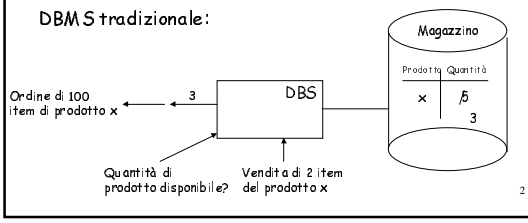
Trigger - Preliminari

- I DBMS tradizionali sono **passivi**: eseguono delle operazioni solo su richiesta
- spesso si ha la necessità di avere capacità **reattive**: il DBMS reagisce autonomamente ad alcuni eventi ed esegue determinate operazioni
- in questo ultimo caso parleremo di **DBMS attivi**, per cui è possibile definire *regole attive* o *trigger*
- esempi di funzioni per cui i trigger sono utili:
  - gestione di vincoli di integrità:
    - per fallimento
    - per modifica
  - auditing:
    - sicurezza
    - statistiche
  - valori derivati memorizzati

Trigger - Preliminari

Esempio: gestione automatizzata di un magazzino in cui se la quantità di un prodotto scende sotto 4 devo ordinare 100 item di tale prodotto

DBMS tradizionale:

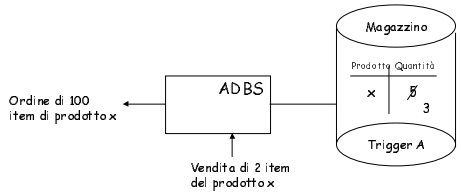


Trigger - Preliminari

Esempio continua:

DBMS attivo:

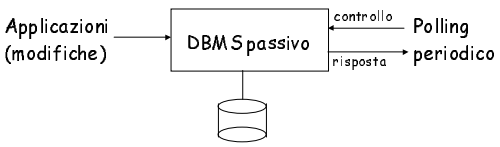
Trigger A: se la quantità diventa <=4 allora ordina 100 item



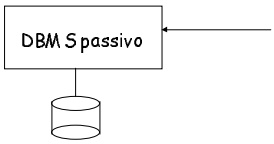
Trigger - Preliminari

- Questo è un esempio di uso delle regole attive per monitoraggio
- altri esempi:
  - vincoli di integrità
  - alerting
  - specifica di azioni consequenziali
  - mantenimento valori derivati
- in generale noi useremo i trigger per realizzare le business rule della nostra applicazione

Approcci architetturali

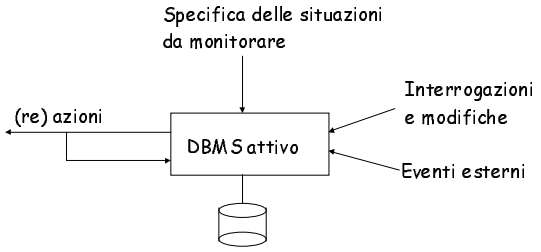


Approcci architetturali



Applicazioni estese con codice per il monitoraggio di situazioni

Approcci architetturali



7

Specifica di trigger

- una *base di dati attiva* è una base di dati nella quale alcune operazioni sono automaticamente eseguite quando si verifica una determinata situazione
- la situazione può corrispondere al fatto che si verifichino eventi specifici, o che siano riscontrati particolari condizioni o particolari stati o transizioni di stato
- una *regola attiva (trigger)* è un costrutto sintattico per definire la reazione del sistema
- il paradigma più noto per la definizione dei trigger è quello

  evento-condizione-azione (ECA)
  - *evento*: se si verifica provoca l'attivazione del trigger
  - *condizione*: se è soddisfatta l'azione del trigger è eseguita
  - *azione*: sequenza di operazioni che può anche modificare la base dati, viene eseguita solo se la condizione è vera

8

Specifica di trigger

- La forma più comune di trigger è quindi:

  ON *evento* IF *condizione* THEN *azione*

  se si verifica l'evento, la condizione è valutata, se la condizione è soddisfatta l'azione viene eseguita
- le regole attive hanno origine da quelle dell'*intelligenza artificiale*
- di solito, però, queste non hanno eventi, ovvero sono della forma IF *condizione* THEN *azione*
- perché è vantaggioso avere l'evento?
  - la condizione è costosa (in termini di efficienza) da valutare, mentre rilevare l'accadere di un evento è molto meno complesso
  - questo problema è ancora più sentito in ambito basi di dati in cui ho grosse moli di dati
  - inoltre, posso specificare azioni diverse per eventi diversi e stessa condizione

9

Eventi & Condizioni

- Un *evento* è qualcosa che accade, o si verifica, che è di interesse e che può essere mappato in un istante di tempo
  - in Oracle: modifica dei dati inserimento, cancellazione, modifica
  - possibilità di definire trigger che possono essere attivati *before* o *after* un evento
  - possibilità di specificare OR di eventi
- Una *condizione* è un ulteriore controllo che viene eseguito quando la regola è considerata e prima che l'azione sia eseguita
  - in Oracle è espressa come un predicato: clausola where di SQL (con alcune limitazioni)
  - se la condizione non c'è si assume vera

10

Azioni

- Un'*azione* è una sequenza di operazioni che viene eseguita quando la regola è considerata e a sua condizione è vera
  - in Oracle l'azione è codice PL/SQL arbitrario
  - l'azione e la condizione possono far riferimento allo stato attuale e precedente dei dati modificati dall'evento
  - sarebbe utile (ma non è possibile in Oracle) passaggio di parametri tra condizione e azione

11

Modello di esecuzione

Due attività fondamentali in un ADBMS:
- rilevare gli eventi
- processo reattivo: selezionare ed eseguire le regole

Algoritmo del processo reattivo:

While ci sono regole da considerare **Do**
(1) trova una regola *R* da considerare (*selezione*)
(2) valuta la condizione di *R (considerazione)*
(3) **If** la condizione di *R* è vera **Then**
esegui l'azione di *R (esecuzione)* **endIf**
endWhile

12

Modello di esecuzione

Granularità del processo reattivo: frequenza di attivazione del processo

Esempi:
- alla fine di un intero comando SQL in cui, per esempio, modifico tante tuple (FOR EACH STATEMENT in Oracle)
- alla fine della modifica di ogni tupla da parte di un comando SQL, più fine della precedente (FOR EACH ROW in Oracle)
- alla fine della transazione, più grossolana, utile per vincoli che possono essere violati in stati intermedi (non supportata in Oracle)

13

Esecuzione delle regole

Due modalità:

- *orientata all'istanza (instance oriented):* la regola è eseguita per ogni elemento della base di dati che attiva la regola e soddisfa la condizione
- *orientata all'insieme (set oriented):* la regola è eseguita una volta per l'insieme di tali elementi

14

Soluzione dei conflitti

Come scegliere una regola fra un insieme di regole attivate?

- arbitrariamente
- priorità
- proprietà statistiche (per es., momento della creazione)
- proprietà dinamiche (per es., regola attivata più di recente)
- alternativa: valutare più regole concorrentemente

15

Modalità di accoppiamento

Sono anche dette *coupling modes* e stabiliscono quando valutare una parte di una regola rispetto ad un'altra

Esempio: modalità di accoppiamento
- tra evento e condizione
- tra condizione e azione

Inoltre è possibile stabilire se l'azione è:
- estensione della transazione corrente
- in una nuova transazione

16

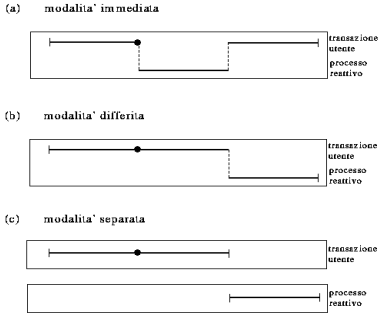
Modalità di accoppiamento

Possibili modalità di accoppiamento sono:

- Immediata: immediatamente nella stessa transazione (è l'unica supportata in Oracle)
- Differita: al momento del commit della transazione corrente
- Separata: in una nuova transazione, in questo caso può essere dipendente o indipenedente

17

Modalità di accoppiamento



18

Terminazione

Problema: non terminazione del processo reattivo

Soluzioni possibili:

- lasciare al progettista il compito di progettare le regole di modo che la non terminazione non si verifichi
- fissare un limite superiore che stabilisce un numero massimo di regole che possono essere attivate
- restrizioni sintattiche sulle regole

19

Trigger in Oracle

Comando per la creazione di trigger:

CREATE [OR REPLACE] TRIGGER *Nome*
{BEFORE | AFTER}
[{delete | insert | update [of [*Colonna* **/,]* } /OR*]**
ON *Relazione*
[[REFERENCING [OLD [AS] *Variabile* **|**
NEW [AS] *Variabile* **/,]*]**
FOR EACH ROW
[WHEN (*Condizione***)]]**
{*Blocco PL/SQL* **|** *Chiamata di procedura***}**

- altri comandi: ALTER TRIGGER con opzioni ENABLE e DISABLE, DROP TRIGGER

20

Trigger in Oracle

Evento:

- possibili eventi: INSERT, DELETE, UPDATE, UPDATE OF Lista attributi
- è possibile specificare che il trigger sia attivato prima (BEFORE) o dopo (AFTER) l'evento che li attiva
- è possibile specificare più di un evento può attivare una regola (in OR)
- in questo caso nel blocco PL/SQL si possono distinguere vari comportamenti in base all'evento mediante predicati condizionali IF inserting, IF updating, IF deleting

Azione:

- può essere blocco PL/SQL o chiamata di procedura (no DDL né comandi transazionali es. ROLLBACK)

21

Trigger in Oracle

Condizione:

- predicato SQL (clausola WHERE) senza sottoquery e funzioni user-defined
- è possibile specificare la condizione solo per row trigger (FOR EACH ROW)
- per gli statement trigger si possono effettuare comunque controlli nel blocco PL/SQL

Modalità di esecuzione:

- due modalità di esecuzione: row level (FOR EACH ROW) e statement level (che è il default!), nonostante ciò i row trigger sono più comuni
- un trigger statement level viene eseguito anche se il comando che attiva il trigger in realtà non ha modificato alcuna tupla

22

Trigger in Oracle

- combinando le opzioni BEFORE e AFTER con l'opzione FOR EACH ROW risultano quattro possibili tipi di trigger

	STATEMENT	ROW
BEFORE	**Trigger before statement:** il trigger è attivato un'unica volta prima dell'esecuzione del comando che lo attiva	**Trigger before row:** il trigger è attivato prima di modificare ogni tupla coinvolta dall'esecuzione del comando che attiva il trigger
AFTER	**Trigger after statement:** il trigger è attivato un'unica volta dopo l'esecuzione del comando che lo attiva	**Trigger after row:** il trigger è attivato dopo aver modificato ogni tupla coinvolta dall'esecuzione del comando che attiva il trigger

23

Trigger in Oracle

Row level vs statement level:

- conviene usare trigger row level se l'azione del trigger dipende dal valore della tupla modificata
- conviene usare trigger statement level se l'azione del trigger è globale per tutte le tuple modificate (fare un controllo di autorizzazione complesso, generare un singolo audit record, calcolare funzioni aggregate)

Before vs after:

- conviene usare trigger before se l'azione deltrigger determina se il comando verrà effettivamente eseguito (si evita di eseguire il comando e di farne eventualmente il rollback) oppure per derivare valori di colonne da utilizzare in un INSERT o un UPDATE

24

Trigger in Oracle

Clausola REFERENCING:

- può essere specificata solo nei row trigger
- per default, la vecchia riga è :old e la nuova è :new nel blocco, old e new nella condizione
- la clausola REFERENCING permette di dichiarare dei nomi alternati per fare riferimento allo stato vecchio (OLD) e nuovo (NEW) della tupla modificata dall'evento
- **problema**: quali tuple sono visibili durante la valutazione della condizione e l'esecuzione dell'azione?
  dipende dall'evento e dal tipo di trigger (before/after)

25

Trigger in Oracle

- **INSERT**:
  - trigger *before*: la tupla inserita non è parte della relazione, ma può essere acceduta usando NEW
  - trigger *after*: la tupla inserita è visibile sia nella relazione che tramite NEW
- DELETE:
  - trigger *before*: la tupla cancellata è parte della relazione e può essere acceduta usando OLD
  - trigger *after*: la tupla cancellata non è parte della relazione ma può essere acceduta usando OLD
- UPDATE:
  - per tutti e due i tipi di trigger, i valori precedenti e correnti della tupla possono essere acceduti usando OLD e NEW, rispettivamente
  - se *before* l'effetto della modifica non è visibile nella relazione, mentre se *after* l'effetto della modifica è visibile nella relazione

26

Trigger in Oracle

- Importante restrizione per i trigger row level
- non possono accedere con SELECT né modificare con INSERT, DELETE, UPDATE le tabelle mutating
- una tabella è mutating se è la tabella su cui è eseguito lo statement che attiva il trigger
- restrizione piuttosto forte
- motivazione: non avere comportamenti che dipendono dall'ordine in cui le tuple della tabella vengono processate nell'esecuzione del comando
- per bypassarla è possibile utilizzare tabelle temporanee o tabelle PL/SQL

27

Trigger in Oracle

- i trigger di diverso tipo vengono eseguiti nel seguente ordine
  - trigger BEFORE STATEMENT
  - per ogni tupla oggetto del comando
    - trigger BEFORE ROW
    - comando e verifica dei vincoli di integrità
    - trigger AFTER ROW
  - verifica dei vincoli che richiedono di aver completato il comando
  - trigger AFTER STATEMENT
- nessuna nozione di priorità: in caso di più trigger dello stesso tipo attivati, quello da eseguire viene scelto non deterministicamente
- modello di esecuzione ricorsivo: se durante l'esecuzione di un trigger se ne attiva un altro, si processa il nuovo trigger
- i trigger fanno parte della stessa transazione di cui fa parte il comando che ha attivato il trigger

28

Trigger in Oracle

Progettazione di trigger

- decidere il tipo di trigger (row/statement, before/after)
- identificare gli eventi
- se row level, determinare se serve condizione e quale
- determinare azione (per violazioni di integrità in generale meglio riparare che impedire: limitare al minimo azioni tipo ROLLBACK e raise_application_error)

Trigger vs vincoli

- i trigger sono più *flessibili* dei trigger, infatti permettono di stabilire come reagire ad una violazione di vincolo
- la flessibilità non è sempre un vantaggio, quindi se è possibile utilizzare i vincoli (es. chiave, integrità referenziale) è meglio (migliore ottimizzazione, meno errori di programmazione)

29

Trigger in Oracle

Esempio 1: si vuole tenere traccia in una tabella Imp_Cancellati degli impiegati cancellati dalla tabella Impiegati

```
CREATE TRIGGER Cancella_Imp
AFTER DELETE ON Impiegati
    REFERENCING OLD AS Old
FOR EACH ROW
    BEGIN
    INSERT INTO Imp_Cancellati
      VALUES (Old.Imp#);
    END;
```

30

Trigger in Oracle

Esempio 2:

- supponiamo che la tabella Impiegati abbia due colonne aggiuntive che memorizzano il numero di telefono di casa (Num_casa) e il numero di telefono dell'ufficio (Num_ufficio), si vuole che il numero di casa sia uguale, di default, a quello dell'ufficio
- non è possibile gestire una situazione di questo tipo con l'uso di DEFAULT perché DEFAULT *Nome colonna* non è un vincolo legale

```
CREATE TRIGGER Default_ Num_casa
AFTER INSERT ON Impiegati
   REFERENCING NEW AS New
FOR EACH ROW
   SET New. Num_casa = NVL(New.Num_casa, New.Num_ufficio);
```


