

Triggers

Antonella Poggi, Claudio Corona

Dipartimento di informatica e Sistemistica
Università di Roma “La Sapienza”

**Progetto di Applicazioni Software
Anno accademico 2008-2009**

Questi lucidi sono stati prodotti sulla base del materiale preparato per il corso di Progetto di Basi di Dati da D. Lembo.

Basi dati attive

- Una base di dati si dice **attiva** quando possiede un sottosistema integrato in grado di gestire **regole di produzione**
- Le regole di produzione seguono il paradigma **Evento-Condizione-Azione** (ECA): ciascuna regola reagisce ad alcuni eventi, valuta una condizione, ed, in base al valore di verità della condizione, esegue una reazione
- Il sottosistema controlla l'esecuzione delle regole
- Il sistema risultante ha un **comportamento reattivo** (che si differenzia dal tipico comportamento passivo delle basi di dati)

Indipendenza della conoscenza

- La conoscenza relativa al comportamento reattivo viene sottratta ai programmi applicativi e codificata sotto forma di regole di produzione
- Tale conoscenza è definita una volta per tutte in modo da essere condivisa da **tutte** le applicazioni che utilizzano la base di dati stessa
- In tal modo si evita di replicare la definizione del comportamento reattivo in ciascuna applicazione

Trigger

- Le regole di produzione messe tipicamente a disposizione dai DBMS sono dette **trigger** (o regole attive)
- I trigger fanno parte dello standard SQL
- La creazione dei trigger fa parte del Data Definition Language
- I trigger creati possono essere cancellati, ed in genere possono essere attivati e disattivati dinamicamente

Trigger (cont.)

- Nel paradigma Evento-Condizione-Azione adottato dai trigger:
 - Gli eventi sono primitive per la manipolazione dei dati: INSERT, DELETE, UPDATE
 - La condizione (che può mancare) è un predicato booleano
 - L'azione è una procedura eseguita quando il trigger è attivato e la condizione è verificata

Trigger (cont.)

- I trigger fanno riferimento ad una tabella, in quanto rispondono ad eventi su questa tabella
- I trigger hanno due livelli di granularità:
 - di tupla (**row-level**): l'attivazione avviene per ogni tupla coinvolta nella operazione (in genere si specifica l'opzione `FOR EACH ROW` nella definizione del trigger)
 - di istruzione (**statement-level**): l'attivazione avviene una sola volta a seguito dell'evento facendo riferimento a tutte le tuple coinvolte nell'operazione

Trigger (cont.)

- L'azione del trigger può essere effettuata **prima** (opzione `BEFORE`) che i cambiamenti previsti dall'evento siano eseguiti sulla tabella associata, oppure **dopo** (opzione `AFTER`)
- L'esecuzione del trigger avviene all'interno della transazione che ha causato l'evento "innescante"
- In SQL è prevista la possibilità di valutare il trigger alla fine della transazione a seguito dell'esecuzione di un `commit work` (**modalità differita**)
N.B. Ad oggi, la maggior parte dei DBMS (tra cui Oracle o MySQL) non supportano la modalità differita.

Trigger in MySQL / 1

- In MySQL, un trigger può essere associato ad una stored procedure precedentemente definita, oppure può definire esso stesso l'azione da intraprendere (codificata in SQL) a seguito dell'evento innescante
- Nel codice SQL della stored procedure associata al trigger oppure nel codice definito dal trigger stesso si verifica la condizione e si definisce l'azione da eseguire

Trigger in MySQL / 2

- È possibile specificare **esclusivamente row-level trigger**: il codice associato al trigger è eseguito **una volta** per ogni tupla interessata dal cambiamento (cioè modificata dall'evento);
- Si utilizzano gli identificatori `OLD` e `NEW` (da passare eventualmente come parametri alla stored procedure associata al trigger), che indicano rispettivamente la tupla “corrente” **prima** e **dopo** l'evento (inserimento, cancellazione, o aggiornamento) che ha innescato il trigger, e consentono di accedere ai suoi campi.

