

I Trigger in MySQL

Definizione: Un **trigger** è una procedura che viene eseguita in maniera automatica in coincidenza di un determinato evento (come ad esempio l'inserimento o la cancellazione o l'aggiornamento di un record di una tabella). In questo modo si ha a disposizione una tecnica per specificare e mantenere eventuali vincoli di integrità anche complessi non esprimibili in altro modo (si pensi ad esempio le clausole CHECK sul valore di uno o più attributi, etc.).

Quindi tale procedura (**trigger**) è associata ad una tabella e viene automaticamente richiamata dal motore del DBMS quando una certa modifica (o evento) avviene all'interno della tabella.

Le modifiche sulla tabella possono includere operazioni *insert*, *update*, e *delete*.

Quando definiamo un trigger, occorre stabilire per quale evento esso deve essere attivato (inserimento, modifica o cancellazione di righe) e se deve essere eseguito prima o dopo tale evento.

Avremo quindi i seguenti tipi di trigger:

- **BEFORE INSERT**
- **BEFORE UPDATE**
- **BEFORE DELETE**

- **AFTER INSERT**
- **AFTER UPDATE**
- **AFTER DELETE**

Il trigger definirà una serie di istruzioni che verranno eseguite per ogni riga interessata dall'evento.

N.B. Non è possibile associare un trigger a una tabella temporanea o una vista.

Sintassi generale

La dichiarazione di un trigger si presenta in questo modo:

CREATE TRIGGER *nome_del_trigger*

[**BEFORE** | **AFTER**]

[**INSERT** | **UPDATE** | **DELETE**]

ON *nome_della_tabella*

FOR EACH ROW

codice_SQL_da_applicare_con_il_trigger

Come possiamo vedere si **contano 6 sezioni** corrispondenti ognuna ad una linea della dichiarazione: analizziamole una per una.

La **prima linea indica il nome del trigger** che stiamo creando. Non possono esistere due trigger con lo stesso nome in un solo database visto che lo scope del trigger stesso è relativo all'intero database e non alla singola tabella di riferimento. Il nome, analogamente ai nomi di tabella o di database, deve essere lungo al massimo 64 caratteri ed è consigliabile inserirlo tra *backtick* (") per evitare di riscontrare errori sull'uso di spazi o di caratteri non ASCII.

La **seconda linea determina il tempo di esecuzione** del trigger che stiamo definendo in base al momento di applicazione. Un trigger infatti può agire prima dell'esecuzione vera e propria della query oppure dopo, ed essere quindi rispettivamente definita come **AFTER** o **BEFORE**. Vedremo più avanti cosa questo comporta.

La **terza linea introduce il tipo di query SQL** che genererà l'esecuzione del trigger. La query può essere solo di tipo esecutivo quindi **INSERT, UPDATE, DELETE** ma non **SELECT**. In base al tipo di query ci troveremo a poter intervenire sul record già presente nel database (nel caso ad esempio di una query **UPDATE** o **DELETE**) oppure sul record nuovo che andremo ad inserire o a sostituire (query **UPDATE** o **INSERT**). Da notare che **INSERT** non rappresenta soltanto le query di tipo **INSERT**, bensì tutte le query che inseriscono dati ex-novo come **LOAD DATA** o **REPLACE** in caso di un record nuovo. Allo stesso modo vanno interpretati **UPDATE** e **DELETE**.

La **quarta linea specifica la tabella da monitorare** per attivare il trigger. Come ricordato prima l'associazione del trigger alla tabella non lo crea nello scope della tabella stessa bensì in quello generale del database. È comunque possibile avere un solo trigger attivo per lo stesso tipo di query sulla stessa tabella.

La **quinta linea illustra l'individualità del trigger** che infatti si applica ad ogni riga singolarmente e non a tutta la tabella nel suo insieme. Questo significa ad esempio che una cancellazione di massa avvia tanti trigger quante sono le righe interessate dalla stessa.

La **sesta linea infine rappresenta il codice SQL da eseguire** all'attivazione dell'evento, considerato come l'insieme di tipo di query, tempo di esecuzione e tabella associata legato ad esso. Il codice SQL utilizzabile presenta una serie di controlli di flusso (in tutto simili a quelli dei vari linguaggi di programmazione) che accenneremo nel prossimo paragrafo e che saranno approfonditi in un prossimo articolo sulle *stored procedures*.

