BASI DI DATI ATTIVE

Vincenzo Calabrò

Basi di dati attive

Supportano la definizione e gestione di regole di produzione (regole attive)

- paradigma Evento-Condizione-Azione
  - a seguito dell'evento, se la condizione è soddisfatta, esegui l'azione
- indipendenza della conoscenza
  - la conoscenza di tipo reattivo viene sottratta ai programmi applicativi e codificata sotto forma di regole attive

Quasi tutti i DBMS hanno supporto per semplici regole attive (trigger)

Trigger (1)

Trigger standardizzati solo in SQL:1999 (SQL-3)

- molti sistemi commerciali hanno introdotto i trigger prima della standardizzazione
- DBMS diversi possono differire nella gestione dell'attivazione e esecuzione dei trigger

Trigger (2)

La creazione dei trigger fa parte del Data Definition Language.

Componenti:

- evento
  - primitive SQL per la manipolazione dei dati
    - insert, delete, update
- condizione
  - predicato booleano espresso in SQL
- azione
  - sequenza di primitive SQL generiche o procedura

Trigger (3)

Due livelli di granularità

- tupla (row-level)
  - l'attivazione avviene per ogni tupla coinvolta nell'evento
- primitiva (statement-level)
  - l'attivazione avviene una sola volta per ogni evento e si applica a tutte le tuple coinvolte nell'evento

Modalità di attivazione

- immediata
  - valutazione immediatamente dopo (opzione after) o prima (opzione before) dell'evento che lo ha attivato
- differita
  - la valutazione avviene alla fine (commit) della transazione

Trigger in SQL: sintassi

```
create trigger NomeTrigger {before|after}
{insert|delete|update [of Colonne] } on Tabella
[referencing
{[old_table [as] TabOld] | [new_table [as] TabNew]} |
{[old[_row] [as] TuplaOld] | [new[_row] [as]TuplaNew]}]
[for each {row|statement}]
[when (PredicatoSQL)]
ComandiSQL
```

- statement-level è default
- *referencing* permette di inserire variabili
  - `old` e `new` si riferiscono allo stato precedente/successivo della tupla
  - `old_table` e `new_table` si riferiscono allo stato precedente/successivo della tabella

Trigger in SQL: livelli e modalità

- sono immediati (before o after)
- granularità di tupla o primitiva
- quattro possibilità
  - before row, before statement, after row, after statement
- i before-trigger
  - vengono usati solo per determinare errori e per modificare i valori delle variabili `new` (possono chiederne la valutazione anticipata)
  - non possono contenere comandi DML
  - non attivano altri trigger

Trigger in SQL: esempi

```
create trigger MonitoraConti
after update on Conto
for each row
when ( old.NomeConto = new.NomeConto
       and  new.Totale > old.Totale)
begin
       insert into SingoliVersamenti
       values (new.NomeConto, new.Totale – old.Totale);
end;
```

```
create trigger ArchiviaFattureCanc
after delete on Fattura
referencing old_table as FattureVecchie
begin
       insert into FattureCancellate
       (select *
       from FattureVecchie);
end;
```

Trigger in SQL: esecuzione (1)

Passi:

- si eseguono i trigger before statement
- si eseguono i trigger before row
- si eseguono test di integrità sulla tabella
- si eseguono i trigger after row
- si eseguono i trigger after statement

Se vi sono più trigger della stessa categoria, l'ordine di esecuzione viene scelto dal sistema in un modo che dipende dall'implementazione

Trigger in SQL: esecuzione (2)

- i trigger sono gestiti in un Trigger Execution Context (TEC)
- l'esecuzione dell'azione di un trigger può produrre eventi che fanno scattare altri trigger, che sono valutati in un nuovo TEC interno
- in ogni istante possono esserci più TEC per una transazione, uno dentro l'altro, ma uno solo può essere attivo (struttura a stack)
- per i trigger row-level il TEC tiene conto di quali tuple sono già state considerate e quali sono da considerare

Evoluzione delle regole attive

Rispetto ai trigger relazionali, alcuni sistemi e prototipi evoluti hanno caratteristiche che aumentano il potere espressivo delle regole attive

