Politecnico di Torino

Basi di dati attive: Trigger

Trigger

- Basi di dati attive
- Trigger in Oracle
- Confronto tra Oracle e DB2
- Linee guida per la scrittura di trigger in Oracle
- Progettazione dei trigger

Basi di dati attive

- Il funzionamento tradizionale del DBMS è *passivo*
  - le interrogazioni e gli aggiornamenti sono richiesti esplicitamente dagli utenti
  - la conoscenza dei processi che operano sui dati è tipicamente incorporata nelle applicazioni
- Basi di dati *attive*
  - la reattività è un servizio fornito da un normale DBMS
    - *vengono monitorati* eventi specifici del database e *attivate* azioni in risposta

Basi di dati attive

- La reattività è assicurata dall'esecuzione automatica di *regole attive o ECA poichè espresso nella forma*
  - Evento
    - operazione di modifica della base di dati
  - Condizione
    - predicato sullo stato della base di dati
    - se la condizione è vera, l'azione viene eseguita
  - Azione
    - sequenza di istruzioni SQL o procedura applicativa

Rule engine

- Componente del DBMS, incaricato di
  - tracciare gli eventi
  - eseguire le regole quando è appropriato, in base alla strategia di esecuzione del DBMS
- L'esecuzione delle regole è alternata all'esecuzione delle transazioni tradizionali

Esempio: riordino di prodotti

- La regola attiva gestisce il riordino delle scorte di un magazzino
  - quando la quantità disponibile di un prodotto scende al di sotto di una determinata soglia, deve essere emesso un nuovo ordine per il prodotto

- Evento
  - aggiornamento della quantità disponibile per il prodotto x
  - inserimento di un nuovo prodotto x
- Condizione
  - la quantità disponibile del prodotto x è inferiore a una determinata soglia

    e non ci sono ordini in sospeso per il prodotto x
- Azione
  - emettere un ordine con una determinata quantità di riordino per il prodotto x

Applicazioni delle regole attive

- Applicazioni interne
  - mantenimento di vincoli di integrità complessi
  - gestione delle repliche
  - manutenzione delle viste materializzate
- Business Rules
  - incorporano nel DBMS la conoscenza delle applicazioni
    - ad esempio, regola di riordino dei prodotti
- Alert
  - utilizzati per le notifiche

Trigger

- Nei DBMS tipicamente le regole attive sono implementate mediante *trigger*
- Il linguaggio SQL fornisce istruzioni per la definizione dei trigger
  - i trigger vengono definiti con l'istruzione DDL CREATE TRIGGER
- La sintassi e la semantica dei trigger sono definite nello standard SQL3
  - alcuni DBMS implementano caratteristiche diverse rispetto allo standard

Struttura del trigger

- Evento
  - inserimento, cancellazione, aggiornamento di una tabella
  - ogni trigger può monitorare solo gli eventi di una *singola* tabella
- Condizione
  - predicato SQL (è opzionale)
- Azione
  - sequenza di istruzioni SQL
  - blocchi di linguaggio di programmazione (proprietario)
    - ad esempio, Oracle PL/SQL
  - codice Java

Processo di esecuzione

Quando gli eventi si verificano [attivazione]

Se la condizione è vera [valutazione]

Allora l'azione viene eseguita [esecuzione]

- Sembra molto semplice, ma...
  - Modalità di esecuzione
  - Granularità dell'esecuzione

Modalità di esecuzione

- Immediato
  - Il trigger viene eseguito *immediatamente prima* o *dopo* l'istruzione di attivazione
- Differito
  - Il trigger viene eseguito immediatamente *prima* del *commit*
- Nei sistemi commerciali è disponibile solo l'opzione immediata

Granularità dell'esecuzione

- Tupla (o livello di riga)
  - Un'esecuzione separata del trigger *per ogni tupla* interessata dall'istruzione di attivazione
- Istruzione
  - Una singola esecuzione del trigger *per tutte le tuple* interessate dall'istruzione attivazione

Esempio di granularità

- Tabella T

A	B
1	5
2	9
8	20

- Evento sulla tabella T

```
UPDATE T
SET A=A+1
WHERE B<10;
```

- Esecuzione del trigger definito sulla tabella T
  - Un trigger a livello di tupla viene eseguito due volte
  - Un trigger a livello di istruzione viene eseguito una sola volta

