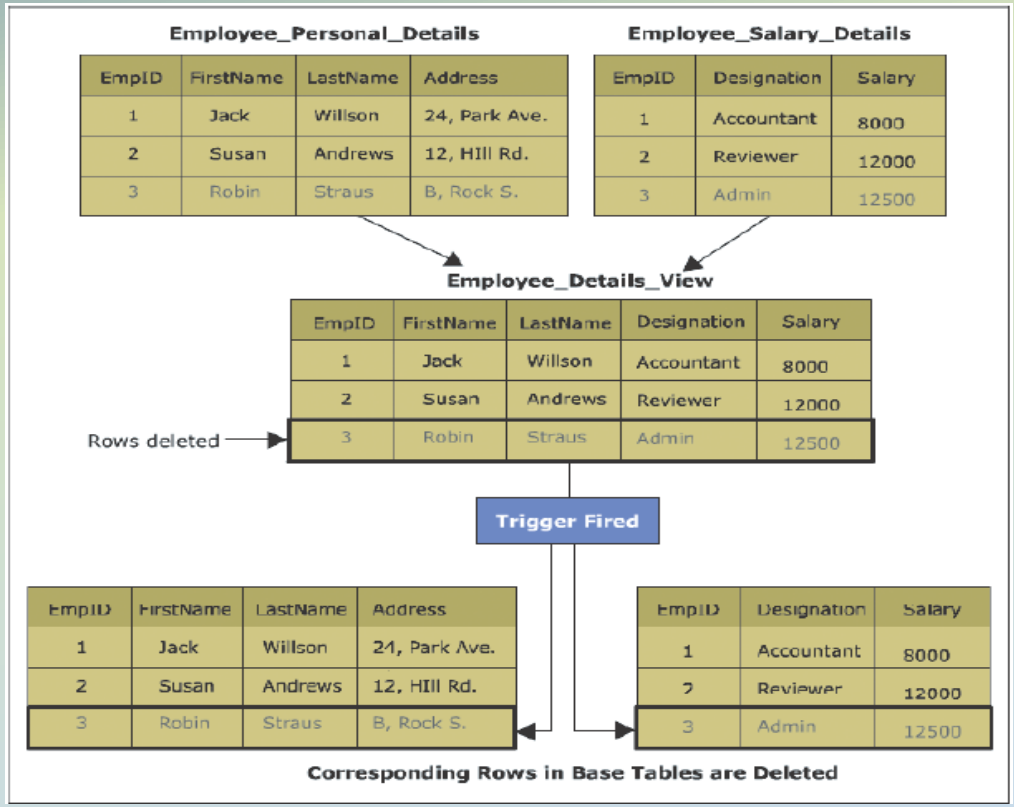
- Following code snippet creates an `INSTEAD OF DELETE` trigger on the **`Account_Transactions`** table:

```
CREATE TRIGGER Delete_AccType
ON Account_Transactions
INSTEAD OF DELETE
AS
BEGIN
DELETE FROM EmployeeDetails WHERE EmployeeID IN
(SELECT TransactionTypeID FROM deleted)
DELETE FROM Account_Transactions WHERE TransactionTypeID IN
(SELECT TransactionTypeID FROM deleted)
END
```

Using INSTEAD OF Triggers with Views 1-3

Can be specified on tables as well as views and provides a wider range and types of updates that the user can perform against a view.

Each table or view is limited to only one `INSTEAD OF` trigger for each triggering action (`INSERT`, `UPDATE`, or `DELETE`) and cannot be used with views having `WITH CHECK OPTION` clause.