Trigger in Oracle

Esempio 3: si vuole controllare che lo stipendio di un impiegato rientri nel range previsto per la sua mansione

```
CREATE TRIGGER Controlla_Stipendio
BEFORE INSERT OR UPDATE OF Stipendio, Manssione ON Impiegati
    FOR EACH ROW
    WHEN (new.Mansione <> 'presidente')
    DECLARE
    minstip number; maxstip number;
    BEGIN
    SELECT minstip, maxstip FROM Stipendi
       WHERE Mansione = :new.Mansione;
    IF (:new.Stipendio < minstip OR :new.Stipendio > maxstip)
    THEN raise_application_error(-20601,'stipendio fuori dal range
       per l'impiegato' || :new.Nome);
    END IF;
    END;
```


Trigger in Oracle

Esempio 4: stesso trigger, con la chiamata di una procedura ControlloStipendio, il cui corpo corrisponde al blocco nell'azione del trigger precedente

```
CREATE TRIGGER Controlla_Stipendio
BEFORE INSERT OR UPDATE OF Stipendio, Manssione ON Impiegati
    FOR EACH ROW
    WHEN (new.Mansione <> 'presidente')
    CALL ControlloStipendio(:new.Mansione, :new.Stipendio, :new.Nome);
```


Trigger in Oracle

Esempio 5: riordinare prodotti quando la disponibilità scende sotto una certa soglia

```
CREATE TRIGGER Riordino
AFTER UPDATE OF Disponibilità ON Magazzino
    FOR EACH ROW
    WHEN (new.Disponibiltà < new.QtaMinima)
    DECLARE
    x number;
    BEGIN
    SELECT COUNT(*) INTO x FROM OrdiniPendenti
       WHERE CodProdotto = :new.CodProdotto;
    IF (x = 0) THEN INSERT INTO OrdiniPendenti
              VALUES (:new.CodProdotto, :new.QtaOrdine, SYSDATE);
    END IF;
    END;
```


Trigger in Oracle

Esempio 6: mantenere colonna derivata che memorizza lo stipendio totale dei membri di un dipartimento

```
CREATE TRIGGER Stipendio_Totale
AFTER DELETE OR INSERT OR UPDATE OF Deptno, Sal ON Emp
    FOR EACH ROW
    BEGIN /* assume che Deptno e Sal siano campi NOT NULL */
    IF DELETING OR (UPDATING AND :old.Deptno != :new.Deptno)
    THEN UPDATE Dept SET TotalSal = TotalSal - :old.Sal
              WHERE Deptno = :old.Deptno;
    END IF;
    IF INSERTING OR (UPDATING AND :old.Deptno != :new.Deptno)
    THEN UPDATE Dept SET TotalSal = TotalSal + :old.Sal
              WHERE Deptno = :new.Deptno;
    END IF;
```

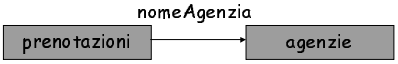

Trigger in Oracle

Esempio 6(segue):

```
IF (UPDATING AND :old.Deptno = :new.Deptno AND :old.Sal != :new.Sal)
    THEN UPDATE Dept SET TotalSal = TotalSal - :old.Sal + :new.Sal
              WHERE Deptno = :new.Deptno;
    END IF;
      END;
```

Trigger in Oracle - Esempio

- Siano *prenotazioni* e *agenzie* due tabelle legate dall'attributo *nomeAgenzia* chiave esterna in *prenotazioni*



- trigger t1 che alla prima prenotazione crea l'agenzia, per le successive ne aggiorna il totale di spese e di prenotazioni
- trigger t1 poi esteso per controllare che ogni agenzia non abbia più di tre prenotazioni (limite massimo consentito), nel caso solleva un'eccezione

