

Basi di dati attive

Basi di dati attive

1

Sommario

- Preliminari
- Approcci architetturali
- Linguaggi per la specifica di regole
 - Eventi
 - Condizioni
 - Azioni
 - Ulteriori caratteristiche
- Modello di esecuzione
 - Esecuzione delle regole
 - Soluzione dei conflitti
 - Modalità di accoppiamento
 - Terminazione

Basi di dati attive

2

Sommario

- Regole attive in Starburst
- Lo standard per le regole attive nei DBMS relazionali: **SQL:1999**
- Le regole attive nei sistemi commerciali
 - Oracle
 - DB2

Basi di dati attive

3

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*

Basi di dati attive

4

Preliminari

Esempi di applicazioni in cui i DBMS attivi sono utili:

- controllo dei processi
- gestione automatizzata del lavoro di ufficio
- controllo di workflow
- sistemi di controllo in ambito medico

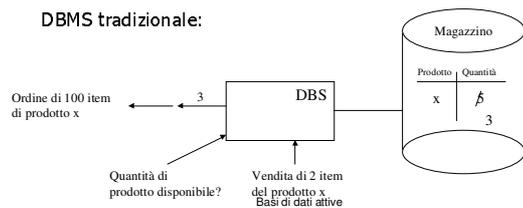
Basi di dati attive

5

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:



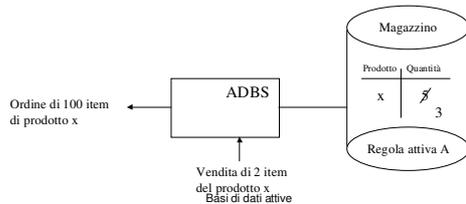
6

Preliminari

Esempio continua:

DBMS attivo:

Regola attiva A: se la quantità diventa ≤ 4 allora ordina 100 item



7

Preliminari

Questo è un esempio di uso delle regole attive per monitoraggio

Altri esempi:

- vincoli di integrità
- alerting
- specifica di azioni consequenziali

Basi di dati attive

8

Approcci architetturali

Basi di dati attive

9

Approcci architetturali

Basi di dati attive

10

Approcci architetturali

Basi di dati attive

11

Linguaggi per la specifica di regole

- 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 si verificano, 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

Basi di dati attive

12

Linguaggi per la specifica di regole

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

Basi di dati attive

13

Linguaggi per la specifica di regole

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

Basi di dati attive

14

Linguaggi per la specifica di regole

- 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*

Basi di dati attive

15

Linguaggi per la specifica di regole

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

Basi di dati attive

16

Eventi

Un *evento* è qualcosa che accade, o si verifica, che è di interesse e che può essere mappato in un istante di tempo

- Modifica dei dati: inserimento, cancellazione, modifica
- Accesso ai dati: interrogazione su una tabella
- Operazione del DBMS: login di un utente, gestione di transazioni, di autorizzazioni
- Eventi temporali: ogni giorno alle 12
- Eventi definiti da applicazioni: data troppo grande

Basi di dati attive

17

Eventi

- Possibilità di definire regole che possono essere attivate *before* o *after* un evento

- Possibilità di combinare gli eventi (*eventi composti*):

- operatori logici: and, or, ecc.
- sequenza: considero un trigger se due o più eventi accadono in un certo ordine
- composizione temporale: considero un trigger quando l'evento E2 avviene 5 secondi dopo l'evento E1

Basi di dati attive

18

Condizioni

Una *condizione* è un ulteriore controllo che viene eseguito quando la regola è considerata e prima che l'azione sia eseguita

- **Predicati**: clausola where di SQL, è vantaggioso avere predicati semplici perché sono più efficienti da valutare
- **Interrogazioni**: condizione vera se e solo se l'interrogazione restituisce l'insieme vuoto
- **Procedure applicative**: chiamata ad una procedura

Basi di dati attive

19

Condizioni

Altre osservazioni:

- la condizione può far riferimento a stati passati o a variabili di sistema
- passaggio di parametri tra condizione e azione (non sempre possibile)
- se la condizione non c'è si assume vera

Basi di dati attive

20

Azioni

Un'azione è una sequenza di operazioni che viene eseguita quando la regola è considerata e a sua condizione è vera

- **Modifica dei dati**: inserimento, cancellazione, modifica
- **Accesso ai dati**: interrogazione su una tabella
- **Altri comandi**: definizione di dati, controllo delle transazioni (commit, rollback), garantire e revocare privilegi
- **Procedure applicative**: chiamata ad una procedura