Trigger in Oracle

Trigger

Sintassi

CREATE TRIGGER TriggerName

Mode Event {*OR Event* }

ON TargetTable

[[*REFERENCING ReferenceName*]

FOR EACH ROW

[*WHEN Predicate*]]

PL/SQL Block

Sintassi

CREATE TRIGGER TriggerName

Mode Event {*OR Event* }

ON TargetTable

[[*REFERENCING ReferenceName*]

FOR EACH ROW

[*WHEN Predicate*]]

PL/SQL Block

- *Mode* è *BEFORE* o *AFTER*
  - anche *INSTEAD OF* ma **è da evitare**

Sintassi

CREATE TRIGGER TriggerName

Mode Event {*OR Event* }

ON TargetTable

[[*REFERENCING ReferenceName*]

FOR EACH ROW

[*WHEN Predicate*]]

PL/SQL Block

- *Event ON TargetTable* è
  - *INSERT*
  - *DELETE*
  - *UPDATE* [*OF ColumnName*]

Sintassi

CREATE TRIGGER TriggerName

Mode Event {*OR Event* }

ON TargetTable

[[*REFERENCING ReferenceName*]

FOR EACH ROW

[*WHEN Predicate*]]

PL/SQL Block

- *FOR EACH ROW* specifica la semantica di esecuzione a livello di tupla
  - se omesso, la semantica di esecuzione è a livello di istruzione

Sintassi

CREATE TRIGGER TriggerName

Mode Event {*OR Event* }

ON TargetTable

[[*REFERENCING ReferenceName*]

FOR EACH ROW

[*WHEN Predicate*]]

PL/SQL Block

- Per rinominare le variabili di stato
  - *REFERENCING OLD AS OldVariableName*
    - analogamente per *NEW*

Sintassi

CREATE TRIGGER TriggerName

Mode Event {*OR Event* }

ON TargetTable

[[*REFERENCING ReferenceName*]

FOR EACH ROW

[*WHEN Predicate*]]

PL/SQL Block

- *Solo* per la semantica di esecuzione a livello di tupla (ossia, *FOR EACH ROW*)
  - è possibile specificare facoltativamente una condizione
  - è possibile accedere alle variabili di stato vecchie e nuove

Sintassi

CREATE TRIGGER TriggerName

Mode Event {*OR Event* }

ON TargetTable

[[*REFERENCING ReferenceName*]

FOR EACH ROW

[*WHEN Predicate*]]

PL/SQL Block

- L'azione è
  - una sequenza di istruzioni SQL
  - un blocco PL/SQL
- *Nessuna* istruzione transazionale e DDL

Semantica

- Modalità di esecuzione
  - immediate before
  - immediate after
- La granularità è
  - riga (tupla)
  - istruzione
- L'esecuzione è innescata da istruzioni di inserimento, cancellazione o aggiornamento in una transazione

Algoritmo di esecuzione

1. Vengono eseguiti i trigger di tipo before a livello di istruzione
2. Per ogni tupla nella tabella target (*TargetTable*) interessata dall'istruzione di trigger
   a) Vengono eseguiti i trigger di tipo before a livello di tupla
   b) Viene eseguita l'istruzione di attivazione
      + i vincoli di integrità definiti a livello di tupla
   c) Vengono eseguiti i trigger di tipo after a livello di tupla
3. Vengono controllati i vincoli di integrità sulle tabelle
4. Vengono eseguiti i trigger di tipo after a livello di istruzione

Semantica

- L'ordine di esecuzione di trigger con lo stesso evento, la stessa modalità e la stessa granularità non viene specificato
  - è una fonte di non determinismo
- Quando si verifica un errore
  - rollback di tutte le operazioni eseguite dai trigger
  - rollback dell'istruzione di attivazione nella transazione di attivazione

Non terminazione

- L'esecuzione di un trigger può attivare altri trigger
  - l'attivazione di trigger in cascata può portare alla non terminazione dell'esecuzione del trigger
- È possibile impostare una lunghezza massima per l'esecuzione di trigger in cascata
  - default = 32 trigger
- Se il limite massimo viene superato
  - viene restituito un errore di esecuzione

