Trigger

DBMS passivi vs. attivi

- I DBMS tradizionali sono **passivi**
  - eseguono delle operazioni solo su richiesta
- Spesso si ha la necessità di avere capacità **reattive**
  - il DBMS reagisce ad eventi eseguendo operazioni definite dal progettista
- DBMS con queste funzionalità sono conosciuti come **DBMS attivi (ADBMS)**
- Negli ADBMS si possono definire *regole attive* o *trigger*

DBMS attivi

- I trigger forniscono (in maniera reattiva) funzionalità altrimenti delegate ai programmi applicativi
- il comportamento reattivo è definito centralmente una sola volta
  - ed è condiviso da tutte le applicazioni che usano il DB
- Benefici in termini di
  - efficienza
  - costi di manutenzione
  - uniformità di gestione dei dati (quindi loro consistenza)
  - integrazione con le altre componenti del DBMS

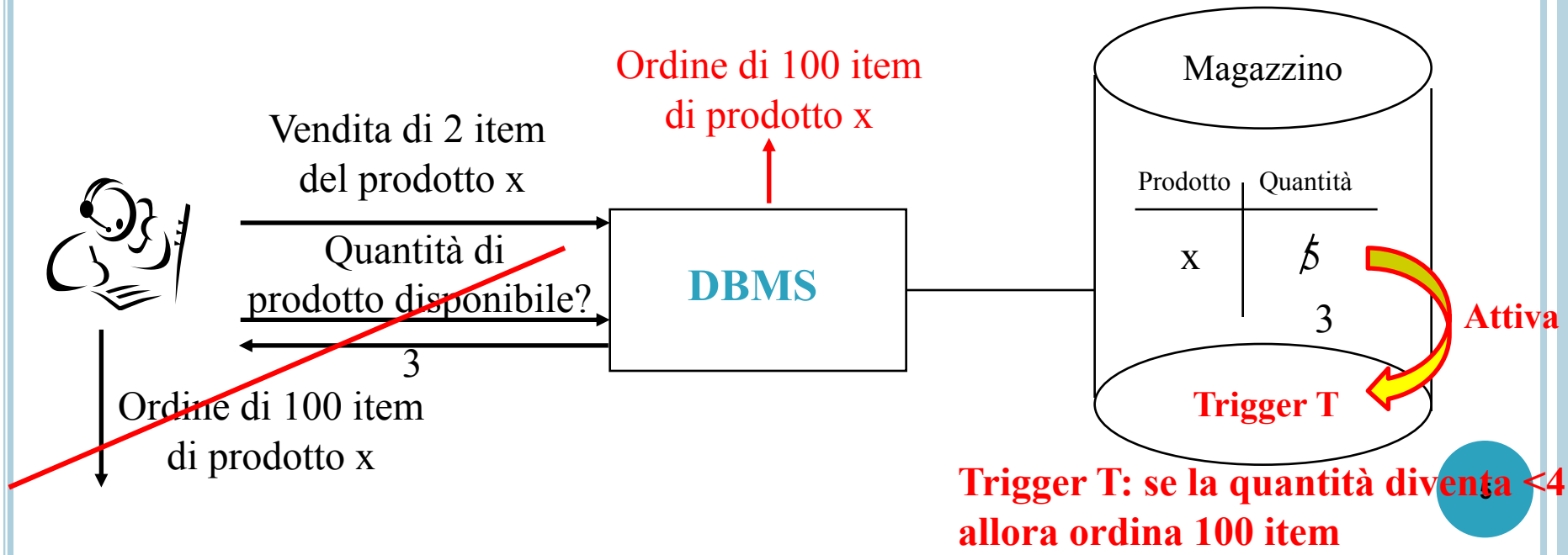
DBMS attivi

- Iniziano ad affermarsi a partire dagli anni ‘90
- I maggiori DBMS commerciali (Oracle, IBM DB2, Microsoft SQL Server) sono stati estesi con la possibilità di specificare trigger
- A partire da SQL:1999 anche lo standard recepisce tali estensioni
- Attualmente i DBMS non sono completamente allineati allo standard attuale