Basi di dati attive

21

Ulteriori caratteristiche

- **Comandi per le regole**: di solito è possibile creare e cancellare una regola, oppure abilitarla e disabilitarla
- **Priorità per le regole**: spesso devo scegliere quale regola attivare fra un insieme di regole
 - priorità relative (per es., ordine di creazione)
 - priorità assolute
- **Strutturare le regole**: in moduli

Basi di dati attive

22

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) end If  
endWhile
```

Basi di dati attive

23

Modello di esecuzione

Granularità del processo reattivo: frequenza di attivazione del processo

Esempi:

- alla fine di un intero comando SQL in cui, per esempio, modifichiamo tante tuple
- alla fine della modifica di ogni tupla da parte di un comando SQL (più fine della precedente)

Basi di dati attive

24

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

Basi di dati attive

25

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

Basi di dati attive

26

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

Basi di dati attive

27

Modalità di accoppiamento

Possibili modalità di accoppiamento sono:

- **Immediata**: immediatamente nella stessa transazione
- **Differita**: al momento del commit della transazione corrente
- **Separata**: in una nuova transazione, in questo caso può essere dipendente o indipendente

Basi di dati attive

28

Modalità di accoppiamento

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

Basi di dati attive

29

Basi di dati attive

30

Starbust

- Progetto di ricerca sviluppato all'IBM
- **Starbust**: DBMS relazionale estensibile al quale è stata aggiunta una componente attiva
- Ha influenzato molto lo standard SQL:1999
- Completa integrazione della componente reattiva del sistema con il linguaggio di interrogazione e le transazioni

Basi di dati attive

31

Starbust

Regola in Starbust:

```
CREATE RULE Nome ON Relazione
WHEN Eventi
[IF Condizione]
THEN Lista Azioni
[PRECEDES Lista Regole]
[FOLLOWS Lista Regole]
```

Nota: più di un evento può attivare una regola

Basi di dati attive

32

Starbust

Possibili **eventi**:

- inserted
- deleted
- updated
- updated(a1,...,an)

Condizione: interrogazione SQL, viene valutata *True* se restituisco almeno una tupla

Nota: non c'è passaggio di parametri

Basi di dati attive

33

Starbust

Possibili **azioni**:

- Comandi di manipolazione: INSERT, DELETE, UPDATE, SELECT
- Comandi di definizione: CREATE/DROP TABLE, CREATE/DROP VIEW, DROP RULE
- Comando transazionale di ROLLBACK

Clausole **PRECEDES/FOLLOWS**: vengono utilizzate per definire delle priorità relative fra le regole

Basi di dati attive

34

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

Basi di dati attive

35

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

Basi di dati attive

36

Starbust

Transition table:

- insieme di tuple che sono state effettivamente inserite, cancellate, modificate
- dette anche *delta table*
- migliorano l'efficienza

Basi di dati attive

37

Starbust

Starbust ammette le seguenti transition table:

- inserted
- deleted
- new-updated, old-updated

Le transition table sono usate nella valutazione della condizione e nell'azione

Basi di dati attive

38

Starbust

Esempio: due tabelle

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

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

Basi di dati attive

39

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;
```

Basi di dati attive

40

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;
```

Basi di dati attive

41

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);
```

Basi di dati attive

42

Starbust

Comandi per la gestione delle regole:

- CREATE
- DROP
- ALTER
- DEACTIVATE
- ACTIVATE

Basi di dati attive

43

Starbust

Esecuzione delle regole:

- le regole sono eseguite alla fine delle transazioni
- possibilità di richiedere esplicitamente l'attivazione del processo reattivo (*processing point*) con il comando PROCESS RULES
- la semantica si basa sulla nozione di *transizione di stato* e di *effetto netto*

Basi di dati attive

44

Starbust

Definizione: Transizione di stato

Una *transizione di stato* è la trasformazione da uno stato ad un altro della base di dati prodotta dall'esecuzione di una sequenza di operazioni SQL di manipolazione dei dati

$S_0 \xrightarrow{\text{Transazione}} S_1$

Basi di dati attive

45

Starbust

Definizione: Effetto netto

L'*effetto netto* di una transizione di stato è costituito dall'insieme delle tuple inserite, da quello delle tuple cancellate e da quello delle tuple modificate

L'effetto netto è usato per calcolare le transition table e per stabilire quali regole sono attivate

Se ho la transizione:

$S_0 \xrightarrow{\text{Transazione}} S_1$

Basi di dati attive

46

Starbust

L'effetto netto sarà composto dai seguenti insiemi:

- tuple inserite stato in S_1
- tuple cancellate stato in S_0
- tuple modificate stato vecchio in S_0 , stato nuovo in S_1 , anche se la modifico più volte

