Esercizi sulle Regole attive

Università

Facendo riferimento alla base dati:

DOTTORANDO (Nome, Disciplina, Relatore)

PROFESSORE (Nome, Disciplina)

CORSO (Titolo, Professore)

ESAMI (NomeStud, TitoloCorso)

Descrivere i trigger che gestiscono i seguenti vincoli di integrità:

1. Ogni dottorando deve lavorare nella stessa area del suo relatore;
2. Ogni dottorando deve aver sostenuto l'esame del corso di cui è responsabile il suo relatore;
3. Ogni dottorando deve aver sostenuto almeno 3 corsi nell'area del suo relatore.

1. Ogni dottorando deve lavorare nella stessa area del suo relatore

```
create trigger T1
  after update of Disciplina on DOTTORANDO
  for each row
  when Disciplina < > ( select Disciplina
                          from PROFESSORE
                         where PROFESSORE.Nome = new.Relatore)
 begin
   select raise(ABORT, “Disciplina sbagliata”);
 end
```

Per questo ed i seguenti 2 triggers, sarebbe necessario anche il trigger che verifica i dati sull’evento INSERT.

2. Ogni dottorando deve aver sostenuto l'esame del corso di cui è responsabile il suo relatore.

```
create trigger T2
  after update of Relatore on DOTTORANDO
  for each row
  when not exists ( select *
        from ESAME join CORSO on TitoloCorso = Titolo
        where NomeStud = new.Nome and
        Professore = new.Relatore )
 begin
   select raise(ABORT, ("Esame mancante");
 end
```

3. Ogni dottorando deve aver sostenuto almeno 3 corsi nell'area del suo relatore

```
create trigger T3
  after update of Disciplina on DOTTORANDO
  for each row
  when 3 < ( select count(*)
        from ESAMI join CORSO on TitoloCorso = Titolo
        join PROFESSORE on Professore = PROFESSORE.Nome
        join DOTTORANDO on NomeStud = DOTTORANDO.Nome
        join PROFESSORE as P2 on Relatore = P2.Nome
       where PROFESSORE.Disciplina = P2.Disciplina
       and DOTTORANDO.Nome=new.Nome )
 begin
   select raise(ABORT, ("Almeno 3 corsi");
 end
```

Borse di studio

Si vuole gestire attraverso un sistema di trigger l'assegnazione di borse di studio agli studenti di laurea magistrale. Le borse sono assegnate agli studenti che ne fanno domanda e che alla data della domanda abbiano sostenuto esami per almeno 50 cfu, con media non inferiore a 27/30.

- Se i requisiti non sono soddisfatti, la domanda è automaticamente respinta; altrimenti accolta.
- In entrambi i casi viene modificato il valore del campo *stato* in **DOMANDABORSA** (inizialmente sempre a NULL), rispettivamente con "respinta" o "accolta".
- In caso di accoglimento, allo studente è automaticamente assegnata una posizione in graduatoria, determinata dalla media dei voti; in caso di parità di media, si considerano dapprima il maggior numero di cfu sostenuti alla data della domanda e infine l'ordine di inserimento delle domande.
- Se uno studente rinuncia alla borsa (*stato* viene modificato in "rinuncia"), la graduatoria viene aggiornata.

Si gestisca il contenuto di **DOMANDABORSA** e **GRADUATORIA** a seguito degli inserimenti di nuove domande e alle eventali rinunce.

DOMANDABORSA (Matricola, DataDomanda, stato)

CORSO (CodCorso, NomeCorso, NumeroCrediti)

GRADUATORIA (Matricola, Media, Cfu, Posizione)