Using INSTEAD OF Triggers with Views 2-3

- Following code snippet creates a table named **Employee_Personal_Details**:

```
CREATE TABLE Employee_Personal_Details
(
EmpID int NOT NULL,
FirstName varchar(30) NOT NULL,
LastName varchar(30) NOT NULL,
Address varchar(30)
)
```

- Following code snippet creates a table named **Employee_Salary_Details**:

```
CREATE TABLE Employee_Salary_Details
(
EmpID int NOT NULL,
Designation varchar(30),
Salary int NOT NULL
)
```

Using INSTEAD OF Triggers with Views 3-3

- Following code snippet creates a view from table named **Employee_Personal_Details** and **Employee_Salary_Details**:

```
CREATE VIEW Employee_Details_View
AS
SELECT e1.EmpID, FirstName, LastName, Designation, Salary
FROM Employee_Personal_Details e1
JOIN Employee_Salary_Details e2
ON e1.EmpID = e2.EmpID
```

- Following code snippet creates an `INSTEAD OF DELETE` trigger **Delete_Employees** on the view:

```
CREATE TRIGGER Delete_Employees
ON Employee_Details_View
INSTEAD OF DELETE
AS
BEGIN
DELETE FROM Employee_Salary_Details WHERE EmpID IN
(SELECT EmpID FROM deleted)
DELETE FROM Employee_Personal_Details WHERE EmpID IN
(SELECT EmpID FROM deleted)
```

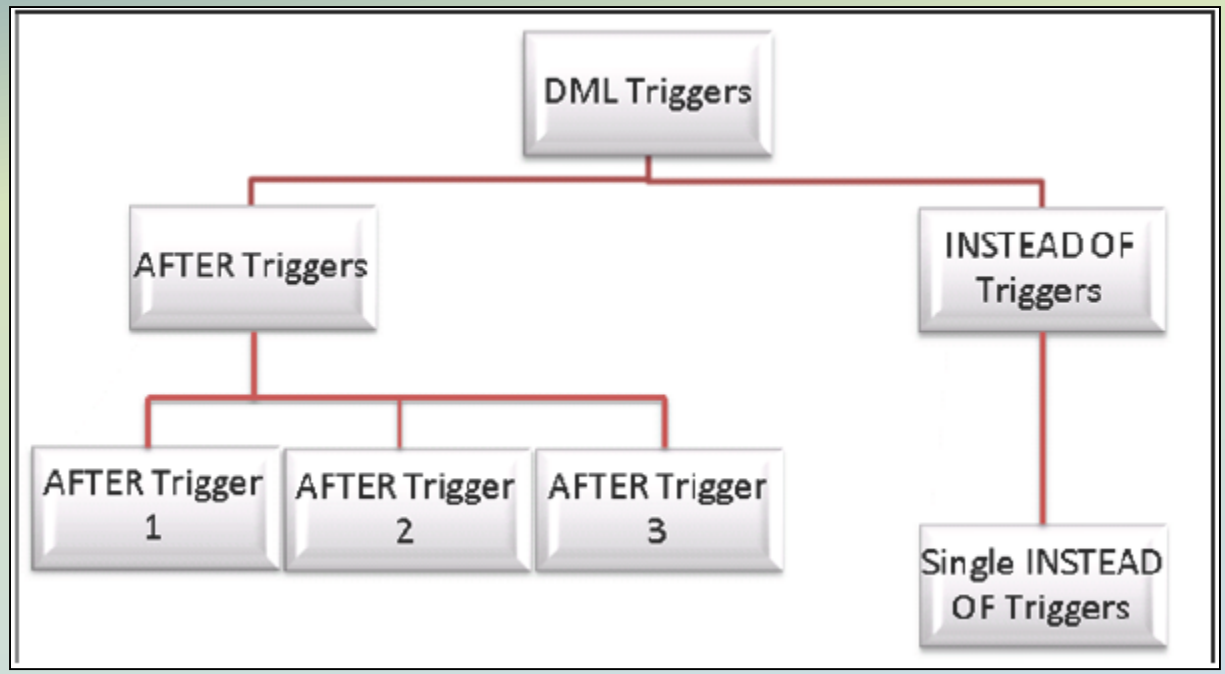
- Following code snippet deletes a row from the view:

```
DELETE FROM Employee_Details _View WHERE EmpID='3'
```

Working with DML Triggers 1-3

When users have multiple `AFTER` triggers on a triggering action, all of these triggers must have a different name.

An `AFTER` trigger can include a number of SQL statements that perform different functions.