Tabelle mutanti

- Una *tabella mutante* è la *tabella target* modificata dall'istruzione (cioè dall'evento) che attiva il trigger
- La tabella mutante
  - *non* è accessibile nei trigger a livello di riga
  - può essere accessibile *solo* nei trigger di istruzione
- L'accesso limitato alle tabelle mutanti caratterizza solo le applicazioni Oracle
  - l'accesso alle tabelle mutanti è *sempre* permesso in SQL3

Esempio: riordino di prodotti

- Trigger per gestire il riordino in un inventario di magazzino
  - quando la quantità di un prodotto in magazzino scende al di sotto di una determinata soglia
  - deve essere emesso un nuovo ordine per il prodotto
- Lo schema della base di dati è il seguente

  Inventory (Part#, QtyOnHand, ThresholdQty, ReorderQty)

  PendingOrders(Part#, OrderDate, OrderedQty)

Esempio: riordino di prodotti

- Trigger per gestire il riordino in un inventario di magazzino
  - quando la quantità di un prodotto in magazzino scende al di sotto di una determinata soglia
  - deve essere emesso un nuovo ordine per il prodotto
- Evento
  - aggiornamento della quantità disponibile per il prodotto x
  - inserimento di un nuovo prodotto x
- Semantica dell'esecuzione
  - dopo l'evento di modifica
  - esecuzione separata per ogni riga della tabella Inventory

Esempio: riordino di prodotti

CREATE TRIGGER Reorder

AFTER UPDATE OF QtyOnHand *OR INSERT ON* Inventory

FOR EACH ROW

Esempio: riordino di prodotti

- Trigger per gestire il riordino in un inventario di magazzino
  - quando la quantità di un prodotto in magazzino scende al di sotto di una determinata soglia
  - deve essere emesso un nuovo ordine per il prodotto
- Condizione
  - la quantità disponibile è inferiore a una determinata soglia

Esempio: riordino di prodotti

```
CREATE TRIGGER Reorder
AFTER UPDATE OF QtyOnHand OR INSERT ON Inventory
FOR EACH ROW
WHEN (NEW.QtyOnHand < NEW.ThresholdQty)
```

Esempio: riordino di prodotti

- Trigger per gestire il riordino in un inventario di magazzino
  - quando la quantità di un prodotto in magazzino scende al di sotto di una determinata soglia
  - deve essere emesso un nuovo ordine per il prodotto
- Condizione
  - la quantità disponibile è inferiore a una determinata soglia

    e non ci sono ordini in sospeso per il prodotto x
    - Questa parte non può essere introdotta nella clausola WHEN
- Azione
  - Emettere un ordine con una determinata quantità di riordino per il prodotto x

Esempio: riordino di prodotti

```
DECLARE
 N number;
BEGIN
 select count(*) into N
 from PendingOrders
 where Part# = :NEW.Part#;
 If (N=0) then
    insert into PendingOrders(Part#,OrderedQty,OrderDate)
    values (:NEW.Part#, :NEW.ReorderQty, SYSDATE);
 end if;
END;
```

Esempio *completo*: riordino di prodotti

```
CREATE TRIGGER Reorder
AFTER UPDATE OF QtyOnHand OR INSERT ON Inventory
FOR EACH ROW
WHEN (NEW.QtyOnHand < NEW. ThresholdQty)
DECLARE
 N number;
BEGIN
 select count(*) into N
 from PendingOrders
 where Part# = :NEW.Part#;
 If (N==0) then
   insert into PendingOrders(Part#,OrderedQty,OrderDate)
   values (:NEW.Part#, :NEW.ReorderQty, SYSDATE);
 end if;
END;
```

Linee guida per la scrittura di trigger in Oracle

Triggers

Linee guida per la scrittura di trigger in Oracle

- La modalità di esecuzione INSTEAD OF è consentita in Oracle, ma dovrebbe essere evitata
- Uso dei trigger before in Oracle per essere conformi allo standard
  - le modifiche della variabile NEW nelle tuple interessate dall'istruzione di attivazione sono consentite nei trigger di tipo before
  - altre modifiche della base di dati, oltre a quelle riportate al punto precedente, non sono consentite nei trigger before
  - i trigger di tipo before non possono innescare altri trigger

