Vincoli

Chiavi esterne

Vincoli locali e globali

Triggers

Leggere Cap 2 Riguzzi et al. Sistemi Informativi

Lucidi derivati da quelli di Jeffrey D. Ullman

Vincoli e Triggers

- Un *vincolo* e' una relazione tra dati che il DBMS deve assicurare.
  - Esempio: vincoli di chiave.
- *I triggers* sono eseguiti solo quando una condizione specifica accade, ad es., l'inserimento di una tupla.
  - Piu' facile da implementare di molti vincoli.

Tipo di vincoli

- Chiavi.
- Chiavi esterne, o integrita' referenziale.
- Vincoli basati su un solo attributo.
  - Vincola i valori di uno specifico attributo.
- Vincoli basati sulla tupla.
  - Relazioni tra attributi.
- Asserzioni: qualsiasi espressione booleana SQL.

Chiavi esterne

- Si consideri la relazione Sells(bar, beer, price).
- Ci possiamo aspettare che un valore beer sia una birra reale, ovvero qualcosa che appare in Beers.name.
- Un vincolo che richiede che una birra in Sells sia una birra in Beers e' chiamato un vincolo di *chiave esterna* (foreign-key constraint).

Esprimere le chiavi esterne

- Si usa la parola chiave REFERENCES:
  1. Nella dichiarazione di un attributo, quando un solo attributo e' coinvolto, oppure
  2. Come un elemento dello schema, come in:

  FOREIGN KEY (<lista di attributi>)

  REFERENCES <relazione> (<attributi>)
- Gli attributi referenziati devono essere dichiarati PRIMARY KEY o UNIQUE.

Esempio: con attributo

```
CREATE TABLE Beers (
   name    CHAR(20) PRIMARY KEY,
   manf    CHAR(20) );
CREATE TABLE Sells (
   bar     CHAR(20),
   beer    CHAR(20) REFERENCES Beers(name),
   price   REAL );
```

Esempio: come elemento

```
CREATE TABLE Beers (
   name    CHAR(20) PRIMARY KEY,
   manf    CHAR(20) );
CREATE TABLE Sells (
   bar     CHAR(20),
   beer    CHAR(20),
   price   REAL,
   FOREIGN KEY(beer) REFERENCES
       Beers(name));
```

Applicare i vincoli di chiave esterna

- Se c'e' un vincolo di chiave esterna da attributi della relazione *R* alla chiave primaria della relazione *S*, sono possibili due violazioni:
  1. Un inserimento o una modifica in *R* introducono valori non presenti in *S*.
  2. Una cancellazione o una modifica in *S* fa si' che alcune tuple di *R* rimangano "a penzoloni"

Azioni intraprese - 1

- Si supponga *R* = Sells, *S* = Beers.
- Un inserimento o una modifica di Sells che introduce un birra non esistente in Beers deve essere rifiutato.
- Una cancellazione o una modifica di Beers che rimuove un valore di birra presente in alcune tuple di Sells puo' essere gestito in quattro modi.

Azioni intraprese -- 2

- I quattro modi possibili per gestire le birre che improvvisamente smettono di esistere sono:
  1. NO ACTION : rifiuta la modifica.
  2. CASCADE : effettua gli stessi cambiamenti in Sells.
     - Birra cancellata: cancella la tupla di Sells.
     - Birra modificata: modifica il valore in Sells.
  3. SET NULL : modifica la birra in NULL.
  4. SET DEFAULT : assegna alla birra il suo valore di default

Esempio: CASCADE

- Si supponga di cancellare la tupla per la Bud da Beers.
  - Allora si cancellano tutte le tuple di Sells che hanno beer = ’Bud’.
- Si supponga di aggiornare la tupla per la Bud cambiando ’Bud’ in ’Budweiser’.
  - Allora si cambiano tutte le tuple di Sells con beer = ’Bud’ in modo che beer = ’Budweiser’.

Esempio: SET NULL

- Si supponga di cancellare la tupla per la Bud da Beers.
  - Si cambiano tutte le tuple di Sells che hanno beer = ’Bud’ in beer = NULL.
- Si supponga di aggiornare la tuple per la Bud cambiando ’Bud’ in ’Budweiser’.
  - Stesso cambiamento.

