




UNIVERSITÀ DEGLI STUDI DI SALERNO

Atzeni, Ceri, Fraternali, Paraboschi,  
Torlone  
**Basi di dati Quinta edizione**  
McGraw-Hill Education, 2018





**BASI DI DATI**

**SQL-DML Avanzato**

**Procedure e Trigger - quarta parte**




DIEM  
DIPARTIMENTO DI  
INGEGNERIA DELL'INFORMAZIONE ED ELETTRICA E MATEMATICA APPLICATA


**Matteo Gaeta**  
Full Professor – Senior Member IEEE



P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi,

©2018 McGraw-Hill Education (Italy) S.r.l.





Le slide utilizzate dai docenti per le attività frontali sono in gran parte riconducibile e riprese dalle slide originali (con alcuni spunti parziali ripresi dai libri indicati) realizzate da:

- autori del libro (Atzeni e altri) di riferimento e sono reperibili su internet su molteplici link oltre che laddove indicato dagli stessi autori del libro;

**Capitolo 5:**

**SQL: caratteristiche evolute**

**PROCEDURE E TRIGGER**

P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e

©2018 McGraw-Hill Education (Italy) S.r.l.

## Stored Procedures



- Una procedura consente di associare un nome a un'istruzione SQL, con la possibilità di specificare dei parametri da utilizzare per lo scambio di informazioni con la procedura.
- Aumenta la comprensibilità del programma
- Una volta che la procedura è definita, può essere usata come se facesse parte dell'insieme di comandi SQL.
- SQL-2 definisce solo procedure semplici (un solo comando SQL).
- Molti sistemi commerciali rimuovono questo limite, in alcuni casi fornendo tutti i costrutti di un linguaggio procedurale (computazionalmente completo)
  - Non Standard

## Stored Procedures



```
procedure AssegnaCitta(:Dip  
varchar(20), :Citta varchar(20))  
  update Dipartimento  
  set Città = :Citta  
  where Nome = :Dip;
```

## Stored Procedures

PL/pgSQL



```
CREATE FUNCTION somefunc(integer, text) RETURNS
integer
AS 'function body text'
LANGUAGE plpgsql;
```

**PL/pgSQL è un linguaggio a blocchi strutturato.**  
**Un blocco è :**

```
[ <<label>> ]
[ DECLARE declarations ]
BEGIN
    statements
END [ label ];
```

## Il Linguaggio PL/pgSQL

- Il linguaggio PL/pgSQL è un linguaggio strutturato a blocchi
- Ciascun blocco ha la forma:

```
DECLARE
    < dichiarazioni di variabili >
BEGIN
    < istruzioni >
END
```

- I blocchi possono essere annidati
- Dichiarazioni e istruzioni devono terminare con un punto e virgola

## Linguaggio PL/pgSQL: Esempio



```
CREATE FUNCTION circ(r INTEGER) RETURNS
NUMERIC AS $BODY$
```

```
DECLARE
```

```
    costante pi CONSTANT NUMERIC := pi();
    risultato NUMERIC;
```

```
BEGIN
```

```
    risultato := 2 * costante pi * r;
    RETURN risultato;
```

```
END;
```

```
$BODY$
```

```
LANGUAGE PLPGSQL;
```

## Linguaggio PL/pgSQL: Esempio



Il risultato su una linea di una query **SELECT** può essere memorizzato in una variabile (di tipo opportuno) mediante il costrutto **INTO**:

```
SELECT <exp> INTO <elemento> FROM...
```

Ex.

```
DECLARE
```

```
risultato corso.crediti integer;
```

```
BEGIN
```

```
    SELECT SUM (crediti) INTO risultato
    FROM corso JOIN frequenza USING id_corso
    WHERE id_studente=studente;
```

```
END;
```

## Il Linguaggio PL/pgSQL: Esempio



DECLARE

minimo INTEGER;

BEGIN

minimo := (SELECT MIN(voto) FROM frequenza  
Where Corso='Basi di Dati');

IF (minimo>27) THEN

RAISE EXCEPTION 'Corso troppo facile';

ENDIF;