Confronto tra Oracle e DB2

Trigger

Differenze tra Oracle e DB2

	Oracle	DB2
Riferimento a Old_Table e New_Table nei trigger delle istruzioni	No	Sì
Clausola When nei trigger a livello di istruzioni	No	Sì
Ordine di esecuzione tra trigger di tupla e istruzione con la stessa modalità	Specificato	Arbitrario
Ordine di esecuzione tra trigger con stesso evento, modalità e granularità	Non specificato	Ordine di creazione
È consentito più di un evento di attivazione	Sì	No
Accesso vietato alla tabella mutante	Sì per i trigger di tupla	No
Disponibilità della semantica instead of	Sì	No
Modifiche al database consentite in trigger di tipo before	Sì	Solo le variabili NEW

Progettazione dei trigger

Triggers

Progettazione dei trigger

- La progettazione di un singolo trigger è solitamente semplice
- Bisogna identificare
  - semantica di esecuzione
  - evento
  - condizione (opzionale)
  - azione

Progettazione dei trigger

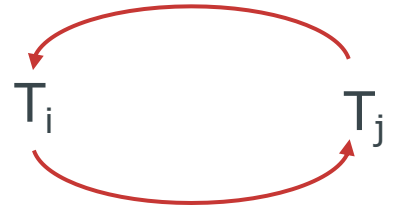
- La comprensione delle interazioni *reciproche* tra i trigger è più complessa
  - l'azione di un trigger può essere l'evento di un trigger differente
    - esecuzione a cascata
- Se si verificano inneschi reciproci
  - è possibile un'esecuzione infinita

Proprietà di esecuzione del trigger

- Terminazione
  - per uno stato arbitrario del database e una transazione utente, l'esecuzione del trigger termina in uno stato finale (anche dopo un abort)
- Confluenza
  - Per uno stato arbitrario del database e una transazione utente, l'esecuzione di un trigger *termina in un unico stato finale*, indipendentemente dall'ordine di esecuzione dei trigger
- La terminazione è la proprietà più importante
- La confluenza è assicurata da un'esecuzione deterministica dei trigger

Garantire la terminazione

- La terminazione è garantita in fase di esecuzione, interrompendo l'esecuzione del trigger dopo un determinato numero di attivazione di trigger in cascata
- La terminazione può essere verificata in fase di progettazione mediante il grafo di attivazione dei trigger
  - un nodo per ogni trigger
  - un arco diretto T_i T_j se il trigger T_i sta eseguendo un'azione che innesca il trigger T_j
- Un ciclo nel grafo mostra esecuzioni che potrebbero causare una condizione di non terminazione



Esempio: gestione dei salari

- Trigger per la gestione degli importi salariali
  - quando viene superato un determinato valore medio di stipendio, viene applicata automaticamente una riduzione dello stipendio
- Viene fornita la seguente tabella

  Employee (Emp#, Ename, …, Salary)
- Semantica di esecuzione
  - dopo gli eventi di modifica
  - esecuzione separata per ogni istruzione di aggiornamento
- Nessuna condizione per l'esecuzione

Esempio: gestione dei salari

```
CREATE TRIGGER SalaryMonitor
AFTER UPDATE OF Salary ON Employee
FOR EACH STATEMENT
BEGIN
  update Employee
  set Salary = Salary * K
  where 2500 < (select AVG (Salary) from Employee);
END;
```

Il valore di K può essere

K = 0.9	l'esecuzione termina
K = 1.1	esecuzione infinita

Trigger per la gestione dei vincoli

- I trigger vengono sfruttati per imporre vincoli di integrità complessi
- Procedura di progettazione
  1. Scrivere il vincolo come predicato SQL
     - fornisce una condizione per l'esecuzione del trigger
  2. Identificare gli eventi che possono violare il vincolo
     - ossia, la condizione
  3. Definire la tecnica di gestione dei vincoli nell'azione

Esempio di progettazione (1)

- Sono riportate le seguenti tabelle
  - Supplier S (S#, SName, …)
  - Part P (P#, PName, …)
  - Supply SP (S#, P#, Qty)
- Vincolo da rispettare
  - un pezzo può essere fornito al massimo da 10 fornitori diversi

Esempio di progettazione (1)

- Predicato di vincolo

```
select P#
from SP
group by P#
having count(*) > 10
```

