**Compound Triggers**

Compound triggers were introduced for the first time is in Oracle 11g. A compound trigger is a single trigger on a table that enables you to specify actions for each of four timing points:

* Before the firing statement
* Before each row that the firing statement affects
* After each row that the firing statement affects
* After the firing statement

Its function just likes the FOLLOWS clause which can also control the firing order or the firing timing if you have many triggers. But it does not have this part of function for the FOLLOWS clause which can arrange some triggers fire in same timing point. A compound trigger can arrange the firing order for many different timing-points because it can clean up the trigger statement once this state ends due to successful completion or an error.

By the way, in a compound trigger, both the statement-level and row-level action can be put up in a single trigger. Plus there is an added advantage: it allows sharing of common state between all the trigger-points using variable. This is because compound trigger in oracle 11g has a declarative section where one can declare variable to be used within trigger. This common state is established at the start of triggering statement and is destroyed after COMPLETION of trigger (regardless of trigger being in error or not). If same had to be done without compound-trigger, it might have been required to share data using packages.

**Syntax of a compound trigger:**

CREATE OR REPLACE TRIGGER <trigger-name>

FOR <trigger-action> ON <table-name>

COMPOUND TRIGGER

-- Global declaration.

g\_global\_variable VARCHAR2(10);

BEFORE STATEMENT IS

BEGIN

NULL; -- Do something here.

END BEFORE STATEMENT;

BEFORE EACH ROW IS

BEGIN

NULL; -- Do something here.

END BEFORE EACH ROW;

AFTER EACH ROW IS

BEGIN

NULL; -- Do something here.

END AFTER EACH ROW;

AFTER STATEMENT IS

BEGIN

NULL; -- Do something here.

END AFTER STATEMENT;

END <trigger-name>;

/

Notice: BEFORE STATEMENT, BEFORE EACH ROW, AFTER EACH ROW and AFTER STATEMENT are optional. You can choose all of them or some of them (at least one time timing-point section) depending on the need. However, a compound trigger cannot have multiple sections for the same timing point.

**When to use Compound Triggers**

The compound trigger is useful when you want to accumulate facts that characterize the “for each row” changes and then act on them as a body at “after statement” time. Two popular reasons to use compound trigger are:

1. To accumulate rows destined for a second table so that you can periodically bulk-insert them.
2. To avoid the infamous ORA-04091: mutating-table error.

**Some Restriction/Catches to note**

1. The body of a compound trigger must be a compound trigger block.
2. A compound trigger must be a DML trigger.
3. A compound trigger must be defined on either a table or a view.
4. The declarative part cannot include PRAGMA AUTONOMOUS\_TRANSACTION.
5. A compound trigger body cannot have an initialization block; therefore, it cannot have an exception section.

This is not a problem, because the BEFORE STATEMENT section always executes exactly once before any other timing-point section executes.

1. An exception that occurs in one section must be handled in that section. It cannot transfer control to another section.
2. If a section includes a GOTO statement, the target of the GOTO statement must be in the same section.
3. OLD, :NEW, and :PARENT cannot appear in the declarative part, the BEFORE STATEMENT section, or the AFTER STATEMENT section.
4. Only the BEFORE EACH ROW section can change the value of :NEW.
5. If, after the compound trigger fires, the triggering statement rolls back due to a DML exception:

* Local variables declared in the compound trigger sections are re-initialized, and any values COMPUTED thus far are lost.
* Side effects from firing the compound trigger are not rolled back.

1. The firing order of compound triggers is not guaranteed. Their firing can be interleaved with the firing of simple triggers.
2. If compound triggers are ordered using the FOLLOWS option, and if the target of FOLLOWS does not contain the corresponding section as source code, the ordering is ignored.