ESAME (CodCorso, Matricola, Data, Voto)

```
Create trigger ValutaDomanda
after insert on DomandaBorsa
for each row
declare M, C number:
M := (  select avg(Voto) from Esame
          where  Matricola=new.Matricola and Data<= new.DataDomanda )
C := (   select sum(NumeroCrediti) from  Esame JOIN Corso ON
          Esame.CodCorso=Corso.CodiceCorso
          where  Matricola=new.Matricola and Data<= new.DataDomanda )
begin
if (M >= 27 and C >= 50)
 then  (
          update DomandaBorsa set stato=”accolta” where Matricola=new.Matricola;
          insert into Graduatoria values (new.Matricola, M, C, NULL);    )
 else
          update DomandaBorsa set stato=”respinta” where Matricola=new.Matricola;
end

Create trigger RitiraDomanda
after update of stato  on DomandaBorsa
for each row
when new.stato = “rinuncia”
begin
     delete * from Graduatoria where Matricola=new.Matricola;
end
```

```
Create trigger AggiornaGraduatoria1
after insert on Graduatoria
for each row
begin
POS:=select count(*)  // calcola quanti studenti hanno media maggiore o
                      // stessa media e più cfu
        from Graduatoria
        where(new.Media < Media) OR (new.Media=Media AND new.Cfu<Cfu)
                                 OR (new.Media=Media AND new.Cfu=Cfu)

/* l'ordine di inserimento delle domande viene così  implicitamente
considerato, privilegiando a pari media e cfu le domande più "antiche":
le regole vengono processate in ordine rispetto all'istante di
inserimento delle domande, quindi la posizione assegnata sarà sempre
l'ultima tra gli studenti con stessa media e stessi cfu */

update Graduatoria set Posizione=Posizione+1 where Posizione > POS;
//sposta in avanti quelli successivi

update Graduatoria set Posizione=POS+1 where Matricola=new.Matricola:
//inserisce in posizione POS+1 il "nuovo" studente
end
```

```
Create trigger AggiornaGraduatoria2
after delete From Graduatoria
for each row
begin
    // modifica la posizione di tutti quelli che lo succedevano
    // in graduatoria

    update Graduatoria set Posizione=Posizione-1
    where Posizione>old.Posizione;
end
```

Classifica

- Si consideri la base di dati:

  RISULTATO (Giornata, SquadraCasa, SquadraOspite, GoalCasa, GoalOspite)

  CLASSIFICA (Giornata, Squadra, Punti)

- Assumendo che la prima tabella sia alimentata tramite inserimenti e che la seconda sia opportunamente derivata dalla prima, scrivere le regole attive che costruiscono la classifica, attribuendo 3 punti alle squadre che vincono, 1 punto a quelle che pareggiano e 0 punti a quelle che perdono.

Classifica

```
create trigger VittoriaInCasa
after insert on RISULTATO
when new.GoalCasa > new.GoalOspite
for each row
begin
  insert into CLASSIFICA C
  select new.Giornata, new.Squadra-Casa, C2.Punti + 3
  from CLASSIFICA C2
  where C2.Squadra = new.Squadra and not exists ( select * from CLASSIFICA
        where Giornata > C2.Giornata )
  insert into CLASSIFICA C
  select new.Giornata, new.Squadra-Ospite, C2.Punti
  from CLASSIFICA C2
  where C2.Squadra = new.Squadra and not exists ( select * from CLASSIFICA
        where Giornata > C2.Giornata )
end
```

Classifica

```
create trigger VittoriaFuoriCasa
after insert on RISULTATO
when new.GoalCasa < new.GoalOspite
for each row
begin
  insert into CLASSIFICA C
  select new.Giornata, new.Squadra-Casa, C2.Punti
  from CLASSIFICA C2
  where C2.Squadra = new.Squadra and not exists ( select * from CLASSIFICA
        where Giornata > C2.Giornata )
  insert into CLASSIFICA C
  select new.Giornata, new.Squadra-Ospite, C2.Punti + 3
  from CLASSIFICA C2
  where C2.Squadra = new.Squadra and not exists ( select * from CLASSIFICA
        where Giornata > C2.Giornata )
end
```

Classifica

```
create trigger Pareggio
after insert on RISULTATO
when new.GoalCasa = new.GoalOspite
for each row
begin
  insert into CLASSIFICA C
  select new.Giornata, new.Squadra-Casa, C2.Punti + 1
  from CLASSIFICA C2
  where C2.Squadra = new.Squadra and not exists ( select * from CLASSIFICA
        where Giornata > C2.Giornata )
  insert into CLASSIFICA C
  select new.Giornata, new.Squadra-Ospite, C2.Punti + 1
  from CLASSIFICA C2
  where C2.Squadra = new.Squadra and not exists ( select * from CLASSIFICA
        where Giornata > C2.Giornata )
end
```