- eventi
- attivazione
- priorità

Evoluzione delle regole: eventi

- eventi temporali (anche periodici)
  - es.: 21/03/2006 alle ore 12:00; ogni giorno alle 17:00
- eventi applicativi (definiti dall'utente)
  - es.: "TemperaturaTroppoAlta"
- combinazioni booleane di eventi
  - SQL:1999 consente di specificare più eventi per un trigger, in disgiunzione (basta che se ne verifichi uno)
  - sono stati proposti anche modelli di composizione più sofisticati (molto complessi e costosi)

Evoluzione delle regole: attivazione (1)

- regole attivabili/disattivabili
  - non presente nello standard
  - disponibile in vari sistemi
- gruppi di regole
  - alcuni sistemi permettono di definire gruppi di regole, ciascun gruppo attivabile/disattivabile
- clausola `instead of`
- modalità `detached`

Evoluzione delle regole: attivazione (2)

- clausola `instead of`
  - quando la condizione è vera, l'azione viene eseguita al posto dell'operazione che ha attivato la regola (evento)
  - è una modalità alternativa a before e after
  - è implementata in diversi sistemi, spesso con forti limitazioni
    - es.: in Oracle si può usare esclusivamente per eventi di modifica su viste, per gestire il problema dell'aggiornamento delle viste

Evoluzione delle regole: attivazione (3)

- modalità `detached`
  - la regola viene gestita in una transazione separata
    - es.: si vogliono gestire in modo efficiente le variazioni del valore di indici di borsa in seguito a numerosi scambi
  - si aggiunge a:
    - immediato (`immediate`): il trigger viene considerato ed eseguito con l'evento che lo ha attivato
    - differito (`deferred`): il trigger viene gestito al termine della transazione

Evoluzione delle regole: priorità

- la priorità regola l'ordine di esecuzione delle regole quando ve ne sono diverse attivate contemporaneamente
  - SQL:1999 specifica un ordine che si basa sul modo di esecuzione e sulla granularità del trigger; a parità modo, la scelta dipende dall'implementazione
  - nei sistemi spesso si usa l'ordine temporale di definizione

Proprietà delle regole attive

Necessario avere garanzie che l'interferenza tra le diverse regole e l'attivazione a catena non generi anomalie

- terminazione
- confluenza
- determinismo delle osservazioni

Terminazione

Per qualunque stato iniziale e qualunque sequenza di modifiche, le regole:

- producono uno stato finale
  - non devono esserci cicli infiniti di attivazione

È la più importante

Confluenza

Per qualunque stato iniziale e qualunque sequenza di modifiche, le regole:

- producono uno stato finale (terminazione)
- producono un unico stato finale, indipendente dall'ordine in cui i trigger vengono eseguiti

È significativa quando il sistema presenta del non determinismo nella scelta delle regole da eseguire

Determinismo delle osservazioni

Per qualunque stato iniziale e qualunque sequenza di modifiche, le regole:

- terminano e producono un unico stato finale, indipendente dall'ordine in cui le regole vengono eseguite (confluenza)

- producono la stessa sequenza di azioni visibili

Analisi di terminazione

Basata sul grafo di attivazione

- un nodo per ogni regola
- un arco da un nodo R_i a un nodo R_j se l'esecuzione dell'azione di R_i può attivare la regola R_j
  - azione di R_i contiene una primitiva che coincide con uno degli eventi di R_j

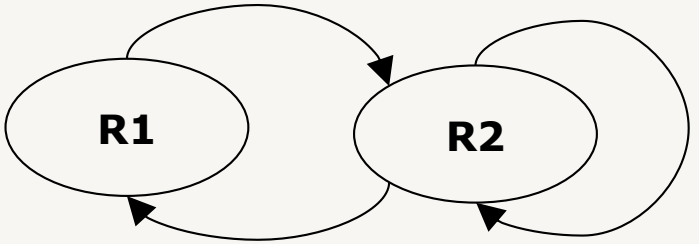
Se il grafo è aciclico, si ha la garanzia che il sistema è terminante

- aciclicità: condizione sufficiente ma non necessaria per la terminazione

Regole non terminanti: esempio

R1:

```
create trigger AggiustaContributi
after update of Stipendio on Impiegato
referencing new_table as NuovoImp
begin
        update Impiegato
        set Contributi = Stipendio * 0.8
        where Matr in (
                select Matr
                from NuovoImp);
end;
```



