Esercizi sulle Regole attive 2

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 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
```

```
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
```

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
```

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
```

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
```

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
```

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
```

Altri Esercizi

Esercizio

- Dato il seguente schema relazionale

  LIBRO(ISBN, Titolo, PrimoAutore, Editore, NpagTotale)

  CAPITOLO(ISN, NUMERO, Titolo, Npag)
- Descrivere in SQL un insieme di trigger che mantengono allineato il numero di pagine totale al variare della composizione dei suoi capitoli. Si assuma che inizialmente il database contenga una sola tupla per ogni libro e nessun capitolo ad esso relativo.

Esercizio

- Si consideri il database:

  CONTO (Codice, Nome, Saldo, Fido)

  TRANSAZIONE (Codice-Conto, Data, Progressivo, Ammontare, Causale)

  SALDO-MENSILE (Codice-Conto, Mese, Saldo)

  WARNING (Codice-Conto, Nome)

- Scrivere le regole attive che al cambio di mese (si supponga il sistema reagisca all'evento change-month()) inseriscono nella tabella SALDO-MENSILE il saldo del mese passato. Scrivere poi un trigger che individui i clienti in warning (cioé quelli per i quali il saldo del mese è inferiore al 10% del saldo del mese precedente). Discutere le proprietà formali del sistema di regole così definito.

Esercizio

- Si consideri il database:

  CORSO (Codice, Nome, Facoltà, Crediti, OreLezione,
  OreEsercitazioni, OreLaboratori)

  DOCENTE (Codice, Nome, Facoltà)

  ASSEGNAMENTI (CodiceCorso, CodiceDocente,
  AnnoAccademico)

- Sapendo che i docenti si dividono in interni ed esterni, e che i docenti esterni sono coloro che hanno l'attributo Facoltà=null, scrivere un trigger che impedisca di assegnare in un certo anno accademico corsi ai docenti esterni finché tutti quelli interni non hanno assegnamenti per un totale di ore pari ad almeno 120.

Esercizio

- Dato il seguente schema (chiave primaria sottolineata):

  CLIENTE(CodCliente, Nome, MinutiTotali, MinutiPremio)

  TELEFONATA(CodCliente, DataInizio, OraInizio, DurataInMinuti, TipoTariffa)

  PREMIO(Codice, SogliaPremio, MinutiPremio)

  RICHIESTA-PREMIO (CodCliente, CodPremio)
- Scrivere il trigger che aggiorna automaticamente, per ogni cliente, i minuti totali di telefonate effettuate (attributo MinutiTotali) a fronte dell'inserimento di una nuova telefonata a suo carico, cioè quando non usufruisce di premi. Descrivere poi un insieme di trigger per la gestione automatica dei premi che, a fronte di una richiesta di premio da parte di un utente oppure a fronte di una nuova telefonata, verifica se l'utente ha raggiunto la soglia di minuti necessaria per ricevere il premio, e in caso positivo attribuisce all'utente minuti aggiuntivi (MinutiPremio) togliendo però al suo totale dei minuti effettuati i minuti relativi alla soglia del premio, eliminando la richiesta così soddisfatta. Scrivere infine un trigger (o modifica un trigger preesistente) che, a fronte di una telefonata fatta da un cliente che usufruisce di un premio, sottrae la durata della telefonata dai suoi minuti premio, fino all'esaurimento del premio stesso. Discutere la terminazione del sistema di trigger.

Esercizio

- Si ha lo schema seguente, che descrive la situazione di un magazzino:

  Prodotto(Cod, Nome, QtaDisp, SogliaRiordino, QtaRiordino,
  DataCons, PrelievoMedioGiornaliero)
  Prelievo(CodProdotto, Data, Qta)
  Allarme(CodProdotto, DataAllarme)
- Si scriva un trigger che all'inserimento di una tupla in Prelievo aggiorna l'attributo PrelievoMedioGiornaliero, assegnandogli un valore pari alla somma dell'80% del valore precedente e del 20% del valore nuovo (ovvero, gestisce una media esponenziale con coefficiente α pari a 0,2).
  Si immagina inoltre che un'applicazione esterna si occupi di riordinare i prodotti la cui quantità scende sotto SogliaRiordino. Si deve scrivere un trigger che inserisce una tupla in Allarme se (1) il prodotto ha QtaDisp inferiore a SogliaRiordino e DataCons presenta il valore nullo; o (2) PrelievoMedioGiornaliero moltiplicato per il numero di giorni mancanti alla consegna supera la QtaDisp. (la funzione today() restituisce la data corrente e la differenza tra date restituisce il numero di giorni di distanza).

