



TRIGGER: ESEMPI DI UTILIZZO

1

TRIGGER E VINCOLI

- I trigger sono più flessibili dei vincoli di integrità
 - permettono di stabilire come reagire ad una violazione di un vincolo arbitrario
- I trigger possono specificare anche vincoli di transizione
- La flessibilità non sempre è un vantaggio
- A volte definire dei vincoli è più vantaggioso:
 - Migliore ottimizzazione
 - Meno errori di programmazione
 - I vincoli sono parte dello standard da lungo tempo, i trigger no

VINCOLI DI INTEGRITÀ

Ogni cliente non può noleggiare più di tre video contemporaneamente

```
CREATE ASSERTION
VerificaNoleggi
CHECK (NOT EXISTS
  (SELECT * FROM Noleggio
   WHERE dataRest IS NULL
   GROUP BY codCli
   HAVING COUNT(*) > 3));
```

BEFORE sarebbe meglio?

```
CREATE TRIGGER VerificaNoleggi
AFTER INSERT ON Noleggio
REFERENCING NEW ROW AS NR
FOR EACH ROW
WHEN
  (SELECT COUNT(*)
   FROM Noleggio
   WHERE dataRest IS NULL AND
     codCli = NR.codCli) > 3
ROLLBACK;
```

VINCOLI DI INTEGRITÀ

Ogni cliente non può noleggiare più di tre video contemporaneamente

Invece di abortire la transazione se il vincolo è violato si vuole annullare l'inserimento

CREATE TRIGGER

VerificaNoleggi

AFTER INSERT ON Noleggio
REFERENCING NEW ROW AS NR
FOR EACH ROW

WHEN (SELECT COUNT(*)
FROM Noleggio
WHERE dataRest IS NULL AND
codCli = NR.codCli) > 3

DELETE FROM Noleggio
WHERE colloc = NR.colloc AND
dataNol = NR.dataNol;

CREATE ASSERTION

VerificaNoleggi

CHECK (NOT EXISTS
(SELECT * FROM Noleggio
WHERE dataRest IS NULL
GROUP BY codCli
HAVING COUNT(*) > 3));

Cancelliamo solo il
noleggio appena
inserito

VINCOLI DI INTEGRITÀ

```
CREATE TRIGGER VerificaNoleggi  
AFTER INSERT ON Noleggio  
REFERENCING NEW TABLE AS NT  
FOR EACH STATEMENT
```

```
WHEN EXISTS
```

```
(SELECT *  
FROM Noleggio  
WHERE Noleggio.dataRest IS NULL AND  
      Noleggio.codCli IN (SELECT codCli FROM NT)  
GROUP BY codCli  
HAVING COUNT(*) > 3)
```

```
DELETE FROM Noleggio
```

```
WHERE (colloc,dataNol) IN (SELECT colloc, dataNol FROM NT)  
AND codCli IN (SELECT codCli
```

```
FROM Noleggio
```

```
WHERE Noleggio.dataRest IS NULL AND
```

```
      Noleggio.codCli IN (SELECT codCli FROM NT)
```

```
GROUP BY codCli
```

```
HAVING COUNT(*) > 3);
```

Ogni cliente non può noleggiare più di tre video contemporaneamente

Se un noleggio causa la violazione del vincolo deve essere impedito

Tutto il noleggio non solo i video eccedenti

CALCOLO DI DATI DERIVATI

Aggiornamento automatico attributo `ptiMancanti` nella tabella `Standard` ad ogni nuovo noleggio:

```
CREATE TRIGGER CalcolaPtiMancanti
AFTER INSERT ON Noleggio
REFERENCING NEW ROW AS NR
FOR EACH ROW
BEGIN ATOMIC
    UPDATE Standard
    SET ptiMancanti = ptiMancanti - 1
    WHERE codCli = NR.codCli AND
           NR.colloc IN (SELECT colloc
                        FROM Video
                        WHERE tipo = 'v');

    UPDATE Standard
    SET ptiMancanti = ptiMancanti - 2
    WHERE codCli = NR.codCli AND
           NR.colloc IN (SELECT colloc FROM Video
                        WHERE tipo = 'd');
END;
```

ogni VHS 1 punto
ogni DVD 2 punti

REGOLE OPERATIVE

Quando il valore dell'attributo `ptiMancanti` per un cliente standard diventa 0, il cliente è rimosso dalla tabella Standard ed inserito nella tabella VIP con bonus pari a 5 euro

```
CREATE TRIGGER OrganizzaClienti
AFTER UPDATE OF ptiMancanti ON Standard
REFERENCING NEW ROW AS NR
FOR EACH ROW
WHEN NR.ptiMancanti <= 0
BEGIN ATOMIC
    INSERT INTO VIP
    VALUES (NR.codCli,5.00);
    DELETE FROM Standard
    WHERE codCli = NR.codCli;
END;
```



