Controllo d’accesso e transazioni

Prof. Alfredo Pulvirenti

Prof. Salvatore Alaimo

(Atzeni-Ceri Capitolo 5)

Controllo dell'accesso

- In SQL è possibile specificare chi (utente) e come (lettura, scrittura, …) può utilizzare la base di dati (o parte di essa)
- Oggetto dei privilegi (diritti di accesso) sono di solito le tabelle, ma anche altri tipi di risorse, quali singoli attributi, viste o domini.
- Un utente predefinito _system (amministratore della base di dati) ha tutti i privilegi.
- Il creatore di una risorsa ha tutti i privilegi su di essa.

Privilegi

- Un privilegio è caratterizzato da:
  - la risorsa cui si riferisce
  - l'utente che concede il privilegio
  - l'utente che riceve il privilegio
  - l'azione che viene permessa
  - la trasmissibilità del privilegio

Tipi di privilegi offerti da SQL

- insert: permette di inserire nuovi oggetti (ennuple)
- update: permette di modificare il contenuto
- delete: permette di eliminare oggetti
- select: permette di leggere la risorsa
- references: permette la definizione di vincoli di integrità referenziale verso la risorsa (può limitare la possibilità di modificare la risorsa)
- usage: permette l'utilizzo in una definizione (per esempio, di un dominio)

grant e revoke

- Concessione di privilegi:

grant < *Privileges* | all privileges > on *Resource*
to *Users* [with grant option]

  - grant option specifica se il privilegio può essere trasmesso ad altri utenti

grant select on Department to Stefano

- Revoca di privilegi

revoke *Privileges* on *Resource* from *Users*
[restrict | cascade]

Autorizzazioni, commenti

- La gestione delle autorizzazioni deve “nascondere” gli elementi cui un utente non può accedere, senza sospetti
- Esempio:
    - Impiegati non esiste (esiste Impiegati)
    - ImpiegatiSegreti esiste, ma l’utente non è autorizzato
  - L’utente deve ricevere lo stesso messaggio

Autorizzazioni, commenti, 2

- Come autorizzare un utente a vedere solo alcune ennuple di una relazione?
  - Attraverso una vista:
    - Definiamo la vista con una condizione di selezione
    - Attribuiamo le autorizzazioni sulla vista, anziché sulla relazione di base

Autorizzazioni, ancora

- (Estensioni di SQL:1999)
- Concetto di ruolo, cui si associano privilegi (anche articolati), poi concessi agli utenti attribuendo il ruolo

Transazione

- Insieme di operazioni da considerare indivisibile ("atomico"), corretto anche in presenza di concorrenza e con effetti definitivi
- Proprietà ("acide"):
  - Atomicità
  - Consistenza
  - Isolamento
  - Durabilità (persistenza)

Le transazioni sono … atomiche

- La sequenza di operazioni sulla base di dati viene eseguita per intero o per niente:
  - trasferimento di fondi da un conto A ad un conto B: o si fanno il prelevamento da A e il versamento su B o nessuno dei due

Le transazioni sono … consistenti

- Al termine dell'esecuzione di una transazione, i vincoli di integrità debbono essere soddisfatti
- "Durante" l'esecuzione ci possono essere violazioni, ma se restano alla fine allora la transazione deve essere annullata per intero ("abortita")

Le transazioni sono … isolate

- L'effetto di transazioni concorrenti deve essere coerente (ad esempio "equivalente" all'esecuzione separata)
  - se due assegni emessi sullo stesso conto corrente vengono incassati contemporaneamente si deve evitare di trascurarne uno

I risultati delle transazioni sono durevoli

- La conclusione positiva di una transazione corrisponde ad un impegno (in inglese commit) a mantenere traccia del risultato in modo definitivo, anche in presenza di guasti e di esecuzione concorrente