END

**RAISE EXCEPTION: Solleva un'eccezione con il messaggio d'errore <messaggio> ed arresta l'esecuzione della funzione.**

## Basi di dati attive



- Una base di dati che contiene regole attive (chiamate *trigger*)
- Presentazione:
  - Definizione dei trigger in SQL:1999
  - Definizione dei trigger in DB2 e Oracle
  - Problemi di progetto per applicazioni basate sull'uso dei trigger

## Basi di dati attive



- I DBMS tradizionale sono passivi: Eseguono delle operazioni solo su richiesta.
- Un **DBMS attivo** ha invece capacita' reattive: **Reagisce autonomamente ad alcuni eventi ed esegue determinate operazioni.**
- Concretamente, in un DBMS attivo e' possibile definire regole attive o trigger.
- Le istruzioni CREATE TRIGGER e CREATE FUNCTION sono presenti nello standard SQL99.
- **PostgreSQL e' un DBMS attivo.**

## Il concetto di trigger



- Un trigger e' una procedura eseguita autonomamente dal sistema, in conseguenza di un inserimento, una cancellazione, o un aggiornamento di una data tabella.
- Quando un trigger e' attivato, esso provoca l'esecuzione di una funzione, specificata al momento della definizione del trigger.
- Tre parti (ECA Rule):
  - 1 Evento: attiva il trigger
  - 2 Condizione: verifica se il trigger deve essere eseguito
  - 3 Azione: funzione da eseguire all'attivazione del trigger

## Il concetto di trigger



- Paradigma: Evento-Condizione-Azione
  - Quando un **evento** si verifica
  - Se la **condizione** è vera
  - Allora l'**azione** è eseguita
- Questo modello consente computazioni reattive
- Non è il solo tipo di regole:
  - Vincoli di integrità
  - Regole datalog
  - Regole di business
- Problema: è difficile realizzare applicazioni complesse

## Evento-Condizione-Azione



- **Evento**
  - Normalmente una modifica dello stato del database: insert, delete, update
  - Quando accade l'evento, il trigger è *attivato*
- **Condizione**
  - Un predicato che identifica se l'azione del trigger deve essere eseguita
  - Quando la condizione viene valutata, il trigger è *considerato*
- **Azione**
  - Una sequenza di update SQL o una procedura
  - Quando l'azione è eseguita anche il trigger è *eseguito*
- I DBMS forniscono tutti i componenti necessari. Basta integrarli.

## SQL:1999, Sintassi

- Lo standard SQL:1999 (SQL-3) sui trigger è stato fortemente influenzato da DB2 (IBM); gli altri sistemi non seguono lo standard (esistono dagli anni 80')
- Ogni trigger è caratterizzato da:
  - nome
  - target (tabella controllata)
  - modalità (**before** o **after**)
  - evento (**insert**, **delete** o **update**)
  - granularità (statement-level o row-level)
  - alias dei valori o tabelle di transizione
  - azione
  - timestamp di creazione

## Definizione di un Trigger

```
CREATE TRIGGER < nome trigger> { BEFORE | AFTER }
    { INSERT | UPDATE | DELETE } [ OR. . . ]
    ON < nome tabella> [ FOR EACH { ROW | STATEMENT }
]
EXECUTE PROCEDURE < nome funzione> ( < argomenti> )
```

**< nome trigger>** : nome del trigger



## Definizione di un Trigger



```
CREATE TRIGGER < nome trigger> { BEFORE | AFTER }
    { INSERT | UPDATE | DELETE } [ OR. . . ]
    ON < nome tabella> [ FOR EACH { ROW | STATEMENT }
]
```

EXECUTE PROCEDURE < nome funzione> ( < argomenti> )

**{ BEFORE | AFTER }** Un trigger puo' essere attivato **prima (BEFORE)** che l'operazione di modifica sia eseguita, **oppure**

**dopo (AFTER)** che l'operazione e' stata eseguita e gli eventuali vincoli di integrita' sono stati controllati.

**Nel primo caso il trigger puo' annullare l'operazione o, nel caso di inserimenti / aggiornamenti, apportare cambiamenti alla riga da inserire / aggiornare.**

## Definizione di un Trigger



```
CREATE TRIGGER < nome trigger> { BEFORE | AFTER }
    { INSERT | UPDATE | DELETE } [ OR. . . ]
    ON < nome tabella> [ FOR EACH { ROW | STATEMENT }
]
```

EXECUTE PROCEDURE < nome funzione> ( < argomenti> )

**[ OR. . . ]:** Permette di specificare piu' eventi per un solo trigger (e.g INSERT OR UPDATE).

E' un'estensione PostgreSQL allo standard SQL.