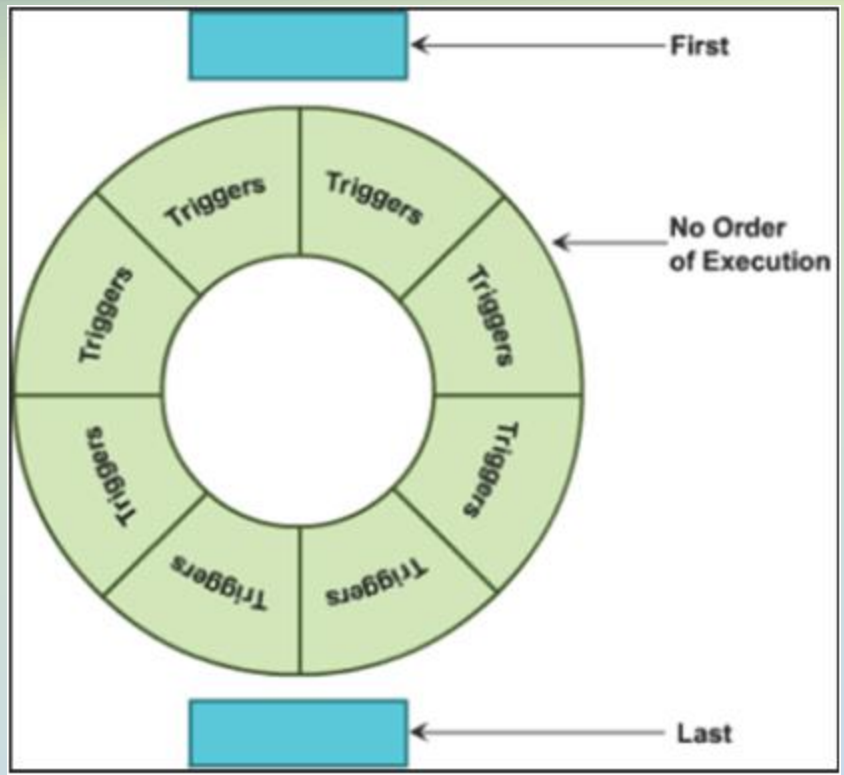


Working with DML Triggers 2-3

- Execution Order of DML Triggers

SQL Server 2012 allows users to specify which `AFTER` trigger is to be executed first and which is to be executed last.

All the triggering actions have a first and last trigger defined for them. However, no two triggering actions on a table can have the same first and last triggers.



Working with DML Triggers 3-3

Syntax:

```
sp_settriggerorder [ @triggername = ] '[ triggerschema. ] triggername'
, [ @order = ] 'value'
, [ @stmttype = ] 'statement_type'
```

where,

`[ triggerschema.] triggername`: is the name of the DML or DDL trigger and the schema to which it belongs and whose order needs to be specified.

`value`: specifies the execution order of the trigger as `FIRST`, `LAST`, or `NONE`. If `FIRST` is specified, then the trigger is fired first.

`statement_type`: specifies the type of SQL statement (`INSERT`, `UPDATE`, or `DELETE`) that invokes the DML trigger.

- Following code snippet executes the **`Employee_Deletion`** trigger defined on the table when the **`DELETE`** operation is performed:

```
EXEC sp_settriggerorder @triggername = 'Employee_Deletion ', @order =
    'FIRST', @stmttype = 'DELETE
```

Viewing Definitions of DML Triggers

A trigger definition includes the trigger name, the table on which the trigger is created, the triggering actions, and the SQL statements that are executed.

SQL Server 2012 provides `sp_helptext` stored procedure to retrieve the trigger definitions.

DML trigger name must be specified as the parameter when executing `sp_helptext`.

Syntax:

```
sp_helptext '<DML_trigger_name>'
```

where,

`DML_trigger_name`: specifies the name of the DML trigger whose definitions are to be displayed.