Trigger in MySQL / 3

- Si noti tuttavia che:
 - I trigger di tipo `INSERT` possono accedere solo ai valori nuovi
 - I trigger di tipo `DELETE` possono accedere solo ai valori vecchi
 - I trigger di tipo `UPDATE` possono accedere sia ai valori vecchi che a quelli nuovi

Sintassi dei Trigger / 1

```
CREATE TRIGGER trigger_name  
[DEFINER = { user | CURRENT_USER }]  
{BEFORE | AFTER}  
{INSERT | DELETE | UPDATE}  
ON tbl_name FOR EACH ROW trigger_stmt
```

– `trigger_name` è il nome del trigger (eventualmente preceduto da `database_name.`).

– `tbl_name` è il nome della tabella su cui si deve verificare l'evento per innescare il trigger.

Sintassi dei Trigger / 2

– `trigger_stmt` è il comando eseguito all'attivazione del trigger

- per eseguire più di un comando si usa il costrutto `BEGIN...END`
- è possibile usare gli stessi comandi ammessi nelle stored procedure
- se si vuole chiamare una stored procedure, si usa l'istruzione `CALL`

Sintassi dei Trigger / 3

- `BEFORE` indica che il `trigger_stmt` deve essere eseguito prima che l'evento che lo ha innescato sia "completato" (l'inserimento, la cancellazione o la modifica vengono di fatto messi in attesa che sia eseguita l'azione del trigger), `AFTER` indica che il `trigger_stmt` deve essere eseguito dopo l'evento.
- Solo uno fra `INSERT` | `DELETE` | `UPDATE` può essere utilizzato, ed indica l'evento che innesca il trigger.

Sintassi dei Trigger / 4

- È possibile usare l'opzione

```
[DEFINER = { user | CURRENT_USER }]
```

all'interno dello statement di creazione di un trigger per specificare con quali privilegi si intende che l'azione sia eseguita

→ di default sono quelli dell'utente che crea il trigger (ovvero sono quelli del `CURRENT_USER`), altrimenti sono quelli dell'utente "user" manuale MySQL).

Cancellare Trigger

Per **eliminare** un trigger si usa il comando

```
DROP TRIGGER trigger_name;
```

Si noti che, nel caso in cui il trigger sia associato ad una stored procedure, cancellare il trigger non significa cancellare la stored procedure associata

Esempio

```
CREATE TABLE test1 (a1 INT);  
CREATE TABLE test2 (a2 INT);  
CREATE TABLE test3 (a3 INT);  
CREATE TABLE test4 (  
    a4 INT,  
    b4 INT  
);
```

```
DELIMITER |  
  
CREATE TRIGGER testref BEFORE INSERT ON test1  
  FOR EACH ROW BEGIN  
    INSERT INTO test2 VALUES (NEW.a1);  
    DELETE FROM test3 WHERE a3=NEW.a1;  
    UPDATE test4 SET b4=b4+1 WHERE a4=NEW.a1;  
  END  
|
```

```
DELIMITER ;
```

```
INSERT INTO test3 (a3) VALUES (1);
```

```
INSERT INTO test4 (a4,b4) VALUES (1,0);
```

```
INSERT INTO test1 VALUES (1);
```

Esempio

```
mysql> SELECT * FROM test1;
```

```
+-----+  
| a1    |  
+-----+  
|      1 |  
+-----+
```

```
mysql> SELECT * FROM test2;
```

```
+-----+  
| a2    |  
+-----+  
|      1 |  
+-----+
```

```
mysql> SELECT * FROM test3;  
Empty set (0.00 sec)
```

```
mysql> SELECT * FROM test4;
```

a4	b4
1	1

Nota

Un trigger si esegue solo sulle tuple modificate dall'evento innescante: se non ce ne sono, non viene eseguito.

Esempio: Sia data la seguente tabella chiamata `tab_eta`

```
+-----+-----+
|  id   |  eta   |
+-----+-----+
|  A1   |   20   |
+-----+-----+
```

Assumiamo di avere un trigger per una `DELETE` su `tab_eta`

Si esegua il comando

```
DELETE FROM tab_eta WHERE id='A3';
```

Poichè non ci sono righe cancellate, il trigger non scatta.

Esempio

Assumiamo ora di avere le seguenti tabelle

```
CREATE TABLE prodotto(  
nome    VARCHAR(25),  
prezzo  FLOAT);
```

```
CREATE TABLE vendite(  
prodotto VARCHAR(25),  
data     DATE);
```

```
CREATE TABLE incasso_giornaliero(  
data     DATE,  
totale  FLOAT);
```

Esempio (cont.)

