# Triggers

A trigger is a pl/sql block that is triggered to fire automatically when an associated DML statement is executed. It can also be triggered as a result of a DDL but we will only focus on DML triggers.

Reasons to use triggers:

* Enforce business rules that cannot be defined using integrity constraints
  + Both triggers and constraints can constrain data but they differ because constraints apply to both existing and new data. A trigger only applied to new data. For example, a NOT NULL constraint can only be applied to a column of existing data if that data already meets that constraint. Any new data will also have to meet that constraint.
  + But, if you make a trigger to check that data entered is NOT NULL, the trigger will only fire for new data being entered. It will not check existing data (ie data that was entered prior to the trigger being defined or entered while the trigger was disabled.)
* Maintain security rules
* Automatically generate values for derived columns
* Prevent invalid transactions
* Provide value auditing.
  + Important: Many databases use some form of auditing/replicating information for keeping a log of changes.
  + For example, we may have a duplicate table that is just for auditing. If we have for example a Title table, we may also have a Title\_Audit table. We could implement a trigger so that every time data is inserted, updated, or deleted in Title, the corresponding record would be inserted into Title\_Audit (with potentially additional timestamp and user information).

There are two types of triggers:  
**1) Row level trigger**– An event is triggered for each row updated, inserted, or deleted. So, the trigger will be implemented multiple times (once for every row)  
**2) Statement level trigger**– An event is triggered for each SQL statement executed. So the trigger will only be called one time.

**Syntax**

*CREATE [OR REPLACE ] TRIGGER trigger\_name*

*{BEFORE | AFTER | INSTEAD OF }*

*{INSERT [OR] | UPDATE [OR] | DELETE}*

*[OF col\_name]*

*ON table\_name*

*[REFERENCING OLD AS o NEW AS n]*

*[FOR EACH ROW]*

*WHEN (condition)*

*DECLARE*

*BEGIN*

*--- sql statements*

*END;*

* *CREATE [OR REPLACE ] TRIGGER trigger\_name*- create a trigger with the given name or overwrite an existing trigger with the same name.
* *{BEFORE | AFTER | INSTEAD OF }*- This clause indicates when the trigger should fire. i.e before updating a table. INSTEAD OF is used to create a trigger on a view. NOTE: before and after cannot be used to create a trigger on a view.
* *{INSERT [OR] | UPDATE [OR] | DELETE}* - This clause determines the triggering event. More than one triggering events can be used together separated by the OR keyword. The trigger gets fired at all the specified triggering event(s).
* *[OF col\_name]*- This clause is used with update triggers and it triggers an event only when a specific column is updated.
* *[ON table\_name]*- The name of the table or view to which the trigger is associated.
* *[REFERENCING OLD AS o NEW AS n]*- This clause is used to reference the old and new values of the data being changed. By default, you reference the values as :old.column\_name or :new.column\_name. The reference names can also be changed from old (or new) to any other user-defined name. NOTE: You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.
* *[FOR EACH ROW]*- This clause is used to determine whether a trigger is row level or statement level.
* *WHEN (condition)*- This clause is valid only for row level triggers and is fired only for rows that satisfy the condition.

## Execution Hierarchy

The following hierarchy is followed when a trigger is fired.  
**1)** BEFORE statement-level trigger fires first  
**2)**  BEFORE row-level trigger fires, once for each row affected  
**3)**  AFTER row-level trigger fires once for each row affected. NOTE: These events will alternate between BEFORE and AFTER row-level triggers.  
**4)** AFTER statement-level trigger fires

## Determining Information about existing Trigger(s).

Data Dictionary - This is a read-only set of tables that provides info about the database. <http://docs.oracle.com/cd/B10501_01/server.920/a96524/c05dicti.htm>

We can use data dictionary Views to obtain information about triggers:

1. DBA\_TRIGGERS describes all the triggers in the database.
2. ALL\_TRIGGERS describes the triggers on tables accessible to the current user
3. USER\_TRIGGERS describes the triggers owned by the current user.

The below statement shows the structure of the view 'USER\_TRIGGERS'

*DESC USER\_TRIGGERS;*

This view stores information about header and body of the trigger.

*SELECT \* FROM user\_triggers WHERE trigger\_name = 'Before\_Update\_Stat\_product';*

The above sql query provides the header and body of the trigger 'Before\_Update\_Stat\_product'.

You can drop a trigger using the following command.

*DROP TRIGGER trigger\_name;*

## CYCLIC CASCADING TRIGGER

This is an undesirable situation where more than one trigger enters into an infinite loop. While creating a trigger, we should ensure that such a situation does not exist.

The below example shows how Trigger's can enter into cyclic cascading.  
Let's consider we have two tables 'abc' and 'xyz'. Two triggers are created.  
**1)** The INSERT Trigger, triggerA on table 'abc' issues an UPDATE on table 'xyz'.  
**2)**The UPDATE Trigger, triggerB on table 'xyz' issues an INSERT on table 'abc'.

A row inserted in table 'abc', triggerA fires and will update table 'xyz'.   
When the table 'xyz' is updated, triggerB fires and will insert a row in table 'abc'.