Esercizio

- Si consideri la seguente base di dati che descrive la profilazione di un utente, i contenuti di una pagina, e le navigazioni di un utente. L'attributo N-Hit è un contatore del numero di accessi a pagine contenenti una certa keyword, Target e' un booleano, inizialmente falso, che diventa vero quando si accerta l'interesse di un utente per quel target. Progr indica il numero progressivo di accesso a pagina dello stesso utente (si assume che gli utenti siano registrati).

  Utente(ID, Nome)

  Profilo(ID, KEYWORD, N-Hit, Target)

  Pagina(ID, KEYWORD)

  Navigazione(USER-ID, PROGR, PageId)
- Costruire un insieme di regole attive che, a fronte della inserzione di una nuova navigazione, incrementa il profilo utente per tutte le keyword presente nella pagina, e che mette il Target a vero quando il numero di Hit della parola chiave presecelta supera il valore di 20 nel contesto degli ultimi 100 accessi.
- Opzionale: studiare l'ottimizzazione delle regole.

Esercizio

- Si consideri la base di dati:
- CORSO (CodiceCorso, Docente, TipoCorso)
  ISCRIZIONE (CodiceStudente, CodiceCorso)
  STUDENTE (CodiceStudente, Nome, CosoLaurea)
  ABILITAZIONE (CodiceStudente, Nome, CosoLaurea)
- Costruire un insieme di trigger che alla prima iscrizione di studenti non informatici ad un corso informatico li inserisca tra gli studenti abilitati a frequentare il laboratorio e mantenga l'abilitazione a meno che lo studente non cancelli tutti i suoi corsi informatici (si assuma che iscrizioni e abilitazioni si riferiscano al semestre corrente).

Esercizio

- Descrivere i dati in formato relazionale e scrivere un insieme di regole attive per gestire il seguente problema: Una società di acquisti su internet mantiene l'elenco dei siti che vendono CD al miglior prezzo, dando tale prezzo in dollari. Tenere aggiornato l'elenco tenendo presente aggiunte e cancellazioni di offerte, che sono espresse in valuta, il cambio di prezzo, e i cambi delle valute. Si assuma che i CD abbiano un identificatore universale.

Esercizio

- Un sistema di controllo del traffico deve gestire eventi legati all'interruzione di traffico. Per ogni utente che sta compiendo un tragitto, il sistema registra tale tragitto e la posizione corrente:

  TRAGITTO(NumUtente, NumTratto, NodoFrom, NodoTo, IdStrada)

  POSIZIONE(Num-utente, NumTratto)
- Gli eventi che possono accadere sono interruzioni di strade tra due nodi, registrati tramite inserimenti e cancellazioni nella tabella INTERRUZIONESTRADA(Nodo-From, Nodo-To, Id-Strada)
- Quando un utente attivo deve ancora passare attraverso una strada interrotta, il sistema deve inviare un warning all'utente. A questo punto, il sistema calcola un nuovo tragitto. Deve però anche inviare un secondo warning se l'interruzione termina e l'utente cui viene modificato il tragitto ha ancora una posizione attiva, in modo da consentire al sistema di ricalcolare il tragitto.

Esercizio

- Si consideri uno stream di dati costituito da tuple della tabella eventi:

  EVENTO(TIMESTAMP, Tipo, Valore)
- ove Tipo assume vari valori (tra cui "CUT") e Valore e' un numero reale; scrivere una regola che, ogniqualvolta sullo stream si verifica un nuovo evento di tipo "CUT", elimina tutte le tuple precedenti al penultimo evento "CUT" (includendo anche questa tupla), memorizzando, in una tabella di sintesi denominata MEDIA, il valor medio per Tipo delle tuple cancellate, usando come chiave della tabella di sintesi il Tipo e il Timestamp dell'evento che ha scatenato la regola:

  MEDIA(TIPO, TIMESTAMP, Valore)
- Si assuma che ogni stream, gestito correttamente, abbia al più un evento "CUT".

