BD 2 - Trigger

Luca Cosmo

Università Ca' Foscari Venezia

Università
Ca'Foscari
Venezia

Recap: Vincoli di Integrità

Abbiamo discusso due meccanismi per garantire vincoli di integrità.

CHECK: efficienti e disponibili nei principali DBMS commerciali

- senza sotto-query: poco espressivi, perchè operanti su singole tuple
- con sotto-query: molto pericolosi, perchè offrono solo garanzie parziali e quindi i principali DBMS non ammettono sotto-query

Asserzioni: puramente dichiarative e con una semantica chiara

- purtroppo molto inefficienti in termini di performance
- non disponibili nei principali DBMS commerciali

Trigger

I trigger sono lo standard de facto per il mantenimento di invarianti globali nei DBMS, perchè possono essere implementati efficientemente.

Paradigma di tipo Evento - Condizione - Azione:

1. Un trigger è associato ad un evento che ne determina l'attivazione. Esempi tipici di eventi sono INSERT, DELETE o UPDATE su una certa tabella.
2. Quando un trigger viene attivato, esso può controllare una certa condizione. Se la condizione è falsa, il trigger termina.
3. Se invece la condizione è vera, viene eseguita la cosiddetta azione associata al trigger. L'azione è una sequenza arbitraria di operazioni sullo schema relazionale.

Attenzione!

I trigger sono stati standardizzati in SQL-3 (1999), mentre molti DBMS commerciali li avevano introdotti già dalla fine degli anni Ottanta!

- Nessuno dei DBMS principali vuole adeguarsi letteralmente allo standard per evitare problemi di backward compatibility
- Nella prima parte della lezione presenteremo i concetti principali descritti nello standard SQL, simili per tutti i DBMS
- Nella seconda parte della lezione ci focalizzeremo su Postgres ed approfondiremo alcune sottigliezze relative ad esso

Trigger per Riga e per Statement

Un evento può coinvolgere righe multiple, quindi SQL fornisce due tipi di trigger diversi:

1. Trigger per riga: eseguiti per ognuna delle righe coinvolte dall'evento scatenante. Si può usare `OLD ROW` e `NEW ROW` per riferirsi alla tupla coinvolta dall'evento prima e dopo la sua occorrenza.
2. Trigger per statement: eseguiti una sola volta per evento scatenante. Si può usare `OLD TABLE` e `NEW TABLE` per riferirsi a tutte le tuple coinvolte dall'evento prima e dopo la sua occorrenza.

E' consentito fare uso di `OLD TABLE` e `NEW TABLE` anche all'interno di un trigger per riga (la riga viene interpretata come una tabella con una singola tupla al suo interno).

BEFORE ed AFTER Trigger

In fase di definizione di un trigger è possibile specificare se l'azione debba essere eseguita prima o dopo l'evento scatenante:

1. BEFORE trigger: attivati prima dell'evento scatenante. Di solito vengono utilizzati per impedire l'esecuzione di un'operazione o per modificarne preventivamente il comportamento.
2. AFTER trigger: attivati dopo l'evento scatenante. Hanno visibilità dello stato della base di dati dopo l'esecuzione di un'operazione e quindi sono talvolta necessari per motivi di espressività.

Un AFTER trigger può simulare l'annullamento di un'operazione facendo un rollback dello stato della base di dati alla situazione precedente, ma l'uso di BEFORE trigger è preferibile quando possibile.

Progettazione di Trigger

I trigger forniscono un modo indiretto per mantenere invarianti globali: essi non adottano lo stile dichiarativo delle asserzioni.

Metodologia per il mantenimento di invarianti tramite trigger:

1. Quali operazioni possono violare l'invariante?
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?
3. Cosa bisogna fare prima o dopo dell'operazione per garantire il mantenimento dell'invariante?

Chiaramente il terzo punto è quello dove abbiamo maggiore libertà...

Esempio 1

Supponiamo di volere utilizzare un trigger per garantire che non sia mai possibile abbassare uno stipendio:

1. Quali operazioni possono violare l'invariante?

Esempio 1

Supponiamo di volere utilizzare un trigger per garantire che non sia mai possibile abbassare uno stipendio:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di aggiornamento

Esempio 1

Supponiamo di volere utilizzare un trigger per garantire che non sia mai possibile abbassare uno stipendio:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di aggiornamento
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?