Transazioni in SQL

- Una transazione inizia al primo comando SQL dopo la "connessione" alla base di dati oppure alla conclusione di una precedente transazione (lo standard indica anche un comando start transaction, non obbligatorio, e quindi non previsto in molti sistemi)
- Conclusione di una transazione
  - commit [work]: le operazioni specificate a partire dall'inizio della transazione vengono eseguite sulla base di dati
  - rollback [work]: si rinuncia all'esecuzione delle operazioni specificate dopo l'inizio della transazione
- Molti sistemi prevedono una modalità autocommit, in cui ogni operazione forma una transazione

Una transazione in SQL

```
start transaction                    (opzionale)
update ContoCorrente
 set Saldo = Saldo – 10
 where NumeroConto = 12345 ;
update ContoCorrente
 set Saldo = Saldo + 10
 where NumeroConto = 55555 ;
commit work;
```

Basi di dati attive

Prof. Alfredo Pulvirenti

Prof. Salvatore Alaimo

Basi di dati attive

- Una base di dati che contiene regole attive (chiamate *trigger*)
- Presentazione:
  - Definizione dei trigger in SQL:1999
  - Definizione dei trigger in DB2 e Oracle
  - Problemi di progetto per applicazioni basate sull'uso dei trigger
  - Caratteristiche evolute dei trigger
  - Esempi di applicazioni

Il concetto di trigger

- Paradigma: Evento-Condizione-Azione
  - Quando un evento si verifica
  - Se la condizione è vera
  - Allora l'azione è eseguita
- Questo modello consente computazioni reattive
- Non è il solo tipo di regole:
  - Vincoli di integrità
- Problema: è difficile realizzare applicazioni complesse

Evento-Condizione-Azione

- Evento
  - Normalmente una modifica dello stato del database: insert, delete, update
  - Quando accade l'evento, il trigger è *attivato*
- Condizione
  - Un predicato che identifica se l'azione del trigger deve essere eseguita
  - Quando la condizione viene valutata, il trigger è *considerato*
- Azione
  - Una sequenza di update SQL o una procedura
  - Quando l'azione è eseguita anche il trigger è *eseguito*
- I DBMS forniscono tutti i componenti necessari. Basta integrarli.

SQL:1999, Sintassi

- Lo standard SQL:1999 (SQL-3) sui trigger è stato fortemente influenzato da DB2 (IBM); gli altri sistemi non seguono lo standard (esistono dagli anni 80')
- Ogni trigger è caratterizzato da:
  - nome
  - target (tabella controllata)
  - modalità (**`before`** o **`after`**)
  - evento (**`insert`**, **`delete`** o **`update`**)
  - granularità (statement-level o row-level)
  - alias dei valori o tabelle di transizione
  - azione
  - timestamp di creazione

SQL:1999, Sintassi

```
create trigger TriggerName
  { before | after }
  { insert | delete | update [of Column] } on
  Table
  [referencing
      {[old_table [as] OldTableAlias]
       [new_table [as] NewTableAlias] } |
      {[old [row] [as] OldTupleName]
       [new [row] [as] NewTupleName] }]
  [for each { row | statement }]
  [when Condition]
  SQLStatements
```

Tipi di eventi

- BEFORE
  - Il trigger è considerato e possibilmente eseguito prima dell'evento (i.e., la modifica del database)
  - I trigger before non possono modificare lo stato del database; possono al più condizionare i valori "new" in modalità row-level (set t.new=expr)
  - Normalmente questa modalità è usata quando si vuole verificare una modifica prima che essa avvenga e "modificare la modifica"
- AFTER
  - Il trigger è considerato e eseguito dopo l'evento
  - E' la modalità più comune, adatta alla maggior parte delle applicazioni

Esempio "before" e "after"

1. "Conditioner" (agisce prima dell'update e della verifica di integrità)

```
create trigger LimitaAumenti
before update of Salario on Impiegato
for each row
when (New.Salario > Old.Salario * 1.2)
set New.Salario = Old.Salario * 1.2
```