Trigger in PostgreSQL abbastanza diversi da quanto prescrive lo standard

TRIGGER: ESEMPIO

- Gestione automatizzata di un magazzino in cui se la quantità di un prodotto scende sotto le 4 unità devo ordinare 100 item di tale prodotto
- DBMS tradizionale attivo

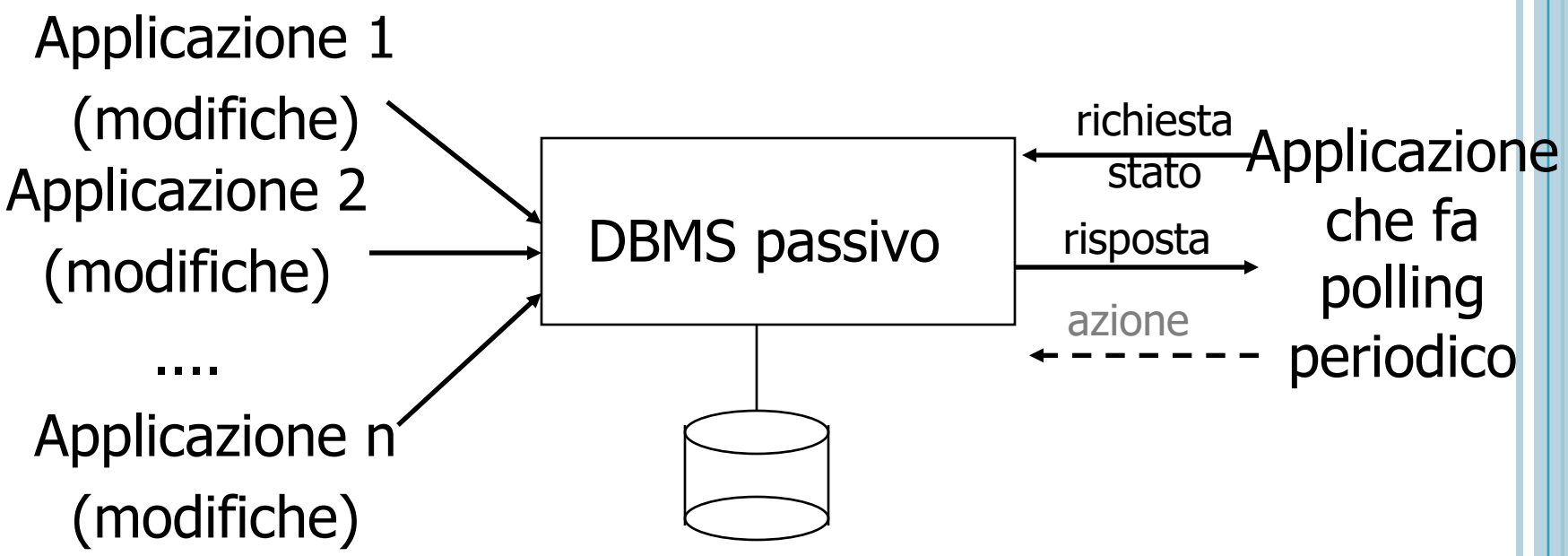


DBMS attivi - Usi

- Monitoraggio (come nella slide precedente)
- Vincoli di integrità
- Alerting
- Auditing
- Sicurezza
- Statistiche
- Eccezioni

Approcci architetturali

DBMS passivi: approccio 1

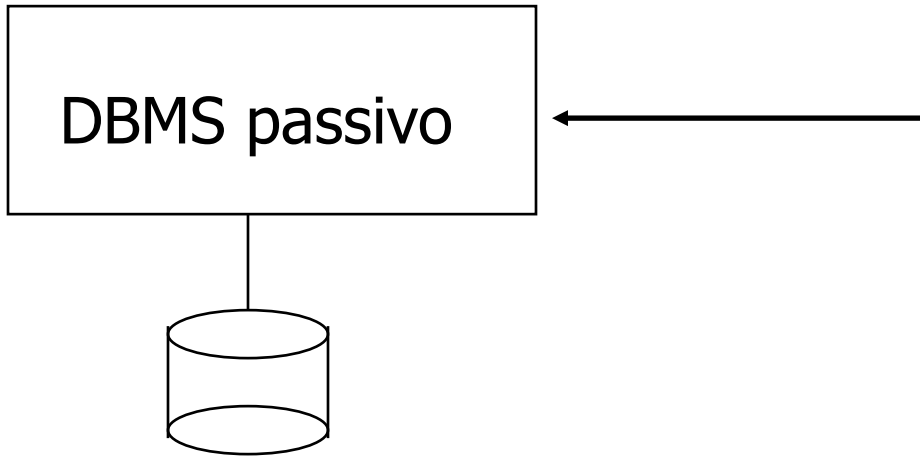


Problemi

- minore efficienza (controlla anche se non è cambiato nulla)
- determinare la frequenza ottima di polling