REGOLE ATTIVE IN POSTGRESQL

8

TRIGGER IN POSTGRESQL

```
CREATE TRIGGER Nome
{ BEFORE | AFTER | INSTEAD OF } { Evento [ OR Evento ] * }
ON Relazione
* [ REFERENCING { { OLD | NEW } TABLE [ AS ] <variabile> } ]
[ FOR EACH { ROW | STATEMENT } ]
** [ WHEN ( Condizione ) ]
EXECUTE FUNCTION NomeFunzione ( Argomenti )
```

- il default è **FOR EACH STATEMENT**
- trigger attivato **BEFORE** o **AFTER**
 - comandi di **INSERT, DELETE, UPDATE** su relazione (di base, no viste)
 - è possibile specificare **UPDATE OF** *Lista attributi*
- trigger attivato **INSTEAD OF** solo per trigger su viste
 - Solo **FOR EACH ROW**
- è possibile specificare più di un evento (in **OR**)
- Clausola referencing **solo per transition table**
- * **dalla versione 10**, non usabile per UPDATE su specifici attributi
- ** **dalla versione 9**

TRIGGER IN POSTGRESQL

```
CREATE CONSTRAINT TRIGGER  
{DEFERRABLE {INITIALLY IMMEDIATE | INITIALLY  
DEFERRED} | NOT DEFERRABLE}
```

- Il momento di attivazione del trigger può essere **immediato** (alla fine del comando che ha generato l'evento) o **differito**, alla fine della transazione che contiene il comando
- la modalità di attivazione si modifica con **SET CONSTRAINTS**, come già visto per i vincoli
- **Solo FOR EACH ROW**
- **No tabelle di transizione**

TRIGGER POSTGRESQL - EVENTI

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
- Funzioni specificabili in diversi linguaggi, 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**

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
 - Non esistono variabili di transizione a livello di tabella

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
 - l'operazione corrispondente all'evento non viene eseguita
 - i trigger successivi non sono eseguiti
 - 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
- Nessun tipo di limitazione nè controllo per terminazione
- L'esecuzione di un trigger “estende” sempre la transazione che ha generato l'evento che ha attivato il trigger

POSTGRESQL VS SQL200N

	PostgreSQL	SQL200N
Evento	OR di insert, delete, update	Singolo insert, delete, update, update of
Tuple e tabelle transizione	ROW e STATEMENT, tupla di transizione disponibile anche senza clausola REFERENCING Tabella di transizione disponibile solo con clausola REFERENCING (non disponibile in constraint trigger e per UPDATE di specifici attributi)	ROW e STATEMENT, tupla e tabella
Condizione	Senza sottoquery ROW e STATEMENT (ma senza tuple/tabelle transiz.)	ROW e STATEMENT
Azione	Invocazione di trigger function	Sequenza di comandi
Ordinamento	Tipo + ordine alfabetico su nome	Tempo creazione
Terminazione	Nessuna condizione	Nessuna indicazione
Constraint trigger	Deferrable, solo for each row	

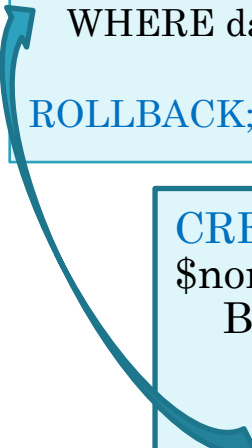


ESEMPI

TRIGGER POSTGRESQL – ESEMPIO 1

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

```
CREATE TRIGGER VerificaNoleggi
AFTER INSERT ON Noleggio
REFERENCING NEW ROW AS NR
FOR EACH ROW
WHEN
  (SELECT COUNT(*)
   FROM Noleggio
   WHERE dataRest IS NULL AND
        codCli = NR.codCli) > 3
ROLLBACK;
```



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

```
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;
    -- non viene effettuato return new, la tupla non è inserita
  ELSE
    RETURN NEW;
  END IF;
END;
$non_tre$ LANGUAGE plpgsql;
```

TRIGGER POSTGRESQL – ESEMPIO 2

- Trigger che implementa dato derivato punti mancanti

```
CREATE TRIGGER CalcolaPtiMancanti
AFTER INSERT ON Noleggio
REFERENCING NEW ROW AS NR
FOR EACH ROW
BEGIN ATOMIC
    UPDATE Standard
    SET ptiMancanti = ptiMancanti - 1
    WHERE codCli = NR.codCli AND
           NR.colloc IN (SELECT colloc
                        FROM Video
                        WHERE tipo = 'v');

    UPDATE Standard
    SET ptiMancanti = ptiMancanti - 2
    WHERE codCli = NR.codCli AND
           NR.colloc IN (SELECT colloc FROM Video
                        WHERE tipo = 'd');
END;
```

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

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');


        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 3

- Trigger che implementa regola operativa migrazione clienti

```
CREATE TRIGGER OrganizzaClienti
AFTER UPDATE OF ptiMancanti ON Standard
REFERENCING NEW ROW AS NR
FOR EACH ROW
WHEN NR.ptiMancanti <= 0
BEGIN ATOMIC
    INSERT INTO VIP
    VALUES (NR.codCli,5.00);
    DELETE FROM Standard
    WHERE codCli = NR.codCli;
END;
```



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

```
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;
```