2. "Re-installer" (agisce dopo l'update)

```
create trigger LimitaAumenti
after update of Salario on Impiegato
for each row
when (New.Salario > Old.Salario * 1.2)
UPDATE Impiegato SET Salario = Old.Salario * 1.2 WHERE Matricola = NEW.Matricola
```

Granularità degli eventi

- Modalità statement-level (di default, opzione **`for each statement`**)
  - Il trigger viene considerato e possibilmente eseguito solo una volta per ogni statement (comando) che lo ha attivato, independentemente dal numero di tuple modificate
  - In linea con SQL (set-oriented)
- Modalità row-level (opzione **`for each row`**)
  - Il trigger viene considerato e possibilmente eseguito una volta per ogni tupla modificata
  - Scrivere trigger row-level è più semplice

Clausola `referencing`

- Dipende dalla granularità
  - Se la modalità è row-level, ci sono due *variabili di transizione* (**`old`** and **`new`**) che rappresentano il valore precedente o successivo alla modifica di una tupla
  - Se la modalità è statement-level, ci sono due *tabelle di transizione* (**`old table`** and **`new table`**) che contengono i valori precedenti e successivi delle tuple modificate dallo statement
- **`old`** e **`old_table`** non sono presenti con l'evento **`insert`**
- **`new`** e **`new_table`** non sono presenti con l'evento **`delete`**

Esempio di trigger row-level

```
create trigger AccountMonitor
 after update on Account
 for each row
 when new.Total > old.Total
 insert
    into Payments
    values
     (new.AccNumber,new.Total-old.Total)
```

Esempio di trigger statement-level

```
create trigger FileDeletedInvoices
 after delete on Invoice
 referencing old_table as OldInvoiceSet
 insert into DeletedInvoices
 (select *
  from OldInvoiceSet)
```

Trigger in DB2

- Seguono la sintassi e semantica di SQL:1999
- Esempio: gestione salari

```
CREATE TRIGGER CheckDecrement
AFTER UPDATE OF Salary ON Employee
FOR EACH ROW
WHEN (NEW.Salary < OLD.Salary * 0.97)
BEGIN
  update Employee
  set Salary=OLD.Salary*0.97
  where RegNum = NEW.RegNum;
END;
```

Esecuzione di Trigger in conflitto

- Quando vi sono più trigger associati allo stesso evento (in conflitto) vengono eseguiti come segue:
  - Per primi i BEFORE triggers (statement-level e row-level)
  - Poi viene eseguita la modifica e verificati i vincoli di integrità
  - Infine sono eseguiti gli AFTER triggers (row-level e statement level)

- Quando vari trigger appartengono alla stessa categoria, l'ordine di esecuzione è definito in base al loro timestamp di creazione (i trigger più vecchi hanno priorità più alta)

Modello di esecuzione ricorsivo

- In SQL:1999 i trigger sono associati ad un “Trigger Execution Context” (TEC)
- L’azione di un trigger può produrre eventi che attivano altri trigger, che verranno valutati con un nuovo TEC interno:
  - Lo stato del TEC includente viene salvato e quello del TEC incluso viene eseguito. Ciò può accadere ricorsivamente
  - Alla fine dell’esecuzione di un TEC incluso, lo stato di esecuzione del TEC includente viene ripristinato e la sua esecuzione ripresa
- L’esecuzione termina correttamente in uno “stato quiescente”
- L’esecuzione termina in errore quando si raggiunge una data profondità di ricorsione dando luogo ad una eccezione di non-terminazione
- Se si verifica un errore o eccezione durante l’esecuzione di una catena di trigger attivati inizialmente da uno statement S, viene fatto un rollback parziale di S

Trigger in Oracle

- Si usa una sintassi differente: sono consentiti eventi multipli, non sono previste variabili per le tabelle, i before trigger possono prevedere update, la condizione è presente solo con trigger row-level, l'azione è un programma PL/SQL

```
create trigger TriggerName
  { before | after } event [, event [,event ]]
   [[referencing
        [old [row] [as] OldTupleName]
        [new [row] [as] NewTupleName] ]
  for each { row | statement } [when Condition]]
  PL/SQLStatements

Event ::= { insert | delete | update [of Column] } on Table
```

