Basi di dati attive

Basi di dati attive

- Una base di dati che offre regole attive
- Si parla normalmente di *trigger*
- Si vogliono descrivere:
  - definizione in SQL:1999
  - varianti ed evoluzioni
  - terminazione e progetto
  - diversi esempi d'uso

Struttura di base dei trigger

- Paradigma di base: Evento-Condizione-Azione
  - quando capita un evento
  - se è vera la condizione
  - si esegue l'azione
- Il modello a regole è un modo intuitivo per rappresentare una computazione
- Altri esempi di regole nel mondo dei DBMS:
  - vincoli di integrità
  - regole Datalog
  - business rules
- Problema: difficile realizzare sistemi complessi

Evento-Condizione-Azione

- Evento
  - normalmente una modifica dello stato della base di dati: INSERT, DELETE, UPDATE

Quando avviene l'evento, il trigger viene *attivato*

- Condizione
  - Un predicato che identifica le situazioni in cui è necessaria l'applicazione del trigger
  - Quando si valuta la condizione il trigger viene *considerato*
- Azione
  - Un generico comando di modifica o una stored procedure
  - Quando si elabora l'azione il trigger viene *eseguito*

- Un DBMS mette già a disposizione tutti i componenti necessari. Si tratta solo di integrarli

Sintassi SQL:1999 dei trigger

- SQL:1999 (anche detto SQL-3) propone una sintassi simile a quella offerta da Oracle Server e IBM DB2
- I sistemi tenderanno a uniformarsi a essa
- Ogni trigger è caratterizzato da
  - nome
  - nome della tabella che viene monitorata
  - modo di esecuzione (BEFORE o AFTER)
  - l'evento monitorato (INSERT, DELETE o UPDATE)
  - granularità (statement-level o row-level)
  - nomi e alias per transition values e transition tables
  - la condizione
  - l'azione
  - il timestamp di creazione

Sintassi SQL:1999 dei trigger

create trigger *NomeTrigger*
{**before** | **after**}
{ **insert** | **delete** | **update** [**of** *Colonne*] } **on**
Tabella
[**referencing**
{[**old table** [**as**] *AliasTabellaOld*]
[**new table** [**as**] *AliasTabellaNew*] } |
{[**old** [**row**] [**as**] *NomeTuplaOld*]
[**new** [**row**] [**as**] *NomeTuplaNew*] }]
[**for each** { **row** | **statement** }]
[**when** *Condizione*]
ComandiSQL

Esecuzione di un singolo trigger

- Modo di esecuzione:
  - BEFORE
    - Il trigger viene considerato ed eventualmente eseguito prima che venga applicata sulla base di dati l'azione che lo ha attivato
    - Di norma viene utilizzata questa modalità quando si vuole verificare la correttezza di una modifica, prima che venga applicata
  - AFTER
    - Il trigger viene considerato ed eventualmente eseguito dopo che è stata applicata sulla base di dati l'azione che lo ha attivato
    - È il modo più comune, adatto a quasi tutte le applicazioni
    - È più semplice da utilizzare correttamente

Granularità degli eventi

- Modo statement level (modo di default)
  - Il trigger viene considerato ed eventualmente eseguito una volta sola per ogni comando che lo ha attivato, indipendentemente dal numero di tuple modificate
  - È il modo più vicino all'approccio tradizionale dei comandi SQL, che sono di norma set-oriented
- Modo row-level (opzione **`for each row`**)
  - Il trigger viene considerato ed eventualmente eseguito una volta per ciascuna tupla che è stata modificata dal comando
  - Consente di scrivere i trigger in modo più semplice
  - Può essere meno efficiente

Clausola **`referencing`**

- Il formato dipende dalla granularità
  - Per il modo row-level, si hanno due *transition variables* **`old`** e **`new`**, che rappresentano rispettivamente il valore precedente e successivo alla modifica della tupla che si sta valutando
  - Per il modo statement-level, si hanno due *transition tables* **`old table`** e **`new table`**, che contengono rispettivamente il valore vecchio e nuovo di tutte le tuple modificate
- Le variabili **`old`** e **`old table`** non sono utilizzabili in trigger il cui evento è **`insert`**
- Le variabili **`new`** e **`new table`** non sono utilizzabili in trigger il cui evento è **`delete`**
- Le variabili e le tabelle di transizione sono estremamente importanti per realizzare i trigger in modo efficiente

Esempio di trigger row-level

```
create trigger MonitoraConti
after update on Conto
referencing old as old new as new
for each row
when (old.NomeConto = new.NomeConto
    and new.Totale > old.Totale)
insert into SingoliVersamenti
  values (new.NomeConto,new.Totale-
old.Totale)
```

Esempio di trigger statement-level

```
create trigger ArchiviaFattureCanc
after delete on Fattura
referencing old table as
SetOldFatture
insert into FattureCancellate
(select *
 from SetOldFatture)
```

Esecuzione di più trigger e loro proprietà

Conflitti tra trigger

- Se vi sono più trigger associati allo stesso evento, SQL:1999 prescrive questa politica di gestione
  - Vengono eseguiti i trigger BEFORE statement-level
  - Vengono eseguiti i trigger BEFORE row-level
  - Si applica la modifica e si verificano i vincoli di integrità definiti sulla base di dati
  - Vengono eseguiti i trigger AFTER row-level
  - Vengono eseguiti i trigger AFTER statement-level
- Se vi sono più trigger della stessa categoria, l'ordine di esecuzione viene scelto dal sistema in un modo che dipende dall'implementazione