Esempio: SET DEFAULT

- Si supponga di cancellare la tupla per la Bud da Beers.
  - Si assegna a tutte le tuple di Sells che hanno beer = ’Bud’ il valore di default per beer
- Si supponga di aggiornare le tuple per la Bud cambiando ’Bud’ in ’Budweiser’.
  - Stesso cambiamento.
- La colonna beer deve avere un vincolo di default. Se non ce l’ha ed è nullable, allora NULL diventa il valore della colonna. Altrimenti errore

Scegliere una politica

- Quando dichiariamo una chiave esterna, possiamo scegliere la politica SET NULL, CASCADE o SET DEFAULT indipendentemente per le cancellazioni e le modifiche.
- Far seguire la dichiarazione di chiave esterna da:

ON [UPDATE | DELETE][SET NULL | CASCADE | SET DEFAULT]

- Due clausole di questo tipo possono essere usate.
- Altrimenti, il default (impedisci (NO ACTION)) e' usato.

Esempio

```
CREATE TABLE Sells (
    bar CHAR(20),
    beer        CHAR(20),
    price       REAL,
    FOREIGN KEY(beer)
        REFERENCES Beers(name)
        ON DELETE SET NULL
        ON UPDATE CASCADE );
```

Check basati sugli attributi

- Mettono un vincolo sul valore di un particolare attributo.
- CHECK(<condizione>) deve essere aggiunto alla dichiarazione dell'attributo.
- La condizione puo' essere una qualunque condizione che puo' apparire in una clausola WHERE
- La condizione puo' usare il nome dell'attributo ma ogni altra relazione o nome di attributo devono apparire in una sottoquery

Esempio

```
CREATE TABLE Sells (
   bar CHAR(20),
   beer       CHAR(20)   CHECK ( beer IN
                (SELECT name FROM Beers)),
   price      REAL CHECK ( price <= 5.00 )
);
```

Temporizzazione delle verifiche

- Un check basato sugli attributi e' verificato solo quando un valore per quell'attributo viene inserito o modificato.
  - Esempio: CHECK (price <= 5.00) verifica ogni nuovo prezzo e lo rifuta se e' piu di $5.
  - Esempio: CHECK (beer IN (SELECT name FROM Beers)) non e' verificato se una birra viene cancellata da Beers (a differenza delle chiavi esterne).

Check basati sulle tuple

- CHECK (<condizione>) puo' essere aggiunto come un altro elemento di una definizione di schema.
- La condizione puo' essere una qualunque condizione che puo' apparire in una clausola WHERE
- La condizione puo' fare riferimento a ogni attributo della relazione ma ogni altro attributo o relazione richiede una sottoquery.
- Verificato solo agli inserimenti e alle modifiche.

Esempio: check

- Solo Joe's Bar puo' vendere birra a piu' di 5$:

```
CREATE TABLE Sells (
        bar             CHAR(20),
        beer            CHAR(20),
        price  REAL,
        CHECK (bar = 'Joe''s Bar' OR
                                price <= 5.00)
);
```

Asserzioni

- Sono elementi dello schema di database, come le relazioni e le viste.
- Definiti da:

  CREATE ASSERTION <nome>

  CHECK (<condizione>);
- La condizione puo' essere una qualunque condizione che puo' apparire in una clausola WHERE
- La condizione puo' fare riferimento a ogni relazione o attributo nello schema del database.

Asserzioni

- Le asserzioni non sono presenti in SQL Server

Esempio: asserzione

- In Sells(bar, beer, price), nessun bar puo' chiedere un prezzo medio superiore a 5$.

```
CREATE ASSERTION NoRipoffBars CHECK (
   NOT EXISTS (
      SELECT bar FROM Sells
      GROUP BY bar
      HAVING AVG(price)>5.00));
```

Bar con un prezzo medio superiore a 5$

Esempio: asserzione

- In Drinkers(name, addr, phone) e Bars(name, addr, license), non ci possono essere piu' bar che drinkers.

```
CREATE ASSERTION FewBar CHECK (
   (SELECT COUNT(*) FROM Bars) <=
   (SELECT COUNT(*) FROM Drinkers)
);
```

Temporizzazione della verifica delle asserzioni

- In principio dobbiamo verificare ogni asserzione dopo ogni modifica a una qualunque relazione del database.
- Un sistema intelligente puo' osservare che solo certe modifiche possono rendere una asserzione violata.
  - Esempio: nessuna modifica a Beers puo' avere effetto su FewBar. Neppure un inserimento in Drinkers o una cancellazione da Bars.

Triggers: motivazione

- I checks basati su attributi e su tuple hanno capacita' limitate.
- Le asserzioni sono sufficientemente generali per la maggior parte delle applicazioni dei vincoli, ma sono difficili da implementare in maniera efficiente.
  - Il DBMS deve aver una vera intelligenza per evitare di verificare asserzioni che non possono essere violate.