Conflitti tra i trigger in Oracle

- Quando molti trigger sono associati allo stesso evento, ORACLE segue il seguente schema:
  - Per primi, i BEFORE statement-level trigger
  - Poi, i BEFORE row-level trigger
  - Poi viene eseguita la modifica e verificati i vincoli di integrità
  - Poi, gli AFTER row-level trigger
  - Infine, gli AFTER statement-level trigger
- Quando vari trigger apparttengono alla stessa categoria, l'ordine di esecuzione è definito in base al loro timestamp di creazione (i trigger più vecchi hanno priorità più alta)
- "Mutating table exception": scatta se la catena di trigger attivati da un before trigger T cerca di modifcare lo stato della tabella target di T

Esempio di Trigger in Oracle

Evento: **update** of QtyDisponibile in Magazzino

Condizione: Quantità sotto soglia e mancanza ordini esterni

Azione: **insert** of OrdiniEsterni

```
create trigger Riordino
after update of QtyDisponibile on Magazzino
for each row
when (new.QtyDisponibile < new.QtySoglia)
begin
  declare X number;
  select count(*) into X
   from OrdiniEsterni
   where Parte = new.Parte;
  if X = 0 then
   insert into OrdiniEsterni
    values(new.Parte,new.QtyRiordino,sysdate)
  end if;
 end;
```

Proprietà formali dei trigger

- E’ importante garantire che l’interferenza tra trigger in una qualunque loro attivazione non produca comportamenti anomali
- Vi sono tre proprietà classiche:
  - Terminazione: per un qualunque stato iniziale e una qualunque transazione, si produce uno stato finale (stato quiescente)
  - Confluenza: L’esecuzione dei trigger termina e produce un unico stato finale, indipendente dall’ordine di esecuzione dei trigger
  - Univoca osservabilità: I trigger sono confluenti e producono verso l’esterno (messaggi, azioni di display) lo stesso effetto
- La terminazione è la proprietà principale

Analisi della terminazione

- Si usa una rappresentazione delle regole detta **grafo di triggering**:
  - Un nodo per ogni trigger
  - Un arco dal nodo t_i al nodo t_j se la esecuzione dell'azione di t_i può attivare il trigger t_j (ciò può essere dedotto con una semplice analisi sintattica)
- Se il grafo è aciclico, l'esecuzione termina
  - Non possono esservi sequenze infinite di triggering
- Se il grafo ha cicli, esso *può* avere problemi di terminazione: lo si capisce guandando i cicli uno per uno.

