NOTE: Defining triggers differs dramatically between DBMS

DB Fundamentals

TRIGGERS

DML Triggers

A trigger is a special kind of a stored procedure that executes in response to certain action on a table (or in a DB)

The action could be

- On insertion of a new record
- On deletion of existing record
- On Updating of an existing record

- Note that DDL triggers also exist
  - These activate on DDL statements like CREATE TABLE, CREATE LOGIN . . .

DML Triggers

Triggers are mechanisms for DBMS to perform tasks when it detects certain events in a **table**

A trigger is an "**Event – Condition – Action**" Rule

- Event: data update, specified by insert or delete or update, Table Created/Altered, Permission Granted etc
- Condition (optional): SQL predicate (WHERE)
- Action: sequence of SQL statements (or a procedure)

DML Triggers

A trigger is activated when

- A specified event occurs (triggering)
- If the optional condition is satisfied (consideration)
- The trigger will then do the action (execution)
- A trigger is only activated by events occurring on that table

DML Triggers

Why use triggers?

- Create an automatic record history/audit log
  - They can evaluate the state of a table before and after a data modification and take actions based on that difference.
- Update columns containing running totals
- Guard against malicious or incorrect INSERT, UPDATE, and DELETE operations
  - They can enforce restrictions that are more complex than those defined with CHECK constraints.
  - Unlike CHECK constraints, DML triggers can reference columns in other tables.

DML Triggers

Why use triggers?

- Constraints communicate errors through standard (ugly) system error messages.
  - Triggers can be used to customise error messages and allow for more complex error handling

- DML triggers can **roll back** pending data changes thus cancelling the attempted data modification

  - A trigger can un-do an insert, update or delete if some given condition is not met

  - E.g. too many students enrolled in a single tutorial class

Triggers - Definition

MS SQL Server

```
CREATE TRIGGER TriggerName
ON someTable | view
FOR | AFTER | INSTEAD OF
INSERT, UPDATE,  DELETE
AS
BEGIN
-- <some query> – Inserted/Deleted
-- SQL statements here
END
```

SQL Standard

```
CREATE TRIGGER TriggerName
BEFORE | AFTER | INSTEAD OF
INSERT, UPDATE, DELETE
ON TargetTable
        [referencing new/old vals]
        [ for each row ]
        WHEN (condition)
ACTION
```


https://msdn.microsoft.com/en-au/library/ms189799.aspx

Triggers in MS SQL fire once per SQL statement and not once per row. As such, code must be written to handle multiple values because the trigger cannot be declared with “For each row”

Triggers – MS SQL

Triggers in MS-SQL are a little easier

- There are the standard 3 action query types
  1. INSERT
  2. UPDATE
  3. DELETE

- These can be matched with the events and timings that trigger them
  1. FOR <query type above>
  2. AFTER <query type above>
  3. INSTEAD OF <query type above>

- There are 2 special tables
  - Inserted
  - Deleted

This relates directly to MS-SQL

Triggers - Definition

Referencing new/old data is restricted based on the triggering event:

- INSERT
  - Can only refer to new data in the **inserted** table
  - The inserted tale contains all new data rows
- ON DELETE
  - Can only refer to old data rows in the **deleted** table
  - The deleted table contains all deleted data rows
- ON UPDATE
  - Can refer to old data rows in the **deleted** table AND/OR new data rows in the **inserted** table

Triggers – Special Tables

In MS-SQL triggers can utilise two special tables tat cannot be directly manipulated

- **Inserted**
  - Contains all the data referenced in an INSERT statement before it is actually committed to the database
- **Deleted**
  - Contains all the data in the base table referenced in a DELETE statement before it is actually removed from the database
- What about queries involving UPDATE?
  - Both **Inserted and Deleted** are used:
  - The Deleted table contains the **original data** in the base table before the any changes have been committed
  - The Inserted table contains all the **new data** that has not yet been committed to the database

Triggers – MS SQL

FOR Triggers

- These triggers run ~~**BEOFRE**~~ their respective event (INSERT, UPDATE, DELETE)
  - Use these to capture changes ~~**BEFORE**~~ they take place (well… not in SQL SERVER ☹)
  - Can be used to check data, log before and after changes or even block changes!


```
CREATE TRIGGER CheckPurchaseQty ON CustomerPurchases
FOR INSERT, UPDATE
AS
BEGIN
   IF ((SELECT Qty FROM Inserted) > 10)
         BEGIN
           ROLLBACK TRANSACTION;
           RAISEERROR ('Too Greedy', 16,1);
         END
 END
```

Transactions can be used to prevent records being inserted or updated (like a CHECK constraint).

In this example, if a person tries to buy too many of one item, the purchase is rejected

Triggers – MS SQL

AFTER Triggers

- These triggers run **AFTER** their respective event
- As a result, they are not supported for VIEWS
  - Used to perform other actions AFTER the base table record has been created
    - History log, update calculated columns, sync DBs. . .


```
CREATE TRIGGER CharacterUpdate ON Characters
AFTER UPDATE
AS

IF(UPDATE(characterID))
    BEGIN
        RAISERROR('My identity column, my property!', 16, 1)
    END

END
```

AFTER Triggers – MS SQL

AFTER Trigger summary

- In SQL Server, BEFORE and AFTER triggers are really just **AFTER** triggers
- These can query both the **inserted** and **deleted** system tables
- They can be used to terminate a database modification, even AFTER it has taken place
- AFTER triggers are never executed if a constraint violation occurs
  - These triggers should not be used for any processing that might prevent constraint violations

Triggers – MS SQL

INSTEAD OF Triggers

- Useful for extending the types of updates a view can support
- Can provide the logic to modify **multiple base tables through a view**
- Can be used to modify columns containing:
  - Computed values
  - IDENTITY values (overwrite)
- These execute INSTEAD OF the query that initiated it
  - If the query was an insert, then the trigger must manually perform the insert into the base table


```
CREATE TRIGGER fixView ON FamilyGuyActors
INSTEAD OF INSERT
AS
BEGIN

INSERT INTO People
       SELECT actorName
       FROM inserted

              INSERT INTO Actors
              SELECT SCOPE_IDENTITY()
END;
```

BEFORE Triggers – MS SQL

INSTEAD OF Trigger Summary

- They override the standard actions of the triggering statement.
- They can be used to perform error or value checking on one or more columns
- They can perform additional actions before inserting, updating or deleting rows

- 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

Triggers

EXAMPLE FOR TRIGGERS

Triggers – Update Visit Count

```
CREATE TRIGGER customerHistoryVisit ON CustomerVisits
FOR INSERT
AS
BEGIN
UPDATE Customers SET TotalVisits = ISNULL(TotalVisits, 0) + 1
FROM Customers AS C JOIN Inserted AS I
ON C.CustomerID = I.CustomerID
END
```

Triggers – Basic Notes

Triggers are great for creating audit logs

- Every aspect of the data and changes can be captured (within reason)
  - Changes can be captured so that problem data can be traced and rolled back
- Great for performing calculations for derived values
  - Totals
  - More complex calculations
  - These can speed up queries involving totals
- What can't triggers do?
  - Record in an audit log the details of the person who deleted a record
    - Can only record the details of the person who edited a record if this detail is also saved in the normal record (eg, lastEditBy column)