Esercizi sulle Regole attive

Transazioni

Dato il seguente schema:

TITOLO (CodTitolo, Nome, Tipologia)

TRANSAZIONE (CodTrans, CodVenditore, CodAcquirente, CodTitolo, Qta, Valore, Data, Istante)

OPERATORE (Codice, Nome, Indirizzo, Disponibilità)

Costruire un sistema di trigger che mantenga aggiornato il valore di Disponibilità di Operatore in seguito all'inserimento di tuple in Transazione, tenendo conto che per ogni transazione in cui l'operatore vende l'ammontare della transazione deve far crescere la disponibilità e per ogni acquisto deve invece diminuire.

Inserire inoltre gli operatori la cui disponibilità scende sotto lo zero in una tabella che elenca gli operatori “scoperti”. Ipotizzando che esista

SCOPERTO (Codice, Nome, Indirizzo)

Transazioni

```
Create trigger TrasferisciAmmontare
after insert on TRANSAZIONE
for each row
begin
   update OPERATORE
   set Disponibilità = Disponibilità – new.Qta*new.Valore
   where Codice = new.CodAcquirente;
   update OPERATORE
   set Disponibilità = Disponibilità + new.Qta*new.Valore
   where Codice = new.CodVenditore;
end
```

Transazioni

```
CreateTrigger DenunciaScoperto
after update of Disponibilità on Operatore
for each row
when new.Disponibilità < 0 and old.Disponibilità >= 0
begin
   insert into OperatoreScoperto values (new.Codice, new.Nome,
                                                    new.Indirizzo)
end
```

Transazioni

```
CreateTrigger RitiraDenuncia
after update of Disponibilità on Operatore
for each row
when old.Disponibilità < 0 and new.Disponibilità >= 0
begin
   delete from OperatoreScoperto where Codice = new.Codice
end
```

Esercizio - Viaggi

Facendo riferimento alla base dati:

DIPENDENTE (codice, nome, qualifica, settore)

VIAGGIO (codice, dip, destinazione, data, km, targa-auto)

AUTO (targa, modello, costo-km)

DESTINAZIONE (nome, stato)

Si consideri la vista:

VIAGGI (dipendente, km-tot, costo-totale)

Scrivere due regole attive che calcolano il valore della vista a seguito di inserzioni di nuovi viaggi, una in modo incrementale e una ricalcolando l'intera vista.

```
Create trigger CalcolaVistaIncrementale
after insert into VIAGGIO
for each row
begin

    update VIAGGI
    set km-tot = km-tot + new.km
        costo-totale= costo-totale + ( select new.km * costo-km
                                       from AUTO
                                       where targa = new.targa-auto )
    where dipendente = new.dip

end
```

```
Create trigger CalcolaVistaOneShot
after insert into VIAGGIO
for each statement
begin

    delete from VIAGGI;

    insert into VIAGGI
    select dip, sum( km ), sum( km * costo-km )
    from VIAGGIO join AUTO on ( targa-auto = targa )
    group by dip;

end
```

Campus

Dato il seguente schema relazionale:

STUDENTI (Matr,Nome,Residenza,Telefono,
CorsoDiLaurea,AnnoCorso,Sede,TotCrediti)

ISCRIZIONI (MatrStud, Corso, Anno, Data)

CORSIANNI (CodCorso, Anno, Docente,
Semestre, NroStudenti, NroFuoriSede)

ABBINAMENTI (CodCorso, CorsoLaurea)

CORSI (CodCorso, Titolo, Crediti, Sede)

https://goo.gl/hW18QA

1

Scrivere una regola attiva che controlla ogni inserimento di una tupla nella tabella ISCRIZIONI e nel caso in cui non esistano elementi corrispondenti in STUDENTI o CORSIANNI, [cioè l'iscrizione non sia una iscrizione significativa, perché lo studente è sconosciuto o il corso non attivo] annulli l'inserimento.

```
create trigger CheckIntRef
before insert on Iscrizioni
for each row
when not exists (  select *
                   from Studenti
                   where Matr = new.MatrStud )
     OR not exists
          ( select *
            from CorsiAnni
            where CodCorso = new.Corso
                  and Anno = new.Anno )
Rollback;
```

https://goo.gl/tj2HHJ

2

Supponendo che l'attributo NroFuoriSede di CORSIANNI rappresenti il numero di studenti iscritti al corso che fanno riferimento a una sede diversa da quella del corso, scrivere una regola attiva che reagisce alle modifiche dell'attributo Sede di STUDENTE, aggiornando se necessario il valore dell'attributo Nro-FuoriSede.