Esempio 1

Supponiamo di volere utilizzare un trigger per garantire che non sia mai possibile abbassare uno stipendio:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di aggiornamento
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?
   Sì, perchè l'informazione è contestuale alla riga modificata

Esempio 1

Supponiamo di volere utilizzare un trigger per garantire che non sia mai possibile abbassare uno stipendio:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di aggiornamento
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?
   Sì, perchè l'informazione è contestuale alla riga modificata
3. Cosa bisogna fare prima o dopo dell'operazione per garantire il mantenimento dell'invariante?

Esempio 1

Supponiamo di volere utilizzare un trigger per garantire che non sia mai possibile abbassare uno stipendio:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di aggiornamento
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?
   Sì, perchè l'informazione è contestuale alla riga modificata
3. Cosa bisogna fare prima o dopo dell'operazione per garantire il mantenimento dell'invariante?
   Impedire l'aggiornamento della riga (BEFORE) oppure riportare lo stipendio al valore originale (AFTER)

Esempio 1

Proibire qualsiasi abbassamento di stipendio:

```
CREATE TRIGGER NetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec
REFERENCING OLD ROW AS OldTuple, NEW ROW AS NewTuple
FOR EACH ROW
WHEN (OldTuple.netWorth > NewTuple.netWorth)
    UPDATE MovieExec
    SET netWorth = OldTuple.netWorth
    WHERE code = NewTuple.code;
```

Esempio 2

Supponiamo di volere utilizzare un trigger per garantire che la media degli stipendi non scenda mai sotto 500.000:

1. Quali operazioni possono violare l'invariante?

Esempio 2

Supponiamo di volere utilizzare un trigger per garantire che la media degli stipendi non scenda mai sotto 500.000:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di inserimento, aggiornamento o cancellazione

Esempio 2

Supponiamo di volere utilizzare un trigger per garantire che la media degli stipendi non scenda mai sotto 500.000:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di inserimento, aggiornamento o cancellazione
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?

Esempio 2

Supponiamo di volere utilizzare un trigger per garantire che la media degli stipendi non scenda mai sotto 500.000:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di inserimento, aggiornamento o cancellazione
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?
   Visto che la media è un'informazione globale della tabella, non possiamo ricorrere ad un controllo puntuale per riga

Esempio 2

Supponiamo di volere utilizzare un trigger per garantire che la media degli stipendi non scenda mai sotto 500.000:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di inserimento, aggiornamento o cancellazione
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?
   Visto che la media è un'informazione globale della tabella, non possiamo ricorrere ad un controllo puntuale per riga
3. Cosa bisogna fare prima o dopo dell'operazione per garantire il mantenimento dell'invariante?

Esempio 2

Supponiamo di volere utilizzare un trigger per garantire che la media degli stipendi non scenda mai sotto 500.000:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di inserimento, aggiornamento o cancellazione
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?
   Visto che la media è un'informazione globale della tabella, non possiamo ricorrere ad un controllo puntuale per riga
3. Cosa bisogna fare prima o dopo dell'operazione per garantire il mantenimento dell'invariante?
   Possiamo mantenere l'invariante annullando l'operazione che l'ha violata, cioè riportando la tabella allo stato originale (AFTER)

Esempio 2

Garantire che la media degli stipendi non scenda mai sotto 500.000:

```
CREATE TRIGGER AvgNetWorthTrigger
AFTER UPDATE ON MovieExec
REFERENCING OLD TABLE AS OldStuff, NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (500000 > (SELECT AVG(netWorth) FROM MovieExec))
BEGIN
    DELETE FROM MovieExec
    WHERE (name, address, code, netWorth)
    IN (SELECT * FROM NewStuff);
    INSERT INTO MovieExec (SELECT * FROM OldStuff);
END;
```

Servono trigger analoghi per INSERT e DELETE

Esempio 3

Supponiamo di volere utilizzare un trigger per garantire che la data di uscita di un film non possa mai essere NULL, usando il valore di default 1915 in tal caso:

1. Quali operazioni possono violare l'invariante?

Esempio 3

Supponiamo di volere utilizzare un trigger per garantire che la data di uscita di un film non possa mai essere NULL, usando il valore di default 1915 in tal caso:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di inserimento o aggiornamento

Esempio 3

Supponiamo di volere utilizzare un trigger per garantire che la data di uscita di un film non possa mai essere NULL, usando il valore di default 1915 in tal caso:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di inserimento o aggiornamento
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?

