Triggers in SQL Server 2005

Triggers in SQL Server 2005

- Two types:
  - DML triggers
  - DDL triggers

DML Triggers

- Rules Event-Action
- There is no condition

DML Triggers

CREATE TRIGGER [*schema_name* .]*trigger_name*
ON { *table* | *view* }
{ FOR | AFTER | INSTEAD OF }
{ [INSERT] [,] [UPDATE] [,] [DELETE] }
AS { *sql_statement* [;] [,...*n*] |
EXTERNAL NAME <method specifier [;] > }

Execution

- EXTERNAL NAME <method specifier [;] > : allows to specify a CLR method to be executed
- FOR, AFTER: the sql statement is executed after the event
- INSTEAD OF: the sql statement is executed instead of the event
- There is no BEFORE case

Execution

- AFTER triggers fire after the triggering action (INSERT, UPDATE, or DELETE), INSTEAD OF triggers and constraints are processed.
- INSTEAD OF triggers fire in place of the triggering action and before constraints are processed.
- If the constraints are violated, the INSTEAD OF trigger actions are rolled back and the AFTER trigger is not executed

Execution

- Each table or view can have one INSTEAD OF trigger for each triggering action (UPDATE, DELETE, and INSERT).
- A table can have several AFTER triggers for each triggering action

Deleted and Inserted Table

- DML triggers use the **deleted** and **inserted** logical (conceptual) tables.
- They have the same structure of the table on which the trigger is defined, that is, the table on which the user action is tried.
- The **deleted** and **inserted** tables hold the old values or new values of the rows that are changed by the user action.
- For example, to retrieve all values in the deleted table, use:

```
SELECT *
FROM deleted
```

Deleted and Inserted Table

- DELETE: the deleted table contains all the rows that have been deleted, the inserted table is empty
- INSERT: the inserted table contains all the rows that have been inserted, the deleted table is empty
- UPDATE: it is seen as a delete followed by an insert, deleted contains the old versions of the rows updated, inserted the new versions
- The table on which the trigger is defined is modified when the trigger fires if the trigger is AFTER, it is not modified if it is INSTEAD OF

Deleted and Inserted Table

- For a trigger on a table the format of the **inserted** and **deleted** tables is the same as the format of the table. Each column in the **inserted** and **deleted** tables maps directly to a column in the base table.
- For a trigger on a view the format of the **inserted** and **deleted** tables passed to a trigger matches the select list of the SELECT statement defined for the view

Trigger Level

- There is no distinction between row-level triggers and statement-level triggers
- All the triggers are statement-level
- If multiple rows are affected, the inserted and deleted tables will have more than one row
- Special care must be taken to consider these cases

Triggers and Constraints

- AFTER triggers are never executed if a constraint violation occurs; therefore, these triggers cannot be used for any processing that might prevent constraint violations.
- INSTEAD OF triggers are executed instead of the triggering action. These triggers are executed after the **inserted** and **deleted** tables reflecting the changes to the base table are created, but before any other actions are taken. They are executed before any constraints, so can perform preprocessing that supplements the constraint actions.

Execution

- If an INSTEAD OF trigger defined on a **table** executes a statement against the table that would usually fire the INSTEAD OF trigger again, the trigger is not called recursively.
- Instead, the statement is processed as if the table had no INSTEAD OF trigger and starts the chain of constraint operations and AFTER trigger executions.
- For example, if a DML trigger is defined as an INSTEAD OF INSERT trigger for a table, and the trigger executes an INSERT statement on the same table, the INSERT statement executed by the INSTEAD OF trigger does not call the trigger again.
- The INSERT executed by the trigger starts the process of performing constraint actions and firing any AFTER INSERT triggers defined for the table.

Execution

- If an INSTEAD OF trigger defined on a **view** executes a statement against the view that would usually fire the INSTEAD OF trigger again, it is not called recursively. Instead, the statement is resolved as modifications against the base tables underlying the view. In this case, the view definition must meet all of the restrictions for an updatable view.
- For example, if a DML trigger is defined as an INSTEAD OF UPDATE trigger for a view, and the trigger executes an UPDATE statement referencing the same view, the UPDATE statement executed by the INSTEAD OF trigger does not call the trigger again.
- The UPDATE executed by the trigger is processed against the view as if the view did not have an INSTEAD OF trigger. Each modification to an underlying base table starts the chain of applying constraints and firing AFTER triggers defined for the table.