  - insieme di parti che violano il vincolo
- Eventi
  - insert on SP
  - update of P# on SP
- Azione
  - rifiutare la transazione in violazione

Esempio di progettazione (1)

- Semantica di esecuzione
  - *dopo* la modifica
  - *livello di istruzione*
    - per catturare l'effetto dell'intera modifica
    - (Oracle) per consentire l'accesso alla tabella mutante
- (Oracle) Nessuna condizione
  - la condizione non può essere specificata nella clausola WHEN
  - viene verificata nel corpo del trigger
- Progettazione della semantica del trigger Oracle

Esempio di progettazione (1)

```
CREATE TRIGGER TooManySuppliers
AFTER UPDATE OF P# OR INSERT ON SP
DECLARE
 N number;
BEGIN
 select count(*) into N
 from SP
 where P# IN (select P# from SP
                group by P#
                having count(*) > 10);
 if (N <> 0) then
   raise_application_error (xxx, 'constraint violated');
 end if;
END;
```

Esempio di progettazione (2)

- Sono riportate le seguenti tabelle
  - Supplier S (S#, SName, …)
  - Part P (P#, PName, …)
  - Supply SP (S#, P#, Qty)
- Vincolo da rispettare
  - la quantità di un prodotto fornito non può essere superiore a 1000. Se è maggiore, ridurla a 1000.
- I vincoli di controllo non consentono azioni di compensazione
  - implementare con un trigger

Esempio di progettazione (2)

- Predicato di vincolo
  - Qty > 1000
  - è anche la condizione di attivazione
- Eventi
  - insert on SP
  - update of Qty on SP
- Azione
  - Qty = 1000

Esempio di progettazione (2)

- Semantica dell'esecuzione
  - *prima* che avvenga la modifica
    - il suo effetto può essere modificato prima che il vincolo venga controllato
  - *livello di tupla*
    - ogni tupla viene modificata separatamente

Esempio di progettazione (2)

```
CREATE TRIGGER ExcessiveQty
BEFORE UPDATE OF Qty OR INSERT ON SP
FOR EACH ROW
WHEN (NEW.Qty > 1000)
BEGIN
 :NEW.Qty := 1000;
END;
```

Mantenimento delle viste materializzate

- Le viste materializzate sono query memorizzate in modo persistente nella base di dati
  - forniscono maggiori prestazioni
  - contengono informazioni ridondanti
    - ad esempio, calcoli aggregati
- I trigger possono essere utilizzati per aggiornare le viste materializzate
  - propagare le modifiche dei dati nelle tabelle alla vista materializzata

Esempio di progettazione (3)

- Tabelle
  - Studente S (SId, SName, DCId)
  - Corso di laurea DC (DCId, DCName)
- Vista materializzata
  - Studenti iscritti ES (DCId, TotalStudents)
    - per ogni corso di laurea, TotalStudents conta il numero totale degli studenti iscritti
    - definito dalla query

```
SELECT DCId, COUNT(*)
FROM S
GROUP BY DCId;
```

Esempio di progettazione (3)

- Tabelle
  - Studente S (SId, SName, DCId)
  - Corso di laurea DC (DCId, DCName)
- Vista materializzata
  - Studenti iscritti ES (DCId, TotalStudents)
    - per ogni corso di laurea, TotalStudents conta il numero totale degli studenti iscritti
  - un nuovo corso di laurea viene inserito nella vista materializzata ES quando il primo studente vi si iscrive
  - un corso di laurea viene cancellato dalla vista materializzata ES quando l'ultimo studente lo abbandona

Esempio di progettazione (3)

- Schema del database

  S (SId, SName, DCId)

  DC (DCId, DCName)

  ES (DCId, TotalStudents)

- Propagare le modifiche sulla tabella S alla vista materializzata (tabella) ES
  - inserimento di nuove tuple in S
  - cancellazione di tuple da S
  - aggiornamento dell'attributo DCId in una o più tuple di S

Esempio di progettazione (3)

- Progettare tre trigger per gestire separatamente ogni modifica dei dati
  - trigger di inserimento, trigger di cancellazione, trigger di aggiornamento
  - tutti i trigger condividono la stessa semantica di esecuzione
- Semantica di esecuzione
  - *dopo* che la modifica ha avuto luogo
    - la tabella ES viene aggiornata dopo che la tabella S è stata modificata
  - *livello di tupla*
    - esecuzione separata per ogni tupla della tabella S
      - significativamente più semplice da implementare