## Definizione di un Trigger



```
CREATE TRIGGER < nome trigger> { BEFORE | AFTER }
    { INSERT | UPDATE | DELETE } [ OR. . . ]
    ON < nome tabella> [ FOR EACH { ROW | STATEMENT } ]
EXECUTE PROCEDURE < nome funzione> ( < argomenti> )
```

**ON < nome tabella >** : Nome della tabella su cui e' definito il trigger.

## Definizione di un Trigger



```
CREATE TRIGGER < nome trigger> { BEFORE | AFTER }
    { INSERT | UPDATE | DELETE } [ OR. . . ]
    ON < nome tabella> [ FOR EACH { ROW | STATEMENT } ]
EXECUTE PROCEDURE < nome funzione> ( < argomenti> )
```

[ **FOR EACH { ROW | STATEMENT }** ] :

Se il trigger è definito **FOR EACH ROW** , allora viene attivato tante volte quante sono le righe coinvolte nell'operazione di modificazione.

Se il trigger è definito **FOR EACH STATEMENT** (default), allora viene attivato una sola volta per ciascuna operazione di modificazione, indipendentemente dalle tuple coinvolte.

## Definizione di un Trigger

```
CREATE TRIGGER < nome trigger> { BEFORE | AFTER }
    { INSERT | UPDATE | DELETE } [ OR. . . ]
    ON < nome tabella> [ FOR EACH { ROW | STATEMENT } ]
EXECUTE PROCEDURE < nome funzione> ( < argomenti> )
```


**< nome funzione>:** Nome della funzione da eseguire all'attivazione del trigger. La funzione deve essere definita prima di creare il trigger.

**< argomenti>:** Lista di argomenti


## Trigger: Funzioni che specificano l'azione

**La funzione che specifica l'azione** di un trigger può essere scritta:

- In un linguaggio procedurale nativo del DBMS, ovvero "compatibile" con il modello logico del DB e con SQL (e.g. PL/SQL per Oracle, PL/pgSQL per PostgreSQL, ecc.)
- In un linguaggio procedurale esterno (e.g. C)
- **Noi utilizzeremo nello specifico trigger le cui funzioni sono scritte in PL/pgSQL: Estensione procedurale di SQL per PostgreSQL**




## SQL:1999, Sintassi




```

create trigger NomeTrigger
{ before | after }
{ insert | delete | update [of Column] } on
TabellaTarget
[referencing
    {[old table [as] VarTuplaOld]
    [new table [as] VarTuplaNew] } |
    {[old [row] [as] VarTabellaOld]
    [new [row] [as] VarTabellaNew] }]
[for each { row | statement } ]
[when Condizione]
StatementProceduraleSQL
  
```

P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e
©2018 McGraw-Hill Education (Italy) S.r.l.



## Sintassi PL/pgSQL: Funzione richiamata dal Trigger



nel linguaggio PL/pgSQL da usare in un trigger e':

```

CREATE [ OR REPLACE ] FUNCTION < nome funzione >()
RETURNS TRIGGER AS $BODY$
    DECLARE
    <variabili>
    BEGIN
    <istruzioni>
    END;
$BODY$
LANGUAGE PLPGSQL;
  
```

La funzione deve restituire il tipo **TRIGGER**

Per cancellare una funzione: DROP FUNCTION < nome funzione >() [ { CASCADE | RESTRICT } ]

P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e
©2018 McGraw-Hill Education (Italy) S.r.l.