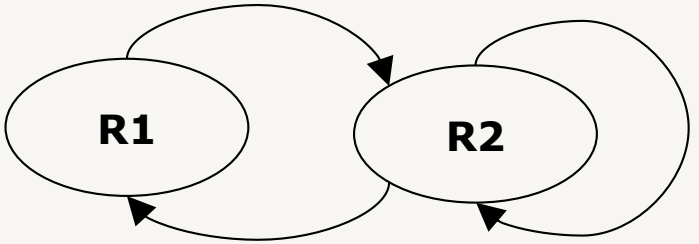
R2:

```
create trigger ControllaSogliaBudget
after update on Impiegato
when 50000 > (
        select sum(Stipendio+Contributi)
        from Impiegato)
begin
        update Impiegato
        set Stipendio = 0.9*Stipendio;
end;
```

Regole terminanti: esempio

R1:

```
create trigger AggiustaContributi
after update of Stipendio on Impiegato
referencing new_table as NuovoImp
begin
        update Impiegato
        set Contributi = Stipendio * 0.8
        where Matr in (
                select Matr
                from NuovoImp);
end;
```



R2:

```
create trigger ControllaSogliaBudget
after update on Impiegato
when 50000 < (
        select sum(Stipendio+Contributi)
        from Impiegato)
begin
        update Impiegato
        set Stipendio = 0.9*Stipendio;
end;
```

Applicazioni delle regole attive

Regole interne alla base di dati

- generate dal sistema e non visibili all'utente
  - gestione dei vincoli di integrità referenziale
  - derivazione o replicazione dei dati
  - gestione di versioni, privatezza, sicurezza, logging delle azioni, registrazione degli eventi, …

Regole esterne (o regole aziendali)

- esprimono conoscenza di tipo applicativo
  - l'implementazione con regole attive la rende valida, per tutte le applicazioni, a livello di schema
    - indipendenza della conoscenza

Gestione dell'integrità referenziale: es. (1)

```
create table Impiegato(… …
  foreign key(NDip) references
          Dipartimento(NroDip)
     on delete set null,
     on update cascade … …);
```

Operazioni che possono violare questo vincolo:

- `insert into Impiegato`
- `update Impiegato.NDip`
- `delete from Dipartimento`
- `update Dipartimento.NroDip`

Gestione dell'integrità referenziale: es. (2)

- evento:
  - inserimento in Impiegato
- condizione:
  - il nuovo valore di NDip non è nella tabella Dipartimento
- azione:
  - proibisci inserimento e segnala errore

```
create trigger ControllaInsDipImpiegato
before insert on Impiegato
for each row
when (not exists (select * from Dipartimento
      where NroDip = new.NDip))
begin
      signal sqlstate '70006' ('Dip non valido');
end;
```

Gestione dell'integrità referenziale: es. (3)

- evento:
  - modifica dell'attributo NDip in Impiegato
- condizione:
  - il nuovo valore di NDip non è nella tabella Dipartimento
- azione:
  - proibisci la modifica e segnala errore

```
create trigger ControllaModDipImpiegato
before update of NDip on Impiegato
for each row
when (not exists (select * from Dipartimento
      where NroDip = new.NDip))
begin
      signal sqlstate '70006' ('Dip non valido');
end;
```

Gestione dell'integrità referenziale: es. (4)

- evento:
  - cancellazione in Dipartimento
- condizione:
  - il valore di NroDip da cancellare è nella tabella Impiegato
- azione:
  - mette a null l'attributo NDip delle tuple coinvolte

```
create trigger ControllaCancDipartimento
after delete on Dipartimento
for each row
when (exists (select * from Impiegato
      where NDip = old.NroDip))
begin
      update Impiegato
      set NDip = null
      where NDip = old.NroDip;
end;
```

Gestione dell'integrità referenziale: es. (5)

- evento:
  - modifica dell'attributo NroDip in Dipartimento
- condizione:
  - il vecchio valore di NroDip è usato nella tabella Impiegato
- azione:
  - si modifica anche NDip delle tuple coinvolte in Impiegato

```
create trigger ControllaModAttribNDip
after update of NroDip on Dipartimento
for each row
when (exists (select * from Impiegato
      where NDip = old.NroDip))
begin
      update Impiegato
      set NDip = new.NroDip
      where NDip = old.NroDip;
end;
```

Regole aziendali

Vincoli o controlli non esprimibili con i costrutti del DBMS, esempi:

- integrità
  - es.: un impiegato deve avere uno stipendio minore del direttore del dipartimento al quale afferisce
  - implementazione con asserzioni o check costringono all'utilizzo dello standard per la reazione alla violazione
- derivazione
  - es.: il fatturato totale è la somma dei totali delle fatture emesse
  - es.: relazione di contenimento (gerarchia) fra prodotti

Regole aziendali (integrità): es.