Basi di dati attive

47

Starbust

Osservazioni: t tupla

- inserisco t, modifico t: considero l'inserimento di t già modificata
- modifico t, cancello t: considero la cancellazione di t
- modifico t più volte: vecchio valore in S_0 , nuovo valore in S_1
- inserisco t, cancello t: la tupla non è considerata nell'effetto netto

Basi di dati attive

48

Starbust

Una regola viene attivata se una o più operazioni dei suoi eventi sono occorse nella transizione che determina il passaggio dallo stato all'inizio della transazione (S0) allo stato alla fine della transazione (S1)

Caso particolare: processing point

Le transition table sono calcolate analogamente

Basi di dati attive

49

SQL:1999

Regola in SQL:1999:

```
CREATE TRIGGER Nome
[BEFORE | AFTER] Evento ON Relazione
[REFERENCING OLD [ROW] [AS] Variabile |
NEW [ROW] [AS] Variabile |
OLD TABLE [AS] Variabile |
NEW TABLE [AS] Variabile]
[FOR EACH ROW | FOR EACH STATEMENT]
[WHEN Condizione]
[Comando SQL | BEGIN ATOMIC Lista Comandi] END
```

Nota: esiste anche il comando **DROP TRIGGER Nome**

Basi di dati attive

50

SQL:1999

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
- **un solo** evento può attivare una regola, quindi una sola operazione su una sola tabella

Condizione:

predicato SQL arbitrario (clausola WHERE)

Basi di dati attive

51

SQL:1999

Azione:

- trigger *before*: definizione di dati, dichiarazione di un cursore ecc., ma **non** è possibile effettuare operazioni che modificano lo stato della base di dati
- trigger *after*: tutto quello che posso mettere in un trigger *before* più operazioni di manipolazione dei dati (INSERT, DELETE, UPDATE)
- due modalità di esecuzione: FOR EACH ROW e FOR EACH STATEMENT (che è il default!), nonostante ciò i trigger FOR EACH ROW sono più comuni

Basi di dati attive

52

SQL:1999

Clausola REFERENCING:

- La clausola REFERENCING "implementa" le transition table
- Il default è ROW
- **Problemi**:
 - Quali tuple sono visibili durante la valutazione della condizione e l'esecuzione dell'azione?
 - Posso usare la clausola REFERENCING OLD **ROW AS** Variable in un trigger FOR EACH **STATEMENT**?

Basi di dati attive

53

SQL:1999

Quesito: Quali tuple sono visibili durante la valutazione della condizione e l'esecuzione dell'azione?

Risposta: dipende dall'evento che ha attivato il trigger e dal tipo di trigger before/after

- **INSERT**:
 - trigger *before*: le tuple inserite non sono visibili come parte della relazione, ma possono essere accedute usando la clausola REFERENCING NEW
 - trigger *after*: le tuple inserite sono visibili sia nella relazione che tramite la clausola REFERENCING (ovviamente NEW)

Basi di dati attive

54

SQL:1999

- **DELETE:**
 - trigger *before*: le tuple cancellate sono visibili come parte della relazione, e possono essere accedute usando la clausola REFERENCING OLD
 - trigger *after*: le tuple cancellate non sono visibili come parte della relazione, ma possono essere accedute usando la clausola REFERENCING OLD
- **UPDATE:**
 - per tutti e due i tipi di trigger, i valori precedenti e correnti delle tuple possono essere acceduti usando la clausola REFERENCING OLD e NEW, rispettivamente)
 - se *before* l'effetto della modifica non è visibile nella relazione, mentre se di tipo *after* l'effetto della modifica è visibile nella relazione

Basi di dati attive

55

SQL:1999

Quesito: Posso usare la clausola REFERENCING OLD ROW in un trigger FOR EACH STATEMENT?

Risposta:

	ODL ROW	NEW ROW	ODL TABLE	NEW TABLE
before statement	-	-	-	-
before row	delete, update	insert, update	-	-
after statement	-	-	delete, update	insert, update
after row	delete, update	insert, update	delete, update	insert, update

Basi di dati attive

56

SQL:1999

Quando l'evento accade il trigger è attivato

Frase vaga che lascia spazio a diverse interpretazioni, ovvero diverse esecuzioni che portano a risultati diversi

L'esecuzione dipende dal tipo di trigger (*before/after*) e dalla priorità

In SQL:1999 associo delle **priorità** relative in base all'ordine di creazione: un trigger "vecchio" è eseguito prima di un trigger "giovane"

Basi di dati attive

57

SQL:1999

Problema: come "interferiscono" i trigger con il controllo dei vincoli?