## PL/pgSQL da usare in un trigger

### Variabili New e Old



All'interno del blocco di istruzioni di una funzione PL/pgSQL per un trigger sono disponibili alcune variabili speciali tra cui:

- **NEW:** In operazioni INSERT o UPDATE, rappresenta la nuova riga della tabella che si vuole aggiornare.
- **OLD:** In operazioni DELETE o UPDATE, rappresenta la riga della tabella che si vuole cancellare o modificare
- Le variabili NEW e OLD sono di tipo RECORD
- I singoli attributi sono denotati NEW.<nome colonna> e OLD.<nome colonna>

## PL/pgSQL da usare in un trigger

### Un esempio per il DB Segreteria




**L'attributo crediti della tabella corso puo' essere NULL: uno studente non dovrebbe potersi iscrivere ad un corso con crediti non noti.**

Definiamo un trigger che generi tale vincolo.


Implementiamo il Trigger che ci consente di effettuare il controllo sulla riga con cui l'utente si vuole iscrivere ad un corso...ecco il Trigger

```
CREATE TRIGGER iscrizione_valida
BEFORE INSERT OR UPDATE
ON frequenza
FOR EACH ROW
EXECUTE PROCEDURE iscrizione_valida();
```



## PL/pgSQL da usare in un trigger

### Un esempio per il DB Segreteria



L'attributo crediti della tabella corso puo' essere NULL: uno studente non dovrebbe potersi iscrivere ad un corso con crediti non noti. Definiamo un trigger che generi tale vincolo

```


CREATE FUNCTION iscrizione_valida()
RETURNS TRIGGER AS
$BODY$
DECLARE
    c INTEGER;
BEGIN
    SELECT crediti INTO c FROM corso WHERE id_corso=NEW.id_corso
    IF c IS NULL THEN RAISE EXCEPTION $$'Non è possibile iscriversi
    al corso % perchè tale corso e' in via di denizione'$$, NEW.id_corso;
    ELSE RETURN NEW;
    END IF;
END;
$BODY$
LANGUAGE PLPGSQL;
  
```

**Il trigger, passa la nuova riga su cui effettuare il controllo nella variabile NEW.**

**Se la funzione non rileva eccezioni, restituisce la riga da inserire nella variabile NEW.**


P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e

©2018 McGraw-Hill Education (Italy) S.r.l.



## PL/pgSQL da usare in un trigger

### Variabili di Ritorno



- Una funzione ha un valore di ritorno specifico dall'istruzione RETURN
- Se il trigger è definito BEFORE e FOR EACH ROW, tale valore può essere NULL (i.e. l'operazione sulla riga corrente e' annullata) oppure una variabile di tipo RECORD
- Se il trigger è definito AFTER oppure FOR EACH STATEMENT, il valore di ritorno è ignorato

P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e

©2018 McGraw-Hill Education (Italy) S.r.l.

## Tipi di eventi



- BEFORE
  - Il trigger è considerato e possibilmente eseguito prima dell'evento (i.e., la modifica del database)
  - I trigger before non possono modificare lo stato del database; possono al più condizionare i valori "new" in modalità row-level (set t.new=expr)
  - Normalmente questa modalità è usata quando si vuole verificare una modifica prima che essa avvenga e "modificare la modifica"
- AFTER
  - Il trigger è considerato e eseguito dopo l'evento
  - After è la modalità più comune, adatta alla maggior parte delle applicazioni

## Esempio "before" e "after"



- 1. "Conditioner" (agisce prima dell'update e della verifica di integrità)
 

```
create trigger LimitaAumenti
before update of Salario on Impiegato
for each row
when (New.Salario > Old.Salario * 1.2)
set New.Salario = Old.Salario * 1.2
```
- 2. "Re-installer" (agisce dopo l'update)
 

```
create trigger LimitaAumenti
after update of Salario on Impiegato
for each row
when (New.Salario > Old.Salario * 1.2)
set New.Salario = Old.Salario * 1.2
```

## Granularità degli eventi



- **Modalità statement-level** (di default, opzione **for each statement**)
  - Il trigger viene considerato e possibilmente eseguito solo una volta per ogni statement (comando) che lo ha attivato, indipendentemente dal numero di tuple modificate
  - In linea con SQL (set-oriented)
- **Modalità row-level** (opzione **for each row**)
  - Il trigger viene considerato e possibilmente eseguito una volta per ogni tupla modificata
  - Scrivere trigger row-level è più semplice

