Triggers

event

1. DML on table or view
2. DDL on schema or DB

or system event (e.g. server start)

fires:

BEFORE

AFTER

INSTEAD OF

procedure

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

(P: required; since ver. 1.0 (year 2017))

(L)

(O)

(L: only for tables)

PMO L

CREATE [OR REPLACE] TRIGGER [IF NOT EXISTS] trigger1 Std: <trigger definition>

this or this

optionally several combined

(PO; )

(L: default)

{BEFORE|AFTER|INSTEAD OF} {INSERT|DELETE|UPDATE [OF col1, col2, …]} [OR …]

before or

after

execution

not allowed: „OF col1, col2, …” & „WHEN(…)” (L: without these restrictions)

(PO: only at row-level)

only for views; the developer has to modify underlying base table(s) of view1 by himself

SQL Server not described because its syntax is different

P:

M:

O:

L:

„BEGIN … END”

should be ATOMIC

only DML statements, SELECT, RAISE (IGNORE) and

RAISE({ROLLBACK|ABORT|FAIL}, message) – see doc

possible „DECLARE” before BEGIN

EXECUTE FUNCTION fun1([…])

BEGIN … END|CALL proc1([…])|any other <SQL procedure statement>

BEGIN … END|CALL proc1[(…)]

BEGIN … END

for each row <0, …, ∞)

being modified

(row-level trigger)

[FOR EACH ROW] [WHEN (<search condition>)]

only for these rows

(ML: only row-level

triggers are possible)

(O: only for row-level triggers)

a before/after image of all rows deleted/inserted

or updated – see Postgres doc & example 42.7.

(O: not allowed)

(ML: no)

[REFERENCING {OLD|NEW} [TABLE] AS alias1 […]]

ON {table1|view1}

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

(PO: without subqueries)

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.

* action type variables – is it INSERT, UPDATE or DELETE?

P O

|  |  |
| --- | --- |
| Oracle | PostgreSQL |
| IF (INSERTING) | IF (TG\_OP = ’INSERT’) | |
| IF (UPDATING) | IF (TG\_OP = ’UPDATE’) | |
| IF (DELETING) | IF (TG\_OP = ’DELETE’) | |

PMOL

string variables

boolean variables

* 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

statement-level

AFTER triggers

row -level

AFTER triggers

row-level

BEFORE triggers

1. 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;