- Following code snippet creates a table named **`Employee_Salary_Details`**:

```
sp_helptext 'Employee_Deletion'
```

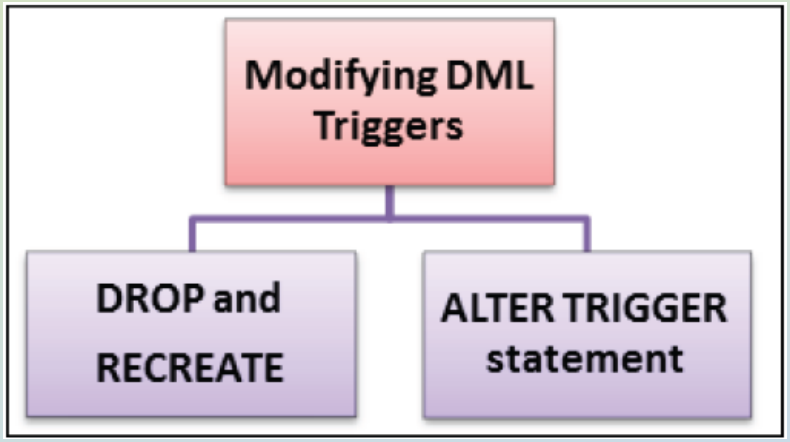
Modifying Definitions of DML Triggers 1-3

Trigger parameters are defined at the time of creating a trigger and include the type of triggering action that invokes the trigger and the SQL statements that are executed.

User can modify any of these parameters for a DML trigger in any one of two ways:

- Drop and re-create the trigger with the new parameters.
- Change the parameters using the `ALTER TRIGGER` statement.

A DML trigger can be encrypted to hide its definition.



Modifying Definitions of DML Triggers 2-3

Syntax:

```
ALTER TRIGGER <trigger_name>
ON { <table_name> | <view_name> }
[WITH ENCRYPTION]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS <sql_statement>
```

where,

`WITH ENCRYPTION`: specifies that the DML trigger definitions are not displayed.

`FOR | AFTER`: specifies that the DML trigger executes after the modification operations are complete.

`INSTEAD OF`: specifies that the DML trigger executes in place of the modification operations.

Modifying Definitions of DML Triggers 3-3

- Following code snippet alters the **CheckEmployeeID** trigger created on the **EmployeeDetails** table using the `WITH ENCRYPTION` option:

```
ALTER TRIGGER CheckEmployeeID
ON EmployeeDetails
WITH ENCRYPTION
FOR INSERT
AS
IF 'E01' IN (SELECT EmployeeID FROM inserted)
BEGIN
PRINT 'User cannot insert the customers of Austria'
ROLLBACK TRANSACTION
END
```

Output:

```
The text for object CheckEmployeeID is encrypted.
```

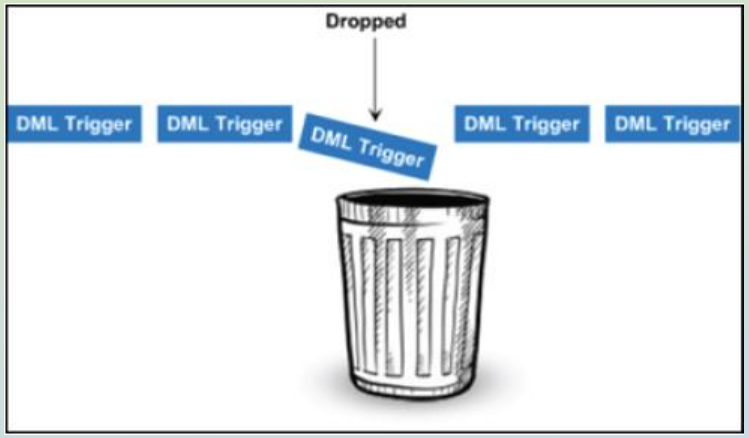
Dropping DML Triggers 1-2

Trigger can be dropped using the `DROP TRIGGER` statement.

Multiple triggers can also be dropped using a single drop trigger statement.

- When a table is dropped, all the triggers defined on that table are also dropped.

When the DML trigger is deleted from the table, the information about the trigger is also removed from the catalog views.



Dropping DML Triggers 2-2

Syntax:

```
DROP TRIGGER <DML_trigger_name> [ ,...n ]
```

where,

`DML_trigger_name`: specifies the name of the DML trigger to be dropped.

`[ ,...n ]`: specifies that multiple DML triggers can be dropped.