Soluzione: triggers

- Un trigger consente all'utente di specificare quando deve essere fatta la verifica.
- Come una asserzione, un trigger ha una condizione generale e puo' inoltre effettuare una qualunque sequenza di modifiche SQL al database.

Triggers in SQL-2003

- Vediamo prima i trigger in SQL-2003

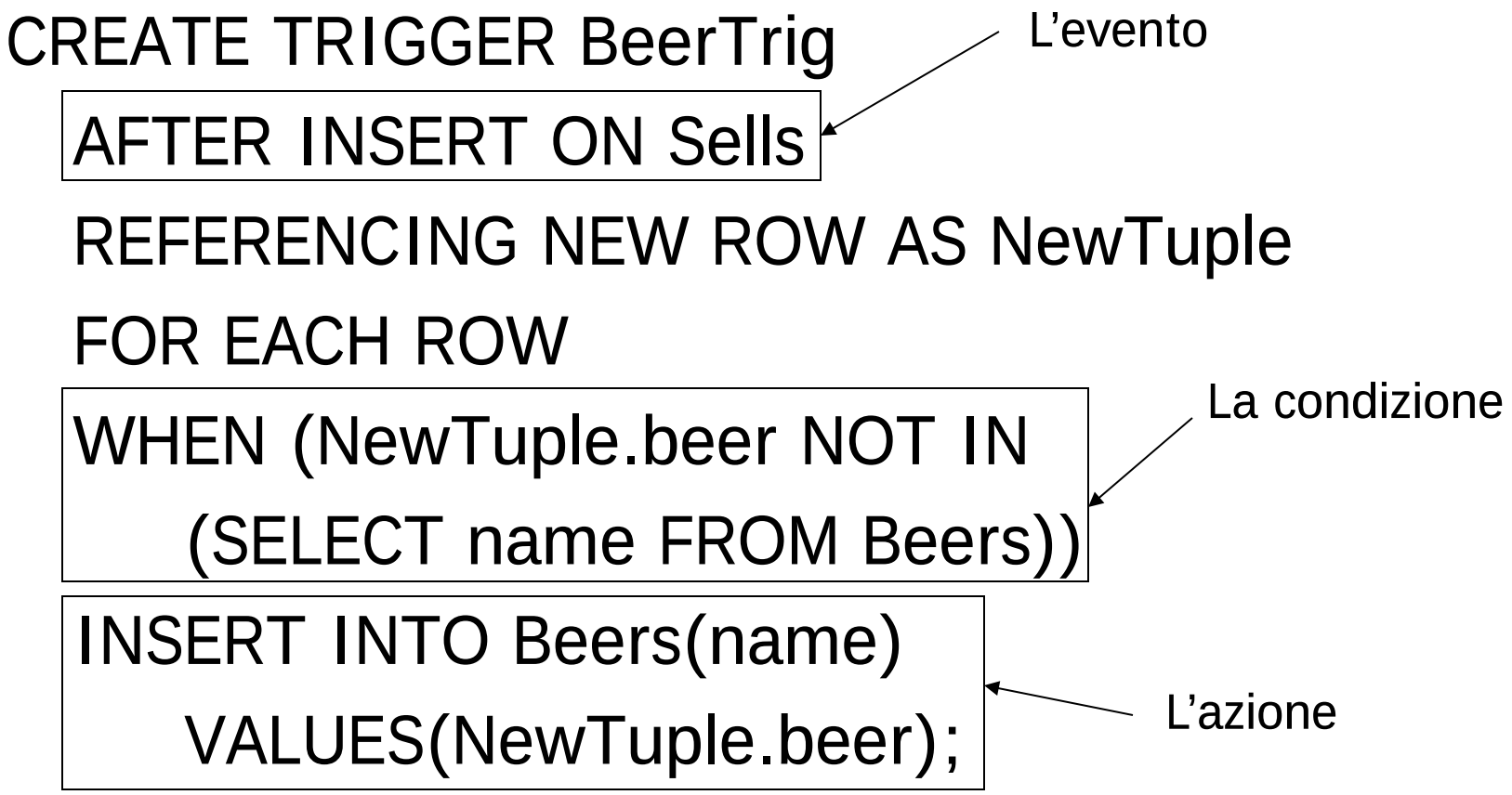
Regole Evento-Condizione-Azione

- Un altro nome per “trigger” e’ *regola ECA*, o regola evento-condizione-azione.
- *Evento* : tipicamente un tipo di modifica di database, ad es. “insert on Sells.”
- *Condizione* : una qualunque espressione SQL con valore booleano.
- *Azione* : una qualunque istruzione SQL.