Il codice da eseguire

Il codice SQL da eseguire all'attivazione del trigger deve essere **una singola istruzione seguita da punto e virgola**, anche se in realtà le istruzioni possono essere in numero superiore ma ognuna di esse deve necessariamente terminare con il punto e virgola.

L'istruzione fondamentale nel blocco di codice eseguito dal trigger è **SET**. Tramite questa istruzione possiamo infatti modificare i valori da inserire in un campo o impostare delle nostre variabili (entrambi i casi sono esemplificati più avanti).

Il codice da eseguire **dovrebbe essere un monoblocco**, e cioè una singola istruzione. Per coniugare questa esigenza con il bisogno di codici più complessi della singola riga si introduce l'uso di blocchi logici **BEGIN/END** simili a quelli utilizzati in Pascal. Tutto quello compreso tra il begin e l'end verrà interpretato come una singola istruzione.

Poichè ogni istruzione deve terminare con un punto e virgola, esso non potrà essere utilizzato come delimitatore per terminare la query: per questo si usa il comando **DELIMITER** seguito dal carattere o sequenza di caratteri da interpretare come separatore tra query diverse (Esempio **DELIMITER //** oppure **DELIMITER \$\$**)

Altre istruzioni utilizzabili sono **IF/THEN/ELSE**, **CASE**, **WHILE**, **REPEAT**, **LOOP**, **LEAVE** e saranno trattate più approfonditamente in un articolo futuro.

Esempi di utilizzo

Vediamo due esempi pratici abbastanza semplici di cosa si possa ottenere con i trigger:

- a) somma statica
- b) controllo dei dati inseriti in una tabella.

Ipotizziamo quindi di avere a disposizione la seguente tabella:

```
CREATE TABLE Prodotto  
(  
  Id          INT NOT NULL AUTO_INCREMENT,  
  Nome       VARCHAR(255) NOT NULL,  
  Qta        INT NOT NULL,  
  Prezzo     DECIMAL(10,2) NOT NULL,  
  PRIMARY KEY(Id)  
);
```

Tramite la funzione **SUM()** di MySQL è possibile ottenere una sommatoria di uno specifico campo di una tabella, in questo caso ad esempio è possibile conoscere la quantità totale di merce presente tramite questa query:

```
SELECT SUM(Qta) FROM Prodotto
```

In alcuni sistemi sono applicati degli approcci di *catching* per evitare di eseguire una simile query ogni volta che ce ne sia bisogno.

Ad esempio VBulletin (il software che con cui è gestito il Forum di Html.it) salva nella tabella del database anche il numero complessivo di messaggi aggiornando tale dato ad ogni inserimento o cancellazione. È possibile ottenere qualcosa del genere anche con i trigger: basterà definirne uno per ogni tipo di query in questo modo:

```
CREATE TRIGGER prodotti_insert_somma
BEFORE INSERT ON Prodotto
FOR EACH ROW SET @somma = @somma + NEW.Qta;
```

```
CREATE TRIGGER prodotti_delete_somma
BEFORE DELETE ON Prodotto
FOR EACH ROW SET @somma = @somma - OLD.Qta;
```

```
CREATE TRIGGER prodotti_update_somma
BEFORE UPDATE ON Prodotto
FOR EACH ROW SET @somma = @somma - OLD.Qta + NEW.Qta;
```

Nella variabile *@somma* troveremo la somma aggiornata dei prodotti presenti nella tabella.

Sarà sufficiente eseguire una query del tipo:

```
SELECT @somma
FROM Prodotto;
```

Questo stratagemma permette di **velocizzare** l'operazione di richiesta snellendo le operazioni di accesso in scrittura al database.

