Atzeni, Ceri, Fraternali, Paraboschi, Torlone

Basi di dati

Quarta edizione

McGraw-Hill, 2013

Capitolo 5:

SQL: caratteristiche evolute

Vincoli di integrità generici: check

- Specifica di vincoli di ennupla (e anche vincoli più complessi, non sempre supportati)

  check (Condizione)

Check, esempio

```
create table Imp
(
Matricola integer primary key,
Cognome character(20),
Nome character(20),
Sesso character not null check (sesso in ('M','F')) ,
Stipendio integer check (Stipendio > 0) ,
Superiore integer,
check (Stipendio <= (select Stipendio
                     from Imp J
                     where Superiore = J.Matricola) )
)
```

non supportata

Check, esempio 2


```
create table Impiegato
(
Matricola character(6),
Cognome character(20),
Nome character(20),
Sesso character not null check (sesso in ('M','F'))
Stipendio integer,
Ritenute integer,
Netto integer,
Superiore character(6),
check (Netto = Stipendio - Ritenute )
)
```

ok

Check, esempio 3

```
insert into Imp values (
1 , 'Rossi', 'Mario', '', 100, 20, 80 );

insert into Imp values (
2 , 'Neri', 'Mario', 'M', 100, 10, 80 );

insert into Imp values (
3 , 'Rossini', 'Luca', 'M', 70, 20, 50 )
```

Vincoli di integrità generici: asserzioni

- Specifica vincoli a livello di schema

```
create assertion NomeAss check ( Condizione )

create assertion AlmenoUnImpiegato
     check (1 <= (  select count(*)
                    from Impiegato ))
```

non supportata

Viste


```
create view NomeVista [ ( ListaAttributi ) ] as SelectSQL
[ with [ local | cascaded ] check option ]

create view ImpiegatiAmmin
        (Nome, Cognome, Stipendio) as
    select Nome, Cognome, Stipendio
    from Impiegato
    where Dipart = 'Amministrazione' and
          Stipendio > 10
```

Interrogazioni sulle viste

- Possono fare riferimento alle viste come se fossero relazioni di base

```
select * from ImpiegatiAmmin
```

equivale a (e viene eseguita come)

```
select Nome, Cognome, Stipendio
from Impiegato
where Dipart = 'Amministrazione' and
      Stipendio > 10
```

Aggiornamenti sulle viste

- Ammessi (di solito) solo su viste definite su una sola relazione
- Alcune verifiche possono essere imposte


```
create view ImpiegatiAmminPoveri as
    select *
    from ImpiegatiAmmin
    where Stipendio < 50
    with check option
```

- check option permette modifiche, ma solo a condizione che la ennupla continui ad appartenere alla vista (non posso modificare lo stipendio portandolo a 60)


```
create view ImpiegatiAmminPoveri as
    select *
    from ImpiegatiAmmin
    where Stipendio < 50
    with check option

update ImpiegatiAmminPoveri
    set stipendio = 60
    where nome = 'Paola'
```

Un'interrogazione non standard

- Interrogazione scorretta

```
select avg(count(distinct Ufficio))
from Impiegato
group by Dipart
```

- Con una vista

```
create view DipartUffici(NomeDip,NroUffici) as
    select Dipart, count(distinct Ufficio)
    from Impiegato
    group by Dipart;

select avg(cast(NroUffici as decimal(5,2))) as NumeroMedioUffici
from DipartUffici
```

Ancora sulle viste

- La nidificazione nella having non è ammessa in alcuni sistemi

```
select Dipart
from Impiegato
group by Dipart
having sum(Stipendio) >= all
   (select sum(Stipendio)
    from Impiegato
    group by Dipart)
```

Soluzione con le viste

```
create view BudgetStipendi(Dip,TotaleStipendi) as
    select Dipart, sum(Stipendio)
    from Impiegato
    group by Dipart

select Dip
from BudgetStipendi
where TotaleStipendi =(select max(TotaleStipendi)
                       from BudgetStipendi)
```

Viste ricorsive

- Per ogni persona, trovare tutti gli antenati, avendo
  Paternita (Padre, Figlio)

- Serve la ricorsione; in Datalog:

  Discendenza (Antenato: p, Discendente: f) ←
  Paternita (Padre: p, Figlio: f)

  Discendenza (Antenato: a, Discendente: d) ←
  Paternita (Padre: a, Figlio: f) ,
  Discendenza (Antenato: f, Discendente: d)

Viste ricorsive in SQL:1999


```
with Discendenza(Antenato,Discendente) AS
(       select Padre, Figlio
        from Paternita
   union all
        select D.Antenato, Figlio
        from Discendenza D, Paternita
        where D.Discendente = Padre)
select *
from Discendenza
```

Funzioni scalari

- Funzioni a livello di ennupla che restituiscono singoli valori
- Temporali
  - current_date, extract(year from …)
- Manipolazione stringhe
  - char_length, lower
- Conversione
  - cast
- Condizionali
  - …

Funzioni condizionali

- Case, coalesce, nullif

```
select Nome, Cognome, coalesce(Dipart,'Ignoto')
from Impiegato

select Targa,
       case Tipo
              when 'Auto' then 2.58 * KWatt
              when 'Moto' then (22.00 + 1.00 * KWatt)
              else null
       end as Tassa
  from Veicolo
  where Anno > 1975
```

Basi di dati attive

- Una base di dati che contiene regole attive (chiamate *trigger*)
- Presentazione:
  - Definizione dei trigger in SQL:1999
  - Definizione dei trigger in DB2 e Oracle
  - Problemi di progetto per applicazioni basate sull'uso dei trigger

Il concetto di trigger

- Paradigma: Evento-Condizione-Azione
  - Quando un evento si verifiva
  - Se la condizione è vera
  - Allora l'azione è eseguita
- Questo modello consente computazioni reattive
- Non è il solo tipo di regole:
  - Vincoli di integrità
  - Regole datalog
  - Regole di business
- Problema: è difficile realizzare applicazioni complesse

Evento-Condizione-Azione

- Evento
  - Normalmente una modifica dello stato del database: insert, delete, update
  - Quando accade l'evento, il trigger è *attivato*
- Condizione
  - Un predicato che identifica se l'azione del trigger deve essere eseguita
  - Quando la condizione viene valutata, il trigger è *considerato*
- Azione
  - Una sequenza di update SQL o una procedura
  - Quando l'azione è eseguita anche il trigger è *eseguito*
- I DBMS forniscono tutti i componenti necessari. Basta integrarli.

SQL:1999, Sintassi

- Lo standard SQL:1999 (SQL-3) sui trigger è stato fortemente influenzato da DB2 (IBM); gli altri sistemi non seguono lo standard (esistono dagli anni 80')
- Ogni trigger è caratterizzato da:
  - nome
  - target (tabella controllata)
  - modalità (**`before`** o **`after`**)
  - evento (**`insert`**, **`delete`** o **`update`**)
  - granularità (statement-level o row-level)
  - alias dei valori o tabelle di transizione
  - azione
  - timestamp di creazione

SQL:1999, Sintassi


```
create trigger NomeTrigger
  { before | after }
  { insert | delete | update [of Column] } on TabellaTarget
  [referencing
        {[old table [as] VarTuplaOld]
         [new table [as] VarTuplaNew] } |
        {[old [row] [as] VarTabellaOld]
         [new [row] [as] VarTabellaNew] }]
  [for each { row | statement }]
  [when Condizione]
  StatementProceduraleSQL
```

Tipi di eventi

- BEFORE
  - Il trigger è considerato e possibilmente eseguito prima dell'evento (i.e., la modifica del database)
  - I trigger before non possono modificare lo stato del database; possono al più condizionare i valori "new" in modalità row-level (set t.new=expr)
  - Normalmente questa modalità è usata quando si vuole verificare una modifica prima che essa avvenga e "modificare la modifica"
- AFTER
  - Il trigger è considerato e eseguito dopo l'evento
  - E' la modalità più comune, adatta alla maggior parte delle applicazioni

Esempio “before” e “after”

- 1. “Conditioner” (agisce prima dell’update e della verifica di integrità)

```
create trigger LimitaAumenti
before update of Salario on Impiegato
for each row
when (New.Salario > Old.Salario * 1.2)
set New.Salario = Old.Salario * 1.2
```

