Regole Attive in PostgreSQL

TRIGGER IN POSTGRESQL

CREATE TRIGGER *Nome*
{ **BEFORE | AFTER**} {*Evento* [**OR** *Evento*]*}
ON *Relazione*
[**FOR EACH** {**ROW | STATEMENT** }]
[**WHEN (***Condizione***)**]
EXECUTE PROCEDURE *NomeFunzione* (*Argomenti*)

il default è **FOR EACH STATEMENT**

- **CREATE CONSTRAINT TRIGGER**
  {DEFERRABLE {INITIALLY IMMEDIATE |INITIALLY DEFERRED} | NOT DEFERRABLE}
  come visto per i vincoli (momento di attivazione si modifica con **SET CONSTRAINTS**)

- **INSTEAD OF** *Evento* solo per trigger definiti su viste

Trigger PostgreSQL - Eventi

- Comandi di **INSERT, DELETE, UPDATE** su relazione (di base, no viste)
- è possibile specificare **UPDATE OF** *Lista attributi*
- è possibile specificare più di un evento (in **OR**)
- trigger attivato **BEFORE** o **AFTER**
- **INSTEAD OF** solo per trigger su viste

Altri comandi:

- **DROP TRIGGER**
- **ALTER TRIGGER** (permette solo di modificare il nome)
- Clausole **DISABLE TRIGGER** [*NomeTrigger* | **ALL** | **USER**] e **ENABLE TRIGGER** [*NomeTrigger* | **ALL** | **USER**] del comando di **ALTER TABLE**

Trigger PostgreSQL – Azione

- L'azione deve essere l'invocazione di una *funzione* definita dall'utente
  - deve essere definita senza parametri
  - deve restituire tipo **trigger**
- A tale funzione si può passare una lista di *argomenti* (stringhe, separate da virgola)
- La stessa trigger function può essere utilizzata come azione di più trigger, l'uso dei parametri permette di specializzarne l'utilizzo
- Vedremo solo trigger function specificate in PL/pgSQL

Trigger function in PL/pgSQL

- PL/pgSQL può essere utilizzato per definire trigger function
- Una trigger function è una funzione senza parametri e con tipo di ritorno **trigger** creata con il comando **CREATE FUNCTION**
- La funzione deve essere dichiarata senza parametri anche se ci si aspetta che riceva gli argomenti specificati nel **CREATE TRIGGER**
- Gli argomenti del trigger sono passati alla funzione attraverso l'array **TG_ARGV**

Trigger function in PL/pgSQL

- Quando una funzione PL/pgSQL è dichiarata come trigger function vengono automaticamente create diverse variabili speciali nel top-level block
- Queste variabili sono:
  - **NEW:** contiene la nuova versione della tupla per le operazioni INSERT/UPDATE nei trigger row-level, NULL nei trigger statement-level
  - **OLD:** contiene la vecchia versione della tupla per le operazioni DELETE/UPDATE nei trigger row-level, NULL nei trigger statement-level

Trigger function in PL/pgSQL

- **TG_NAME:** contiene il nome del trigger correntemente attivato
- **TG_WHEN:** contiene la stringa AFTER o BEFORE a seconda della definizione del trigger
- **TG_LEVEL:** contiene la stringa ROW o STATEMENT a seconda della definizione del trigger
- **TG_OP:** contiene la stringa INSERT, UPDATE o DELETE a seconda dell'operazione che ha attivato il trigger
- **TG_TABLE_NAME:** contiene il nome della relazione su cui è definito il trigger
- **TG_NARGS:** contiene il numero degli argomenti passati alla trigger function nel comando CREATE TRIGGER
- **TG_ARGV[ ]:** array di stringhe, che contiene gli argomenti passati alla trigger function nel comando CREATE TRIGGER
  - l'indice parte da 0
  - accessi all'array con indici invalidi risultano in valore NULL