La modifica della sede comporta due variazioni: lo studente diventa “fuori sede” per tutti i corsi (a cui è iscritto) erogati nella sede abbandonata, e diventa “in sede” per tutti i corsi (a cui è iscritto) erogati nella nuova sede dello studente (per i quali prima era contato tra i “fuori sede”).
Sono due, quindi, i contatori da aggiornare:

```
create trigger CheckSede1
after update of Sede on Studenti
for each row
begin
  update CorsiAnni
  set NroFuoriSede = NroFuoriSede + 1
  where (Corso,Anno) in (select Corso, Anno
                         from Iscrizioni
                         where MatrStud=old.Matr)
        and Corso in ( select CodCorso
                       from Corsi
                       where Sede = old.Sede);
end
```

```
Create trigger CheckSede2
after update of Sede on Studenti
for each row
begin
update CorsiAnni
  set NroFuoriSede = NroFuoriSede - 1
  where (Corso,Anno) in (select Corso,Anno
                         from Iscrizioni
                         where MatrStud=new.Matr)
            and Corso in ( select CodCorso
                           from Corsi
                           where Sede = new.Sede);
end
```

L’uso corretto di old e new è cruciale per distinguere i due valori dell’attributo “Sede”, mentre è indifferente per “Matr” (che non cambia).
Si noti anche che un eventuale update che riassegni a sede il suo stesso valore lascerebbe tutti i contatori inalterati.

https://goo.gl/H96ruw

Non tutti i database permettono la clausola IN applicata su colonne multiple.

Spesso è però possibile concatenare stringhe, quindi si potrebbe fare:

where Corso || Anno) in (select
Corso || Anno …

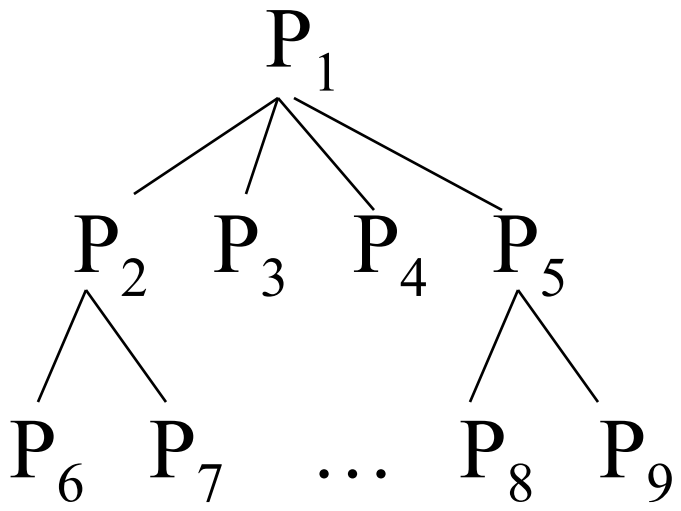
Si valutino però sempre le implicazioni di questa strategia.

Prodotti

Lo schema seguente descrive un insieme gerarchico di prodotti, dove per i prodotti non contenuti in altri prodotti assumiamo Livello (che descrive il livello di profondità a cui il prodotto è situato nella gerarchia) pari a zero e SuperProdotto pari a null:

Product (Code, Name, Description, SuperProduct, Level)

Esempio di gerarchia:



https://goo.gl/HMpKEw

1

Scrivere una regola attiva che alla cancellazione di un prodotto *cancelli tutti i sottoprodotti corrispondenti*

```
create trigger DeleteProduct
after delete on Product
for each row
delete from Product
where SuperProduct = old.Code
```

La cancellazione di **un** prodotto fa scattare subito un'attivazione della regola che cancella tutti i suoi sottoprodotti diretti.