Approcci architetturali

DBMS passivi: approccio 2

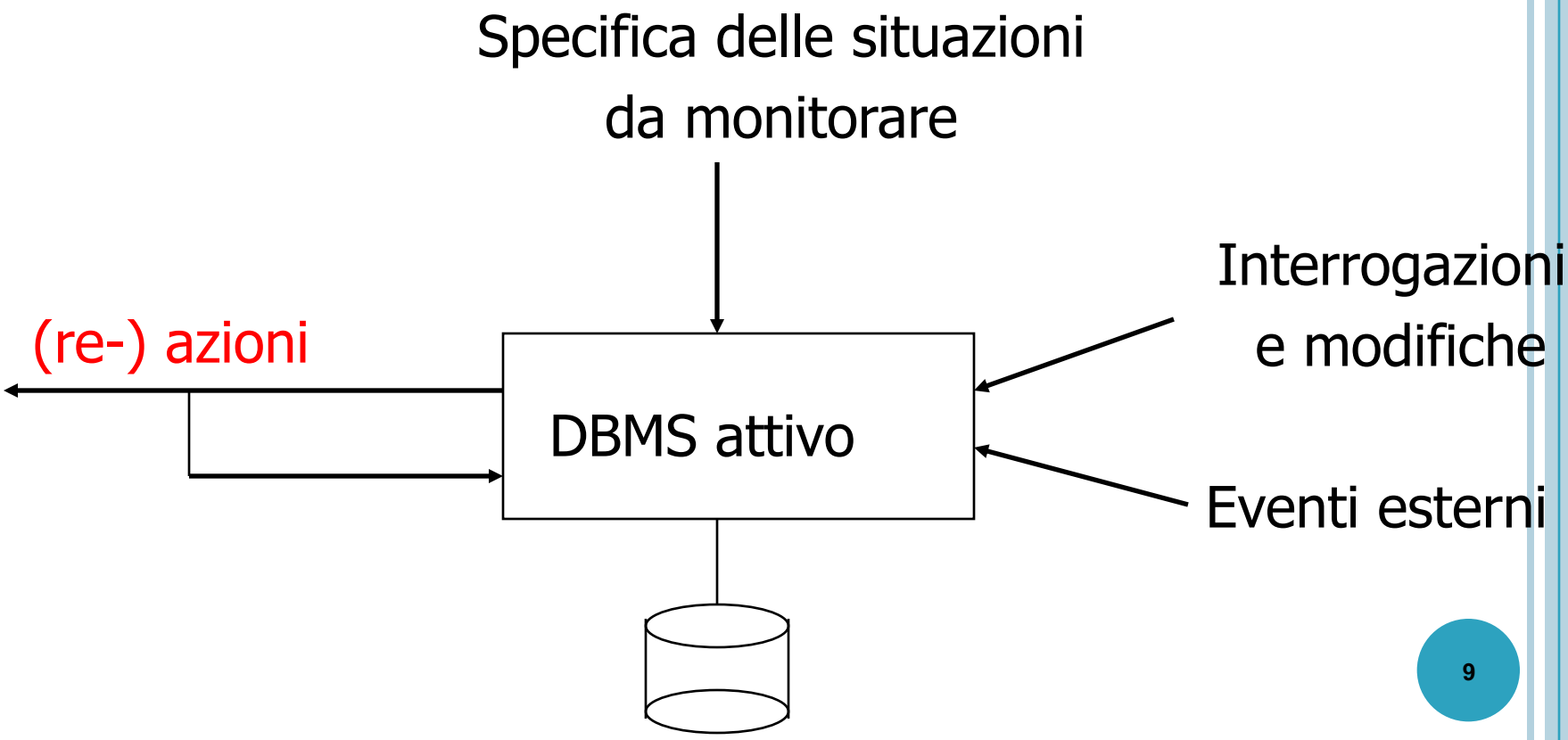


Problemi

- Compromette la modularità e la riusabilità del codice
- La correttezza di ciascuna applicazione dipende dalla correttezza e integrazione di tutte