Esempio: un trigger

- Ci sono molti dettagli da imparare riguardo i triggers.
- Ecco un esempio per illustrarli.
- Invece di usare un vincolo di chiave esterna e rifiutare gli inserimenti in Sells(bar, beer, price) con beer sconosciuto, un trigger puo' aggiungere la birra a Beers, con un produttore NULL.

Esempio: definizione di trigger



Opzioni: CREATE TRIGGER

- CREATE TRIGGER <nome>
- Opzione:

CREATE OR REPLACE TRIGGER <nome>

  - Utile se s'e' gia' un trigger con quel nome e lo si vuole modificare.

Opzioni: evento

- AFTER puo' essere BEFORE.
  - Anche, INSTEAD OF, se la relazione e' una vista.
    - Un modo efficace di eseguire modifiche di viste: usare i trigger per tradurle in modifiche delle tabelle originarie.
- INSERT puo' essere DELETE o UPDATE.
  - e UPDATE puo' essere UPDATE OF <attributo> ON <tabella>.

Opzioni: FOR EACH ROW

- I triggers sono a *livello di riga* o a *livello di istruzione*.
- FOR EACH ROW indica il livello di riga; la sua assenza indica il livello di istruzione.
- I triggers a livello di riga sono eseguiti una volta per ogni tupla modificata.
- I trigger a livello di istruzione sono eseguiti una volta per una istruzione SQL, indipendentemente da quante tuple sono modificate.

Opzioni: REFERENCING

- Le istruzioni INSERT implicano una nuova tupla (per il livello riga) o un nuovo insieme di tuple (per il livello di istruzione).
- DELETE implica una tupla o una tabella vecchia.
- UPDATE le implica entrame.
- Ci si riferisce ad esse con

[NEW| OLD][ROW |TABLE] AS <nome>

Opzioni: la condizione

- Una qualunque condizione a valori booleani.
- Viene valutata prima o dopo l'evento, a seconda che BEFORE o AFTER sia usato nell'evento.
- Si accede alla nuova/vecchia tupla o insieme di tuple attraverso i nomi dichiarati nella clausola REFERENCING.

Opzioni: l'azione

- Ci puo' essere piu' di una istruzione SQL nell'azione.
  - Si racchiudono tra BEGIN . . . END se ce n'e' piu' di una.
- Ma le query non hanno senso nell'azione, quindi siamo in pratica limitati a modifiche.

Un altro esempio

- Usando Sells(bar, beer, price) e una relazione unaria RipoffBars(bar) creata allo scopo, mantieni una lista di bar che aumentano il prezzo di una qualunque birra di piu’ di 1$.

Il trigger

```
CREATE TRIGGER PriceTrig
    AFTER UPDATE OF price ON Sells
    REFERENCING
        OLD ROW as old
        NEW ROW as new
    FOR EACH ROW
    WHEN(new.price > old.price + 1.00)
    INSERT INTO RipoffBars
        VALUES(new.bar);
```

L'evento- solo cambia- menti di prezzo

Gli agg. ci consentono di parlare di vecchie e nuove tuple

Dobbiamo considerare ogni cambiamento di prezzo

Condizione: un increm. di pr. > $1

Quando il camb. di prezzo e' abbastanza grande, aggiungi il bar a RipoffBars

Trigger sulle viste

- Generalmente, e' impossibile modificare una vista ottenuta dal join di più tabelle.
- Ma un trigger INSTEAD OF ci consente di interpretare le modifche alle viste in una maniera che ha senso.
- Esempio: progetteremo una vista Synergy che ha triple (drinker, beer, bar) tali che il bar serve la birra, il drinker frequenta il bar e quella birra gli piace.
- Si usano le relazioni Sells(bar, beer, price) Frequents(bar,drinker) e Likes(drinker,beer)

Esempio: la vista

Prendi una copia di ogni attributo

```
CREATE VIEW Synergy AS
    SELECT Likes.drinker, Likes.beer, Sells.bar
    FROM Likes, Sells, Frequents
    WHERE Likes.drinker = Frequents.drinker
        AND Likes.beer = Sells.beer
        AND Sells.bar = Frequents.bar;
```

Join naturale di Likes, Sells e Frequents

Interpretare un inserimento in una vista

- Non possiamo fare un inserimento in Synergy --- e' una vista ottenuta con un join.
- Ma possiamo usare un trigger INSTEAD OF per trasformare una tripla (drinker, beer, bar) in tre inserimenti di coppie proiettate, una per ciascuna relazione Likes, Sells, and Frequents.
  - Sells.price avra' valore NULL.

