

Il linguaggio SQL: trigger

Sistemi Informativi T

Versione elettronica: [04.7.SQL.trigger.pdf](#)

DBMS attivi

- Un DBMS si dice **attivo** quando dispone di un sottosistema integrato per definire e gestire **regole**
- I **trigger** sono un caso specifico di regole di tipo **ECA** (**E**vento, **C**ondizione, **A**zione):

*Un trigger si attiva a fronte di un dato **evento** e,
se sussiste una data **condizione**, allora esegue una data **azione***

```
CREATE TRIGGER EmpSalaryTooHigh
AFTER UPDATE OF Salary ON Employee           -- evento
FOR EACH ROW MODE DB2SQL
WHEN (NEW.Salary >                             -- condizione
      SELECT Salary FROM Employee
      WHERE EmpCode = NEW.EmpManager)
UPDATE Employee                                -- azione
SET Salary = OLD.Salary
WHERE EmpCode = NEW.EmpCode
```

I componenti di un trigger

- Un trigger fa sempre riferimento a una singola tabella (**target**)
`AFTER UPDATE OF Salary ON Employee`
- **Evento**: istruzione SQL di manipolazione dati (**INSERT, DELETE, UPDATE**)
`AFTER UPDATE OF Salary ON Employee`
- **Condizione**: predicato Booleano
`NEW.Salary > SELECT Salary FROM Employee
WHERE EmpCode = NEW.EmpManager`
- **Azione**: sequenza di una o più istruzioni SQL
`UPDATE Employee
SET Salary = OLD.Salary
WHERE EmpCode = NEW.EmpCode`
- Se sono presenti più istruzioni, vengono eseguite in maniera **atomica**:
`BEGIN ATOMIC
<istruzione 1>; <istruzione 2>;
END`

Perché i trigger

- I trigger vengono tipicamente usati per **gestire vincoli di integrità**, **calcolare dati derivati**, **gestire eccezioni**, ecc.
- In tutti i casi è bene evitare che il compito di mantenere il DB in uno stato consistente sia lasciato al controllo delle singole applicazioni
- Mediante i trigger è il sistema che si fa carico di garantire la consistenza del DB, sollevando quindi le applicazioni da tale onere e, al tempo stesso, prevenendo rischi di inconsistenze dovuti ad applicazioni mal progettate

Trigger: attivazione e granularità

- Un trigger può attivarsi prima (**BEFORE**) o dopo (**AFTER**) l'evento corrispondente:
 - i "before trigger" vengono usati per "condizionare" l'esito dell'operazione oppure per bloccarla segnalando errore
 - Gli "after trigger" servono a "reagire" alla modifica del DB mediante opportune azioni
- Nel caso di eventi che coinvolgano più tuple della tabella target, un trigger può essere attivato:
 - Row trigger: per ognuna di queste tuple (**FOR EACH ROW**)
 - Ad es. per controllare che tutti gli update siano legali
 - Statement trigger: solo una volta per la data istruzione (**FOR EACH STATEMENT**), di fatto lavorando in modo "aggregato"
 - Ad es. per contare quanti inserimenti sono stati eseguiti

Variabili e tabelle di transizione

- Un trigger ha spesso necessità di fare riferimento allo **stato del DB prima e/o dopo l'evento** (ad es. per verificare che il nuovo stipendio non sia maggiore di più del 20% di quello vecchio)
- A tale scopo si possono usare **2 variabili** e **2 tabelle di transizione**:
 - **OLD**: valore della tupla prima della modifica
 - **NEW**: valore della tupla dopo la modifica
 - **OLD_TABLE**: una ipotetica table che contiene tutte le tuple modificate, con i valori prima della modifica
 - **NEW_TABLE**: idem, ma dopo la modifica
- Non tutti i riferimenti hanno senso per tutti i tipi di eventi:
 - **OLD** e **OLD_TABLE** non hanno senso in caso di **INSERT**
 - **NEW** e **NEW_TABLE** non hanno senso in caso di **DELETE**
- Tali nomi si possono modificare con la clausola **REFERENCING**:

REFERENCING NEW AS NewEmployee

NEW_TABLE AS NewEmpTable

Esempio di statement trigger

- Si vuole evitare che lo stipendio medio superi il valore di 1000

```
CREATE TRIGGER CheckSalariesUpdates
AFTER UPDATE OF Salary ON Employee
REFERENCING NEW_TABLE AS NewEmpTable
FOR EACH STATEMENT MODE DB2SQL
WHEN ((SELECT AVG(Salary) FROM Employee) > 1000)
UPDATE Employee
SET Salary = 0.9*Salary
WHERE EmpCode IN (SELECT EmpCode FROM NewEmpTable)
```

- Serve anche un trigger analogo per gli inserimenti

Before trigger (1)

- I trigger di questo tipo sono tipicamente usati per far rispettare dei **vincoli di integrità**
- Un before trigger è necessariamente un **row trigger**, e può usare solo le variabili **NEW** e **OLD**
- Inoltre non può includere azioni che modifichino il DB (quindi non può nemmeno attivare altri trigger: **NO CASCADE**), ad eccezione della tupla in esame

```
CREATE TRIGGER CheckEmpSalary
NO CASCADE BEFORE INSERT ON Employee
REFERENCING NEW AS NewEmp
FOR EACH ROW MODE DB2SQL
WHEN (NewEmp.Salary > (SELECT Salary From Employee
                        WHERE EmpCode = NewEmp.EmpManager))
  SIGNAL SQLSTATE '70000' ('Stipendio troppo elevato!')
```

- L'istruzione **SIGNAL** annulla gli effetti dell'evento che ha attivato il trigger

Before trigger (2)

- Le azioni di un before trigger possono essere:
 - **SELECT**
 - **SIGNAL**
 - **SET**: serve a modificare uno o più campi di una nuova tupla (**NEW**)
- Ad es. se si ha una business rule che impone di assegnare ad ogni nuovo impiegato lo stipendio minimo del suo dipartimento (e automaticamente anche il manager di quel dipartimento):

```
CREATE TRIGGER EmpMinSalary
NO CASCADE BEFORE INSERT ON Employee
REFERENCING NEW AS NewEmp
FOR EACH ROW MODE DB2SQL
SET NewEmp.Salary = (SELECT MIN(Salary) FROM Employee
                     WHERE Dept = NewEmp.Dept),
    NewEmp.EmpManager = (SELECT MgrCode From Department
                        WHERE DeptCode = NewEmp.Dept)
```

Before trigger (3)

- I before trigger possono essere efficacemente usati per far rispettare vincoli persi nella traduzione da schema concettuale a schema relazionale
- Ad es. per i **vincoli di mutua esclusione**: se le matricole degli studenti iscritti devono essere distinte da quelle degli studenti Erasmus:

```
CREATE TRIGGER DisjointFromErasmus
NO CASCADE BEFORE INSERT ON StudIscritti
REFERENCING NEW AS NewStud
FOR EACH ROW MODE DB2SQL
WHEN (EXISTS(SELECT * FROM StudErasmus E
              WHERE E.Matr = NewStud.Matr))
SIGNAL SQLSTATE '70000' ('Matricola già presente!')
```

- Analogo trigger per inserimenti in **StudErasmus**

After trigger

- Un after trigger può includere le seguenti azioni:
`SELECT, INSERT, DELETE, UPDATE, SIGNAL`
- Un tipico esempio d'uso riguarda il **calcolo di dati derivati**:

```
FATTURE(NumFattura,Data,Importo)
```

```
VENDITE(NumFattura,CodProd,Quantita)
```

```
GIACENZE(CodProd,Qtaresidua,ScortaMinima)
```

```
-- aggiorna la quantità residua di tutti i prodotti venduti
```

```
-- inseriti in una nuova fattura
```

```
CREATE TRIGGER UpdateQtaResidua
```

```
AFTER INSERT ON Vendite
```

```
REFERENCING NEW AS NuovaVendita
```

```
FOR EACH ROW MODE DB2SQL
```

```
UPDATE Giacenze
```

```
SET QtaResidua = Qtaresidua - NuovaVendita.Quantita
```

```
WHERE CodProd = NuovaVendita.CodProd
```