Un impiegato deve avere uno stipendio minore del direttore del dipartimento al quale afferisce

IMPIEGATO(ImpNum,Mgr,Salario,DipNum)
DIPARTIMENTO(DipNum,Direttore)

```
create trigger SalarioImpiegato
after update on Salario of Impiegato
for each row
when (new.Salario > select Salario
                    from Impiegato
                    where ImpNum = new.Mgr
                    and ImpNum in (select Direttore
                                   from Dipartimento
                                   where DipNum = new.DipNum))
begin
       signal sqlstate '70005' ('Salario troppo elevato');
end;
```

Regole aziendali (derivazione): es. (1)

PRODOTTO(Codice, Nome, SuperProdotto, Livello)

- gerarchia di prodotti
  - ogni prodotto ha un super-prodotto e un livello di profondità nella gerarchia
  - prodotti non contenuti in altri prodotti: SuperProdotto = NULL e Livello = 0

Regole

- se un prodotto è cancellato: cancellare tutti i sottoprodotti che lo compongono
- se un prodotto è inserito: calcolare il suo livello

Regole aziendali (derivazione): es. (2)

- evento:
  - cancellazione di un prodotto
- azione:
  - cancellare tutti i suoi sottoprodotti

```
create trigger CancellaProdotto
after delete on Prodotto
for each row
begin
      delete from Prodotto
      where SuperProdotto = old.Codice;
end;
```

Regole aziendali (derivazione): es. (3)

- evento:
  - inserimento di un prodotto
- azione:
  - calcolare il suo livello

```
create trigger LivelloProdotto
after insert on Prodotto
for each row
begin
	if new.SuperProdotto is not null
		update Prodotto
		set Livello = 1 + (
			select Livello from Prodotto
			where Codice = new.SuperProdotto)
		where Codice = new.Codice
	else
		update Prodotto
		set Livello = 0
		where Codice = new.Codice;
	endif;
end;
```

Basi di dati attive – Esercizi

Esercizio 1 (1)

CLIENTE(CF, Nome, Cognome, NumTel, PianoTarif)

PIANOTAR(Codice, CostoScattoRisp, CostoAlSec)

TELEFONATA(CF, Data, Ora, NumeroDest, Durata)

BOLLETTA(CF, Mese, Anno, Importo)

1. Ipotizziamo bollette da aggiornare inserite automaticamente a 0 all'inizio mese (evento *CHANGE_MONTH*)
2. A seguito di ogni telefonata aggiornare la bolletta del cliente che ha chiamato
3. Alla fine di ogni mese (evento *END_MONTH*) scontare dalle bollette 5 centesimi per ogni telefonata diretta a utenti della compagnia (verso numeri in CLIENTE) se l'importo della bolletta è maggiore di 100 Euro

Esercizio 1 (2)

1. Ipotizziamo bollette da aggiornare inserite automaticamente a 0 all'inizio mese (evento *CHANGE_MONTH*)

```
create trigger CominciaMese
after CHANGE_MONTH
begin
    insert into BOLLETTA
    select CF, sysdate().month, sysdate().year, 0
    from Cliente
end
```

N.B. sintassi alternativa per l'estrazione del mese e dell'anno dalla *data di sistema* è:

`extract( month from sysdate() )` e
`extract( year from sysdate() )`

Esercizio 1 (3)

2. A seguito di ogni telefonata aggiornare la bolletta del cliente che ha chiamato

```
create trigger AddebitaChiamata
after insert on TELEFONATA
for each row
begin
    update BOLLETTA as B
    set Importo = Importo +
            (select PT.CostoScattoRisp +
            PT.CostoAlSec * new.Durata
            from PIANOTAR as PT join CLIENTE as CL
                    on PT.Codice = CL.PianoTarif
            where new.CF = CL.CF )
    where B.CF = new.CF
end
```

Esercizio 1 (4)