Si consideri il seguente problema:

Per ogni articolo venduto in una data d si aggiorni automaticamente l'incasso totale relativo alla data d

Nel nostro schema questo vuole dire che:

All'inserimento di una tupla $\langle p, d \rangle$ all'interno della tabella vendite, si aggiorna automaticamente la tabella `incasso_giornaliero` in modo che

1. se non esiste una tupla in `incasso_giornaliero` del tipo $\langle d, t \rangle$, vi si inserisce $\langle d, p \rangle$ dove p è il prezzo dell'articolo venduto
2. altrimenti si aggiorna la tupla $\langle d, t \rangle$ nella tupla $\langle d, t' \rangle$, con $t' = t + p$

Esempio (cont.)

```
CREATE PROCEDURE incrementa (n VARCHAR(25), d DATE)
BEGIN
  DECLARE pr FLOAT;
  DECLARE dd DATE;
  DECLARE present BOOL DEFAULT 1;
  DECLARE verifica_presenza CURSOR FOR SELECT data
                                         FROM incasso_giornaliero
                                         WHERE data=d;

  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET present=0;
  SELECT prezzo INTO pr FROM prodotto WHERE nome=n;
  OPEN verifica_presenza;
  FETCH verifica_presenza INTO dd;
  IF NOT present THEN
    INSERT INTO incasso_giornaliero VALUES (d,pr);
  ELSE UPDATE incasso_giornaliero SET totale=totale+pr WHERE data=d;
  END IF;
END
```

Esempio (cont.)

```
CREATE TRIGGER incrementa_tgr  
AFTER INSERT ON vendite  
FOR EACH ROW  
CALL incrementa(NEW.prodotto,NEW.data);
```

Nota: Ricordarsi di cambiare il delimitatore dei comandi quando si definisce la stored procedure.

Esempio

Assumiamo ora di avere la seguente relazione

```
CREATE TABLE impiegato(  
  codice VARCHAR(25),  
  nome VARCHAR(25),  
  salario INT  
);
```

e la seguente stored procedure MySQL

```
CREATE PROCEDURE allinea_salario (livello INT)  
UPDATE IMPIEGATO SET SALARIO=livello  
WHERE SALARIO<livello;
```

Esempio

Creiamo un trigger che esegue la stored procedure

```
CREATE TRIGGER trig_salario  
AFTER UPDATE ON impiegato  
FOR EACH ROW CALL allinea_salario(100);
```

Esempio

Creiamo un trigger che esegue la stored procedure

```
CREATE TRIGGER trig_salario  
AFTER UPDATE ON impiegato  
FOR EACH ROW CALL allinea_salario(100);
```

Anche se il trigger è corretto, se tentiamo di aggiornare la tabella `impiegato` otteniamo il messaggio di **errore**:

```
Can't update table 'impiegato' in stored function/trigger  
because it is already used by statement which invoked this  
stored function/trigger.
```

Limite di MySQL: Nel trigger non è possibile modificare la tabella interessata dall'evento che innesca il trigger

Altri limiti nell'utilizzo dei trigger in MySQL

- Non è possibile definire due trigger sulla stessa tabella, per la stessa coppia (evento, momento);
- Le cancellazioni in cascata (ottenute tramite il comando `CASCADE` associato ad un foreign key) non innescano eventuali trigger associati a tali cancellazioni;
- Un trigger non può innescare istruzioni di tipo `RETURN`
→ Non è possibile inviare **esplicitamente** un messaggio di errore dal corpo di un trigger (o da una stored procedure) in modo da bloccare l'esecuzione dell'evento che ha innescato il trigger (o della transazione che ha invocato la stored procedure); in altre parole, non si può simulare un comportamento come quello che si ottiene in `JAVA` tramite l'uso delle eccezioni.

Cicli nei trigger

Un trigger può attivarne un altro; in tal caso i trigger si dicono **in cascata**. Le reciproche attivazioni dei trigger si rappresentano col **grafo di attivazione**. Quando c'è un **ciclo** nel grafo di attivazione **può** esserci una serie infinita di esecuzioni di trigger.

In genere, i DBMS presentano un limite massimo consentito di attivazioni consecutive dello stesso trigger (per evitare cicli infiniti).

Tuttavia, spesso i cicli nel grafo di attivazione sono innocui.

Occorre comunque fare attenzione ai cicli quando si realizzano trigger.