Il trigger

```
CREATE TRIGGER ViewTrig
    INSTEAD OF INSERT ON Synergy
    REFERENCING NEW ROW AS n
    FOR EACH ROW
    BEGIN
        INSERT INTO LIKES VALUES(n.drinker, n.beer);
        INSERT INTO SELLS(bar, beer) VALUES(n.bar, n.beer);
        INSERT INTO FREQUENTS VALUES(n.drinker, n.bar);
    END;
```

Ordine di esecuzione dei trigger

- Se ci sono piu' trigger associati allo stesso evento essi vengono eseguiti in questo ordine:
  - Trigger BEFORE a livello di istruzione
  - Trigger BEFORE a livello di riga
  - Si applica la modifica e si verificano i vincoli di integrita'
  - Trigger AFTER a livello di riga
  - Trigger AFTER a livello di istruzione

Ordine di esecuzione dei trigger

- Se ci sono piu' trigger nella stessa categoria vengono eseguiti in un ordine scelto dal sistema che dipende dall'implementazione (non determinismo)

Proprieta' dei trigger

- 3 proprieta' classiche
  - Terminazione: per qualunque stato iniziale e sequenza di modifiche i trigger producono uno stato finale (non ci sono cicli infiniti di attivazione)
  - Confluenza: i trigger terminano e producono un unico stato finale, indipendente dall'ordine in cui vengono eseguiti

Proprieta' dei trigger

- Determinismo delle osservazioni: I trigger sono confluenti e inoltre producono la stessa sequenza di azioni (inclusi i messaggi)

- La proprieta' di terminazione e' di gran lunga la piu' importante

Analisi di terminazione

- Si esegue costruendo un grafo di attivazione:
  - Si crea un nodo per ogni trigger
  - Si crea un arco da un trigger T1 a un trigger T2 se l'azione di T1 contiene una primitiva che coincide con uno degli eventi di T2 (si puo' verificare con una semplice analisi sintattica)

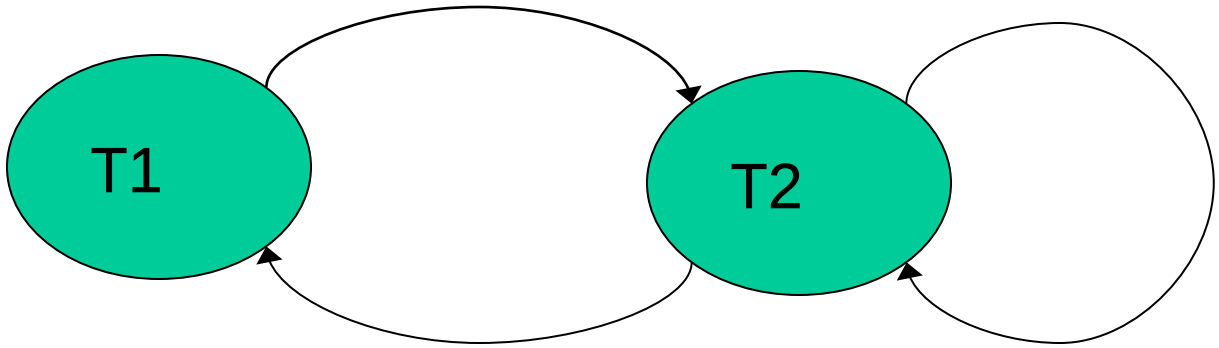
Analisi di terminazione

- Se il grafo e’ aciclico, il sistema termina
- Se il grafo ha dei cicli, ci potrebbero essre delle attivazioni infinite (ma non e’ detto)

Esempio

- T1: CREATE TRIGGER AggiustaContributi
  AFTER UPDATE OF Stipendio ON Impiegato
  REFERENCING NEW TABLE AS NuovoImp
  UPDATE Impiegato SET Contributi=Stipendio*0.8
  WHERE Matr IN (SELECT Matr from NuovoImp)
- T2: CREATE TRIGGER ControlloSogliaBudget
  AFTER UPDATE ON Impiegato
  WHEN 50000<(SELECT SUM(Stipendio+Contributi)
  FROM Impiegato)
  UPDATE Impiegato SET Stipendio=0.9*Stipendio

Esempio



- Ci sono due cicli pero' il sistema e' terminanante
- Se si inverte il verso del confronto in T2 diventa non terminante

Modi di esecuzione rispetto alle transazioni

- 3 alternative
  - Immediato (immediate): il trigger viene considerato ed eseguito con l'evento che lo ha attivato
  - Differito (deferred): il trigger viene eseguito al termine della transazione
  - Distaccato (detached): il trigger viene gestito in una transazione separata. Ad es., si aggiornano le valutazioni di un titolo di borsa dopo diversi scambi