Basi di dati attive

Paolo Atzeni
Stefano Ceri

Basi di dati attive

- BD con componente per la gestione di regole **Evento-Condizione-Azione** (regole di produzione):
  - eventi: normalmente modifiche della base di dati
  - valuta una condizione e, in base al valore di verit\`a
- hanno comportamento **reattivo** (in contrasto con **passivo**): eseguono non solo le transazioni utenti ma anche le regole
- le regole “mettono a fattor comune” parte dell’applicazione che è così “condivisa”
  - “indipendenza della conoscenza”
- esistono molti prototipi sperimentali
- i DBMS commerciali hanno i **trigger** (standardizzati in SQL:1999, molto in ritardo e quindi i sistemi sono molto diversi)

Trigger

- definiti con istruzioni DDL (`create trigger`)
  - basati sul paradigma evento-condizione-azione (ECA):
    evento: modifica dei dati, specificata con `insert, delete, update`
    condizione (opzionale) predicato SQL
    azione: sequenza di istruzioni SQL (o estensioni, ad esempio PL/SQL in Oracle)
  - intuitivamente:
    quando si verifica l'evento (attivazione)
    se la condizione è soddisfatta (considerazione)
    allora esegui l'azione (esecuzione)
  - ogni trigger fa riferimento ad una tabella (target): risponde ad eventi relativi a tale tabella

Trigger: granularità e modalità

- granularità
  - di ennupla (row-level): attivazione per ogni ennupla coinvolta nell'operazione
  - di operazione (statement-level): una sola attivazione per ogni istruzione SQL, con riferimento a tutte le ennuple coinvolte ("set-oriented")
- modalità
  - immediata: subito dopo (o subito prima) dell'evento
  - differita: al commit

Sintassi SQL:1999 dei trigger

- SQL:1999 propone una sintassi simile a quella offerta da Oracle Server e IBM DB2
- I sistemi tenderanno a uniformarsi a essa
- Ogni trigger è caratterizzato da
  - nome
  - nome della tabella che viene monitorata
  - modo di esecuzione (BEFORE o AFTER)
  - l'evento monitorato (INSERT, DELETE o UPDATE)
  - granularità (statement-level o row-level)
  - nomi e alias per transition values e transition tables
  - l'azione
  - il timestamp di creazione

Sintassi SQL:1999 dei trigger

create trigger *NomeTrigger*
{**before** | **after**}
{ **insert** | **delete** | **update** [**of** *Colonne*] } **on** *Tabella*
[**referencing**
  {[**old table** [**as**] *AliasTabellaOld*]
  [**new table** [**as**] *AliasTabellaNew*] } |
  {[**old** [**row**] [**as**] *NomeTuplaOld*]
  [**new** [**row**] [**as**] *NomeTuplaNew*] }]
[**for each** { **row** | **statement** }]
[**when** *Condizione*]
ComandiSQL

Esecuzione di un singolo trigger

- Modo di esecuzione:
  - BEFORE
    - Il trigger viene considerato ed eventualmente eseguito prima che venga applicata sulla base di dati l'azione che lo ha attivato
    - Di norma viene utilizzata questa modalità quando si vuole verificare la correttezza di una modifica, prima che venga applicata
  - AFTER
    - Il trigger viene considerato ed eventualmente eseguito dopo che è stata applicata sulla base di dati l'azione che lo ha attivato
    - È il modo più comune, adatto a quasi tutte le applicazioni
    - È più semplice da utilizzare correttamente

Granularità degli eventi

- Modo statement level (modo di default)
  - Il trigger viene considerato ed eventualmente eseguito una volta sola per ogni comando che lo ha attivato, indipendentemente dal numero di tuple modificate
  - È il modo più vicino all'approccio tradizionale dei comandi SQL, che sono di norma set-oriented
- Modo row-level (opzione **`for each row`**)
  - Il trigger viene considerato ed eventualmente eseguito una volta per ciascuna tupla che è stata modificata dal comando
  - Consente di scrivere i trigger in modo più semplice
  - Può essere meno efficiente

Clausola `referencing`

- Il formato dipende dalla granularità
  - Per il modo row-level, si hanno due *transition variables* **`old`** e **`new`**, che rappresentano rispettivamente il valore precedente e successivo alla modifica della tupla che si sta valutando
  - Per il modo statement-level, si hanno due *transition tables* **`old table`** e **`new table`**, che contengono rispettivamente il valore vecchio e nuovo di tutte le tuple modificate
- Le variabili **`old`** e **`old table`** non sono utilizzabili in trigger il cui evento è **`insert`**
- Le variabili **`new`** e **`new table`** non sono utilizzabili in trigger il cui evento è **`delete`**
- Le variabili e le tabelle di transizione sono importanti per realizzare i trigger in modo efficiente

Esempio di trigger row-level

```
create trigger MonitoraConti
after update on Conto
referencing old as old new as new
for each row
when (old.NomeConto = new.NomeConto and
new.Totale > old.Totale)
insert values
(new.NomeConto,new.Totale-old.Totale)
    into SingoliVersamenti
```

Esempio di trigger statement-level

```
create trigger ArchiviaFattureCanc
after delete on Fattura
referencing old table as SetOldFatture
insert into FattureCancellate
(select *
 from SetOldFatture)
```


Conflitti tra trigger

- Se vi sono più trigger associati allo stesso evento, SQL:1999 prescrive questa politica di gestione
  - Vengono eseguiti i trigger BEFORE statement-level
  - Vengono eseguiti i trigger BEFORE row-level
  - Si applica la modifica e si verificano i vincoli di integrità definiti sulla base di dati
  - Vengono eseguiti i trigger AFTER rwo-level
  - Vengono eseguiti i trigger AFTER statement-level
- Se vi sono più trigger della stessa categoria, l'ordine di esecuzione viene scelto dal sistema in un modo che dipende dall'implementazione