3. Alla fine di ogni mese (evento *END_MONTH*) scontare dalle bollette 5 centesimi per ogni telefonata diretta a utenti della compagnia (verso numeri in CLIENTE) se l'importo della bolletta è maggiore di 100 Euro

```
create trigger Promozione5per100
after END_MONTH
begin
    update BOLLETTA as B
    set Importo = Importo - 0,05 *
            (select count(*) from TELEFONATA as T
            where T.CF = B.CF
            and T.Data.month=(sysdate()-1).month
            and T.Data.year=(sysdate()-1).year
                    and T.NumeroDest in
                    (select numTel from CLIENTE)
    where B.Importo > 100
end
```

Esercizio 2 (1)

RISULTATO (Giornata, Casa, Ospite, GoalCasa, GoalOspite)

CLASSIFICA (Squadra, Punti)

Ipotizzando che:

- la prima tabella sia alimentata tramite inserimenti;
- la seconda tabella sia derivata dalla prima.

1. Scrivere le regole attive che costruiscono la classifica, attribuendo 3 punti alle squadre che vincono, 1 a quelle che pareggiano, e 0 a quelle che perdono

Esercizio 2 (2)

```
create trigger InserisciVittoria
after insert on RISULTATO
for each row
when new.GoalCasa <> new.GoalOspite
begin
       update CLASSIFICA
       set Punti = Punti + 3
       where (new.GoalCasa > new.GoalOspite
              and Squadra = new.Casa)
              or
              (new.GoalCasa < new.GoalOspite
              and Squadra = new.Ospite)
end
```

Esercizio 2 (3)

```
create trigger InserisciPareggio
after insert on RISULTATO
for each row
when new.GoalCasa = new.GoalOspite
begin
       update CLASSIFICA
       set Punti = Punti + 1,
       where Squadra = new.Casa
              or Squadra = new.Ospite
end
```

Esercizio 3 (1)

RISULTATO (Giornata, Casa, Ospite, GoalCasa, GoalOspite)

CLASSIFICA (Giornata, Squadra, Punti)

SQUADRA(Nome)

Ipotizzando che:

- la prima tabella sia alimentata tramite inserimenti;
- la seconda tabella sia derivata dalla prima.

1. Scrivere una regola attiva che inserisca tutte le squadre nella relazione CLASSIFICA con punteggio 0 alla giornata 0 al verificarsi dell'evento NEW_CHAMPIONSHIP
2. Scrivere le regole attive che costruiscono la classifica, attribuendo 3 punti alle squadre che vincono, 1 a quelle che pareggiano, e 0 a quelle che perdono

N.B. Rispetto all'esercizio precedente è cambiato lo schema della relazione *CLASSIFICA* ed è stata aggiunta la relazione *SQUADRA*

Esercizio 3 (2)

1. Scrivere una regola attiva che inserisca tutte le squadre nella relazione CLASSIFICA con punteggio 0 alla giornata 0 al verificarsi dell'evento NEW_CHAMPIONSHIP

```
create trigger NewChampionship
after NEW_CHAMPIONSHIP
begin
    insert into CLASSIFICA
    select 0, SQUADRA.Nome, 0
    from SQUADRA
end
```

Esercizio 3 (3)

2.Scrivere le regole attive che costruiscono la classifica, attribuendo 3 punti alle squadre che vincono, 1 a quelle che pareggiano, e 0 a quelle che perdono

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

       insert into CLASSIFICA as C
       select new.Giornata, new.Ospite, C2.Punti
       from CLASSIFICA as C2
       where C2.Squadra = new.Ospite and not exists
              (select * from CLASSIFICA
              where Giornata > C2.Giornata
                     and Squadra = C2.squadra)
end
```

Esercizio 3 (4)

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

        insert into CLASSIFICA as C
        select new.Giornata, new.Ospite, C2.Punti + 3
        from CLASSIFICA as C2
        where C2.Squadra = new.Ospite and not exists
                (select * from CLASSIFICA
                where Giornata > C2.Giornata
                        and Squadra = C2.squadra)
end
```

Esercizio 3 (5)

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

        insert into CLASSIFICA as C
        select new.Giornata, new.Ospite, C2.Punti + 1
        from CLASSIFICA as C2
        where C2.Squadra = new.Ospite and not exists
                (select * from CLASSIFICA
                where Giornata > C2.Giornata
                        and Squadra = C2.squadra)
end
```

Esercizio 3 – Soluzione Alternativa (6)

2.Scrivere le regole attive che costruiscono la classifica, attribuendo 3 punti alle squadre che vincono, 1 a quelle che pareggiano, e 0 a quelle che perdono

