Trigger

Introduzione

- L'introduzione di trigger all'interno di una Base di Dati permette la gestione automatica di particolari procedure in risposta a determinati eventi esterni;
- Basi di Dati di questo tipo vengono dette *Basi di Dati **attive***
  - i trigger rendono reattivo il comportamento del sistema alle sollecitazioni esterne.

Il livello di astrazione dei dati

- L'utilizzo di regole attive in un DB aumenta
  - il livello di astrazione dei dati introducendo una nuova dimensione detta *Indipendenza della conoscenza* che nasconde, all'esterno, l'esecuzione delle regole attive.
- Il vantaggio introdotto da questo tipo di astrazione è lo spostamento di alcuni controlli e procedure automatiche da livello applicativo a quello dello schema.

Il paradigma E-C-A

- **Evento:**
  - Qualsiasi modifica (insert,update,delete), o anche un altro trigger(in questo caso si dice che i trigger sono in *cascata*), introdotta su una tabella della Base di Dati(*tabella di target*).
- **Condizione** [opzionale]:
  - Un predicato booleano espresso mediante sintassi SQL.
- **Azione:**
  - Sequenza di primitive SQL generiche, talvolta arricchite da un linguaggio di programmazione integrato disponibile nell'ambito di uno specifico DBMS.

Tipi di Trigger

- **Trigger a livello di tupla**
  - I trigger a livello di riga vengono eseguiti una volta sola per ogni riga su cui agisce un'istruzione DML.

utili per mantenere sincronizzati i dati distribuiti.

Tipi di Trigger

- **Trigger a livello di istruzione**
  - I trigger a livello di istruzione vengono eseguiti una sola volta per ogni istruzione DML.
    - Esempio: se una singola istruzione INSERT avesse inserito 500 righe in una tabella, un trigger a livello di istruzione su tale tabella verrebbe eseguito una sola volta.
- I trigger a livello di istruzione non vengono utilizzati di frequente per attività a correlate ai dati.

Tipi di Trigger

- **Trigger BEFORE e AFTER**

  Poichè i trigger vengono innescati da eventi (istruzioni DML)

  - È possibile impostarli in modo che si verifichino immediatamente prima o dopo tali eventi.

Tipi di Trigger

- **Trigger INSTEAD OF**
  - In alcuni DBMS (Oracle ad esempio) è possibile utilizzare trigger INSTEAD OF per indicare cosa fare invece di eseguire le azioni che hanno invocato il trigger.
- Ad Esempio:
  - con un trigger INSTEAD OF è possibile specificare al DBMS come eseguire aggiornamenti, cancellazioni e inserimenti di record nelle tabelle base della vista quando un utente cerca di modificare i valori attraverso la vista stessa.

Tipi di Trigger

- **Trigger a livello di schema e database**
  - sono trigger che si innescano su operazioni eseguite a livello di schema, come create table, alter table, drop table, audit, rename, truncate e revoke.
  - I trigger a livello di schema offrono principalmente due capacità:
    - **impedire l'esecuzione di operazioni DDL**
    - **fornire ulteriore controllo della sicurezza quando si verificano operazioni DDL.**

I Trigger SQL:2003

Un Trigger è una specifica per una determinata azione che deve essere eseguita ogniqualvolta avviene un'azione su un determinato oggetto.

- Definizioni
  - *triggered action* (azione d'innesco), può essere sia una dichiarazione di una procedura SQL oppure una lista di dichiarazioni.
  - *target table è* una tabella della base di dati persistente, detta anche (tabella obiettivo) del trigger.
  - *Event trigger*, può essere un inserimento, una cancellazione o la modifica di un insieme di righe.