Partite

Si consideri il seguente schema relazionale, relativo al campionato europeo di pallavolo:

GIOCATORE (NumTessera, Nome, Squadra, Altezza, DataNascita, PresenzeInNazionale)

SQUADRA (Nazione, Allenatore, NumPartiteVinte)

PARTITA (IdPartita, Data, Squadra1, Squadra2, SetVintiSquadra1, SetVintiSquadra2, Arbitro)

PARTECIPAZIONE (IdPartita, TesseraGiocatore, Ruolo, PuntiRealizzati)

Partite

1. Costruire un trigger che mantenga aggiornato il valore di NumPartiteVinte di Squadra in seguito agli inserimenti in Partita, tenendo conto che NumPartiteVinte è relativo a tutta la storia della nazionale, non solo al campionato corrente, e che una squadra vince una partita quando vince 3 set.

2. Costruire inoltre un trigger che tenga aggiornato il numero delle presenze dei giocatori.

Partite

```
create trigger IncrementaVittorie
after insert on PARTITA
for each row
begin
update SQUADRA
    set NumPartiteVinte = NumPartiteVinte + 1
    where
        new.SetVintiSquadra1=3 and Nazione=new.Squadra1 or
        new.SetVintiSquadra2=3 and Nazione=new.Squadra2
end
```

Partite

```
create trigger IncrementaPresenze
after insert on PARTECIPAZIONE
for each row
begin
    update GIOCATORE
        set PresenzeInNazionale = PresenzeInNazionale + 1
        where NumTessera = new.TesseraGiocatore
end
```

Bollette

Un database gestisce le bollette telefoniche di una compagnia di telefonia mobile.

CLIENTE (CodiceFiscale, nome, cognome, numTelefonico, PianoTariffario)

PIANOTARIFFARIO (Codice, costoScattoAllaRisposta, costoAlSecondo)

TELEFONATA (CodiceFiscale, Data, Ora, numeroDestinatario, durata)

BOLLETTA (CodiceFiscale, Mese, Anno, importo)

Bollette

- Scrivere un trigger che a seguito di ogni telefonata aggiorna la bolletta del cliente che ha chiamato.

- Facciamo l'ipotesi che le bollette da aggiornare siano sempre già presenti nella base di dati, demandando a un altro trigger la creazione di una bolletta di importo 0 per ogni cliente registrato all'inizio di ogni mese.

Bollette

T1. Scrivere un trigger che a seguito di ogni telefonata aggiorna la bolletta del cliente che ha chiamato.

T2. Facciamo l'ipotesi che le bollette da aggiornare siano sempre già presenti nella base di dati, demandando a un altro trigger la creazione di una bolletta di importo 0 per ogni cliente registrato all'inizio di ogni mese.

(si supponga che la fine del mese sia segnalata dall'evento END_MONTH)

Bollette – T2

```
Create Trigger CominciaMese
after END_MONTH
begin
    insert into BOLLETTA
    select CodiceFiscale, sysdate().month, sysdate().year, 0
    from CLIENTE
end
```

Bollette – T1

```
Create trigger AddebitaChiamata
after insert of TELEFONATA
for each row
begin
  update BOLLETTA B
  set importo = importo + ( select PT.costoScattoAllaRisposta +
                            PT.costoAlSecondo * new.durata
                            from PIANOTARIFFARIO PT join Cliente C
                                 on C.PianoTariffario = PT.Codice
                            where new.CodiceFiscale = C.CodiceFiscale )
  where B.CodiceFiscale = new.CodiceFiscale
     and B.Anno = new.Data.year and B.Mese = new.Data.month
end
```

Bollette

T3. Scrivere un trigger che alla fine di ogni mese (si supponga che la fine del mese sia segnalata dall'evento END_MONTH) sconti dalle bollette 5 centesimi per ogni telefonata diretta a utenti della compagnia (cioè verso numeri di utenti registrati nella tabella **CLIENTE**) se l'importo complessivo della bolletta mensile supera i 100 euro.