- 2. “Re-installer” (agisce dopo l’update)

```
create trigger LimitaAumenti
after update of Salario on Impiegato
for each row
when (New.Salario > Old.Salario * 1.2)
set New.Salario = Old.Salario * 1.2
```

Granularità degli eventi

- Modalità statement-level (di default, opzione **`for each statement`**)
  - Il trigger viene considerato e possibilmente eseguito solo una volta per ogni statement (comando) che lo ha attivato, independentemente dal numero di tuple modificate
  - In linea con SQL (set-oriented)
- Modalità row-level (opzione **`for each row`**)
  - Il trigger viene considerato e possibilmente eseguito una volta per ogni tupla modificata
  - Scrivere trigger row-level è più semplice

Clausola `referencing`

- Dipende dalla granularità
  - Se la modalità è row-level, ci sono due *variabili di transizione* (**`old`** and **`new`**) che rappresentano il valore precedente o successivo alla modifica di una tupla
  - Se la modalità è statement-level, ci sono due *tabelle di transizione* (**`old table`** and **`new table`**) che contengono i valori precedenti e successivi delle tuple modificate dallo statement
- **`old`** e **`old table`** non sono presenti con l'evento **`insert`**
- **`new`** e **`new table`** non sono presenti con l'evento **`delete`**

Esempio di trigger row-level


```
create trigger AccountMonitor
  after update on Account
  for each row
  when new.Totale > old.Totale
  insert values
      (new.NumeroConto,
       new.Totale-old.Totale)
      into Pagamenti
```

Esempio di trigger statement-level


```
create trigger ArchiviaFattureCancellate
  after delete on Fattura
  referencing old_table as VecchieFatture
  insert into FattureCancellate
  (select *
   from VecchieFatture)
```

Triggers in DB2

- Seguono la sintassi e semantica di SQL:1999
- Esempio: gestione salari

```
create trigger ControllaStipendi
after update of Stipendio on Impiegato
for each row
when (new.Stipendio < old.Stipendio * 0.97)
begin
  update Impiegato
  set Stipendio = old.Stipendio * 0.97
  where Matr = new.Matr;
end;
```

Esecuzione di Trigger in conflitto

- Quando vi sono più trigger associati allo stesso evento (in conflitto) vengono eseguiti come segue:
  - Per primi i **`before`** triggers (*statement-level* e *row-level*)
  - Poi viene eseguita la modifica e verificati i vincoli di integrità
  - Infine sono eseguiti gli **`after`** triggers (*row-level* e *statement level*)
- Quando vari trigger apparteengono alla stessa categoria, l'ordine di esecuzione è definito in base al loro timestamp di creazione (i trigger più vecchi hanno priorità più alta)

Modello di esecuzione ricorsivo

- In SQL:1999 i triggers sono associati ad un “Trigger Execution Context” (TEC)
- L’azione di un trigger può produrre eventi che attivano altri trigger, che verranno valutati con un nuovo TEC interno:
  - Lo stato del TEC includente viene salvato e quello del TEC incluso viene eseguito. Ciò può accadere ricorsivamente
  - Alla fine dell’esecuzione di un TEC incluso, lo stato di esecuzione del TEC includente viene ripristinato e la sua esecuzione ripresa
- L’esecuzione termina correttamente in uno “stato quiescente”
- L’esecuzione termina in errore quando si raggiunge una data profondità di ricorsione dando luogo ad una eccezione di non-terminazione
- Se si verifica un errore o eccezione durante l’esecuzione di una catena di trigger attivati inizialmente da uno statement S, viene fatto un rollback parziale di S

Trigger in Oracle

- Si usa una sintassi differente: sono consentiti eventi multipli, non sono previste variabili per le tabelle, i before trigger possono prevedere update, la condizione è presente solo con trigger row-level, l'azione è un programma PL/SQL

create trigger *NomeTrigger*
{ **before** | **after** } *evento* [, *evento* [, *evento*]]
[[**referencing**
[**old** [**row**] [**as**] *VarTuplaOld*]
[**new** [**row**] [**as**] *VarTuplaNew*]]
for each { **row** | **statement** } [**when** *Condizione*]]
BloccoPL/SQL