- A seconda del momento di azione i trigger sono divisi in AFTER TRIGGER o BEFORE TRIGGER,
- a seconda della natura dell'evento innescante si dividono invece in DELETE TRIGGER, INSERT TRIGGER oppure UPDATE TRIGGER

I Trigger SQL:2003

- Per ogni trigger possono venire adoperate delle tabelle di transizione atte a memorizzare lo stato precedente e successivo del target del trigger.
- In particolare:
  - Per il delete trigger è prevista una sola tabella di transizione detta old transition table
  - per un insert trigger una tabella detta new transition table
  - per un update trigger sono previste en trambe le tabelle prima citate

La granularità

- l'azione del trigger può essere di tipo
  - *statement-level trigger* (eseguito solo una volta dopo l'attivazione del trigger)
  - *Rowlevel trigger*. (eseguito una volta per ogni riga della tabella di transizione associata al trigger.

Variabili dei trigger

- Vengono messe a disposizione dei trigger delle variabili speciali che rendono possibili le modifiche sulle tabelle di transizione.
- **Statement-level trigger**
  - la variabile ha il valore della tabella di transizione
- **Row-level trigger**
  - Variabile di transizione associa ad ogni azione del trigger il valore della tupla nella tabella di transizione da modificare; ed è una *old transition variable* oppure una *new transition variable* a seconda se si parla di delete trigger o insert trigger, nel caso di update trigger

Definizione di trigger

- Nome
- Nome della tabella di target
- Il momento d'azione (BEFORE e AFTER)
- L'evento (insert, update, delete)
- Se è statement-level oppure row-level trigger
- Nomi per le tabelle di transizione o per le variabili
- L'azione
- Un timestamp della creazione del trigger.

Esecuzione

- Durante l'esecuzione di un'istruzione S, possono esistere zero o più contesti d'esecuzione trigger, però al più uno di loro può essere attivo.
- Il contesto di esecuzione di un trigger (TEC) viene creato, attivato, eseguito e distrutto come descritto nelle regole generali per S.

Trigger in MySql

- Trigger di tipo insert:

mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));

Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account

-> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)

Sintassi dei trigger in MySql

CREATE

[DEFINER = { user | CURRENT_USER }]

TRIGGER nome_trigger trigger_time

trigger_event

ON tabella_target

FOR EACH ROW

trigger_stmt

Sintassi dei trigger in MySql

- **DEFINER**, determina i privilegi da applicare quando il trigger è attivo;
- **TRIGGER_TIME**, rappresenta il tipo di trigger: BEFORE o AFTER;
- **TRIGGER_EVENT**, indica il tipo di primitiva che attiva il trigger:
  - INSERT: il trigger è attivato quando una nuova tupla è inserita nella tabella target; ad esempio, attraverso le primitive INSERT, LOAD DATA e REPLACE;
  - UPDATE: il trigger è attivato quando una tupla della tabella target viene modificata; ad esempio, attraverso la primitiva UPDATE;
  - DELETE: il trigger è attivato quando una tupla della tabella target viene cancellata; ad esempio, attraverso le primitive DELETE e REPLACE.

Sintassi dei trigger in MySql

- **TRIGGER_STMT**, è la primitiva SQL da eseguire, una volta che il trigger è attivo; se si desidera eseguire una procedura SQL, si può utilizzare il costrutto **BEGIN...END**.

Variabili OLD e NEW

Per accedere alle colonne della tabella target si utilizzano gli alias OLD e NEW.

- In particolare:
  - OLD.col_name (read-only) si riferisce alla colonna della tabella target prima della modifica effettuata dall'evento associato al trigger.
  - NEW.col_name si riferisce alla colonna della tabella target dopo la modifica effettuata dall'evento.

Variabili OLD e NEW

- In un INSERT trigger, può essere usata solo la variabile NEW;
- in un DELETE trigger, può essere usata solo la variabile OLD.
- In un BEFORE trigger, si può usare il comando SET NEW.column = value, per la modifica o l'inserimento di un nuovo valore all'interno della tupla.