Bollette – T3

```
Create Trigger Promozione
after END_MONTH
begin
    update BOLLETTA B
    set importo = importo – 0,05 * ( select count(*)
        from TELEFONATA T
        where T.CodiceFiscale = B.CodiceFiscale
        and T.Data.month = ( sysdate() – 1 ).month
        and T.Data.year = ( sysdate() – 1 ).year
        and T.NumeroDestinatario in ( select numTelefonico
                                      from CLIENTE ) )
    where B.importo > 100 and B.anno = (sysdate() - 1).year
and B.mese = (sysdate() - 1).month
end
```

Play with me

- Si consideri il seguente schema relativo a un sistema di noleggio di sale prove per gruppi musicali.
- Le indicazioni orarie di inizio e fine sono relative a ore intere (8:00, 21:00, ..., ma non 8:15).
- L'uso effettivo può avvenire solo in corrispondenza di una prenotazione, e al limite iniziare dopo o terminare prima degli orari della prenotazione.
- Inoltre, tutte le sale prove aprono alle 7:00 e chiudono tassativamente alle 24:00.

Cliente (CodiceFiscale, Nome, Cognome, Tipo)
Prenotazione (CodFisCliente, CodiceSala, Giorno, OraInizio, OraFine)
UsoEffettivo (CodFisCliente, CodiceSala, Giorno, OraInizio, OraFine, Costo)
Sala (Codice, CostoOrario)

1) Si scriva un trigger che impedisce di prenotare una sala già prenotata

Play with me

Si deve intercettare l'esistenza di una prenotazione per la stessa sala che sia temporalmente sovrapposta. Possiamo usare una semantica "before" per imporre da subito il rollback della transazione.

N.B.: due intervalli di tempo sono sovrapposti se uno inizia prima della fine e finisce dopo l'inizio dell'altro.

```
create trigger TuNonPuoiPrenotare
before insert into Prenotazione
for each row
when exists ( select *
                from Prenotazione
                where CodiceSala = new.CodiceSala and Giorno = new.Giorno and
                        OraInizio < new.OraFine and OraFine > new.OraInizio )
do
  rollback0
```

Play with me

Si supponga che i dati sull'uso effettivo siano inseriti solo *al termine* dell'uso della sala. Si propongano un arricchimento dello schema per tener traccia del numero di ore prenotate e non utilizzate da ogni cliente, e un insieme di regole che assegni il valore "Inaffidabile" al campo "tipo" dei clienti all'accumulo di 50 ore prenotate e non usate.

- *Teoricamente si può ricalcolare il numero di ore "sprecate" con una query ad ogni verifica, senza alterare lo schema.*
- *Considerazioni di efficienza però suggeriscono di calcolare incrementalmente il numero di sprechi di ogni cliente. Ad esempio aggiungendo un attributo "OreSprecate" alla tabella Cliente.*
- *La corrispondenza tra usi effettivi e prenotazioni si sancisce assumndo che il trigger precedente garantisca la correttezza delle prenotazioni e che gli usi effettivi non violino mai i vincoli delle prenotazioni (solo chi ha prenotato può presentarsi, la prenotazione c'è di sicuro, e l'uso non inizia prima e non termina dopo gli orari indicati).*

Play with me

```
create trigger AggiornaOreSprecate
after insert into UsoEffettivo
for each row
 update Cliente
 set OreSprecate = OreSprecate +
                        ( select OraFine – OraInizio – ( new.OraFine – new.OraInizio )
                          from Prenotazione
                          where CodiceSala=new.CodiceSala and Giorno = new.Giorno
                              and OraInizio<= new.OraInizio and  OraFine >= new.OraFine )
 where CodiceFiscale = new.CodFisCliente
```

Questa è la soluzione più semplice: aggiorna sempre il campo, eventualmente con un contributo pari a 0 se l'utilizzatore è stato puntuale.