Evento ::= { **insert** | **delete** | **update** [**of** *Attributo*] } **on** *Tabella*

Conflitti tra i trigger in Oracle

- Quando molti trigger sono associati allo stesso evento, ORACLE segue il seguente schema:
  - Per primi, i **`before`** statement-level trigger
  - Poi, i **`before`** row-level trigger
  - Poi viene eseguita la modifica e verificati i vincoli di integrità
  - Poi, gli **`after`** row-level trigger
  - Infine, gli **`after`** statement-level trigger
- Quando vari trigger appartengono alla stessa categoria, l'ordine di esecuzione è definito in base al loro timestamp di creazione (i trigger più vecchi hanno priorità più alta)
- "*Mutating table exception*": scatta se la catena di trigger attivati da un **`before`** trigger T cerca di modificare lo stato della tabella target di T

Esempio of Trigger in Oracle

Evento: `update` of QtaDisponibile in Magazzino
Condizione: Quantità sotto soglia e mancanza ordini esterni
Azione: `insert` of OrdiniEsterni

```
create trigger Riordino
after update of QtaDisponibile on Magazzino
when (new.QtaDisponibile < new.QtaSoglia)
for each row
 declare
  X number;
   select count(*) into X
   from OrdiniEsterni
   where Parte = new.Parte;
  if X = 0
  then
   insert into OrdiniEsterni
    values(new.Parte,new.QtaRiordino,sysdate)
  end if;
 end;
```

Proprietà formali dei trigger

- E' importante garantire che l'interferenza tra trigger in una qualunque loro attivazione non produca comportamenti anomali
- Vi sono tre proprietà classiche:
  - Terminazione: per un qualunque stato iniziale e una qualunque transazione, si produce uno stato finale (stato quiescente)
  - Confluenza: L'esecuzione dei trigger termina e produce un unico stato finale, indipendente dall'ordine di esecuzione dei trigger
  - Univoca osservabilità: I trigger sono confluenti e producono verso l'esterno (messaggi, azioni di display) lo stesso effetto
- La terminazione è la proprietà principale

Analisi della terminazione

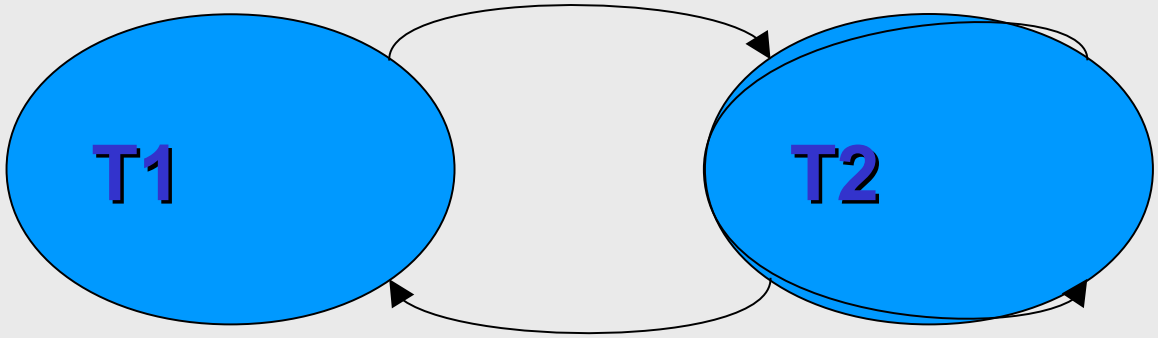
- Si usa una rappresentazione delle regole detta grafo di triggering:
  - Un nodo per ogni trigger
  - Un arco dal nodo t_i al nodo t_j se l'esecuzione dell'azione di t_i può attivare il trigger t_j (ciò può essere dedotto con una semplice analisi sintattica)
- Se il grafo è aciclico, l'esecuzione termina
  - Non possono esservi sequenze infinite di triggering
- Se il grafo ha cicli, esso *può* avere problemi di terminazione: lo si capisce guardando i cicli uno per uno.