Trigger di inserimento (3)

- Evento
  - insert on S
- Nessuna condizione
  - è sempre eseguito
- Azione
  - se la tabella ES contiene il DCId a cui è iscritto lo studente
    - incrementare TotaleStudents
  - altrimenti
    - aggiungere una nuova tupla nella tabella ES per il corso di laurea, con TotalStudents impostato a 1

Trigger di inserimento (3)

```
CREATE TRIGGER InsertNewStudent
AFTER INSERT ON S
FOR EACH ROW
DECLARE
 N number;
BEGIN
 --- verificare se la tabella ES contiene la tupla per il corso di laurea
 --- corso NEW.DCId a cui lo studente si iscrive
 select count(*) into N
 from ES
 where DCId = :NEW. DCId;
```

Trigger di inserimento (3)

```
if (N <> 0) then
    --- la tupla per il corso di laurea NEW.DCId è disponibile in ES
    update ES
    set TotalStudents = TotalStudents +1
    where DCId = :NEW.DCId;
else
    --- nessuna tupla per il corso di laurea NEW.DCId è disponibile in ES
    insert into ES (DCId, TotalStudents)
    values (:NEW.DCId, 1);
end if;
END;
```

Trigger di cancellazione (3)

- Evento
  - delete from S
- Nessuna condizione
  - è sempre eseguito
- Azione
  - se lo studente era l'unico iscritto al corso di laurea
    - cancellare la tupla corrispondente da ES
  - altrimenti
    - decrementare TotaleStudents

Trigger di cancellazione (3)

```
CREATE TRIGGER DeleteStudent
AFTER DELETE ON S
FOR EACH ROW
DECLARE
 N number;
BEGIN
 --- leggere il numero di studenti iscritti al corso di laurea OLD.DCId
select TotalStudents into N
 from ES
 where DCId = :OLD.DCId;
```

Trigger di cancellazione (3)

```
if (N > 1) then
    --- ci sono molti studenti iscritti
    update ES
    set TotalStudents = TotalStudents – 1
    where DCId = :OLD.DCId;
else
    --- c'è un solo studente iscritto
    delete from ES
    where DCId = :OLD.DCId;
end if;
END;
```

Trigger di aggiornamento (3)

- Evento
  - Update of DCId on S
- Nessuna condizione
  - è sempre eseguito
- Azione
  - aggiornare la tabella ES per il corso di laurea a cui lo studente *era* iscritto
    - decrementare TotalStudents o cancellare la tupla se l'ultimo studente è stato immatricolato
  - aggiornare la tabella ES per il corso di laurea a cui lo studente *è attualmente* iscritto
    - incrementare TotaleStudenti, o inserire una nuova tupla se si tratta del primo studente

Trigger di aggiornamento (3)

```
CREATE TRIGGER UpdateDegreeCourse
AFTER UPDATE OF DCId ON S
FOR EACH ROW
DECLARE
 N number;
BEGIN
--- leggere il numero di studenti iscritti al corso di laurea OLD.DCId
 select TotalStudents into N
 from ES
 where DCId = :OLD.DCId;
```

Trigger di aggiornamento (3)

```
if (N > 1) then
    --- ci sono molti studenti iscritti
    update ES
    set TotalStudents = TotalStudents – 1
    where DCId = :OLD.DCId;
else
    --- c'è un solo studente iscritto
    delete from ES
    where DCId = :OLD.DCId;
end if;
```

Trigger di aggiornamento (3)

```
--- controllare se la tabella ES contiene la tupla per il corso di laurea
--- NEW.DCId a cui lo studente è iscritto
 select count(*) into N
 from ES
 where DCId = :NEW. DCId;
```

Trigger di aggiornamento (3)

```
if (N <> 0) then
 --- la tupla per il corso di laurea NEW.DCId è disponibile in ES
       update ES
       set TotalStudents = TotalStudents +1
       where DCId = :NEW.DCId;
else
 --- non ci sono tuple per il corso di laurea NEW.DCId disponibili in ES
        insert into ES (DCId, TotalStudents)
        values (:NEW.DCId, 1);
end if;
END;
```