Triggers

event → procedure

a) DML on table or view
b) DDL on schema or DB
or system event (e.g. server start)

fires:
BEFORE
AFTER
INSTEAD

Usage

- tracking users, checking rights, logging
- declaring complex integrity constraints (SQL Standard defines normal integrity constraints on tables as triggers!)
- advanced filling of attributes with default values (e.g. auto-increment) – see the example below
- DML on views that aren't "automatically updatable"
- replication

DML trigger

SQL Server not described because its syntax is different

PMO L CREATE [OR REPLACE] TRIGGER [IF NOT EXISTS] trigger1 — Std: <trigger definition>
(O) (L)

(L: default)
before or after → execution
{BEFORE|AFTER|INSTEAD OF} {INSERT|DELETE|UPDATE [OF col1, col2, …]} [OR …]
this or this
(PO; optionally several combined)
(L: only for tables)
→ only for views; the developer has to modify underlying base table(s) of view1 by himself
→ not allowed: „OF col1, col2, …" & „WHEN(…)" (L: without these restrictions)
→ (PO: only at row-level)

ON {table1|view1}

(ML: no)
(O: not allowed)
(P: required; since ver. 1.0 (year 2017))
[REFERENCING {OLD|NEW} [TABLE] AS alias1 […]] → i.e. currently use it only for "transition tables" in Postgres, if needed
→ „transition table" visible as alias1 (else only the current row)
→ for AFTER trigger only
a before/after image of all rows deleted/inserted or updated – see Postgres doc & example 42.7.

for each row <0, …, ∞) being modified (row-level trigger)
only for these rows
[FOR EACH ROW] [WHEN (<search condition>)]
(ML: only row-level triggers are possible)
→ (PO: without subqueries)
→ (O: only for row-level triggers)

P: EXECUTE FUNCTION fun1([…])
M: BEGIN … END|CALL proc1([…])|any other <SQL procedure statement> ← „BEGIN … END" should be ATOMIC
O: BEGIN … END|CALL proc1[(…)] ← possible „DECLARE" before BEGIN
L: BEGIN … END ← only DML statements, SELECT, RAISE (IGNORE) and RAISE({ROLLBACK|ABORT|FAIL}, message) – see doc

MySQL doesn't allow: „ON view1" (& „INSTEAD OF") and „OF col1, col2, …" & "WHEN(…)"

example of body in Oracle:

```
IF (:NEW.employee_id is null) THEN
    SELECT seq_employee.NEXTVAL  INTO  :NEW.employee_id  FROM dual;
END IF;
```

PMOL • OLD/NEW – current row in table1 (in "WHEN(…)" and in trigger body in row-level triggers)
(O: :OLD/:NEW in trigger body, can be changed using "REFERENCING …")
(M: "WHEN(…)" isn't allowed)

OLD.col1 in INSERT and NEW.col1 in DELETE is: (PO: NULL, ML: invalid).

Only in Postgres we can read the whole record variable OLD/NEW (for example to check if it is NULL), without referring to some column (OLD.col1/NEW.col1); so don't do this as it isn't necessary and isn't portable.

P O • action type variables – is it INSERT, UPDATE or DELETE?

Oracle	PostgreSQL
IF (INSERTING)	IF (TG_OP = 'INSERT')
IF (UPDATING)	IF (TG_OP = 'UPDATE')
IF (DELETING)	IF (TG_OP = 'DELETE')
↖ boolean variables	↖ string variables

PMOL • Can a trigger body contain DML statements referring to the table on which we run the trigger?
PL: yes
MO: no (O: it can contain them with PRAGMA AUTONOMOUS_TRANSACTION;
it also cannot contain SELECT statements)

PMOL • A trigger cannot begin or end transaction, i.e. run COMMIT/ROLLBACK, START TRANSACTION
(O: unless it has PRAGMA AUTONOMOUS_TRANSACTION) (M: ROLLBACK TO SAVEPOINT is permitted)

• If the trigger body raises an exception, then the DBMS makes ROLLBACK of all changes performed by:
MO: the DML statement which fired the trigger (including all triggers)
P: the entire transaction containing the DML statement which fired the trigger (i.e. this ROLLBACK is the same or wider than in MySQL and Oracle)
L: see doc
(this is consistent with exception handling in Oracle and Postgres functions)

PMOL • Order of firing several triggers defined on the same table:
1. All BEFORE triggers are fired before all AFTER triggers (it is specified by SQL Standard)
(PO: additionally this order is preserved:
statement-level BEFORE triggers → row-level BEFORE triggers → row -level AFTER triggers → statement-level AFTER triggers)
2. If the 1st point isn't enough, then: (P: alphabetical) (SQL Standard & MySQL: time of trigger creation) (OS: undefined) order;
(MO: order can be changed/set using: "{FOLLOWS|PRECEDES} trigger2")

DDL triggers & additional notes about DML triggers in Postgres – see "Triggers – other info.txt" file.

Deleting triggers

(O: no) (P: trigger names are unique only for given table)

PMO L DROP TRIGGER [IF EXISTS] trigger1 ON table1; Std: <drop trigger statement>
P DROP EVENT TRIGGER [IF EXISTS] trigger1; - for DDL trigger

Modifying triggers

P ALTER EVENT TRIGER trigger1 {DISABLE|ENABLE}; - for DDL trigger
O ALTER TRIGER trigger1 {DISABLE|ENABLE};
O ALTER TABLE table1 {DISABLE|ENABLE} ALL TRIGGERS;
P ALTER TABLE table1 {DISABLE|ENABLE} TRIGGER trigger1;