Trigger function in PL/pgSQL

- Una trigger function deve restituire NULL oppure una tupla con la stessa struttura della tabella su cui è stato attivato il trigger
- Il valore di ritorno è utilizzato solo dai trigger *before row*
  - valore di ritorno NULL segnala al trigger manager di non eseguire il resto dell'operazione per tale tupla
    - i trigger successivi non sono eseguiti
    - l'operazione corrispondente all'evento non viene eseguita
  - valore di ritorno diverso dal valore NEW originale modifica la tupla che verrà inserita o aggiornata
    - per modificare tale riga è possibile modificare i campi di NEW e restituire la NEW modificata o costruire una nuova tupla e restituirla
- Per tutti gli altri tipi di trigger il valore di ritorno è ignorato (e può essere o meno NULL)

PostgreSQL - Modalità di Esecuzione

- Scelta regola:
  - dipende dal tipo di trigger come in SQL200N
    - trigger BEFORE STATEMENT
    - per ogni tupla oggetto del comando
      - trigger BEFORE ROW
      - comando e verifica dei vincoli di integrità
      - trigger AFTER ROW
    - verifica dei vincoli che richiedono di aver completato il comando
    - trigger AFTER STATEMENT
  - se esistono più trigger dello stesso tipo: ordine alfabetico
- Possibilità di esecuzione differita per CONSTRAINT TRIGGER

PostgreSQL – Tabella riassuntiva

Eventi primitivi	Operazioni sulla base di dati
Eventi compositi	OR
Transition tuple/table	tuple
Constraint trigger	deferrable, solo for each row
Terminazione	Nessuna condizione
Ordinamento regole	Tipo + ordine alfabetico

PostgreSQL vs SQL200N

	PostgreSQL	SQL200N
Evento	**OR** di insert, delete, update	Singolo insert, delete, update, **update of**
Tuple e tabelle transizione	Solo per ROW trigger, solo tupla di transizione No clausola REFERENCING	ROW e STATEMENT, tupla e tabella
Condizione	Senza sottoquery ROW e STATEMENT (ma senza tabelle transiz.)	ROW e STATEMENT
Azione	Invocazione di trigger function	Sequenza di comandi
Ordinamento	Alfabetico su nome	Tempo creazione

Trigger PostgreSQL – Esempio 1

- Trigger che implementa vincolo 'non più di tre video contemporaneamente'

```
CREATE OR REPLACE FUNCTION non_tre() RETURNS trigger AS
   $non_tre$
   BEGIN
      IF (SELECT COUNT(*) FROM Noleggio
         WHERE dataRest IS NULL AND codCli = NEW.codCli) > 3
      THEN
          RAISE EXCEPTION '%ne ha gia tre', NEW.codCli;
          -- nb nn viene effettuato return new, la tupla nn è inserita
      ELSE
         RETURN NEW;
      END IF;
   END;
$non_tre$ LANGUAGE plpgsql;
```

Trigger PostgreSQL – Esempio 1

- Trigger che implementa vincolo ‘non più di tre video contemporaneamente’

```
CREATE TRIGGER non_piu_di_tre
BEFORE INSERT OR UPDATE ON Noleggio
FOR EACH ROW
EXECUTE PROCEDURE non_tre();
```

Trigger PostgreSQL – Esempio 2

- Trigger che implementa dato derivato punti mancanti

```
CREATE OR REPLACE FUNCTION agg_pti() RETURNS trigger AS
   $agg_pti$
   BEGIN
      IF (NEW.codCli IN (SELECT codCli FROM Standard))
      THEN
         UPDATE Standard
         SET ptiMancanti = ptiMancanti - 1
         WHERE codCli = NEW.codCli AND NEW.colloc IN
                                          (SELECT colloc
                                           FROM VIDEO
                                           WHERE tipo = 'v');
```