Esercizio

- Si consideri uno schema relazionale per la memorizzazione dei risultati di un test a scelta multipla:

  RispostaCorretta(CODDOMANDA,Valore)

  RispostaStudente (MATR,CODDOMANDA, Valore)

  Graduatoria(MATR,PuntiTotali,Posizione)
- Il meccanismo di valutazione deve attribuire 1 punto allo studente per ogni risposta corretta e sottrarre 0,25 punti per ogni risposta scorretta (ovvero, ogni risposta dello studente che presenti un valore diverso da quello registrato in RispostaCorretta).
- Realizzare tramite trigger il meccanismo che aggiorna l'attributo *PuntiTotali* in seguito a inserimenti nella tabella RispostaStudente.
- Gestire tramite trigger anche il valore dell'attributo Posizione in seguito a variazioni dell'attributo PuntiTotali, in modo tale che esso rappresenti la posizione in graduatoria dello studente (ad esempio, per lo studente con il punteggio massimo, l'attributo vale 1)

Esercizio

- Si consideri una base di dati che comprende due tabelle. La prima, GIOCA, descrive il calendario delle partite previste, ed è preparata all'inizio del campionato; ogni giornata ha un numero progressivo, ogni partita di ogni giornata è identificata dalla coppia <giornata,partita>. La seconda, SEGNA, viene aggiornata tramite inserimenti subito dopo ogni goal segnato da un giocatore di una delle due squadre che giocano quella partita:

  GIOCA(Giornata, Partita, Squadra1, Squadra2)

  SEGNA(Giornata, Partita, NumeroGoal, Squadra, Giocatore)
- Costruire regole attive che reagiscono ad inserimenti in SEGNA e aggiornano le tabelle RISULTATO, CLASSIFICA (si ricorda che una sconfitta vale 0 punti, un pareggio 1 e una vittoria 3 punti) e MARCATORI:

  RISULTATO(Giornata, Partita, Goal1, Goal2).

  CLASSIFICA(Giornata, Squadra, Punti).

  MARCATORI(Giocatore, Squadra, Goal)
- Si supponga che prima dell'inizio di una giornata di campionato il database venga aggiornato manualmente inserendo le tuple delle partite programmate nella giornata, con Goal1 e Goal2 uguali a zero, e le tuple della classifica relativa alla giornata, aggiungendo ad ogni squadra un punto rispetto alla giornata precedente, mentre la classifica dei marcatori comprende una tupla per giocatore (identificato dalla coppia <giocatore,squadra>) solo per quei giocatori che hanno fatto qualche goal.

Esercizio

- Si consideri un sistema di allarmi caratterizzato da sensori che sono in grado di registrare informazione nel seguente formato ALLARME(Cod-sensore, data, ora, tipo-allarme, citta, regione) costruire regole attive che scattano all'inserimento di un nuovo allarme e che registrano una "emergenza" quando si verificano 5 allarmi dello stesso tipo nella stessa regione e nello stesso giorno. Ogni "emergenza" deve essere registrata una sola volta per ogni regione e giorno.

Esercizio

- Attraverso un sistema di soglie di allarme a più livelli, la Protezione Civile dà l'allerta per fenomeni di maltempo (bufere, piogge violente, nevicate intense...). Le allerte sono riferite ai codici di avviamento postale. I dati della tabella ALLERTAMETEO sono mantenuti costantemente aggiornati sulla base dei calcoli di un sistema software esterno che elabora le previsioni meteo. Se per un certo CAP in una certa data sono presenti più allerte, si considera (e si mette in evidenza nella tabella ALLERTAINEVIDENZA) solo quella con livello di gravità più elevato. Se in base a previsioni successive un'allerta rientra, questa viene rimossa, e si pone in evidenza l'eventuale allerta di livello immediatamente inferiore. Si progetti un insieme di regole attive per mantenere aggiornata la tabella ALLERTAINEVIDENZA.
- ALLERTAMETEO (ID, Data, CAP, TipoAllarme)
- TIPOALLERTA (TipoAllarme, LivelloGravità, Descrizione)

  // *Su LivelloGravità sono specificati i vincoli* unique *e* not null
- ALLERTAINEVIDENZA (CAP, Data, TipoAllarme)

Esercizio

- Un portale di e-commerce vuole gestire tramite trigger il "carrello", in cui gli utenti possono inserire e rimuovere i prodotti in vendita e modificare le quantità ordinate. Un carrello vuoto con totale pari a 0 è creato dall'applicazione di e-commerce ad ogni sessione di navigazione degli utenti registrati. Il prezzo totale dei prodotti nel carrello dev'essere calcolato in automatico ed essere sempre aggiornato. Inoltre, se l'importo totale raggiunge certe soglie, al totale si applica un bonus, e sono previste soglie diverse con bonus crescenti (ad es. se il totale supera 100 il bonus è di 5, se supera 200 è di 15, e così via). I bonus non sono cumulabili tra loro, ma vale solo quello relativo alla soglia più alta. Si scrivano i trigger per la gestione di tutti i tipi di operazione sulla tabella ITEMCARRELLO, badando a realizzare una soluzione incrementale per il calcolo del totale.