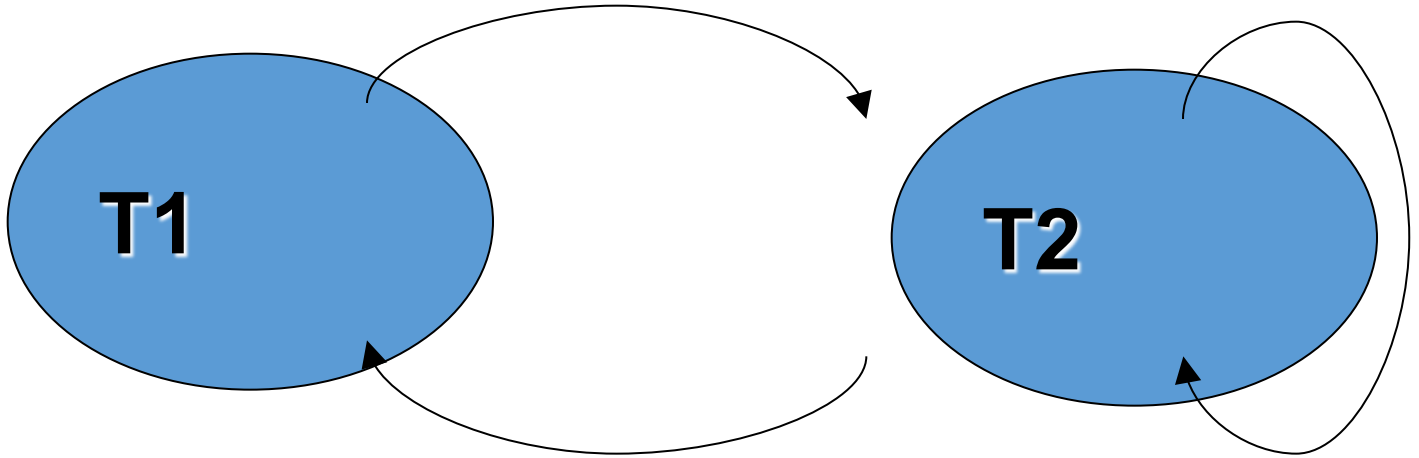
Esempio con due trigger

```
T1:   create trigger AdjustContributions
      after update of Salary on Employee
      referencing new table as NewEmp
      update Employee
      set Contribution = Salary * 0.8
      where RegNum in ( select RegNum
                        from NewEmp)



T2:   create trigger CheckBudgetThreshold
      after update on Employee
      referencing new_table as NewEmp1
      when 50000 < ALL (select (Salary+Contribution)
                        from NewEmp1)
      update Employee
      set Salary = 0.9*Salary
```

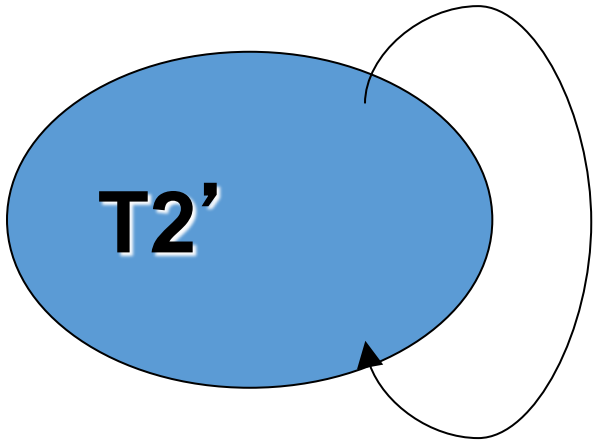
Grafo di triggering corrispondente



- Ci sono due cicli ma il sistema termina.
- Per renderlo non terminante basta cambiare il comparatore nella condizione di T2 oppure moltiplicare per un fattore più grande di 1 nella azione di T2.

Esempio di non terminazione

```
T2': create trigger CheckBudgetThreshold
     after update on Employee
     for each row
     when New.Salary < 50000
     update Employee
     set Salary = 0.9*Salary
```



Esercizio 1

- Dato lo schema relazionale:
  - IMPIEGATO (Nome, Salario, DipNum)
  - DIPARTIMENTO (DipNum, NomeManager)
- Definire le seguenti regole attive in Oracle e DB2:

1. una regola, che quando il dipartimento cancellato, mette ad un valore di default (99) il valore di DipNum degli impiegati appartenenti a quel dipartimento;
2. una regola che cancella tutti gli impiegati appartenenti a un dipartimento quando quest'ultimo cancellato;
3. una regola che, ogni qual volta il salario di un impiegato supera il salario del suo manager, pone tale salario uguale al salario del manager;
4. una regola che, ogni qual volta vengono modificati i salari, verifica che non vi siano dipartimenti in cui il salario medio cresce più del tre per cento, e in tal caso annulla la modifica.

```
create trigger T1
after delete on DIPARTIMENTO
     for each row
     when (exists (select *
                   from IMPIEGATO
                   where DipNum=Old.DipNum))
udpade IMPIEGATO set DipNum = 99 where
DipNum=Old.DipNum
```

```
create trigger T2
after delete on DIPARTIMENTO
for each row
    when (exist (select *
                 from IMPIEGATO
                 where DipNum=Old.DipNum))
delete from IMPIEGATO where DipNum=Old.DipNum
```

```
create trigger T3
after update of Salario on IMPIEGATO
for each row
	declare x number;
	begin
		select Salary into x
		from IMPIEGATO join DIPARTIMENTO on
		Nome = NomeManager
		Where DIPARTIMENTO.DipNum = New.DipNum
		if new.Salario > x then
			update IMPIEGATO set Salario = x
			where Nome = New.Nome
	end
```

```
create trigger T4
after update of Salario on IMPIEGATO
for each row
        declare x number;
        declare y number;
        declare l number;
        begin
                select avg(salario), count(*) into x, l
                from IMPIEGATO
                where DipNum=new.DipNum;
                y=((x*l)-new.Salario+old.Salario)/l;
                if (x>(y*1.03)) then
                        update IMPIEGATO set Salario=old.Salario
                        where DipNum=new.DipNum AND nome=new.nome;
        end
```

```
create trigger T4
after update of Salario on IMPIEGATO
for each row
        declare x number;
        declare y number;
        begin
                select SUM(salario) into x
                from IMPIEGATO
                where DipNum=new.DipNum;
                y=x-new.Salario+old.Salario;
                if (x/y>1.03) then
                        update IMPIEGATO set Salario=old.Salario
                        where DipNum=new.DipNum AND nome=new.nome;
        end
```

```
UPDATE impiegato
SET salario = v
WHERE nome=n1 AND dipnum=d1
```

```
create trigger T4
after update of Salario on IMPIEGATO
for each row
when((select SUM(salario)
          from IMPIEGATO
          where DipNum=new.DipNum ) /
      (select SUM(salario)-new.Salario+old.Salario
          from IMPIEGATO
          where DipNum=new.DipNum ) > 1.03)
update IMPIEGATO set Salario=old.Salario
where DipNum=new.DipNum AND nome=new.nome;
end
```

```
create trigger T4
after update of Salario on IMPIEGATO
referencing old as oldSalary;
referencing new as newSalary;
Declare x,l number;
Declare d number;
Declare oldSal, newSal number;
Begin
        SELECT DISTINCT dipnum into d FROM newSalary;
        SELECT SUM(salario) into oldSal FROM oldSalary;
        SELECT SUM(salario) into newSal FROM newSalary;
        SELECT AVG(salario), count(*) into x, l
        from IMPIEGATO
        where DipNum=d;
        y=((x*l)-newSal+oldSal)/l;
        if (x>(y*1.03)) then
                DELETE FROM impiegato WHERE dipnum = d;
                INSERT INTO impiegato as (select * from oldSalary);
end
```

```
UPDATE impiegato
SET salario = v
WHERE dipnum=d1
```

Esercizio 2

- Riferendosi alla base di dati dell'esercizio precedente:
  - definire in DB2 e in Oracle un trigger R1 che, quando cancellato un impiegato che svolge il ruolo di manager di un dipartimento, cancella quel dipartimento e tutti i suoi dipendenti.
  - Definire inoltre un trigger R2 che, ogni qual volta sono modificati i salari, verifica la loro media, e se essa supera i 50.000 cancella tutti gli impiegati il cui salario è stato modificato e attualmente supera gli 80.000.

Esercizio 3

- Dato lo schema relazionale:
  - DOTTORANDO (Nome, Disciplina, Relatore)
  - PROFESSORE (Nome, Disciplina)
  - CORSO (Titolo, Professore)
  - ESAMI (NomeStud, TitoloCorso)
- Descrivere i trigger che gestiscono i seguenti vincoli di integrità (business rules):
  1. ogni dottorando deve lavorare nella stessa area del suo relatore;
  2. ogni dottorando deve aver sostenuto almeno 3 corsi nell'area del suo relatore;
  3. ogni dottorando deve aver sostenuto l'esame del corso di cui è responsabile il suo relatore.