Modello di esecuzione

- SQL:1999 prevede che i trigger vengano gestiti in un Trigger Execution Context (TEC)
- L'esecuzione dell'azione di un trigger può produrre eventi che fanno scattare altri trigger, che dovranno essere valutati in un nuovo TEC interno
- In ogni istante possono esserci più TEC per una transazione, uno dentro l'altro, ma uno solo può essere attivo
- Per i trigger row-level il TEC tiene conto di quali tuple sono già state considerate e quali sono da considerare
- Si ha quindi una struttura a stack
  - TEC0 -> TEC1 -> ... -> TECn
- Quando un trigger ha considerato tutti gli eventi, il TEC si chiude e si passa al trigger successivo
- È un modello complicato, ma preciso e relativamente semplice da implementare

Esempio di esecuzione

Gestione dei salari

Impiegato

Matricola	Nome	Salario	NDip	NProg
50	Rossi	59.000	1	20
51	Verdi	56.000	1	10
52	Bianchi	50.000	1	20

Dipartimento

NroDip	MatricolaMGR
1	50

Progetto

NroProg	Obiettivo
10	NO
20	NO

Trigger T1: Bonus

Evento: **update** di `Obiettivo` in `Progetto`
Condizione: `Obiettivo` **= 'SI'**
Azione: incrementa del 10% il salario degli impiegati coinvolti nel progetto

```
CREATE TRIGGER Bonus
AFTER UPDATE OF Obiettivo ON Progetto
FOR EACH ROW
WHEN NEW.Obiettivo = 'SI'
BEGIN
  update Impiegato
            set Salario = Salario*1.10
            where NProg = NEW.NroProg;
END;
```

Trigger T2: ControllaIncremento

Evento: **update** di `Salario` in `Impiegato`
Condizione: nuovo salario maggiore di quello del manager
Azione: decrementa il salario rendendolo uguale a quello del manager

```
CREATE TRIGGER ControllaIncremento
AFTER UPDATE OF Salario ON Impiegato
FOR EACH ROW
DECLARE X number;
BEGIN
            SELECT Salario into X
            FROM Impiegato JOIN Dipartimento
            ON Impiegato.Matricola =
            Dipartimento.MatricolaMGR
            WHERE Dipartimento.NroDip= NEW.NDip;
            IF NEW.Salario > X
              update Impiegato set Salario = X
              where Matricola = NEW.Matricola;
            ENDIF;
END;
```


Trigger T3: ControllaDecremento

Evento: **update** di `Salario` in `Impiegato`
Condizione: decremento maggiore del 3%
Azione: decrementa il salario del solo 3%

```
CREATE TRIGGER ControllaDecremento
AFTER UPDATE OF Salario ON Impiegato
FOR EACH ROW
WHEN (NEW.Salario < OLD.Salario * 0.97)
BEGIN
            update Impiegato
            set Salario=OLD.Salario*0.97
            where Matricola = NEW.Matricola;
END;
```

Attivazione di T1

```
Update Progetto
set Obiettivo = 'SI' where NroProg = 10
```

Progetto

NroProg	Obiettivo
10	SI
20	NO

Evento: update dell'attributo **Obiettivo** in **Progetto**

Condizione: vera

Azione: si incrementa del 10% il salario di Verdi

Matricola	Nome	Salario	Ndipart	NProg
50	Rossi	59.000	1	20
51	Verdi	61.600	1	10
52	Bianchi	50.000	1	20

Attivazione di T2

Evento: **update** di **Salario** in **Impiegato**

Condizione: vera (il salario dell'impiegato Verdi supera quello del manager Rossi)

Azione: si modifica il salario di Verdi rendendolo uguale a quello del manager Rossi

Matricola	Nome	Salario	Ndipart	NProg
50	Rossi	59.000	1	20
51	Verdi	59.000	1	10
52	Bianchi	50.000	1	20

- Si attiva nuovamente T2 - condizione è falsa
- Si attiva T3

Attivazione di T3

Evento: update dell'attributo **`salario`** in **`Impiegato`**

Condizione: vera (il salario di Verdi è stato decrementato per più del 3%)

Azione: si decrementa il salario di Verdi del solo 3%

Matricola	Nome	Salario	Ndipart	NProg
50	Rossi	59.000	1	20
51	Verdi	59.752	1	10
52	Bianchi	50.000	1	20

- Si attiva nuovamente T3 - condizione è falsa
- Si attiva T2 – condizione vera

Attivazione di T2

Matricola	Nome	Salario	Ndipart	NProg
50	Rossi	59.000	1	20
51	Verdi	59.000	1	10
52	Bianchi	50.000	1	20

Attivazione di T3

- La condizione del trigger è falsa
  - Il salario è stato decrementato per meno del 3%
- L'attivazione dei trigger ha raggiunto lo stato di terminazione

Estensioni (di solito non disponibili)

- eventi temporali (anche periodici) o “definiti dall’utente”
- combinazioni booleane di eventi
- clausola `instead of`: non si esegue l'operazione che ha attivato l'evento, ma un’altra azione
- esecuzione “distaccata”: si attiva transazione autonoma
- definizione di priorità
- regole a gruppi, attivabili e disattivabili
- regole associate anche a interrogazioni (non solo aggiornamenti)

Proprietà delle regole

- terminazione (essenziale)
- confluenza
- determinismo delle osservazioni

Applicazioni

- funzionalità interne
  - controllo dei vincoli di integrità
  - replicazione
  - gestione delle viste
    - materializzate: propagazione
    - virtuali: modifica delle interrogazioni
- funzionalità applicative: descrizione della dinamica (comportamento) delle basi di dati