- Following code snippet drops the **`CheckEmployeeID`** trigger created on the **`EmployeeDetails`** table:

```
DROP TRIGGER CheckEmployeeID
```

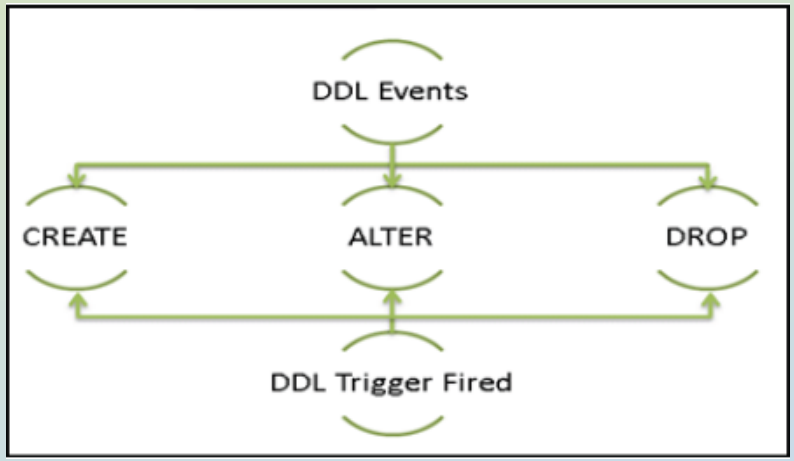
DDL Triggers 1-2

Data Definition Language (DDL) triggers execute stored procedures when DDL events such as `CREATE`, `ALTER`, and `DROP` statements occur in the database or the server.

DDL triggers can operate only on completion of the DDL events.

DDL triggers can be used to prevent modifications in the database schema. A schema is a collection of objects such as tables, views, and so forth in a database.

DDL triggers can invoke an event or display a message based on the modifications attempted on the schema and are defined either at the database level or at the server level.



DDL Triggers 2-2

Syntax:

```
CREATE TRIGGER <trigger_name>
ON { ALL SERVER | DATABASE }
[WITH ENCRYPTION]
{ FOR | AFTER } { <event_type> }
AS <sql_statement>
```

where,

`ALL SERVER`: specifies that the DDL trigger executes when DDL events occur in the current server.

`DATABASE`: specifies that the DDL trigger executes when DDL events occur in the current database.

`event_type`: specifies the name of the DDL event that invokes the DDL trigger.

- Following code snippet creates a DDL trigger for dropping and altering a table:

```
CREATE TRIGGER Secure
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
PRINT 'You must disable Trigger "Secure" to drop or alter tables!'
ROLLBACK
```

Scope of DDL Triggers 1-2

DDL triggers are invoked by SQL statements executed either in the current database or on the current server.

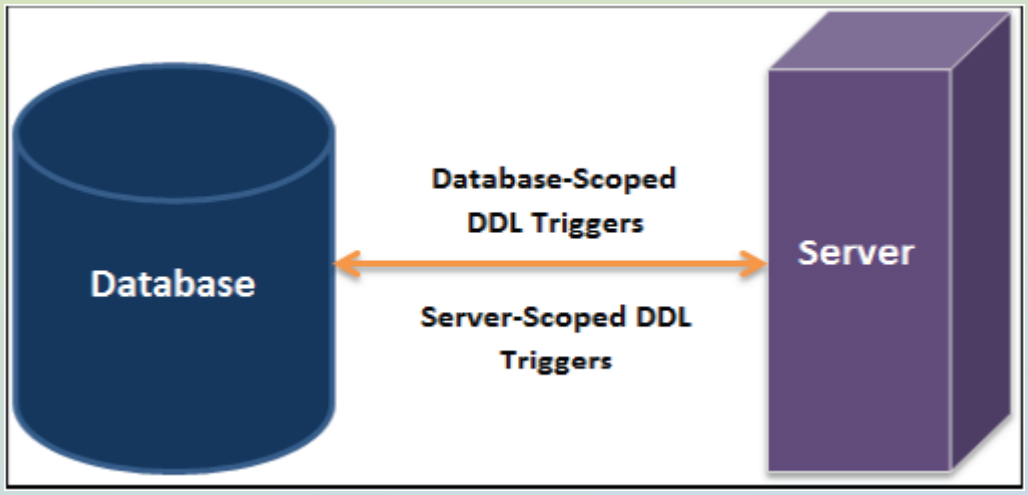
A DDL trigger created for a `CREATE LOGIN` statement executes on the `CREATE LOGIN` event in the server.