Esempio 3

Supponiamo di volere utilizzare un trigger per garantire che la data di uscita di un film non possa mai essere NULL, usando il valore di default 1915 in tal caso:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di inserimento o aggiornamento
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?
   Sì, perchè l'informazione è contestuale alla riga inserita o modificata

Esempio 3

Supponiamo di volere utilizzare un trigger per garantire che la data di uscita di un film non possa mai essere NULL, usando il valore di default 1915 in tal caso:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di inserimento o aggiornamento
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?
   Sì, perchè l'informazione è contestuale alla riga inserita o modificata
3. Cosa bisogna fare prima o dopo dell'operazione per garantire il mantenimento dell'invariante?

Esempio 3

Supponiamo di volere utilizzare un trigger per garantire che la data di uscita di un film non possa mai essere NULL, usando il valore di default 1915 in tal caso:

1. Quali operazioni possono violare l'invariante?
   L'invariante può essere violata da un'operazione di inserimento o aggiornamento
2. Il mantenimento dell'invariante può essere controllato per ogni riga coinvolta dall'operazione oppure no?
   Sì, perchè l'informazione è contestuale alla riga inserita o modificata
3. Cosa bisogna fare prima o dopo dell'operazione per garantire il mantenimento dell'invariante?
   Possiamo mantenere l'invariante correggendo il valore della data nella riga, prima o dopo l'operazione

Esempio 3

La data di uscita di un film non può mai essere NULL (default a 1915):

```
CREATE TRIGGER FixYearTrigger
BEFORE INSERT ON Movies
REFERENCING NEW ROW AS NewRow, NEW TABLE AS NewStuff
FOR EACH ROW
WHEN NewRow.year IS NULL
UPDATE NewStuff SET year = 1915;
```

Serve un trigger analogo per UPDATE

Uso dei Trigger

Trigger passivi: tali trigger provocano il fallimento di un'operazione sotto determinate condizioni. Usi tipici:

- Definizione di vincoli di integrità (es. no abbassamenti di stipendio)
- Controlli dinamici di autorizzazione (es. si possono inserire dati solo se il codice del dipartimento coincide con quello dell'utente che ha richiesto l'operazione)

Trigger attivi: tali trigger modificano, anche in modo complesso, lo stato della base di dati in corrispondenza di certi eventi. Usi tipici:

- Definizione di vincoli di integrità (es. pensate a CASCADE)
- Meccanismi di auditing e logging
- Definizione di business rules (regole aziendali)

Uso dei Trigger

I trigger sono molto potenti e ci permettono di programmare meccanismi che abbiamo già discusso nell'ambito di questo corso.

Example (Chiavi Esterne)

E' possibile utilizzare i trigger per implementare vincoli di tipo FOREIGN KEY, gestendo sia la politica CASCADE che la politica SET NULL.

Example (Dipendenze Funzionali)

E' possibile utilizzare i trigger per garantire che un'arbitraria dipendenza funzionale $X \rightarrow Y$ sia sempre rispettata.

Vantaggi dei Trigger

Poichè i trigger sono gestiti centralmente dal DBMS, non c'è alcun modo di scavalcarli!

- Se sviluppate applicazioni che si appoggiano ad un DBMS, è più robusto centralizzare un'invariante in un trigger che sparpagliare i controlli all'interno del codice applicativo
- Chiunque utilizzi la base di dati, anche all'esterno dell'applicazione che state sviluppando, è soggetto al controllo dei trigger
- Se volete fare auditing e logging, i trigger sono l'unico strumento veramente robusto per tali compiti, dato che il DBMS ha completa visibilità delle operazioni effettuate sulle tabelle

Svantaggi dei Trigger

I trigger sono purtroppo poco standardizzati e DBMS differenti fanno scelte diverse riguardo ad alcuni aspetti delicati:

- Che linguaggio utilizzare per scrivere i trigger? Qui abbiamo visto SQL, ma perchè limitarsi ad esso?
- Se vi sono più trigger registrati sullo stesso evento, in che ordine devono essere eseguiti?
- E' possibile per un trigger T_1 attivare un altro trigger T_2?
- In caso affermativo, è possibile che T_2 attivi nuovamente T_1? Come gestire la presenza di trigger ricorsivi?