Approcci architetturali

DBMS attivi



TRIGGER

- Alcune operazioni sono automaticamente eseguite quando si verifica una determinata ***situazione*** interna o esterna alla base di dati
- La situazione può corrispondere a
  - eventi specifici (insert, update, ecc.)
  - particolari condizioni o particolari stati o transizioni di stato
- Un **trigger** (**o regola attiva**)
  - è il costrutto sintattico per definire la reazione del sistema
  - è specificato nel DDL del DBMS

Paradigma ECA

- Paradigma più noto per la definizione dei trigger è Evento-Condizione-Azione (ECA):

ON ***evento***

IF ***condizione***

THEN ***azione***

1. Al verificarsi dell'evento si valuta la condizione
2. Se la condizione è soddisfatta si esegue l'azione

Paradigma ECA

- Paradigma più noto per la definizione dei trigger è Evento-Condizione-Azione (ECA):

ON ***evento***

IF ***condizione***

THEN ***azione***

Un evento è qualcosa che accade, che è di interesse (per la definizione delle regole) e che può essere mappato, dal punto di vista del sistema, in un istante di tempo

Es. più comuni: INSERT, DELETE, UPDATE di relazioni

1. Al verificarsi dell'evento si valuta la condizione
2. Se la condizione è soddisfatta si esegue l'azione

Paradigma ECA

- Paradigma più noto per la definizione dei trigger è Evento-Condizione-Azione (ECA):

ON ***evento***

IF ***condizione***

THEN ***azione***

Una condizione è un ulteriore controllo che viene eseguito quando il trigger è considerato e prima che l'azione sia eseguita

Predicato SQL (clausola WHERE)

1. Al verificarsi dell'evento si valuta la condizione
2. Se la condizione è soddisfatta si esegue l'azione

Paradigma ECA

- Paradigma più noto per la definizione dei trigger è Evento-Condizione-Azione (ECA):

ON ***evento***

IF ***condizione***

THEN ***azione***

Una condizione è un ulteriore controllo che viene eseguito quando il trigger è considerato e prima che l'azione sia eseguita

Predicato SQL (clausola WHERE)

1. Al verificarsi dell'evento si valuta la condizione
2. Se la condizione è soddisfatta si esegue l'azione

Paradigma ECA

- Paradigma più noto per la definizione dei trigger è Evento-Condizione-Azione (ECA):

ON ***evento***

IF ***condizione***

THEN ***azione***

Un'azione è una sequenza di operazioni che viene eseguita quando il trigger è considerato e la sua condizione è vera

Es. aggiornamenti, invocazioni di procedure, rollback

1. Al verificarsi dell'evento si valuta la condizione
2. Se la condizione è soddisfatta si esegue l'azione

Eventi

- Possibilità di definire trigger che possono essere attivati *before* o *after* un evento
  - BEFORE: il trigger viene eseguito prima di eseguire l'evento
    - utile per trigger che verificano pre-condizioni
    - se non sono vere ⇒ abort, si previene l'esecuzione dell'evento
  - AFTER: il trigger viene eseguito dopo l'esecuzione dell'evento
    - utile per mantenere consistenti i dati, monitoraggio, alerting etc.

- Possibilità di combinare gli eventi (*eventi composti*) tramite, ad esempio:
  - *Operatori logici*: and, or, ecc.
  - *Sequenza*: seleziono un trigger se due o più eventi accadono in un certo ordine

Evento Condizione Azione - Esempio

- Relazione Film
- Trigger T: descrizione informale
  - assegna un valore di default all'attributo valutaz, se questo non è specificato al momento dell'inserimento della tupla
  - il valore di valutaz è posto uguale alla media delle valutazioni, calcolata su tutte le tuple presenti nella relazione Film, aumentata del 10%
- Trigger T: descrizione ECA
  - Evento: INSERT INTO Film
  - Condizione: valutaz IS NULL
  - Azione: assegnamento del valor medio di valutaz moltiplicato per 1.1 all'attributo valutaz delle tuple inserite