Scope of the DDL trigger depends on whether the trigger executes for database events or server events.

Scope of DDL Triggers 2-2

DDL triggers are classified into two types, which are as follows:

- Database-Scoped DDL Triggers:
  - are invoked by the events that modify the database schema.
  - stores the triggers in the database and execute on DDL events, except those related to temporary tables.
- Server-Scoped DDL Triggers:
  - are invoked by DDL events at the server level.
  - are stored in the master database.



Nested Triggers 1-2

Both DDL and DML triggers are nested when a trigger implements an action that initiates another trigger.

DDL and DML triggers can be nested up to 32 levels.

Nested triggers can be used to perform the functions such as storing the backup of the rows that are affected by the previous actions.

A Transact-SQL trigger executes the managed code through referencing a CLR routine, aggregate, or type, that references the counts as one level against the 32-level nesting limit.

Users can disable nested triggers, by setting the nested triggers option of `sp_configure` to 0 or off.

Nested Triggers 2-2

- Following code snippet creates an `AFTER DELETE` trigger named **`Employee_Deletion`** on the **`Employee_Personal_Details`** table:

```
CREATE TRIGGER Employee_Deletion
ON Employee_Personal_Details
AFTER DELETE
AS
BEGIN
PRINT 'Deletion will affect Employee_Salary_Details table'
DELETE FROM Employee_Salary_Details WHERE EmpID IN
(SELECT EmpID FROM deleted)
END
```

UPDATE 1-2

Returns a Boolean value that specifies whether an `UPDATE` or `INSERT` action was performed on a specific view or column of a table.

Can be used anywhere inside the body of a Transact-SQL `UPDATE` or `INSERT` trigger to test whether the trigger should execute some actions.

Syntax:

```
UPDATE ( column )
```

where,

`column`: is the name of the column to test for either an `INSERT` or `UPDATE` action.

UPDATE 2-2

- Following code snippet creates a trigger `Accounting` on the **`Account_Transactions`** table to update the columns **`TransactionID`** or **`EmployeeID`**:

```
CREATE TRIGGER Accounting
ON Account_Transactions
AFTER UPDATE
AS
IF ( UPDATE (TransactionID) OR UPDATE (EmployeeID) )
BEGIN
RAISERROR (50009, 16, 10)
END;
GO
```

Handling of Multiple Rows in a Session

When a user writes the code for a DML trigger, then the statement that causes the trigger to fire will be single statement.

Single statement will affect multiple rows of data, instead of a single row.

When the functionality of a DML trigger involves automatically recalculating summary values of one table and storing the result in another table, then multirow considerations are important.

- Following code snippet stores a running total for a single-row insert:

```
USE AdventureWorks2012;
GO
CREATE TRIGGER PODetails
ON Purchasing.PurchaseOrderDetail
AFTER INSERT AS
UPDATE PurchaseOrderHeader
SET SubTotal = SubTotal + LineTotal
FROM inserted
WHERE PurchaseOrderHeader.PurchaseOrderID = inserted.PurchaseOrderID;
```

Performance Implication of Triggers

Triggers do not carry overheads, rather they are quite responsive.

Many performance issues can occur because of the logic present inside the trigger.

A good rule will be to keep the logic simple within the triggers and avoid using cursors while executing statements against another table and different tasks that cause performance slowdown.

Summary

- A trigger is a stored procedure that is executed when an attempt is made to modify data in a table that is protected by the trigger.
- Logon triggers execute stored procedures when a session is established with a LOGON event.
- DML triggers are executed when DML events occur in tables or views.
- The INSERT trigger is executed when a new record is inserted in a table.
- The UPDATE trigger copies the original record in the Deleted table and the new record into the Inserted table when a record is updated.
- The DELETE trigger can be created to restrict a user from deleting a particular record in a table.
- The AFTER trigger is executed on completion of INSERT, UPDATE, or DELETE operations.