## Clausola referencing



- Dipende dalla granularità
  - Se la modalità è row-level, ci sono due *variabili di transizione* (**old** and **new**) che rappresentano il valore precedente o successivo alla modifica di una tupla
  - Se la modalità è statement-level, ci sono due *tabelle di transizione* (**old table** and **new table**) che contengono i valori precedenti e successivi delle tuple modificate dallo statement
- **old** e **old table** non sono presenti con l'evento **insert**
- **new** e **new table** non sono presenti con l'evento **delete**



## Esempio di trigger row-level

```
create trigger AccountMonitor (Nome Trigger)
after update on Account (Nome Tabella)
for each row (Modalità Riga)
when new.Totale > old.Totale
insert values
    (new.NumeroConto,
     new.Totale-old.Totale)
into Accredito
```

Il Trigger controlla il totale nuovo su account e se > del vecchio inserisce una nuova riga in una tabella diversa Accredito. New e Old si riferiscono alla riga della tabella referenziata dal Trigger, nell'esempio Account

## Esempio di trigger statement-level

```
create trigger ArchiviaFattureCancellate
after delete on Fattura
/* Stiamo omettendo [for each statement] */
REFERENCING OLD TABLE AS
VECCHIE_FATTURE
insert into FattureCancellate
(select *
 from VECCHIE_FATTURE)
```

Osservare con for each statement il referencing è obbligatorio

Il Trigger dopo un delete di una o più fatture, ovvero quando termina lo statement, sulla tabella **Fattura**, allora inserisce all'interno della tabella fatturecancellate le righe cancellate che risultano dalla variabile old denominata attraverso il referencing VECCHIE\_FATTURE.

## Esecuzione di Trigger in conflitto



- Quando vi sono più trigger associati allo stesso evento (in conflitto) vengono eseguiti come segue:
  - Per primi i **before** triggers (*statement-level* e *row-level*)
  - Poi viene eseguita la modifica e verificati i vincoli di integrità
  - Infine sono eseguiti gli **after** triggers (*row-level* e *statement level*)
- Quando vari trigger appartengono alla stessa categoria, l'ordine di esecuzione è definito in base al loro timestamp di creazione (i trigger più vecchi hanno priorità più alta)

## Modello di esecuzione ricorsivo



- In SQL:1999 i triggers sono associati ad un "Trigger Execution Context" (TEC)
- L'azione di un trigger può produrre eventi che attivano altri trigger, che verranno valutati con un nuovo TEC interno:
  - Lo stato del TEC includente viene salvato e quello del TEC incluso viene eseguito. Ciò può accadere ricorsivamente
  - Alla fine dell'esecuzione di un TEC incluso, lo stato di esecuzione del TEC includente viene ripristinato e la sua esecuzione ripresa
- L'esecuzione termina correttamente in uno "stato quiescente"
- L'esecuzione termina in errore quando si raggiunge una data profondità di ricorsione dando luogo ad una eccezione di non-terminazione
- Se si verifica un errore o eccezione durante l'esecuzione di una catena di trigger attivati inizialmente da uno statement S, viene fatto un rollback parziale di S

## Proprietà formali dei trigger



- E' importante garantire che l'interferenza tra trigger in una qualunque loro attivazione non produca comportamenti anomali
- Vi sono tre proprietà classiche:
  - **Terminazione**: per un qualunque stato iniziale e una qualunque transazione, si produce uno stato finale (stato quiescente)
  - **Confluenza**: L'esecuzione dei trigger termina e produce un unico stato finale, indipendente dall'ordine di esecuzione dei trigger
  - **Univoca osservabilità**: I trigger sono confluenti e producono verso l'esterno (messaggi, azioni di display) lo stesso effetto
- La terminazione è la proprietà principale

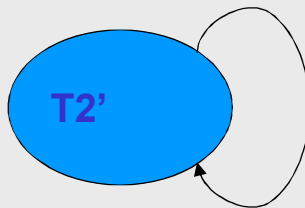
## Analisi della terminazione



- Si usa una rappresentazione delle regole detta **grafo di triggering**:
  - Un nodo per ogni trigger
  - Un arco dal nodo  $t_i$  al nodo  $t_j$  se l'esecuzione dell'azione di  $t_i$  può attivare il trigger  $t_j$  (ciò può essere dedotto con una semplice analisi sintattica)
- Se il grafo è aciclico, l'esecuzione termina
  - Non possono esservi sequenze infinite di triggering
- Se il grafo ha cicli, esso *può* avere problemi di terminazione: lo si capisce guardando i cicli uno per uno.