Privilegi di sistema richiesti per creare Trigger

- La clausola **DEFINER** specifica l'account utente associato al trigger; all'atto dell'attivazione del trigger saranno controllati i privilegi del suddetto account. Il formato della clausola deve essere del tipo:

  'user_name'@'host_name'

Privilegi in MYSQL

- Al momento della creazione del trigger, l'utente che crea il trigger deve avere i privilegi di **SuperUtente**;
- Al momento dell'attivazione del trigger, vengono controllati i privilegi dell'account utente definito nella clausola DEFINER. Il suddetto utente deve possedere i seguenti privilegi:
  - I privilegi di SuperUtente;
  - Il privilegio di SELECT sulla tabella target, se sono contenuti riferimenti alle colonne della stessa (attraverso le variabili OLD e NEW);
  - I privilegi di UPDATE sulla tabella target, se sono presenti primitive che modificano
  - i valori delle tabelle (attraverso istruzioni del tipo: SET NEW.colname = value);
  - Tutti i privilegi che sono necessari per l'esecuzione dello statement SQL.

Chiamata di procedure all'interno dei trigger

- I trigger non possono invocare procedure che restituiscono dati all'utente o che utilizzano SQL dinamico;
- I trigger non possono invocare procedure che esplicitamente iniziano o terminano una transazione (primitive START TRANSACTION, END TRANSACTION, COMMIT o ROLLBACK);

Gestione degli Errori

- Se un BEFORE trigger fallisce, l'operazione che ha generato l'evento di attivazione, non viene eseguita;
- Un AFTER trigger viene eseguito, se precedentemente sono stati eseguiti i BEFORE trigger e la primitiva di attivazione;
- Un errore durante un BEFORE o un AFTER trigger, genera il fallimento dell'intera procedura che causa l'attivazione del trigger.

In un sistema transazionale, il fallimento di un trigger causa il rollback di tutte le modifiche effettuate dal trigger; in un sistema non transazionale, invece, tutte le modifiche apportate prima dell'errore, rimangono effettive.

Limitazione all'utilizzo dei trigger

- In MySql non possono essere creati due trigger con evento e tipo evento uguali, su una stessa tabella target.
- Ad esempio:
  - non possono essere definiti due BEFORE INSERT trigger o due AFTER UPDATE trigger per una stessa tabella.

Esempio di trigger: Inserimento utente

```
CREATE TABLE 'bd2'.'utente' (
'idutente' INTEGER UNSIGNED NOT NULL
    AUTO_INCREMENT,
'nome' VARCHAR(15) NOT NULL DEFAULT '',
'cognome' VARCHAR(25) NOT NULL DEFAULT '',
'password' VARCHAR(45) NOT NULL DEFAULT '',
PRIMARY KEY('idutente')
)
ENGINE = InnoDB;
```

Creazione Funzione

```
mysql> delimiter //
mysql> create function valida_stringa (str varchar(25))
-> returns varchar(25)
-> deterministic
-> begin
-> declare testa varchar(1);
-> declare coda varchar(24);
-> set str = trim(str);
-> set testa = upper(left(str,1));
-> set coda = lower(right(str,length(str)-1));
-> return concat(testa,coda);
-> end;
-> //
Query OK, 0 rows affected (0.00 sec)
```

Creazione Trigger

```
mysql> create trigger UTENTE_BEF_INS_ROW
-> before insert on utente
-> for each row
-> begin
-> set NEW.nome = valida_stringa(NEW.nome);
-> set NEW.cognome = valida_stringa(NEW.cognome);
-> set NEW.password = password(NEW.password);
-> end;
-> //
Query OK, 0 rows affected (0.00 sec)
```

Esempio 1

```
mysql> delimiter ;
mysql> insert into utente values
    (1,'alessio','meola','passwd1010');
Query OK, 1 row affected (0.29 sec)

mysql> insert into utente values
    (2,'gIuSePPe','MAZZARELLA','passwd2020');
Query OK, 1 row affected (0.27 sec)
```

Risultato

```
mysql> select * from utente;
+----------+----------+-----------+-------------------------------------+
| idutente | nome     | cognome   | password                            |
+----------+----------+-----------+-------------------------------------+
|        1 | Alessio  | Meola     | *4BD714D8E708D3D6BF85BB246E0DCE6C32 |
|        2 | Giuseppe | Mazzarella | *80BCC99B46B82A5A8C18B98EDD7585AE51 |
+----------+----------+-----------+-------------------------------------+
2 rows in set (0.00 sec)
```

Esempio 2: Gestione magazzino e analisi dei dati

- Le operazioni previste sono:
- riordino della merce
- statistiche sulle vendite.