Si può aggiungere una clausola when per intercettare i casi di contributo nullo e non effettuare l'aggiornamento

Play with me

Resta poi solo da intercettare il limite di 50 ore:

```
create trigger AggiornaTipoCliente
after update of OreSprecate on Cliente
for each row
when old.OreSprecate < 50 and new.OreSprecate >= 50
do
    update Cliente
    set Tipo = “Inaffidabile”
    where CodiceFiscale = old.CodiceFiscale
```

ATTENZIONE: manca del tutto il contributo di chi non si presenta affatto

Play with me

How to deal with users who don't show up at all?

- We may not accept new any new reservation for users who didn't show up in the past (but this would be a new business rule – we should simply try to count the hours as wasted hours... What we miss, in this case, is the triggering event)
- We may consider a new reservation a triggering event and, before reserving, check for previous "dangling" reservations
  - And delete them, once dealt with (in order not to count them again in the future)
  - Or, more likely, in order not to delete potentially useful data, mark them with a flag that needs to be added to the schema
- Most likely (and simply), periodically check for these situations → we need a triggering event that is not a data modification, but rather a system event (example: check all reservations daily, after the change of date), as in the following trigger

Play with me

```
create trigger GhostMusicians
after change-date()      // each vendor has its own extended event language
do
 update User U
  set WastedHours = WastedHours +
                    ( select sum( P.EndTime – P.StartTime )
                      from Reservation P
                      where P.Date = today()–1 and P.UserSSN = U.SSN
                         and not exists( select *
                                         from Usage S
                                         where S.Date = P.Date
                                               and P.StartTime <= S.StartTime
                                               and S.EndTime >= P.EndTime ) )
end;
```

Also note that this solution, by means of aggregation, also accounts for the case in which the same user has left more than one pending reservation in the same day.

Società

Si consideri la seguente tabella, relativa al possesso di quote azionarie di società:

POSSIEDE (Società1, Società2, Percentuale)

Si consideri, per semplicità, che le tuple della tabella POSSIEDE siano inserite a partire da una tabella vuota, che poi non viene più modificata. Si costruisca tramite trigger la relazione CONTROLLA (una società A controlla una società B se A possiede **direttamente o indirettamente** più del 50% di B).

Si assuma che le percentuali di controllo siano numeri decimali, e si tenga presente che la situazione di controllo avviene in modo *diretto* quando una società possiede più del 50% della "controllata" o *indiretto* quando una società controlla altre società che possiedono percentuali di B e la somma delle percentuali possedute e controllate da A supera il 50%.

Schema della tabella ausiliaria:

CONTROLLA (SocControllante, SocControllata)

A può controllare B in due modi distinti:

1. *direttamente*: ne possiede più del 50%
2. *indirettamente*: la somma della percentuale di possesso diretto e del possesso mediato (tramite altre società **controllate**) è superiore al 50% (anche se nessuna di tali percentuali supera il 50 %)

Gli inserimenti in POSSIEDE possono essere gestiti con un trigger a granularità di statement che traduce in inserimenti in CONTROLLA i possessi che rappresentano un controllo diretto. Gli inserimenti nella tabella CONTROLLA innescano poi la ricerca dei controlli indiretti.

PRIMA SOLUZIONE

I possessi maggioritari sono direttamente tradotti in controlli

```
create trigger ControlloDiretto
after insert on POSSIEDE
for each statement
do
     insert into CONTROLLA select Società1, Società2
                                  from newTable
                                  where percentuale > 50
```

A fronte di un nuovo controllo, occorre propagare l'eventuale controllo indiretto

```
Create view PercentualeIndiretta( s1, s2, perc ) as
select C.SocControllante, P.Società2, sum( P.Percentuale )
from CONTROLLA C join POSSIEDE P on C.SocControllata = P.Società1
where ( C.SocControllante, P.Società2 ) not in ( select *
                                                 from CONTROLLA )
group by C.SocControllante, P.Società2
```