```
create trigger VittoriaInCasa
after insert on RISULTATO
for each row
when new.GoalCasa > new.GoalOspite
begin
       insert into CLASSIFICA as C
       select new.Giornata, new.Casa, C2.Punti + 3
       from CLASSIFICA as C2
       where C2.Squadra = new.Casa and
       C2.Giornata =
              (select max(Giornata) from CLASSIFICA
              where Squadra = C2.Squadra)

       insert into CLASSIFICA as C
       select new.Giornata, new.Ospite, C2.Punti
       from CLASSIFICA as C2
       where C2.Squadra = new.Ospite and
       C2.Giornata =
              (select max(Giornata) from CLASSIFICA
               where Squadra = C2.Squadra)
end
```

Esercizio 3 – Soluzione Alternativa (7)

```
create trigger VittoriaFuoriCasa
after insert on RISULTATO
for each row
when new.GoalCasa < new.GoalOspite
begin
        insert into CLASSIFICA as C
        select new.Giornata, new.Casa, C2.Punti
        from CLASSIFICA as C2
        where C2.Squadra = new.Casa and
        C2.Giornata =
                (select max(Giornata) from CLASSIFICA
                where Squadra = C2.Squadra)

        insert into CLASSIFICA as C
        select new.Giornata, new.Ospite, C2.Punti + 3
        from CLASSIFICA as C2
        where C2.Squadra = new.Ospite and
        C2.Giornata = (select max(Giornata) from CLASSIFICA
                where Squadra = C2.Squadra)
end
```

Esercizio 3 – Soluzione Alternativa (8)

```
create trigger Pareggio
after insert on RISULTATO
for each row
when new.GoalCasa = new.GoalOspite
begin
       insert into CLASSIFICA as C
       select new.Giornata, new.Casa, C2.Punti + 1
       from CLASSIFICA as C2
       where C2.Squadra = new.Casa and
       C2.Giornata =
              (select max(Giornata) from CLASSIFICA
              where Squadra = C2.Squadra)

       insert into CLASSIFICA as C
       select new.Giornata, new.Ospite, C2.Punti + 1
       from CLASSIFICA as C2
       where C2.Squadra = new.Ospite and
       C2.Giornata =
              (select max(Giornata) from CLASSIFICA
              where Squadra = C2.Squadra)
end
```

Esercizio 4 (1)

LIBRO(Isbn, Titolo, NumCopieVendute)

SCRITTURA(Isbn, Nome)

AUTORE(Nome, NumCopieVendute)

1. Mantenere aggiornato l'attributo NumCopieVendute di Autore rispetto a modifiche sull'attributo NumCopieVendute di Libro e a inserimenti su Libro

Esercizio 4 (2)

```
create trigger AggiornaDopoVenditeLibroEsistente
after update of NumCopieVendute on LIBRO
for each row
begin
       update AUTORE
       set NumCopieVendute = NumCopieVendute +
                             new.NumCopieVendute -
                             old.NumCopieVendute
       where Nome in
              (select Nome
              from SCRITTURA
              where Isbn = new.Isbn)
end
```

N.B. NumCopieVendute di Autore (quello su cui agisce la regola) è diverso da NumCopieVendute di Libro

Esercizio 4 (3)

```
create trigger AggiornaDopoAggiuntaLibro
after insert on LIBRO
for each row
begin
       update AUTORE
       set NumCopieVendute = NumCopieVendute +
                             new.NumCopieVendute
       where Nome in
              (select Nome
              from SCRITTURA
              where Isbn = new.Isbn)
end
```

Esercizio 5 (1)

STUDENTI(Matr, Nome, CdL, AnnoCorso, Sede, TotCrediti)

ISCRIZIONI(MatrStud, Corso, Anno, Data)

CORSIANNI(CodCorso, Anno, NroStudenti, NroFuoriSede)

ABBINAMENTI(CodCorso, CorsoLaurea)

CORSI(CodCorso, Titolo, Crediti, Sede)

1. Dopo ogni inserimento di una tupla nella tabella ISCRIZIONI e nel caso in cui non esistano elementi corrispondenti in STUDENTI o CORSIANNI (studente sconosciuto o corso non attivo) annullare l'inserimento
2. A seguito di aggiornamenti dell'attributo Sede di STUDENTE, aggiornare, se necessario, il valore dell'attributo NroFuoriSede di CORSIANNI.
   Si noti che servono due trigger.

Esercizio 5 (2)

1. Dopo ogni inserimento di una tupla nella tabella ISCRIZIONI e nel caso in cui non esistano elementi corrispondenti in STUDENTI o CORSIANNI (studente sconosciuto o corso non attivo) annullare l'inserimento