  PRODOTTO (ProdottoID, Descrizione, Categoria, Prezzo)

  CARRELLO (CarrelloID, UserID, DataCreazione, OraCreazione, Totale)

  ITEMCARRELLO (CarrelloID, ProdottoID, Quantità)

  BONUS (Soglia, Bonus)

Esercizio

- A Milano l' "Area C" definisce le regole per l'accesso dei veicoli al centro città. Per accedere occorre pagare un ticket, che vale per l'intera giornata; all'interno della stessa giornata sono pertanto consentiti più accessi con lo stesso ticket. L'attivazione del ticket, però, deve avvenire il giorno dell'ingresso o entro le ore 23:59 del giorno successivo. Le attivazioni effettuate il giorno prima dell'accesso o oltre le ore 24:00 del giorno successivo non coprono l'ingresso. Il passaggio del veicolo nel giorno scoperto sarà dunque sanzionato. Quando sono attive, le telecamere rilevano l'ingresso di tutti i veicoli e ne trasmettono i dati a un sistema che riconoscere il mezzo e la sua tipologia (residenti, mezzi di servizio, accesso libero), sicché il valore del ticket ad esso applicato è rilevato automaticamente.
- Considerando il seguente schema dei dati, si scriva un insieme di trigger che 1) a seguito dei passaggi delle auto verifichino se l'auto deve pagare il ticket, e nel caso se è già stato pagato il giorno stesso; 2) a seguito di attivazioni di ticket verifica se è associabile al passaggio dell'auto. Nel caso in cui il ticket avvenga successivamente all'ingresso nell'area, il sistema associa il pagamento al primo ingresso avvenuto tra il giorno precedente ed il giorno in cui il ticket viene attivato. In caso di irregolarità (pagamento non effettuato entro il giorno successivo o pagamento insufficiente) viene registrata una sanzione, altrimenti viene registrato l'avvenuto pagamento.

AUTO (Targa, Tipologia)

COSTO (Tipologia, ValoreDaPagare)

TELECAMERA (ID, Indirizzo)

SANZIONI (Targa, Data)

PASSAGGIAUTO (IDTelecamera, Targa, Data, Ora, Pagato_Si/No)

ATTIVAZIONETICKET (Targa, Data, Ora, Importo)

Esercizio

- Per il 150° anniversario il Politecnico organizza, presso le sue strutture e in alcuni musei, degli eventi per famiglie, dedicati ai bambini (a cui ogni accompagnatore può iscrivere al più 5 bambini). Le richieste di prenotazione si possono effettuare fino al giorno precedente l'evento e nel limiti di 5 posti assegnabili, altrimenti sono rifiutate. Se vi sono posti a sufficienza, sono accettate come prenotazioni, altrimenti sono inserite in una lista di attesa.

EVENTO (Nome, Data, TotalePosti)

RICHIESTA (NomeEvento, Data, NomeAccompagnatore, NumeroPosti)

PRENOTAZIONE (NomeEvento, Data, NomeAccompagnatore, NumeroPosti)

LISTAATTESA (Posizione, NomeEvento, Data, NomeAccompagnatore, NumeroPosti)

- Le richieste non possono essere accettate per una frazione dei posti, ma solo per il totale di posti richiesti. Le prenotazioni accettate e le richieste in lista di attesa possono anche essere annullate dai richiedenti. Quando si liberano dei posti, questi sono assegnati alla prima richiesta in lista di attesa per la quale i posti disponibili siano sufficienti.
- Si progetti un insieme di regole che gestisca le richieste di prenotazione, le eventuali cancellazioni e la lista di attesa.

Esercizio

- Un portale di e-commerce vuole gestire tramite trigger il "carrello", in cui gli utenti possono inserire e rimuovere i prodotti in vendita e modificare le quantità ordinate. Un carrello vuoto con totale pari a 0 è creato dall'applicazione di e-commerce ad ogni sessione di navigazione degli utenti registrati. Il prezzo totale dei prodotti nel carrello dev'essere calcolato in automatico ed essere sempre aggiornato. Inoltre, se l'importo totale raggiunge certe soglie, al totale si applica un bonus, e sono previste soglie diverse con bonus crescenti (ad es. se il totale supera 100 il bonus è di 5, se supera 200 è di 15, e così via). I bonus non sono cumulabili tra loro, ma vale solo quello relativo alla soglia più alta. Si scrivano i trigger per la gestione di tutti i tipi di operazione sulla tabella ITEMCARRELLO, badando a realizzare una soluzione incrementale per il calcolo del totale.