Evento Condizione Azione

- Perché è vantaggioso avere l'evento?
  - valutare una condizione è costoso
  - rilevare l'accadere di un evento è immediato
- Inoltre, si possono specificare azioni diverse per eventi diversi e stessa condizione

Azioni

- Le azioni ammesse in un trigger dipendono in genere dalla modalità BEFORE/AFTER con cui è stato specificato l'evento
- Le azioni dei trigger BEFORE sono soggette a varie limitazioni, spesso è ammesso solo ROLLBACK

CREAZIONE TRIGGER IN SQL:200N

CREATE TRIGGER <nome trigger>
{BEFORE | AFTER} <evento> ON <tabella soggetto>
[WHEN <condizione>]
{<comando SQL> |
BEGIN ATOMIC <sequenza di comandi SQL> END} ;

SQL:200N - EVENTO

- Possibili eventi: INSERT, DELETE, UPDATE, UPDATE [OF <lista attributi>] per la tabella soggetto
- Se si specifica UPDATE OF a1,…,an, il trigger viene attivato solo da un evento che modifica **tutti** e **soli** gli attributi a1,…,an
- **Un solo** evento può attivare un trigger, quindi non sono possibili eventi composti
- È possibile specificare che il trigger sia attivato prima (before) o dopo (after) l'esecuzione dell'operazione associata all'evento

SQL: 200N - CONDIZIONE ED AZIONE

- **Condizione**
  - Espressione booleana SQL arbitraria
- **Azione**
  - Un singolo comando SQL
  - Una sequenza di comandi SQL
  - Non possono contenere parametri di connessione
  - Nel caso di trigger di tipo BEFORE, SQL sconsiglia l'esecuzione di comandi di aggiornamento dei dati nel contesto dell'azione, ma non lo vieta
  - Un trigger di tipo BEFORE potrebbe aggiornare alcuni dati prima dell'esecuzione dell'evento che ha attivato il trigger, generando comportamenti anomali
    - soprattutto se vi sono più trigger BEFORE per lo stesso evento

SQL:200N - MODALITÀ DI ESECUZIONE

Due modalità

- ***Orientata all'istanza*** *(instance oriented):* il trigger viene eseguito una volta **per ogni** tupla coinvolta nell'evento che attiva il trigger e soddisfa la condizione
  - FOR EACH ROW
- ***Orientata all'insieme*** *(set oriented):* il trigger viene eseguito una sola volta per tutte le tuple coinvolte nell'evento
  - FOR EACH STATEMENT

- Possono esserci differenze nel risultato

SQL:200N – MODALITÀ DI ESECUZIONE

- **Esecuzione orientata all'insieme**: valutazione condizione ed esecuzione azione vengono eseguiti una sola volta, per l'insieme di tuple coinvolte nell'evento che ha attivato il trigger
  - L'insieme delle tuple aggiornate dall'evento viene chiamato **tabella di transizione**
- **Esecuzione orientata all'istanza**: valutazione condizione ed esecuzione azione vengono eseguiti una volta per ogni tupla coinvolta nell'evento che ha attivato il trigger
  - La tupla coinvolta nell'evento viene chiamata **variabile** o **tupla di transizione**

Tuple/abelle di transizione

- La tupla/***l'insieme*** delle tuple che sono state modificate nell'operazione che ha attivato il trigger ("versione" prima e dopo la modifica)
  - OLD o OLD ROW (equivalenti)
  - NEW o NEW ROW (equivalenti)
  - ***OLD TABLE***
  - ***NEW TABLE***
- Si possono usare nella condizione e/o nell'azione
- È possibile assegnare alle tabelle/tuple di transizione un alias

CREAZIONE TRIGGER (SINTASSI PIÙ COMPLETA)

CREATE TRIGGER <nome trigger>
{BEFORE | AFTER} <evento> ON <tabella soggetto>
[**REFERENCING** { **OLD [ROW] AS** <variabile> |
NEW [ROW] AS <variabile> |
OLD TABLE AS <variabile> |
NEW TABLE AS <variabile> }]
[**FOR EACH {ROW | STATEMENT}**]
[WHEN <condizione>]
{<comando SQL> |
BEGIN ATOMIC <sequenza di comandi SQL> END} ;