Trigger PostgreSQL – Esempio 2

- Trigger che implementa dato derivato punti mancanti

```
        UPDATE Standard
        SET ptiMancanti = ptiMancanti - 2
        WHERE codCli = NEW.codCli AND NEW.colloc IN
                                    (SELECT colloc
                                     FROM VIDEO
                                     WHERE tipo = 'd');
    END IF;
    RETURN NEW;
  END;
$agg_pti$ LANGUAGE plpgsql;
```

Trigger PostgreSQL – Esempio 2

- Trigger che implementa dato derivato punti mancanti

```
CREATE TRIGGER agg_pti
AFTER INSERT ON Noleggio
FOR EACH ROW
EXECUTE PROCEDURE agg_pti();
```

Trigger PostgreSQL – Esempio 3

- Trigger che implementa regola operativa migrazione clienti

```
CREATE OR REPLACE FUNCTION diventa_vip() RETURNS trigger
   AS $diventa_vip$
   BEGIN
      IF (NEW.ptiMancanti <= 0)
      THEN
         INSERT INTO Vip
         VALUES (NEW.codCli,5.00);
         DELETE FROM Standard
         WHERE codCli = NEW.codCli;
      END IF;
      RETURN NEW;
   END;
$diventa_vip$ LANGUAGE plpgsql;
```

Trigger PostgreSQL – Esempio 3

- Trigger che implementa regola operativa migrazione clienti

```
CREATE TRIGGER diventa_vip
AFTER UPDATE ON Standard
FOR EACH ROW
EXECUTE PROCEDURE diventa_vip();
```

Regole PostgreSQL

- PostgreSQL prevede anche una nozione di regola che consente di dire al query executor come riscrivere una determinata operazione
- Quelle che coinvologono le operazioni di aggiornamento prendono il nome di *update rule*

CREATE [OR REPLACE] RULE *Nome* **AS**
ON E*vento* **TO** *Relazione*
[**WHERE** C*ondizione*]
DO [ALSO | INSTEAD]
{ **NOTHING** | *Comando* | (C*omando* [;*Comando*]*) }

PosgreSQL: regole vs trigger

- Le regole manipolano il comando o generano un nuovo comando da eseguire
- L'effetto dell'esecuzione di una regola è specificato in termini del *query tree*
- Molte cose che si possono fare con i trigger si possono fare anche con le regole
  - l'approccio dei trigger è più aderente allo standard e concettualmente più semplice dell'approccio delle regole
  - i trigger sono in generale più adatti per gestire i vincoli di integrità
  - le regole permettono di gestire in maniera flessibile le modifiche attraverso le viste

Regole PosgreSQL

- Esempio: aggiornamenti su vista Nol3gg

```
CREATE VIEW Nol3gg AS
SELECT colloc, dataNol, codCli
FROM Noleggio
WHERE dataRest IS NULL AND
(CURRENT_DATE - dataNol) > INTERVAL '3' DAY;
```

Regole PosgreSQL

- Esempio: aggiornamenti su vista Nol3gg

```
CREATE RULE nol3gg_ins AS ON INSERT
  TO Nol3gg
   DO INSTEAD
   INSERT INTO Noleggio VALUES (
         NEW.colloc,
         NEW.dataNol,
         NEW.codCli);
```

Regole PosgreSQL

- Esempio: aggiornamenti su vista Nol3gg

```
CREATE RULE nol3gg_del AS
  ON DELETE TO Nol3gg
  DO INSTEAD
  DELETE FROM Noleggio
  WHERE colloc = OLD.Colloc
                AND dataNol = OLD.dataNol;
```

Regole PosgreSQL

- Esempio: aggiornamenti su vista Nol3gg

```
CREATE RULE nol3gg_upd AS ON UPDATE TO Nol3gg
 DO INSTEAD
 UPDATE Noleggio
 SET codCli = NEW.codCli,
        colloc = NEW.colloc,
        dataNol = NEW.dataNol
 WHERE colloc = OLD.Colloc AND dataNol = OLD.dataNol;
```