Esempio:

```
CREATE TRIGGER Trigger1 AFTER UPDATE ON Tabella1 ...;
CREATE TRIGGER Trigger2 BEFORE UPDATE ON Tabella1 ...;
CREATE TRIGGER Trigger3 AFTER UPDATE ON Tabella1 ...;
ALTER TABLE Tabella1 ADD CONSTRAINT Vincolo1...;
```

Basi di dati attive

58

SQL:1999

Esempio (continua) quale è l'effetto di eseguire UPDATE su Tabella1?

Le seguenti cose nel seguente ordine avvengono:

- Trigger2 è attivato
- Esecuzione dell'operazione di UPDATE su Tabella1
- Controllo Vincolo1 (il controllo dei vincoli avviene alla fine dell'esecuzione del comando)
- Trigger1 è attivato
- Trigger3 è attivato (più giovane di Trigger1)

Basi di dati attive

59

SQL:1999

Riassumendo valuto le seguenti cose nel seguente ordine:

- *Before* trigger
- Comando SQL in esame
- Controllo vincoli
- *After* trigger

Basi di dati attive

60

SQL:1999

In ogni insieme di trigger:

- eseguo i trigger più "vecchi" prima dei più "giovani"
- lo standard non fornisce indicazioni per ordine fra i trigger FOR EACH STATEMENT e ROW

Nota: i trigger fanno parte della stessa transazione di cui fa parte il comando che ha attivato il trigger

Basi di dati attive

61

SQL:1999

Tabella riassuntiva

Modello dei dati	Relazionale ad oggetti
Eventi primitivi	Operazioni sulla base di dati
Eventi compositi	No
Eventi parametrici	No
Passaggio di parametri	No
Condizioni su stati passati	Sì
Net effect	No
Modalità di accoppiamento	Immediata
Costruito a cui associa il trigger	Relazione

Basi di dati attive

62

SQL:1999

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

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

Basi di dati attive

63

SQL:1999

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 il vincolo DEFAULT perché DEFAULT *Nome colonna* non è un vincolo legale

Basi di dati attive

64

SQL:1999

Esempio 2 (continua):

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

Dove: `COALESCE(valore1, valore2) = CASE WHEN valore1 IS NOT NULL THEN valore1 ELSE valore2`

Basi di dati attive

65

SQL:1999

Esempio 3: supponiamo che la tabella Dipartimenti abbia un attributo Budget e che il budget di un dipartimento non può essere modificato dopo le 17m

```
CREATE TRIGGER Update_Dipartimenti
AFTER UPDATE ON Dipartimenti
WHEN (CURRENT_TIME > TIME '17:00:00')
SELECT MAX(Budget)/0 FROM Dipartimenti;
```

N.B. il default è FOR EACH STATEMENT

Basi di dati attive

66

SQL:1999

Esempio 3 (continua):

Problema: I comandi SQL

```
UPDATE Dipartimenti SET Budget = v1, Dirigente = v3;
```

```
UPDATE Dipartimenti SET Budget = NULL;
```

potrebbero non attivare il trigger

Basi di dati attive

67

SQL:1999

Esempio 4: la prima volta che viene eletto Bob, le tasse vengono diminuite dell'1%, inoltre, ogni modifica delle tasse influenza il debito nazionale e diminuisce la popolarità di Bob

```
CREATE TRIGGER Update_Primi_Ministri
AFTER UPDATE OF Nome ON Primi_Ministri
REFERENCING OLD ROW AS Old, NEW ROW AS New
FOR EACH ROW
WHEN (New.Nome='Bob' AND New.Nome <> Old.Nome)
UPDATE Contribuenti SET Tasse=Tasse * 0.99;
```

Basi di dati attive

68

SQL:1999

Esempio 4 (continua):

```
CREATE TRIGGER Update_Contribuenti
AFTER UPDATE OF Tasse ON Contribuenti
REFERENCING OLD ROW AS Old, NEW ROW AS New
FOR EACH ROW
BEGIN ATOMIC
  UPDATE Debito_Nazionale
  SET Quantità = Quantità+(Old.Tasse-New.Tasse);
  UPDATE Primi_Ministri
  SET Popolarità = Popolarità - 0.01
END;
```

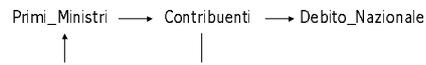
Basi di dati attive

69

SQL:1999

Esempio 4 (continua):

Problema: i trigger sembrerebbero attivarsi a vicenda dando vita ad un processo reattivo infinito



Il ciclo è solo apparente perché gli UPDATE su Primi_Ministri sono su colonne diverse

Basi di dati attive

70

SQL:1999

Trigger e vincoli

I trigger sono più *flessibili* dei trigger, infatti permettono di stabilire come reagire ad una violazione di vincolo

La flessibilità è sempre un vantaggio? Non sempre

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

Basi di dati attive

71