Ognuna di queste cancellazioni, a sua volta, origina un'attivazione della stessa regola, per cancellare i sotto-sottoprodotti, e così via ricorsivamente, fino a cancellare tutti i prodotti discendenti di quello cancellato originariamente.

https://goo.gl/JS9Gzt

2

Scrivere una regola attiva che alla creazione di un nuovo prodotto (*eventualmente* "figlio" di un prodotto esistente) calcola il valore dell'attributo **Livello**.

```
create trigger ProductLevel
after insert on Product
for each row
begin

  update Product
  set Level = 1 + ( select Level
                    from Product
                    where Code = new.SuperProduct )
   where Code = new.Code and new.SuperProduct is not null;

   update Product
   set Level = 0
   where Code = new.Code and new.SuperProduct is null;

end;
```

https://goo.gl/Vh7yLT

Alcuni accorgimenti:

•Occorre trattare separatamente il caso dell'inserimento di un prodotto "radice" (Level = 0) e di un prodotto contenuto in un atro prodotto.

•La condizione "Code=**new**.Code" serve a individuare la tupla appena inserita (che deve essere modificata).

•ATTENZIONE!! "set **new**.Level = …" sarebbe un grave errore! "**new**" non è un puntatore all'oggetto creato.

•Si noti che con questa regola il sistema gestisce interamente in automatico il calcolo del livello di profondità, e in una eventuale tupla inserita con un livello non coerente con quello del predecessore il valore errato verrebbe sovrascritto.

Se permettessimo l'aggiornamento dell'attributo *SuperProduct,* (cambiando la gerarchia nel tempo) dobbiamo anche cambiare *Level*

```
create trigger TrigSupPro_A
after update of SuperProduct on Product
when new.SuperProduct is null
 update Product set Level = 0
   where Code = old.Code

create trigger TrigSupPro_B
after update of SuperProduct on Product
when new.SuperProduct is not null
 update Product set Level = 1 + ( select Level
                                  from Product
                                  where Code = new.SuperProduct)
   where Code = old.Code
```

https://goo.gl/bvyEh3

Oppure possiamo utilizzare dei trigger *before*:

```
create trigger ProductLevel_Before_childProduct
before insert into Product
for each row
when new.SuperProduct is not null
set new.Level = 1 + ( select Level
                      from Product
                      where Code = new.SuperProduct )
```

```
create trigger ProductLevel_Before_rootProduct
before insert into Product
for each row
when new.SuperProduct is null
set new.Level = 0
```

E quindi propagare le modifiche:

```
create trigger TrigSupPro_A
after update of Level on Product
begin
   update Product set Level = 1 + new.Level
   where SuperProduct = old.Code
end
```

Questo trigger si attiva ricorsivamente "a cascata" e aggiorna tutti i sottoprodotti. ((N.B. l'azione e l'evento sono uguali)
La computazione termina se e solo se non ci sono cicli nei dati (l'albero è un albero correttamente strutturato)

https://goo.gl/bvyEh3

3

Implementare mediante regole attive un vincolo di pseudo-integrità referenziale sull'attributo superProduct, per cui gli unici valori ammessi sono:

- Null
- il codice di un *altro* prodotto (occorre vietare anche che un prodotto sia SuperProdotto di se stesso).

```
create trigger PseudoCostraint
after insert on Product
for each row
when new.SuperProduct is not null and
        new.SuperProduct not in  ( select Code
                                   from Product )
rollback;
```

https://goo.gl/v2uonr

Questa formulazione ancora NON impedisce a un prodotto di essere Super di se stesso (l'inserimento di un tale prodotto vedrebbe prima l'aggiunta della tupla alla tabella e poi [il modo è after] l'attivazione della regola, quindi la query annidata restituirebbe anche il Codice appena inserito). Si può introdurre una regola a parte:

```
create trigger PseudoConstraintSpecial
after insert on Product
for each row
when new.SuperProduct = new.Code
rollback;
```

https://goo.gl/sZIx7E

Altre opzioni:

- Aggiungere alla query annidata nella prima regola: “where Code <> **new**.Code”.
- Aggiungere “**new**.SuperProduct = **new**.Code” (cioè la condition della seconda regola) in OR alla clausola when della prima.
- Utilizzare la semantica "before" (così la query annidata non restituirebbe il Codice preso dalla tupla in via di inserimento, proprio perché la regola scatta prima).

Si noti che le regole definite non impediscono la creazione di riferimenti circolari, a meno di disabilitare gli updates.

Infatti, se il database è consistente (l'albero della gerarchia è ben formato) al momento della definizion dei trigger, e solo inserimenti e cancellazioni sono permessi (no updates), non c'è modo di introdurre dei cicli.

https://goo.gl/w9kF8k