Clausola REFERENCING

- Con la clausola REFERENCING si specificano alias a livello di tabella o tupla di transizione
- La parola chiave **OLD**/**NEW** specifica alias per la tabella/tupla di transizione **prima**/**dopo** dell'esecuzione dell'evento

Esempio

- Trigger T
  - Evento: inserimento nella relazione Film
  - Condizione: valutaz IS NULL
  - Azione: calcolo del valor medio di valutaz ed assegnazione di tale valore moltiplicato per 1.1 all'attributo valutaz delle tuple inserite
- Operazione scatenante: Comando SQL che inserisce 5 tuple in Film

Esempio

- **Esecuzione orientata all'insieme**
  - La condizione viene valutata e l'azione viene eseguita una sola volta, indipendentemente dal numero di film inseriti
  - tutti i 5 film inseriti avranno lo stesso valore per l'attributo valutaz
- **Esecuzione orientata all'istanza**
  - La condizione viene valutata e l'azione viene eseguita una volta per ogni film inserito (quindi 5 volte)
  - i 5 film inseriti avranno valori dell'attributo valutaz potenzialmente diversi
    - la media è calcolata su insiemi differenti di valori

Esempio

```
CREATE TRIGGER ModificaValNull
AFTER INSERT ON Film
REFERENCING NEW TABLE AS NT
FOR EACH STATEMENT
WHEN EXISTS(SELECT *
                    FROM NT
                    WHERE valutaz IS NULL)
UPDATE Film
SET valutaz = (SELECT AVG(valutaz)*1.1 FROM Film)
WHERE (titolo,regista) IN (SELECT titolo,regista FROM NT)
            AND valutaz IS NULL;
```

Esempio

```
CREATE TRIGGER ModificaValNull
AFTER INSERT ON Film
REFERENCING NEW ROW AS NR
FOR EACH ROW
WHEN (NR.valutaz IS NULL)
UPDATE Film
SET valutaz = (SELECT AVG(valutaz)*1.1 FROM Film)
WHERE titolo = NR.titolo AND regista = NR.regista;
```

Il risultato dei due trigger può essere diverso

Visibilità di tuple/tabelle di transizione

- Quali tuple sono visibili durante la valutazione della condizione e l'esecuzione dell'azione?
- Dipende:
  - Dal tipo di trigger (before/after)
  - Dal tipo di esecuzione (row/statement)
  - Dall'evento che ha attivato il trigger

Visibilità di tuple/tabelle di transizione Rispetto al modo di esecuzione

tupla su cui l'evento dovrà essere eseguito

	FOR EACH ROW		**FOR EACH STATEMENT**	
BEFORE	tuple sì	tabelle no	tuple no	tabelle no
AFTER	tuple sì	tabelle sì	tuple no	tabelle sì

Si esegue il trigger **prima** di completare l'esecuzione dell'evento

$\Rightarrow$

la tabella di transizione non esiste ancora

Visibilità di tuple/tabelle di transizione Evento = INSERT

- Non si possono specificare clausole REFERENCING OLD
  - le tuple inserite dall'evento non esistevano prima della sua esecuzione
- Se il trigger è di tipo BEFORE le tuple inserite
  - non sono visibili nella tabella soggetto
  - ma possono essere accedute una alla volta usando la tupla di transizione NEW
- Se il trigger è di tipo AFTER le tuple inserite
  - sono visibili nella tabella soggetto
  - e possono essere accedute mediante la tupla o la tabella di transizione NEW

Visibilità di tuple/tabelle di transizione Evento = DELETE

- Non si possono specificare clausole REFERENCING NEW
  - le tuple cancellate dall'evento non esistono più dopo la sua esecuzione
- Se il trigger è di tipo BEFORE le tuple cancellate
  - sono visibili nella tabella soggetto
  - e possono essere accedute usando la tupla di transizione OLD
- Se il trigger è di tipo AFTER le tuple cancellate
  - non sono visibili nella tabella soggetto
  - ma possono essere accedute usando la tupla o la tabella di transizione OLD

