

# **Esercizi sulle Regole attive**

Autori

Dato il seguente schema relazionale:

**Libro** (Isbn, Titolo, NumCopieVendute)

**Scrittura** (Isbn, Nome)

**Autore** (Nome, NumCopieVendute)

Definire un insieme di regole attive in grado di mantenere aggiornato l'attributo NumCopieVendute di *Autore* rispetto a:

- Modifiche sull'attributo *NumCopieVendute* di *Libro*
- Inserimenti su *Scrittura*.

<https://goo.gl/S5F80J>

create trigger AggiornaCopieAutoreDopoNuoveVendite  
after **update** of NumCopieVendute on Libro  
for each row  
update Autore

set NumCopieVendute = NumCopieVendute +  
new.NumCopieVendute –  
old.NumCopieVendute

where Nome in ( **select Nome**  
**from Scrittura**  
**where Isbn = new.Isbn**)

ATTENZIONE: NumCopieVendute di Autore (quello su cui agisce la regola) è diverso dal NumCopieVendute di Libro.

<https://goo.gl/DVbg8r>

```
create trigger AggiornaCopieAutoreDopoAggiuntaInScrittura
after insert on Scrittura
for each row
update Autore
  set NumCopieVendute = NumCopieVendute +
      (select NumCopieVendute
       from Libro
       where Isbn=new.Isbn)
where Nome = new.Nome
```

<https://goo.gl/PsM4dD>

*Potrebbe sembrare che dobbiamo anche reagire agli inserimenti in LIBRO, Tuttavia, perché la paternità del libro venga assegnata, deve essere comunque aggiunta una tupla in SCRITTURA, e abbiamo già definito un trigger che aggiorna le copie in quel caso. Sarebbe quindi sbagliato scrivere un trigger del tipo:*

```
create trigger AggiornaCopiePerNuovoLibro  
after insert on Libro  
for each row  
update Autore  
set NumCopieVendute = NumCopieVendute +  
new.NumCopieVendute  
where Nome in ( select Nome  
from Scrittura  
where Isbn = new.Isbn)
```

Esami

Data la base di dati:

**Esame** (Matr, Corso, Data, Voto)

**Precedenza** (CorsoPrecedente, CorsoPreceduto)

Con il seguente vincolo di integrità:

*"non e' possibile registrare un esame se non esiste la registrazione di tutti gli esami che lo precedono"*

- a. Indicare quali operazioni possono violare un vincolo.
- b. Scegliere, in base all'esperienza, quale operazione può realmente violare il vincolo.
- c. Scrivere una regola attiva che, reagendo a tale operazione, forzi il rollback della transazione.

<https://goo.gl/sEqGyY>



Vi sono 5 possibili operazioni che possono violare la regola:

- 1.INSERT di un elemento in Esame senza la registrazione degli esami che lo precedono.
- 2.UPDATE di Data in Esame.
- 3.DELETE di un elemento di Esame.
- 4.UPDATE di un record in Precedenza.
- 5.INSERT di un record in Precedenza

L'operazione che può realisticamente violare il vincolo in un caso reale è la prima.

*Aggiornamenti e cancellazioni di esami non sono frequenti.*

*Cambiamenti alle regole di precedenza potrebbero influenzare gli esami già sostenuti. Una soluzione a questo problema sarebbe quella di estendere la tabella Precedenza con un campo **data** dalla quale la regola è attiva.*

create trigger MancanoPrecedenze  
after insert on Esame  
for each row  
when exists

<https://goo.gl/K25Bkb>

```
( select *  
  from Precedenza  
 where CorsoPreceduto = new.Corso AND  
        CorsoPrecedente not in ( select Corso  
                                from Esame  
                                where Matr = new.Matr  
                                AND Data < new.Data ) )
```

rollback;