Position

- The CREATE TRIGGER statement must be the first statement in the batch.
- All other statements that follow in that batch are interpreted as part of the definition of the CREATE TRIGGER statement.

INSTEAD OF Triggers

- The primary advantage of INSTEAD OF triggers is that they enable views that would not be updatable to support updates.
- A view based on multiple base tables must use an INSTEAD OF trigger to support inserts, updates, and deletes that reference data in more than one table.
- Another advantage of INSTEAD OF triggers is that they enable you to code logic that can reject parts of a batch while letting other parts of a batch to succeed.

Updatable Views

- You can modify the data of an underlying base table through a view, as long as the following conditions are true:
  - Any modifications, including UPDATE, INSERT, and DELETE statements, must reference columns from only one base table.
  - The columns being modified in the view must directly reference the underlying data in the table columns. The columns cannot be derived in any other way, such as through the following:
    - An aggregate function: AVG, COUNT, SUM, MIN, MAX, GROUPING
    - A computation. The column cannot be computed from an expression that uses other columns.

Updatable Views

- The columns being modified are not affected by GROUP BY, HAVING, or DISTINCT clauses.
- TOP is not used anywhere in the *select_statement* of the view together with the WITH CHECK OPTION clause.

Nulls and Defaults

- Similarly, when no value is specified in the INSERT statement for a column, a DML trigger is still activated when:
  - An implicit null value is inserted into a column because no DEFAULT definition exists.
  - A default value is inserted into a column because a DEFAULT definition does exist.

Example

- The SubTotal column in the PurchaseOrderHeader table must be the sum of the LineTotal column for all the related rows in the PurchaseOrderDetail table
- The SubTotal column in the PurchaseOrderHeader must be updated when new lines are inserted in PurchaseOrderDetail

Example

```
-- Trigger is valid for single-row inserts.
USE AdventureWorks;
GO
CREATE TRIGGER NewPODetail
ON Purchasing.PurchaseOrderDetail
AFTER INSERT AS
   UPDATE PurchaseOrderHeader
   SET SubTotal = SubTotal + LineTotal
   FROM inserted
   WHERE PurchaseOrderHeader.PurchaseOrderID =
    inserted.PurchaseOrderID ;
```

Example

- If there are more than one row in inserted, it is not defined which one is used for the update

Example 2

```
-- Trigger is valid for multirow and single-row inserts.
USE AdventureWorks;
GO
CREATE TRIGGER NewPODetail2
ON Purchasing.PurchaseOrderDetail
AFTER INSERT AS
   UPDATE PurchaseOrderHeader
   SET SubTotal = SubTotal +
      (SELECT SUM(LineTotal)
      FROM inserted
      WHERE PurchaseOrderHeader.PurchaseOrderID
       = inserted.PurchaseOrderID)
   WHERE PurchaseOrderHeader.PurchaseOrderID IN
      (SELECT PurchaseOrderID FROM inserted);
```

DDL Triggers

- DDL Triggers are a special kind of trigger that fire in response to Data Definition Language (DDL) statements.
- They can be used to perform administrative tasks in the database such as auditing and regulating database operations.
- DDL triggers fire only after the DDL statements that trigger them are run. DDL triggers cannot be used as INSTEAD OF triggers

DDL Triggers

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

Example

- The following example illustrates how a DDL trigger can be used to prevent any table in a database from being modified or dropped:

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

Syntax

CREATE TRIGGER *trigger_name* ON
{ ALL SERVER | DATABASE }
{ FOR | AFTER } { *event_type* | *event_group* } [,...*n*]
AS { *sql_statement* [;] [,...*n*] |
    EXTERNAL NAME < method specifier > [;] }

ALL SERVER: the trigger fires for any event in the current server

DATABASE: the trigger fires for any event in the current database