Esempio con due trigger


```
T1:    create trigger CorreggiContributi
       after update of Stipendio on Impiegato
       referencing new table as NewImp
       update Impiegato
       set Contributi = Stipendio * 0.8
       where Matr in (select Matr
                      from NewImp)

T2:    create trigger ControllaSogliaBudget
       after update on Impiegato
       when 50000 < (select    (New.Stipendio
                    + New.Contributi)
                      from Impiegato)
       update Impiegato
       set Stipendio = 0.9 * Stipendio
```

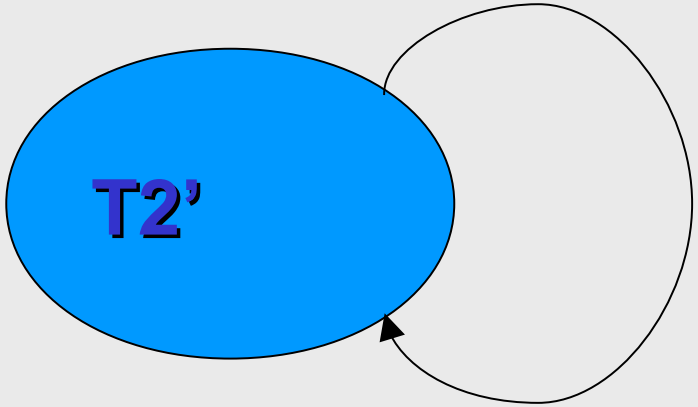
Grafo di triggering corrispondente



- Ci sono due cicli ma il sistema termina.
- Per renderlo non terminante basta cambiare il comparatore nella condizione di T2 oppure moltiplicare per un fattore più grande di 1 nella azione di T2.

Esempio di non terminazione


```
T2′:   create trigger ControllaSogliaBudget
       after update on Impiegato
       when New.Stipendio < 50000
       update Stipendio
       set Stipendio = 0.9 * Stipendio
```



Aspetti evoluti delle basi di dati attive

- Modalità di esecuzione (immediata, differita, distaccata)
- Amministrazione delle regole (priorità, gruppi, attivazione e deattivazione dinamica
- Clausola “Instead-of”
- Altri eventi (di sistema, di utente, system-defined)
- Eventi complessi e calcolo degli eventi
- Una nuova categoria di sistema: “stream database”.

Modalità di esecuzione

- E’ il collegamento tra attivazione (evento) e considerazione/esecuzione (condizione e azione)
- Condizione e azione sono sempre valutate assieme
- Caso “immediato”: considerazione e esecuzione assieme all’evento
- Alternative:
  - Differito: il trigger è valutato alla fine della transazione
    - Esempio d’uso: trigger che gestiscono vincoli di integrità che possono essere violati durante una transazione
  - Distaccato: il trigger è valutato in un’altra transazione
    - Esempio d’uso: gestione di una variazione di valore di titoli della borsa

Priorità, attivazioni e gruppi

- Definizione di priorità:
  - Specifica l'ordine di esecuzione dei trigger quando molti di loro vengono attivati dallo stesso evento
  - SQL:1999 indica la priorità di differenti classi di trigger; all'interno di una classe l'ordine dipende dall'ordine di creazione
- Attivazione/deattivazione dei trigger
  - Non è standard, ma è spesso disponibile
- Organizzazione dei trigger in gruppi
  - Alcuni sistemi consentono di raggruppare trigger e quindi di attivarli/deattivarli come gruppo

Clausola `instead of`

- Alternativa a **`before`** e **`after`**
- Viene eseguita una differente operazione rispetto a quella che ha attivato il trigger
- La semantica è piuttosto pericolosa (l'applicazione fa una cosa e il sistema ne fa un'altra)
- Implementata in alcuni sistemi con limitazioni
  - In Oracle si può usare per ridirigere gli update dalle viste alle tabelle di base in caso di ambiguità

Controllo dell'accesso

- In SQL è possibile specificare chi (utente) e come (lettura, scrittura, …) può utilizzare la base di dati (o parte di essa)
- Oggetto dei privilegi (diritti di accesso) sono di solito le tabelle, ma anche altri tipi di risorse, quali singoli attributi, viste o domini
- Un utente predefinito _system (amministratore della base di dati) ha tutti i privilegi
- Il creatore di una risorsa ha tutti i privilegi su di essa