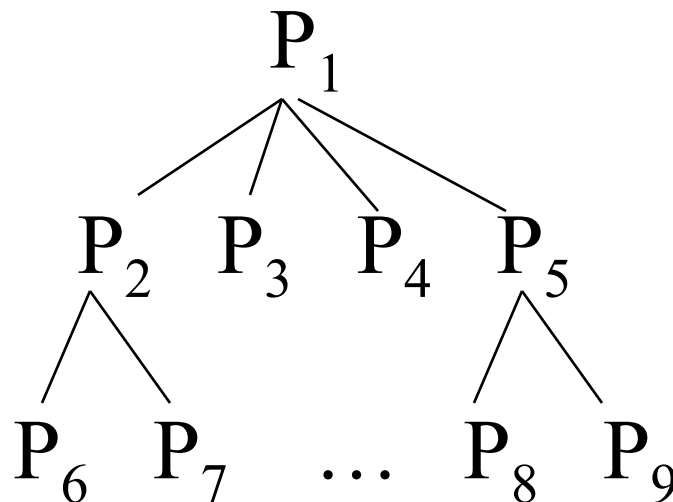
Cioè l'inserimento si impedisce se esiste almeno una tupla in *Precedenza* che indichi una dipendenza dell'esame appena registrato [CorsoPreceduto = **new.Corso**] da un corso che non sia tra quelli già registrati [Data < **new.Data**] nella tabella *Esame* dallo studente a cui si riferisce la tupla appena inserita [Matr = **new.Matr**]

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 altro 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 PseudoConstraint
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  $\neq$  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 definizione dei trigger, e solo inserimenti e cancellazioni sono permessi (no updates), non c'è modo di introdurre dei cicli.

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 )

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

after 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.

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.

# Esercizio - Il teatro lirico “social”

Un teatro lirico gestisce gli spettacoli tramite un sistema di row-level trigger. I visitatori del sito si iscrivono e registrano le parole chiave di loro interesse.

- Quando (a) viene pubblicato un nuovo spettacolo (descritto con parole chiave) i visitatori interessati ricevono una e-mail, e potranno acquistare un biglietto.
- In caso di (b) cancellazioni o (c) cambi di orario, chi ha acquistato un biglietto riceve una notifica.
- Si scrivano *solo* i trigger relativi agli eventi descritti (a,b,c).
- Si assuma disponibile una funzione *send-mail(e-mailDest, Subj, ... AltriAttr...)* con opportuni parametri utili. *e-mailDest* può essere una lista di destinatari.
- Si ipotizzi che l’inserimento delle keyword sia contestuale a quello dello spettacolo

**VISITATORE** ( Id-Vis, Nome, Email )

**INTERESSE** ( Id-Vis, Keyword )

**SPETTACOLO** ( Id-Spet, Titolo, Data, OraInizio )

**DESCRIZIONE** ( Id-Spet, Keyword )

**ACQUISTO** ( Id-Vis, Id-Spet, DescrizionePosti )

# Il teatro lirico “social”

## Trigger (a)

create trigger NuovoSpettacolo  
after **insert** into SPETTACOLO  
for each row

```
    send-mail( ( select Email  
                  from ( VISITATORE V join INTERESSE I on V.Id-Vis = I.Id-Vis  
                        join DESCRIZIONE D on D.Keyword = I.Keyword )  
                  where D.Id-Spet = new.Id-Spet ),  
              “Nuovo spettacolo”,  
              new.Titolo,  
              new.Data,  
              new.OraInizio )
```

# Il teatro lirico “social”

## Trigger (b)

```
create trigger CancellazioneSpettacolo
after delete from SPETTACOLO
for each row
  send-mail( ( select Email
               from VISITATORE V join ACQUISTO A on V.Id-Vis = A.Id-Vis
               where A.Id-Spet = old.Id-Spet),
             “Cancellazione” ,
             old.Titolo,
             old.Data,
             old.OraInizio )
```

# Il teatro lirico “social”

## Trigger (c)

```
create trigger CambioOrario
after update of OraInizio on SPETTACOLO
for each row
  send-mail( ( select Email
               from VISITATORE V join ACQUISTO A on V.Id-Vis = A.Id-Vis
               where A.Id-Spet = old.Id-Spet),
             “Cambio Orario”,
             old.Titolo,
             old.Data,
             new.OraInizio )
```

# 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

# 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

[illegible]

# 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

# 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
```