Per queste due operazioni sono stati creati due trigger e una funzione.

Creazione Tabelle

```
mysql> CREATE TABLE ‘bd2‘.‘prodotto‘ (
    ->   ’idprodotto’ INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    ->   ’nome’ VARCHAR(45) NOT NULL DEFAULT ’’,
    ->   ’marca’ VARCHAR(45) NOT NULL DEFAULT ’’,
    ->   ’vendita’ FLOAT NOT NULL DEFAULT 0,
    ->   ’acquisto’ FLOAT NOT NULL DEFAULT 0,
    ->   PRIMARY KEY(’idprodotto’)
    -> )
    -> ENGINE = InnoDB;
Query OK, 0 rows affected (0.14 sec)

mysql> CREATE TABLE ’bd2’.’magazzino’ (
    ->   ’idmagazzino’ INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    ->   ’citta’ VARCHAR(45) NOT NULL DEFAULT ’’,
    ->   PRIMARY KEY(’idmagazzino’)
    -> )
    -> ENGINE = InnoDB;
Query OK, 0 rows affected (0.13 sec)
```

Creazione Tabelle

```
mysql> CREATE TABLE 'bd2'.'prodotti_magazzino' (
    ->   'idprodotto' INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,


->   'idmagazzino' INTEGER UNSIGNED NOT NULL DEFAULT 0,
->   'qta' INTEGER UNSIGNED NOT NULL DEFAULT 0,
->   'qta_limite' INTEGER UNSIGNED NOT NULL DEFAULT 0,
->   'qta_riordino' INTEGER UNSIGNED NOT NULL DEFAULT 0,
->   PRIMARY KEY('idprodotto', 'idmagazzino'),
->   CONSTRAINT 'prod_mag_prod' FOREIGN KEY 'prod_mag_prod' ('idprodotto')
->     REFERENCES 'prodotto' ('idprodotto')
->     ON DELETE RESTRICT
->     ON UPDATE RESTRICT,
->   CONSTRAINT 'prod_mag_mag' FOREIGN KEY 'prod_mag_mag' ('idmagazzino')
->     REFERENCES 'magazzino' ('idmagazzino')
->     ON DELETE RESTRICT
->     ON UPDATE RESTRICT
-> )
-> ENGINE = InnoDB;
```

Creazione Tabelle

```
mysql> CREATE TABLE 'bd2'.'ordine' (
    ->   'idprodotto' INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    ->   'idmagazzino' INTEGER UNSIGNED NOT NULL DEFAULT 0,
    ->   'data' DATETIME NOT NULL DEFAULT 0,
    ->   PRIMARY KEY('idmagazzino', 'idprodotto'),
    ->   CONSTRAINT 'ord_prod' FOREIGN KEY 'ord_prod' ('idprodotto')
    ->     REFERENCES 'prodotto' ('idprodotto')
    ->     ON DELETE RESTRICT
    ->     ON UPDATE RESTRICT,
    ->   CONSTRAINT 'ord_mag' FOREIGN KEY 'ord_mag' ('idmagazzino')
    ->     REFERENCES 'magazzino' ('idmagazzino')
    ->     ON DELETE RESTRICT
    ->     ON UPDATE RESTRICT
    -> )
    -> ENGINE = InnoDB;
Query OK, 0 rows affected (0.16 sec)

mysql> CREATE TABLE 'bd2'.'analisi' (
    ->   'idprodotto' INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    ->   'venduto' INTEGER UNSIGNED NOT NULL DEFAULT 0,
    ->   'guadagno' FLOAT NOT NULL DEFAULT 0,
    ->   PRIMARY KEY('idprodotto'),

->   CONSTRAINT 'analisi_prod' FOREIGN KEY 'analisi_prod' ('idprodotto')
->     REFERENCES 'prodotto' ('idprodotto')
-> )
-> ENGINE = InnoDB;
```