I trigger sono difficili da debuggare e poco visibili in generale.

Trigger o Vincoli?

Se avete possibilità di scelta, è sempre preferibile utilizzare i diversi tipi di vincoli messi a disposizione dal DBMS invece dei trigger:

- i vincoli sono standard e gestiti in modo uniforme da tutti i DBMS
- i vincoli hanno una semantica semplice e non presentano problemi in termini di debugging
- i vincoli garantiscono che una certa proprietà valga già al momento della loro definizione, a differenza dei trigger che sono reattivi

Talvolta i trigger sono necessari per la loro espressività, per esempio:

- per invarianti che coinvolgono più di una tabella
- per invarianti che coinvolgono più righe di una stessa tabella

Trigger in Postgres

Postgres offre un sistema di trigger potente e fedele allo standard SQL:

```
CREATE TRIGGER name { BEFORE | AFTER } { evt [ OR ... ] }
  ON table_name
  [ REFERENCING { { OLD | NEW } TABLE AS tab } [ ... ] ]
  [ FOR EACH { ROW | STATEMENT } ]
  [ WHEN ( condition ) ]
  EXECUTE FUNCTION func ( args )
```

Differenze rispetto allo standard:

- è possibile usare `OR` per associare uno stesso trigger a più eventi
- non è possibile riferire `OLD ROW` e `NEW ROW` in `REFERENCING`, ma c'è un modo custom per accedere a tali righe
- il corpo del trigger deve essere definito in una funzione separata

Funzioni e Trigger

Postgres supporta la definizione di funzioni scritte in vari linguaggi. Il suo linguaggio nativo è chiamato PL/pgSQL.

PL/pgSQL può essere usato per definire trigger functions, cioè funzioni:

- con `trigger` come tipo di ritorno
- senza argomenti: il passaggio di parametri avviene in modo custom in fase di creazione del trigger, perchè non esiste un chiamante

```
CREATE FUNCTION my_trigger() RETURNS trigger AS $$
   trigger function definition
$$ LANGUAGE plpgsql;
```

Il comando `CREATE TRIGGER` accetta solamente trigger functions!

Trigger Functions

Tolti i due vincoli già menzionati, una trigger function può essere una funzione PL/pgSQL arbitraria.

In questa lezione ci focalizziamo su un formato semplificato:

```
BEGIN
    statement_1;
    ...
    statement_n;
END;
```

dove ogni statement è un'istruzione SQL, un condizionale (`IF`) oppure un `RETURN`. Prossimamente discuteremo PL/pgSQL in dettaglio.

Passaggio di Parametri

Quando una trigger function viene invocata da Postgres, vengono create nel suo scope alcune variabili speciali. Le più importanti:

- `NEW`: la nuova riga per operazioni di INSERT/UPDATE all'interno di un trigger per riga (NULL nel caso di DELETE)
- `OLD`: la vecchia riga per operazioni di DELETE/UPDATE all'interno di un trigger per riga (NULL nel caso di INSERT)
- `TG_NARGS`: numero di argomenti passati tramite la `CREATE TRIGGER`
- `TG_ARGV`: vettore di argomenti passati tramite la `CREATE TRIGGER`

Ci sono inoltre una serie di variabili informative come `TG_OP`, che dicono quale evento ha scatenato il trigger.

Valore di Ritorno

Una trigger function associata ad un BEFORE trigger per riga può:

- ritornare NULL per indicare che l'operazione (INSERT, UPDATE o DELETE) sulla riga deve essere abortita
- nel caso di INSERT o UPDATE: ritornare una riga, che diverrà la nuova riga che sarà inserita o sostituirà la riga aggiornata
- se non si vuole interferire con l'operazione: ritornare `NEW` nel caso di INSERT o UPDATE, ritornare `OLD` nel caso di DELETE

Una trigger function deve ritornare NULL in tutti gli altri casi, cioè nel caso di trigger per statement ed AFTER trigger per riga.