Trigger in Oracle - Esempio

```
create or replace trigger t1
before insert on prenotazioni
for each row
declare
 conta number;
begin
 select count(*) into conta
 from   agenzie
 where  nomeAgenzia  = :new.agenzia;
if (conta = 0)
 then insert into agenzie
       values (:new.agenzia,1,:new.spesa);
 else update agenzie
      set    numPrenotazioni = numPrenotazioni + 1,
             spesaTot = spesaTot + :new.spesa
      where  nomeAgenzia = :new.agenzia;
```


Trigger in Oracle - Esempio

```
create or replace package packPren as
   …
   troppePrenotazioni exception;
end packPren;

create or replace trigger t1
before insert on prenotazioni
for each row
declare
  conta number;
  prenota number;
begin
  select count(*) into conta
  from   agenzie
  where  nomeAgenzia  = :new.agenzia;
```


Trigger in Oracle - Esempio (segue)

```
else begin
       select numPrenotazioni into prenota
       from   agenzie
       where  nomeAgenzia = :new.agenzia;
       if (prenota = 3)
         then raise packPren.troppePrenotazioni;
       end if;

       update agenzie
       set    numPrenotazioni = numPrenotazioni + 1,
              spesaTot = spesaTot + :new.spesa
       where  nomeAgenzia = :new.agenzia;
      end;
```


Trigger in Oracle - Esempio (segue)

```
create or replace package packPren as
   procedure prenota(…);
   troppePrenotazioni exception;
end packPren;

create or replace package body packPren as
   procedure prenota(…)
   begin
      …
      insert into prenotazioni
      values (…)
      …
      exception when packPren.troppePrenotazioni
                then …do something…
   end prenota;
end packPren;
```


Starbust

Esempio: due tabelle

Impiegati(Imp#,Stipendio,Dip#)
Dipartimenti(Dip#,Dirigente)

Vincolo: lo stipendio di un impiegato non può essere maggiore dello stipendio del direttore del dipartimento in cui lavora

Starbust

Esempio (continua):

```
CREATE RULE stipendio_troppo_alto ON Impiegati
WHEN inserted, updated(Stipendio), updated(Dip#)
IF SELECT *
     FROM Impiegati E, Impiegati M, Dipartimenti D
     WHERE E.Stipendio>M.Stipendio AND
                  E.Dip#=D.Dip# AND D.Dirigente = M.Imp#
THEN ROLLBACK;
```

Nota: dovrei definire una regola simile su Dipartimenti

43

Starbust

Esempio: due tabelle

Impiegati(Imp#, Stipendio, Dip#)
Dipartimenti(Dip#, Dirigente)

Vincolo: lo stipendio di un impiegato non può essere aumentato più di 100

44

Starbust

Esempio (continua):

```
CREATE RULE aumento_troppo_alto ON Impiegati
WHEN updated(Stipendio)
IF EXISTS (SELECT *
     FROM old-updated ou, new-updated nu
     WHERE nu.Stipendio-ou.Stipendio>100)
THEN ROLLBACK;
```

45

Starbust

Esempio (continua): inserire nella relazione Ben_Pagato gli impiegati che guadagnano più di 3000

```
CREATE RULE ins_in_bp ON Impiegati
WHEN inserted
THEN INSERT INTO Ben_Pagato
     SELECT * FROM inserted
     WHERE Stipendio > 3000
FOLLOWS aumento_troppo_alto ;
```

46

Starbust

Esempio (continua) se lo stipendio medio degli impiegati inseriti eccede la media dello stipendio di tutti gli impiegati di almeno 1000, assegnare a tutti gli impiegati inseriti uno stipendio pari a 5000

```
CREATE RULE avg_ins ON Impiegati
WHEN inserted
IF (SELECT avg(Stipendio) FROM inserted ) -
   (SELECT avg(Stipendio) FROM Impiegati) > 1000
THEN UPDATE Impiegati
     SET Stipendio = 5000
     WHERE Imp# IN (SELECT Imp# FROM inserted);
```

47