Visibilità di tuple/tabelle di transizione Evento = UPDATE

- I valori precedenti e correnti delle tuple possono essere acceduti usando le clausole REFERENCING OLD e NEW
  - a livello di tupla nei trigger di tipo BEFORE
  - a livello di tupla o di tabella nei trigger di tipo AFTER
- Se il trigger è di tipo AFTER l'effetto della modifica è visibile anche nella tabella soggetto

Visibilità di tuple/tabelle di transizione Riassunto

Tipo trigger e modalità esecuzione	Evento	Tabelle/tuple di transizione
BEFORE ROW	INSERT DELETE UPDATE	NEW OLD NEW, OLD
AFTER ROW	INSERT DELETE UPDATE	NEW, NEW TABLE OLD, OLD TABLE TUTTE
BEFORE STATEMENT	INSERT DELETE UPDATE	- - -
AFTER STATEMENT	INSERT DELETE UPDATE	NEW TABLE OLD TABLE NEW TABLE, OLD TABLE

Modello di esecuzione

Attività fondamentali in un ADBMS:

1. Rilevare gli eventi ed attivare i trigger corrispondenti
2. Processo reattivo: selezionare ed eseguire i trigger
   - In base allo standard, viene attivato dopo l'esecuzione di ogni comando SQL

Possono essere eseguite concorrentemente

attività 1

While true **do**

seleziona eventi

attiva i trigger appropriati

endWhile

attività 2

While ci sono trigger da considerare

Do

(1) **seleziona** un trigger T

(2) **valuta** la condizione di T

(3) **If** la condizione di T è vera **Then**

esegui l'azione di T **endIf**

endWhile

Scelta di uno dei trigger attivati dall'evento

Verifica condizione ed esecuzione sequenziale delle operazioni nell'azione

Valutazione della condizione del trigger selezionato, il trigger viene eliminato dall'insieme dei trigger attivati

Selezione trigger

- Il trigger da eseguire viene selezionato sulla base di
  - tipo di trigger (before/after)
  - modalità di esecuzione (row/statement)
  - Priorità
    - lo standard assegna priorità assolute in base al tempo di creazione
      - un trigger “vecchio” è eseguito prima di un trigger “giovane”
    - PostgreSQL assegna priorità assolute in base al nome
      - i trigger sono selezionati in ordine alfabetico

Selezione trigger

- Possono esistere vincoli specificati per la tabella soggetto
- L'esecuzione degli eventi può violare i vincoli
  - nel determinare l'ordine di esecuzione è necessario considerare anche il controllo dei vincoli

Selezione trigger

I trigger vengono selezionati secondo il seguente ordine:

- Trigger BEFORE, FOR EACH STATEMENT
- Per ogni tupla oggetto del comando che rappresenta l'evento:
  - Trigger BEFORE, FOR EACH ROW:
    - Esecuzione evento per la singola tupla
    - Verifica dei vincoli di integrità sulla tupla, con valutazione immediata
  - Trigger AFTER, FOR EACH ROW
- Verifica dei vincoli con valutazione immediata sulla tabella
- Trigger AFTER, FOR EACH STATEMENT

Esecuzioni in cascata

- L'esecuzione dell'azione di un trigger può provocare nuovi eventi
  - questi possono a loro volta attivare altri trigger
- Tali trigger possono
  - essere aggiunti all'insieme di trigger da considerare (modalità iterativa)
  - dare origine ad una nuova esecuzione dell'algoritmo durante l'esecuzione dell'azione del trigger correntemente attivato (modalità ricorsiva)
    - Assunta dallo standard SQL

Terminazione

- Il processo reattivo potrebbe non terminare
- Lo standard non fornisce indicazioni su questo aspetto
- Non vengono poste restrizioni sintattiche per evitare la non terminazione
- Di solito i DBMS hanno un limite superiore al numero di trigger attivabili ricorsivamente

Cancellazione trigger

- DROP TRIGGER <nome trigger>;

Trigger: esempi di utilizzo

Trigger e vincoli

- I trigger sono più flessibili dei vincoli di integrità
  - permettono di stabilire come reagire ad una violazione di un vincolo
- I trigger possono specificare anche vincoli di transizione
- La flessibilità non sempre è un vantaggio
- A volte definire dei vincoli è più vantaggioso:
  - Migliore ottimizzazione
  - Meno errori di programmazione
  - I vincoli sono parte dello standard da lungo tempo, i trigger no