La view calcola le percentuali di possesso indiretto non ancora rappresentate nella tabella controlla. Il trigger considera *tutte* le percentuali di controllo indiretto perc (incluse quelle implicate dalla nuova tupla in Controlla, ché il trigger è in modo after). Tali percentuali sono sommate alle *eventuali* percentuali di controllo diretto (si fa un left join, per considerare tutte le indirette, e se non c'è una componente diretta per quella coppia di società l'attributo Percentuale avrà valore NULL), e la somma di perc e dell'eventuale componente diretta è confrontata con 50.

```
create rule ControlloIndiretto
after insert on CONTROLLA
for each row
do
    insert into CONTROLLA  select s1, s2
                           from PercentualeIndiretta left join POSSIEDE
                                on s1 = Società1 and s2=Società2
                           where s1 = new.SocControllante and
                                ( Percentuale is NULL and perc > 50 or
                                  Percentuale + perc > 50 )
```

*Si noti quindi che, affinché si verifichi un controllo indiretto, non è necessario un contributo da parte della relazione POSSIEDE (che, se c'è, peraltro è necessariamente inferiore al 50%, o il controllo sarebbe già stato individuate e inserito nella tabella come controllo diretto). È invece necessario un contributo da PercentualeIndiretta, a cui la scelta di usare l'outer join e di controllare separatamente le condizioni con l'**or**.*

```
create rule ControlloIndiretto2
after insert on POSSIEDE
for each row
do
    insert into CONTROLLA  select s1, s2
                           from PercentualeIndiretta left join POSSIEDE
                                on s1 = Società1 and s2=Società2
                           where s1 = new.SocControllante and
                                ( Percentuale is NULL and perc > 50 or
                                  Percentuale + perc > 50 )
```

Questo trigger potrebbe confliggere col precedente, ma il fatto che nella view PercentualeIndiretta non appaiano tuple relative a coppie già inserite in CONTROLLA evita ogni problema.

SECONDA SOLUZIONE

I possessi maggioritari sono direttamente tradotti in controlli

```
create rule ControlloDiretto
after insert on POSSIEDE
for each statement
do
     insert into CONTROLLA select Società1, Società2
                                  from newTable
                                  where percentuale > 50

      if(not exists(select * from CONTROLLA
                    where SocControllante=new.Società1
                          and SocControllata=new.Società1))
            insert into CONTROLLA values (new.Società1, new.Società1)

       if(not exists(select * from CONTROLLA
                    where SocControllante=new.Società2
                          and SocControllata=new.Società2))
            insert into CONTROLLA values (new.Società2, new.Società2)
```

A fronte di un nuovo controllo, occorre propagare l'eventuale controllo indiretto

```
Create view PercentualeDiretta( s1, s2, perc ) as
select C.SocControllante, P.Società2, sum( P.Percentuale )
from CONTROLLA C join POSSIEDE P on C.SocControllata = P.Società1
where ( C.SocControllante, P.Società2 ) not in ( select *
                                                 from CONTROLLA )
group by C.SocControllante, P.Società2
```

La view calcola le percentuali di possesso indiretto non ancora rappresentate nella tabella controlla. Il trigger considera tutte le percentuali di controllo indiretto perc (incluse quelle implicate dalla nuova tupla in Controlla, ché il trigger è in modo after). Tali percentuali sono sommate alle eventuali percentuali di controllo diretto (si fa un left join, per considerare tutte le indirette, e se non c'è una componente diretta per quella coppia di società l'attributo Percentuale avrà valore NULL), e la somma di perc e dell'eventuale componente diretta è confrontata con 50.

```
create rule ControlloIndiretto
after insert on CONTROLLA
for each row
When not exists (Select * from controlla
                 where )
do
    insert into CONTROLLA  select s1, s2
                           from PercentualeDiretta
                           where ( s1 = new.SocControllante or
                                   s2=new.SocControllata )
                                  and perc > 50

create rule ControlloIndiretto
after insert on POSSIEDE
for each row
do
    insert into CONTROLLA  select s1, s2
                           from PercentualeDiretta
                           where ( s1 = new.SocControllante or
                                   s2=new.SocControllata ) and perc > 50
```