Privilegi

- Un privilegio è caratterizzato da:
  - la risorsa cui si riferisce
  - l'utente che concede il privilegio
  - l'utente che riceve il privilegio
  - l'azione che viene permessa
  - la trasmissibilità del privilegio

Tipi di privilegi offerti da SQL

- insert: permette di inserire nuovi oggetti (ennuple)
- update: permette di modificare il contenuto
- delete: permette di eliminare oggetti
- select: permette di leggere la risorsa
- references: permette la definizione di vincoli di integrità referenziale verso la risorsa (può limitare la possibilità di modificare la risorsa)
- usage: permette l'utilizzo in una definizione (per esempio, di un dominio)

grant e revoke

- Concessione di privilegi:

  grant < *Privileges* | all privileges > on *Resource*
  to *Users* [with grant option]

  - grant option specifica se il privilegio può essere trasmesso ad altri utenti

    grant select on Department to Stefano

- Revoca di privilegi

  revoke *Privileges* on *Resource* from *Users*
  [restrict | cascade]

Autorizzazioni, commenti

- La gestione delle autorizzazioni deve "nascondere" gli elementi cui un utente non può accedere, senza sospetti
- Esempio:
  - Impiegati non esiste (esiste Impiegati)
  - ImpiegatiSegreti esiste, ma l'utente non è autorizzato
- L'utente deve ricevere lo stesso messaggio

Autorizzazioni, commenti, 2

- Come autorizzare un utente a vedere solo alcune ennuple di una relazione?
  - Attraverso una vista:
    - Definiamo la vista con una condizione di selezione
    - Attribuiamo le autorizzazioni sulla vista, anziché sulla relazione di base

Autorizzazioni, ancora

- (Estensioni di SQL:1999)
- Concetto di ruolo, cui si associano privilegi (anche articolati), poi concessi agli utenti attribuendo il ruolo

Transazione

- Insieme di operazioni da considerare indivisibile ("atomico"), corretto anche in presenza di concorrenza e con effetti definitivi
- Proprietà ("acide"):
  - Atomicità
  - Consistenza
  - Isolamento
  - Durabilità (persistenza)

Le transazioni sono … atomiche

- La sequenza di operazioni sulla base di dati viene eseguita per intero o per niente:
  - trasferimento di fondi da un conto A ad un conto B: o si fanno il prelevamento da A e il versamento su B o nessuno dei due

Le transazioni sono … consistenti

- Al termine dell'esecuzione di una transazione, i vincoli di integrità debbono essere soddisfatti
- "Durante" l'esecuzione ci possono essere violazioni, ma se restano alla fine allora la transazione deve essere annullata per intero ("abortita")

Le transazioni sono … isolate

- L'effetto di transazioni concorrenti deve essere coerente (ad esempio "equivalente" all'esecuzione separata)
  - se due assegni emessi sullo stesso conto corrente vengono incassati contemporaneamente si deve evitare di trascurarne uno

I risultati delle transazioni sono durevoli

- La conclusione positiva di una transazione corrisponde ad un impegno (in inglese commit) a mantenere traccia del risultato in modo definitivo, anche in presenza di guasti e di esecuzione concorrente

Transazioni in SQL

- Una transazione inizia al primo comando SQL dopo la "connessione" alla base di dati oppure alla conclusione di una precedente transazione (lo standard indica anche un comando start transaction, non obbligatorio, e quindi non previsto in molti sistemi)
- Conclusione di una transazione
  - commit [work]: le operazioni specificate a partire dall'inizio della transazione vengono eseguite sulla base di dati
  - rollback [work]: si rinuncia all'esecuzione delle operazioni specificate dopo l'inizio della transazione
- Molti sistemi prevedono una modalità autocommit, in cui ogni operazione forma una transazione

Una transazione in SQL


```
start transaction                    (opzionale)
update ContoCorrente
   set Saldo = Saldo – 10
   where NumeroConto = 12345 ;
update ContoCorrente
   set Saldo = Saldo + 10
   where NumeroConto = 55555 ;
commit work;
```