  PRODOTTO (ProdottoID, Descrizione, Categoria, Prezzo)

  CARRELLO (CarrelloID, UserID, DataCreazione, OraCreazione, Totale)

  ITEMCARRELLO (CarrelloID, ProdottoID, Quantità)

  BONUS (Soglia, Bonus)

Esercizio

- La società Autostrade controlla i limiti di velocità attraverso il sistema “Safety Tutor”, che registra gli orari di passaggio sotto le telecamere poste all'inizio e alla fine di ogni tratta controllata (a ogni passaggio si inserisce una tupla nella tabella PASSAGGIOVEICOLO). All'interno di ogni tratta il limite di velocità è omogeneo. Il sistema calcola su ciascuna tratta la velocità media in base al tempo di percorrenza (si ignori, per semplicità, il caso di tratte percorse a cavallo della mezzanotte). I dati relativi ai veicoli che non superano i limiti sono subito eliminati, mentre quelli da sanzionare (tenendo conto di una tolleranza del 5%) sono registrati. Inoltre, se lo stesso veicolo nella stessa giornata supera il limite su 3 o più tratte viene applicata una sanzione extra. Si progetti un sistema di regole che implementi il comportamento descritto.

  TELECAMERA (ID, Luogo)

  TRATTA (IdTelecameraInizio, IdTelecameraFine, Distanza, LimiteVelocità)

  PASSAGGIOVEICOLO (Targa, IdTelecameraFine, Data, Ora)

  DASANZIONARE (Targa, IdTelecameraFine, Data, Ora, VelocitàMediaTransito)

  SANZIONEEXTRA (Targa, Data)

Esercizio

- Si consideri il seguente schema di dati relativi a un'associazione sportiva:

  NUOTATORE(CodiceFiscale, Nome, Cognome, Gruppo, TempoMedioVasca)

  ALLENATORE(CodiceFiscale, Nome, Cognome)

  GRUPPO(Codice,Nome, CFAllenatore, TempoMedioVasca)

  TEMPOVASCA(CodiceFiscaleAllievo, Data, Tempo)
- Per mantenere i gruppi (il cui numero è fisso) il più possibile omogenei nel tempo, alla fine di ogni giornata ogni nuotatore fa una vasca cronometrata. Ad ognuno è associato il tempo medio delle sue ultime due prestazioni e ad ogni gruppo è associata la media dei tempi medi dei componenti. Quando un nuotatore ha un tempo medio che differisce dalla media del suo gruppo di più del 10% lo si sposta in un altro gruppo. Il gruppo in cui viene spostato è il gruppo, diverso da quello di provenienza, la cui media globale è più vicina a quella del nuotatore da spostare; lo spostamento avviene anche qualora il gruppo di destinazione abbia una media con uno scostamento superiore a quello di provenienza. Ovviamente, lo spostamento non avviene se un nuotatore migliora (o peggiora) significativamente la sua performance trovandosi già nel gruppo dei più veloci (o, rispettivamente, dei più lenti).
- Si progetti un insieme di trigger per spostare i nuotatori ad ogni nuova registrazione di un tempo di vasca considerando anche l'impatto sulla media del gruppo di ogni registrazione e dell'entrata o dell'uscita di un nuotatore dal gruppo stesso.

Esercizio

- Si consideri il seguente schema di dati relativi a un sito Web che ospita recensioni di libri:

  UTENTE(Email, Password, Cognome, Nome)

  LIBRO(ISBN, Autori, Titolo, Descrizione, NumPagine, AnnoDiPubblicazione, CasaEditrice, VotoMedio)

  RECENSIONE(Email, ISBN, Data, Voto, Motivazioni)

- Ogni utente inserisce recensioni dei libri che legge dando ad essi un voto da 1 (pessimo) a 5 (ottimo) e scrivendo una motivazione per il voto dato. Ad ogni nuovo giudizio la media dei voti del libro viene aggiornata. Tuttavia, per evitare che i giudizi sui libri siano resi inattendibili da lettori inesperti o troppo polarizzati, si decide di non computare nelle medie i voti di chi ha recensito meno di 5 libri né quelli di chi ha sempre dato lo stesso voto a tutti i libri recensiti. Si noti che l'inserimento di una nuova recensione può portare a dover considerare recensioni precedentemente ignorate (mentre non è mai vero che nuove recensioni possano portare a dover ignorare recensioni già considerate). Si proponga un insieme di regole attive per implementare il comportamento descritto in seguito all'inserimento di nuove recensioni. Si ometta, per semplicità, la gestione delle modifiche e delle cancellazioni di recensioni già inserite.