Un'altra applicazione di largo uso dei trigger è data dal **controllo della correttezza dei valori inseriti** nel database: potremmo avere un campo i cui valori possono oscillare tra un massimo e un minimo. Il controllo potrebbe essere fatto a priori tramite il linguaggio di programmazione oppure a posteriori tramite una query di `UPDATE` che aggiorni i record con correggendo eventuali errori, **o, ancora meglio, potrebbe essere impiegato un trigger.**

In questo esempio vengono eliminati i prezzi degli articoli inferiori a 1.00 o superiori a 10.00 sostituendoli rispettivamente con il valore 0 e con il valore 10.00.

```
DELIMITER //
CREATE TRIGGER trg1
BEFORE INSERT ON fornituranew.Articolo
FOR EACH ROW
BEGIN
IF NEW.Prezzo < 1.00
    THEN
        SET NEW.Prezzo = 0;
END IF;
IF NEW.Prezzo > 10.00
    THEN
        SET NEW.Prezzo = 10.00;
END IF;
END //
DELIMITER ;
```

N.B. Il qualificatore **NEW** indica proprio che il nome di colonna che stiamo utilizzando si riferisce al nuovo valore della riga che sta per essere aggiornata.

NEW si può utilizzare in caso di **INSERT** e **UPDATE**.

Analogamente è disponibile il qualificatore **OLD** che fa riferimento ai valori precedenti la modifica, e si può utilizzare in caso di **UPDATE** e **DELETE**.

N.B. La modifica attraverso l'istruzione **SET** è possibile solo per i valori **NEW** e solo nei trigger di tipo **BEFORE**.

```
DELIMITER //
CREATE TRIGGER trg2
BEFORE INSERT ON fornituranew.Articolo
FOR EACH ROW
BEGIN
DECLARE msg VARCHAR(255);
IF NEW.Prezzo < 1.00
    THEN
        SET msg = concat("Prezzo non puo' essere inferiore a 1.00");
        SIGNAL sqlstate '45000' SET message_text = msg;
    ELSE
        IF NEW.Prezzo > 10.00
            THEN
                SET msg = concat("Prezzo non puo' essere superiore a 10.00");
                SIGNAL sqlstate '45000' SET message_text = msg;
            END IF;
    END IF;
END IF;
END //
DELIMITER ;
```

In questo esempio non viene permesso l'inserimento di fornitori nati dopo il 1 gennaio 1960.

```
DELIMITER //  
CREATE TRIGGER trg3  
BEFORE INSERT ON fornituranew.Fornitore  
FOR EACH ROW  
BEGIN  
DECLARE msg VARCHAR(255);  
IF NEW.DataN <= "1960-01-01"  
  THEN  
    SET msg = concat("DataN non puo' essere precedente al 1 gennaio 1960");  
    SIGNAL sqlstate '45000' SET message_text = msg;  
END IF;  
END //  
DELIMITER ;
```

In questo esempio il campo Qta della tabella Fornisce deve essere un valore intero compreso tra 1 e 3.

```
DELIMITER //  
CREATE TRIGGER trg4  
BEFORE INSERT ON fornituranew.Fornisce  
FOR EACH ROW  
BEGIN  
DECLARE msg VARCHAR(255);  
IF NEW.Qta < 1  
  THEN  
    SET msg = concat("Quantita' non puo' essere inferiore a 1");  
    SIGNAL sqlstate '45000' SET message_text = msg;  
  ELSE  
    IF NEW.Qta > 3  
      THEN  
        SET msg = concat("Quantita' non puo' essere superiore a 3");  
        SIGNAL sqlstate '45000' SET message_text = msg;  
      END IF;  
    END IF;  
END IF;  
END //  
DELIMITER ;
```

La cancellazione

Così come è possibile creare un trigger tramite una query, sempre tramite una query sarà necessario cancellarlo eseguendo una query **DROP**:

```
DROP TRIGGER [database.]nome_del_trigger;
```

Bug conosciuti

Al momento per MySQL 5.0 il supporto per i triggers ha qualche “bug” conosciuto giustificato dall’introduzione recente, tanto è vero che con MySQL 5.1 alcuni di essi sono stati risolti. Tra questi:

- È necessario essere superuser (root o similare) per creare un trigger.
- I trigger non vengono attivati dalla cascata di eventi legati alle foreign keys.
- Se passate da una versione precedente ad una successiva alla 5.0.10 dovrete cancellare tutti i trigger, aggiornare MySQL e quindi ricrearli per non aver il problema di non riuscire più ad eliminarli.
- Le versioni precedenti alla 5.0.10 non permettono di eseguire trigger che contengano all’interno del codice SQL delle query vere e proprie.
- Nel codice SQL di un trigger non è possibile gestire transazioni.