Trigger per Riga

```
CREATE TRIGGER name { BEFORE | AFTER } { evt [ OR ... ] }
  ON table_name
  [ REFERENCING { { OLD | NEW } TABLE AS tab } [ ... ] ]
  FOR EACH ROW
  [ WHEN ( condition ) ]
  EXECUTE FUNCTION func ( args )
```

Punti chiave:

- un BEFORE trigger per riga può prevenire operazioni o modificarle
- la clausola `WHEN` può fare riferimento a `OLD` e `NEW` per specificare una condizione di attivazione e non può fare uso di sotto-query
- è possibile usare `REFERENCING` per vedere i cambiamenti complessivi nell'intera tabella, non solo nella riga (solo per AFTER trigger)

Trigger per Statement

```
CREATE TRIGGER name { BEFORE | AFTER } { evt [ OR ... ] }
  ON table_name
  [ REFERENCING { { OLD | NEW } TABLE AS tab } [ ... ] ]
  FOR EACH STATEMENT
  EXECUTE FUNCTION func ( args )
```

Punti chiave:

- un trigger per statement viene eseguito una volta anche se nessuna riga è coinvolta nell'operazione scatenante
- sebbene la clausola `WHEN` possa essere usata anche in un trigger per statement, essa è inutile perché `OLD` e `NEW` non sono popolati
- è possibile usare `REFERENCING` per vedere i cambiamenti complessivi nell'intera tabella (solo per AFTER trigger)

Modello di Esecuzione

L'ordine di esecuzione dei trigger dipende dal loro tipo:

1. i BEFORE trigger per statement si attivano prima di tutti, per la precisione prima che l'evento abbia inizio
2. i BEFORE trigger per riga si attivano immediatamente prima di operare sulla riga coinvolta (anche prima dei CHECK)
3. gli AFTER trigger per riga si attivano alla fine dell'evento, ma prima degli AFTER trigger per statement
4. gli AFTER trigger per statement vengono eseguiti per ultimi

Ci sono però varie sottigliezze descritte nel manuale, vediamo quali sono le più importanti nell'uso comune...

Sottigliezze

Alcune specificità di Postgres da tenere bene a mente:

- un trigger per riga ha visibilità dei cambiamenti effettuati sulle righe precedenti, anche nel caso di BEFORE trigger, ma l'ordine di visita delle righe non è predicibile!
- se più di un trigger viene definito per lo stesso evento sulla stessa tabella, essi sono eseguiti in ordine alfabetico fino a terminazione o finché uno di essi non ritorna NULL (nel caso di BEFORE trigger per riga, in cui altri trigger per la stessa riga vengono ignorati)
- un trigger può attivare ricorsivamente altri trigger, potenzialmente introducendo ricorsioni infinite. Evitare ricorsioni infinite è compito del programmatore.

Esempio: Trigger per Dipendenze Funzionali

Vediamo un modo per garantire la dipendenza funzionale $A \rightarrow B$

```
CREATE TRIGGER FuncDep
  BEFORE INSERT OR UPDATE ON Relation
  FOR EACH ROW
  EXECUTE FUNCTION fix_func_dep ()

CREATE FUNCTION fix_func_dep() RETURNS TRIGGER AS $$
  IF (EXISTS SELECT * FROM Relation
             WHERE A = NEW.A AND B != NEW.B)
  THEN RETURN NULL;
  END IF;
  RETURN NEW;
$$ LANGUAGE plpgsql;
```

Checkpoint

Concetti Chiave

- I trigger e la loro semantica evento - condizione - azione
- Progettazione di trigger per il mantenimento di invarianti globali
- Vantaggi e svantaggi dell'uso dei trigger
- L'utilizzo dei trigger in Postgres

Materiale Didattico

- Database Systems: Sezione 7.5
- Fondamenti di Basi di Dati: Sezione 7.3.2
- Documentazione di Postgres (link)