Vincoli di integrità

Ogni cliente non può noleggiare più di tre video contemporaneamente

```
CREATE ASSERTION
VerificaNoleggi
CHECK (NOT EXISTS
  (SELECT * FROM Noleggio
  WHERE dataRest IS NULL
  GROUP BY codCli
  HAVING COUNT(*) > 3));
```

```
CREATE TRIGGER
VerificaNoleggi
AFTER INSERT ON Noleggio
REFERENCING NEW ROW AS NR
FOR EACH ROW
WHEN (SELECT COUNT(*)
  FROM Noleggio
  WHERE dataRest IS NULL AND
                codCli = NR.codCli) > 3
ROLLBACK;
```

VINCOLI DI INTEGRITÀ

Ogni cliente non può noleggiare più di tre video contemporaneamente

Invece di abortire la transazione se il vincolo è violato si vuole annullare l'inserimento

```
CREATE TRIGGER
VerificaNoleggi
AFTER INSERT ON Noleggio
REFERENCING NEW ROW AS NR
FOR EACH ROW
WHEN (SELECT COUNT(*)
  FROM Noleggio
  WHERE dataRest IS NULL AND
                codCli = NR.codCli) > 3
DELETE FROM Noleggio
WHERE colloc = NR.colloc AND
                dataNol = NR.dataNol;
```

```
CREATE ASSERTION
VerificaNoleggi
CHECK (NOT EXISTS
  (SELECT * FROM Noleggio
  WHERE dataRest IS NULL
  GROUP BY codCli
  HAVING COUNT(*) > 3));
```

VINCOLI DI INTEGRITÀ

Ogni cliente non può noleggiare più di tre video contemporaneamente

Se un noleggio causa la violazione del vincolo deve essere impedito

Tutto il noleggio non solo i video eccedenti

```
CREATE TRIGGER VerificaNoleggi
AFTER INSERT ON Noleggio
REFERENCING NEW TABLE AS NT
FOR EACH STATEMENT
WHEN EXISTS
        (SELECT *
         FROM Noleggio
         WHERE Noleggio.dataRest IS NULL AND
                  Noleggio.codCli IN (SELECT codCli FROM NT)
         GROUP BY codCli
         HAVING COUNT(*) > 3)
DELETE FROM Noleggio
WHERE (colloc,dataNol) IN  (SELECT colloc, dataNol FROM NT)
AND codCli IN (SELECT codCli
                  FROM Noleggio
                  WHERE Noleggio.dataRest IS NULL AND
                           Noleggio.codCli IN (SELECT codCli FROM NT)
                  GROUP BY codCli
                  HAVING COUNT(*) > 3);
```

Calcolo di dati derivati

Aggiornamento automatico attributo ptiMancanti nella tabella Standard ad ogni nuovo noleggio:

```
CREATE TRIGGER CalcolaPtiMancanti
AFTER INSERT ON Noleggio
REFERENCING NEW ROW AS NR
FOR EACH ROW
BEGIN ATOMIC
UPDATE Standard
SET ptiMancanti = ptiMancanti - 1
WHERE codCli = NR.codCLI AND
    NR.colloc IN (SELECT colloc FROM Video
                  WHERE tipo = 'v');
UPDATE Standard
SET ptiMancanti = ptiMancanti - 2
WHERE codCli = NR.codCLI AND
    NR.colloc IN (SELECT colloc FROM Video
                  WHERE tipo = 'd');
END;
```

ogni VHS 1 punto

ogni DVD 2 punti

Regole operative

Quando il valore dell'attributo ptiMancanti per un cliente standard diventa 0, il cliente è rimosso dalla tabella Standard ed inserito nella tabella VIP con bonus pari a 5 euro

```
CREATE TRIGGER OrganizzaClienti
AFTER UPDATE OF ptiMancanti ON Standard
REFERENCING NEW ROW AS NR
FOR EACH ROW
WHEN NR.ptiMancanti <= 0
BEGIN ATOMIC
        INSERT INTO VIP
        VALUES (NR.codCli,5.00);
        DELETE FROM Standard
        WHERE codCli = NR.codCli;
END;
```