I trigger

```
mysql> delimiter //
mysql> create function analisi_vendite (prodotto integer, qta integer)
    -> returns integer
    -> deterministic
    -> begin
    -> declare guad integer;
    -> select (vendita - acquisto) into guad from prodotto
    -> where idprodotto=prodotto;
    -> update analisi set venduto = venduto + qta, guadagno = guadagno
       + (guad * qta)
    -> WHERE idprodotto = prodotto;
    -> return 1;
    -> end;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

I trigger

```
mysql> create trigger PROD_MAG_AFT_UPD_ROW
    -> AFTER update OF qta ON prodotti_magazzino
    -> for each row
    -> begin
    -> declare nordini integer;
    -> declare qtavenduta integer;
    -> set qtavenduta = OLD.qta - NEW.qta;
    -> if  NEW.qta < NEW.qta_limite
    -> then
    -> SELECT COUNT(*) into nordini
    -> FROM ordine WHERE ordine.idprodotto = NEW.idprodotto
    -> AND ordine.idmagazzino = NEW.idmagazzino;
    -> if nordini = 0
    -> then
    -> insert into ordine values (NEW.idprodotto,NEW.idmagazzino,CURDATE());
    -> end if;
```

I trigger

```
    -> end if;
    -> analisi_vendite(NEW.idprodotto, qtavendita)
    -> end;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> create trigger PROD_MAG_AFT_INS_ROW
    -> AFTER insert ON prodotti_magazzino
    -> for each row
    -> begin
    -> if not exists (SELECT *
    ->     FROM analisi
    ->     WHERE idprodotto = NEW.idprodotto)
    -> then
    -> insert into analisi values (NEW.idprodotto,0,0);
    -> end if;
    -> end;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;
```

Esecuzione

- mysql> update prodotti_magazzino set qta = 9
- -> where idprodotto = 1 and idmagazzino = 2;

- Query OK, 0 rows affected (0.00 sec)
- Rows matched: 1 Changed: 0 Warnings: 0

- mysql> update prodotti_magazzino set qta = qta - 41
- -> where idprodotto = 3 and idmagazzino = 1;
- Query OK, 1 row affected (0.08 sec)

- Rows matched: 1 Changed: 1 Warnings: 0
- mysql> select * from prodotti_magazzino;

Risultati

```
+------------+--------------+-----+------------+---------------+
| idprodotto | idmagazzino | qta | qta_limite | qta_riordino |
+------------+--------------+-----+------------+---------------+
|          1 |            1 |  10 |          2 |            10 |
|          1 |            2 |   9 |         10 |            25 |
|          2 |            1 |   5 |          1 |             5 |
|          2 |            2 |  12 |          2 |            10 |
|          3 |            1 |   9 |         10 |            10 |
+------------+--------------+-----+------------+---------------+
5 rows in set (0.00 sec)

mysql> select * from ordine;
+------------+--------------+---------------------+
| idprodotto | idmagazzino | data                |
+------------+--------------+---------------------+
|          3 |            1 | 2006-01-13 00:00:00 |
|          1 |            2 | 2006-01-13 00:00:00 |
+------------+--------------+---------------------+
2 rows in set (0.00 sec)

mysql> select * from analisi;
+------------+----------+-----------+
| idprodotto | venduto | guadagno |
+------------+----------+-----------+
|          1 |        8 |       400 |
|          2 |        0 |         0 |
|          3 |       41 |      3280 |
+------------+----------+-----------+
3 rows in set (0.00 sec)
```