```
create trigger CheckIntRef
after insert on ISCRIZIONI
when not exists
       (select *
       from STUDENTI
       where Matr = new.MatrStud)
       or not exists
       (select *
       from CORSIANNI
       where CodCorso = new.Corso
              and AnnoCorso = new.Corso)
Rollback
```

Esercizio 5 (3)

2. A seguito di aggiornamenti dell'attributo Sede di STUDENTE, aggiornare, se necessario, il valore dell'attributo NroFuoriSede di CORSIANNI.
   - Aumenta di 1 gli studenti fuori sede della sede vecchia

```
create trigger CheckSede1
after update of Sede on STUDENTI
for each row
begin
    update CORSIANNI
    set NroFuoriSede = NroFuoriSede + 1
    where(CodCorso, Anno) in
           (select Corso, Anno from ISCRIZIONI
           where MatrStud = old.Matr)
          and CodCorso in
           (select CodCorso from CORSI
           where Sede = old.Sede)
end
```

Esercizio 5 (4)

- Diminuisci di 1 gli studenti fuori sede della sede nuova

```
create trigger CheckSede2
after update of Sede on STUDENTI
for each row
begin
   update CORSIANNI
   set NroFuoriSede = NroFuoriSede - 1
   where (CodCorso,Anno) in
          (select Corso,Anno from ISCRIZIONI
          where MatrStud = new.Matr)
          and CodCorso in
          (select CodCorso from CORSI
          where Sede = new.Sede)
end
```

Esercizio 6 (1)

STUDENTI(Nome, AreaLavoro, Supervisore)

PROFESSORI(Nome, AreaLavoro)

CORSI(Titolo, Professore)

ESAMI(NomeStudente, TitoloCorso)

Scrivere un trigger per ognuno dei seguenti vincoli di integrità (*business rule*):

1. ogni studente deve lavorare nella stessa area del proprio supervisore;
2. ogni studente deve aver seguito almeno tre corsi tenuti da un professore che lavori nella loro area di interesse;
3. ogni studente deve aver superato almeno un esame tenuto dal proprio supervisore.

Esercizio 6 (2)

1. ogni studente deve lavorare nella stessa area del proprio supervisore

```
create trigger T1
after update of Arealavoro on STUDENTI
for each row
when Arealavoro <>
    (select Arealavoro
    from PROFESSORI
    where PROFESSORI.Nome = new.Supervisore)
then signal sqlstate 70005 (“Area di Lavoro Errata”)
```

N.B. la sintassi `signal sqlstate NUMERO (“DESCRIZIONE”)` permette di sollevare un errore avente codice (univoco) *NUMERO* e descrizione *DESCRIZIONE*

Esercizio 6 (3)

2. ogni studente deve aver seguito almeno tre corsi tenuti da un professore che lavori nell' area di interesse del suo supervisore

```
create trigger T2
after update of Subject on STUDENTI
for each row
when 3 <
    (select count(*)
    from ESAMI join CORSI on TitoloCorso = Titolo
           join PROFESSORI on Professore = PROFESSORI.Nome
           join STUDENTI on NomeStudente = STUDENTI.Nome
           join PROFESSORI as P2 on Supervisore = P2.Nome
    where PROFESSORI.AreaLavoro = P2. AreaLavoro
           and STUDENTI.Nome = new.Nome)
then signal sqlstate 70006 ("Mancano dei Corsi")
```

Esercizio 6 (4)

3. ogni studente deve aver superato almeno un esame tenuto dal proprio supervisore

```
create trigger T3
after update of Supervisore on STUDENTI
for each row
when not exist
    (select *
    from ESAMI join CORSI on TitoloCorsi = Titolo
    where NomeStudente = new.Nome
    and Professore = new.Supervisore)
then signal sqlstate 70007 (“Manca un esame”)
```

VINCENZO CALABRÒ

LinkedIn vincenzocalabro

www.vincenzocalabro.it