Trigger

Massimo Mecella

Dipartimento di Ingegneria informatica automatica e gestionale
Antonio Ruberti
Sapienza Università di Roma

Progetto di Applicazioni Software

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**

Indipendenza della conoscenza

- Il comportamento reattivo è una caratteristica intrinseca del dominio
- Il comportamento reattivo è definito una volta per tutte in modo da essere condiviso da **tutte** le applicazioni che utilizzano la base di dati stessa
  $\rightarrow$ 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

Casi di uso di trigger

I trigger sono generalmente definiti per i seguenti scopi:

- assicurare l'integrità dei dati
- controllare i cambiamenti (e.g. mantenere un log degli utenti e ruoli coinvolti nei cambiamenti)
- eseguire regole di business (e.g. avvisare un utente ogni qualvolta una certa condizione è soddisfatta)

I trigger ed il paradigma ECA

- Nel paradigma ECA adottato dai trigger:
  - Gli eventi sono primitive per la manipolazione dei dati: `INSERT`, `DELETE`, `UPDATE`
    $\rightarrow$ un trigger fa riferimento ad una tabella, in quanto definisce come reagire ad eventi su una tabella
  - 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

Granularità di azione di un trigger

- L'azione di un trigger può essere definita a due livelli di granularità:
  - di tupla (**row-level**): viene eseguita per ogni tupla coinvolta nella operazione (in genere si specifica l'opzione `FOR EACH ROW` nella definizione del trigger)
  - di istruzione (**statement-level**): viene eseguita una sola volta a fronte dell'evento facendo riferimento a tutte le tuple coinvolte nell'operazione

Esecuzione dell'azione del trigger

- L'azione del trigger può essere eseguita:
  - **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) – (opzione `BEFORE`)
  - **dopo** che sia completato l'evento – (opzione `AFTER`)
  - **al posto** dell'operazione corrispondente all'evento stesso – (opzione `INSTEAD`)
- L'esecuzione del trigger avviene all'interno della transazione che ha causato l'evento "innescante"

Modalità di azione differita

- 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 non supportano la modalità differita.

Sintassi standard per la definizione di trigger

```
CREATE TRIGGER nome_trigger
  { BEFORE | AFTER | INSTEAD OF }
  { INSERT | UPDATE [ OF column_name [, ... ] ]
    | DELETE }
  ON tabella
  { REFERENCING OLD ROW AS alias_old
    NEW ROW AS alias_new}
  { NOT DEFERRABLE | [ DEFERRABLE ]
    { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }
  [ FOR [ EACH ] { ROW | STATEMENT } ]
  [ WHEN ( condition ) ]
  EXECUTE sql_statement
```

Variabili speciali `OLD` e `NEW`

Lo standard SQL prevede l'utilizzo delle variabili `OLD` e `NEW` per rappresentare nella definizione della condizione e/o dell'azione del trigger:

- gli insiemi delle tuple interessate prima e dopo l'evento scatenante, se si tratta di un trigger statement-level
- la tupla interessata prima e dopo l'evento scatenante, se si tratta di un un trigger row-level.

N.B. `NEW` non ha senso nei trigger scatenati da eventi `DELETE`, mentre `OLD` non ha senso nei trigger scatenati da eventi `INSERT`.

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.
Occorre quindi fare attenzione ai cicli quando si realizzano trigger.
In genere, i DBMS presentano un limite massimo consentito di attivazioni consecutive dello stesso trigger (per evitare cicli infiniti).

Esempio di trigger

NOTA: negli esempi successivi, usiamo una sintassi in pseudoSQL per le stored procedure in modo da non concentrarsi su dettagli propri del particolare DBMS (in quanto le stored procedure dipendono dal DBMS). Successivamente nel corso/esercitazioni si presenteranno i dettagli in PostgreSQL

Supponiamo di definire le seguenti tabelle:

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

Si definisca poi il seguente trigger:

```
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
```

Cosa fa questo trigger?
Si supponga di popolare le tabelle definite sopra come segue:

```
INSERT INTO test3 (a3) VALUES (1);
INSERT INTO test4 (a4,b4) VALUES (1,0);
INSERT INTO test1 VALUES (1);
```

Che succede?

```
SELECT * FROM test1;
+------+
| a1   |
+------+
|    1 |
+------+

SELECT * FROM test2;
+------+
| a2   |
+------+
|    1 |
+------+
```

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

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.

Altro esempio di trigger

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 <p,d> 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 <d,t>, vi si inserisce <d,p> dove p è il prezzo dell'articolo venduto
2. altrimenti si aggiorna la tupla <d,t> nella tupla <d,t'>, 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
```