## Esempio di non terminazione


```
T2': create trigger ControllaSogliaBudget
after update on Impiegato
when New.Stipendio < 50000
update Stipendio
set Stipendio = 0.9 * Stipendio
```




## Esempio con due trigger

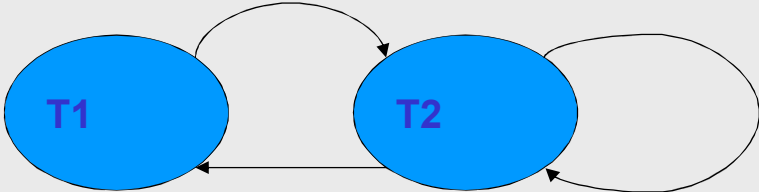
```
T1: create trigger CorreggiContributi
after update of Stipendio on Impiegato
referencing new table as NewImp
update Impiegato
set Contributi = Stipendio * 0.8
where Matr in (select Matr
               from NewImp)
```

```
T2: create trigger ControllaSogliaBudget
after update on Impiegato
when 50000 < (select (New.Stipendio
                    + New.Contributi)
              from Impiegato)
update Impiegato
set Stipendio = 0.9 * Stipendio
```





## Grafo di triggering corrispondente



- Ci sono due cicli ma il sistema termina.
- Per renderlo non terminante basta cambiare il comparatore nella condizione di T2 oppure moltiplicare per un fattore più grande di 1 nella azione di T2.

P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e

©2018 McGraw-Hill Education (Italy) S.r.l.

41





## Aspetti evoluti delle basi di dati attive

- Modalità di esecuzione (immediata, differita, distaccata)
- Amministrazione delle regole (priorità, gruppi, attivazione e deattivazione dinamica)
- Clausola “Instead-of”
- Altri eventi (di sistema, di utente, system-defined)
- Eventi complessi e calcolo degli eventi
- Una nuova categoria di sistema: “stream database”.

P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone, *Basi di dati*, 5e

©2018 McGraw-Hill Education (Italy) S.r.l.

42

## Modalità di esecuzione



- E' il collegamento tra attivazione (evento) e considerazione/esecuzione (condizione e azione)
- Condizione e azione sono sempre valutate assieme
- Caso "immediato": considerazione e esecuzione assieme all'evento
- Alternative:
  - Differito: il trigger è valutato alla fine della transazione
    - Esempio d'uso: trigger che gestiscono vincoli di integrità che possono essere violati durante una transazione
  - Distaccato: il trigger è valutato in un'altra transazione
    - Esempio d'uso: gestione di una variazione di valore di titoli della borsa

## Priorità, attivazioni e gruppi



- Definizione di priorità:
  - Specifica l'ordine di esecuzione dei trigger quando molti di loro vengono attivati dallo stesso evento
  - SQL:1999 indica la priorità di differenti classi di trigger; all'interno di una classe l'ordine dipende dall'ordine di creazione
- Attivazione/deattivazione dei trigger
  - Non è standard, ma è spesso disponibile
- Organizzazione dei trigger in gruppi
  - Alcuni sistemi consentono di raggruppare trigger e quindi di attivarli/deattivarli come gruppo

## Clausola **instead of**



- Alternativa a **before** e **after**
- Viene eseguita una differente operazione rispetto a quella che ha attivato il trigger
- La semantica è piuttosto pericolosa (l'applicazione fa una cosa e il sistema ne fa un'altra)
- Implementata in alcuni sistemi con limitazioni
  - In Oracle si può usare per ridirigere gli update dalle viste alle tabelle di base in caso di ambiguità