Trigger: attivazioni in cascata

- In generale ragionare con i trigger è complesso, in quanto si possono anche avere **attivazioni in cascata**:

```
GIACENZE(CodProd,Qtaresidua,ScortaMinima)  
CARENZE(CodProd,QtaDaOrdinare)
```

```
-- se la giacenza di un prodotto è minore del minimo  
-- provvede a inserire una tupla in CARENZE  
CREATE TRIGGER InsertQtaDaOrdinare  
AFTER UPDATE ON Giacenze  
REFERENCING NEW AS NG  
FOR EACH ROW MODE DB2SQL  
WHEN (NG.QtaResidua < NG.ScortaMinima)  
INSERT INTO CARENZE  
VALUES (NG.CodProd,NG.ScortaMinima-NG.QtaResidua)
```

Trigger: cicli

- Quando si ha attivazione in cascata è possibile avere anche **cicli infiniti**:

```
CREATE TRIGGER T1
AFTER INSERT INTO R
FOR EACH ROW MODE DB2SQL
DELETE FROM R
WHERE id = NEW.id
```

```
CREATE TRIGGER T2
AFTER DELETE FROM R
FOR EACH ROW MODE DB2SQL
INSERT INTO R
VALUES (OLD.id, ...)
```

- I cicli si possono presentare **anche con un solo trigger** (ricorsivo, in quanto ri-attiva se stesso):

```
CREATE TRIGGER CheckSalariesWrong
AFTER UPDATE OF Salary ON Employee
WHEN ((SELECT AVG(Salary) FROM Employee) > 1000)
UPDATE Employee
SET Salary = 1.1*Salary
```

SQL: trigger

Costrutto IF ... THEN ...[ELSE...] END IF

- Un'azione può essere eseguita solo se sussiste una data condizione, ad es:

```
CREATE TRIGGER DisjointFromErasmus
NO CASCADE BEFORE INSERT ON StudIscritti
REFERENCING NEW AS NewStud
FOR EACH ROW MODE DB2SQL
IF (EXISTS(SELECT * FROM StudErasmus E
           WHERE E.Matr = NewStud.Matr))
THEN SIGNAL SQLSTATE '70000' ('Matricola già presente!');
END IF
```

- La condizione può ovviamente anche inserirsi nella clausola WHEN
- Con l'IF si può spezzare una condizione composta in due parti:
 - parte "semplice" da valutare per il DBMS nel WHEN
 - parte "complessa" nell'IF
- Inoltre si può diversificare l'azione se si usa anche ELSE

Esempio di azioni alternative

- Il seguente trigger o segnala errore oppure provvede a completare la tupla dell'impiegato inserito:

```
CREATE TRIGGER DUEAZIONI
NO CASCADE BEFORE INSERT ON Employee
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
IF (N.CodImp > 100) THEN
    SIGNAL SQLSTATE '80000' ('Codice invalido');
ELSE SET N.Salary = (SELECT MIN(Salary) FROM Employee
                     WHERE Dept = N.Dept);
END IF
```

- Notare i punti e virgola...

Attivazione di più trigger in DB2

- In DB2, se un evento attiva più trigger, questi vengono eseguiti rispettando l'ordine con cui sono stati definiti
- Se un trigger T1 attiva un trigger T2, DB2 sospende l'esecuzione di T1 e inizia a eseguire T2, il quale “vede” già gli eventuali effetti prodotti da T1
- Nel caso di attivazioni ricorsive, DB2 pone un limite massimo di 16 livelli di ricorsione