Modello di esecuzione

- SQL:1999 prevede che i trigger vengano gestiti in un Trigger Execution Context (TEC)
- L'esecuzione dell'azione di un trigger può produrre eventi che fanno scattare altri trigger, che dovranno essere 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
- Per i trigger row-level il TEC tiene conto di quali tuple sono già state considerate e quali sono da considerare
- Si ha quindi una struttura a stack
  - TEC0 -> TEC1 -> ... -> TECn
- Quando un trigger ha considerato tutti gli eventi, il TEC si chiude e si passa al trigger successivo
- È un modello complicato, ma preciso e relativamente semplice da implementare

Esempio di esecuzione

Gestione dei salari

https://goo.gl/kp2Cu9

Impiegato

Matricola	Nome	Salario	NDip	NProg
50	Rossi	59.000	1	20
51	Verdi	56.000	1	10
52	Bianchi	50.000	1	20

Dipartimento

NroDip	MatricolaMGR
1	50

Progetto

NroProg	Obiettivo
10	NO
20	NO

Trigger T1: Bonus

Evento: **update** di **`Obiettivo`** in **`Progetto`**

Condizione: **`Obiettivo`** = **'SI'**

Azione: incrementa del 10% il salario degli impiegati coinvolti nel progetto

Trigger T1: Bonus

Evento: **update** di **Obiettivo** in **Progetto**

Condizione: **Obiettivo = 'SI'**

Azione: incrementa del 10% il salario degli impiegati coinvolti nel progetto

```
CREATE TRIGGER Bonus
AFTER UPDATE OF Obiettivo ON Progetto
FOR EACH ROW
WHEN NEW.Obiettivo = 'SI'
BEGIN
  update Impiegato
            set Salario = Salario*1.10
            where NProg = NEW.NroProg;
END;
```

Trigger T2: ControllaIncremento

Evento: **update** di **`Salario`** in **`Impiegato`**

Condizione: nuovo salario maggiore di quello del manager

Azione: decrementa il salario rendendolo uguale a quello del manager

Trigger T2: ControllaIncremento

Evento: **update** di **`Salario`** in **`Impiegato`**

Condizione: nuovo salario maggiore di quello del manager

Azione: decrementa il salario rendendolo uguale a quello del manager

```
CREATE TRIGGER ControllaIncremento
AFTER UPDATE OF Salario ON Impiegato
FOR EACH ROW
DECLARE X number;
BEGIN
            SELECT Salario into X
            FROM Impiegato JOIN Dipartimento
            ON Impiegato.Matricola = Dipartimento.MatricolaMGR
            WHERE Dipartimento.NroDip= NEW.NDip;

            IF NEW.Salario > X
              update Impiegato set Salario = X
              where Matricola = NEW.Matricola;
            ENDIF;
END;
```

Trigger T3: ControllaDecremento

Evento: **update** di **`Salario`** in **`Impiegato`**

Condizione: decremento maggiore del 3%

Azione: decrementa il salario del solo 3%

Trigger T3: ControllaDecremento

Evento: **update** di **`Salario`** in **`Impiegato`**

Condizione: decremento maggiore del 3%

Azione: decrementa il salario del solo 3%

```
CREATE TRIGGER ControllaDecremento
AFTER UPDATE OF Salario ON Impiegato
FOR EACH ROW
WHEN (NEW.Salario < OLD.Salario * 0.97)
BEGIN
        update Impiegato
        set Salario=OLD.Salario*0.97
        where Matricola = NEW.Matricola;
END;
```

https://goo.gl/C8VbDN

Attivazione di T1

```
Update Progetto
set Obiettivo = 'SI' where NroProg = 10
```

Progetto

NroProg	Obiettivo
10	SI
20	NO

Evento: update dell'attributo **Obiettivo** in **Progetto**

Condizione: vera

Azione: si incrementa del 10% il salario di Verdi

Matricola	Nome	Salario	Ndipart	NProg
50	Rossi	59.000	1	20
51	Verdi	61.600	1	10
52	Bianchi	50.000	1	20

Attivazione di T2

Evento: **update** di **`Salario`** in **`Impiegato`**

Condizione: vera (il salario dell'impiegato Verdi supera quello del manager Rossi)

Azione: si modifica il salario di Verdi rendendolo uguale a quello del manager Rossi

Matricola	Nome	Salario	Ndipart	NProg
50	Rossi	59.000	1	20
51	Verdi	59.000	1	10
52	Bianchi	50.000	1	20

- Si attiva nuovamente T2 - condizione è falsa
- Si attiva T3

Attivazione di T3

Evento: update dell'attributo **`salario`** in **`Impiegato`**

Condizione: vera (il salario di Verdi è stato decrementato per più del 3%)

Azione: si decrementa il salario di Verdi del solo 3%

Matricola	Nome	Salario	Ndipart	NProg
50	Rossi	59.000	1	20
51	Verdi	59.752	1	10
52	Bianchi	50.000	1	20

- Si attiva nuovamente T3 - condizione è falsa
- Si attiva T2 – condizione vera

Attivazione di T2

Matricola	Nome	Salario	Ndipart	NProg
50	Rossi	59.000	1	20
51	Verdi	59.000	1	10
52	Bianchi	50.000	1	20

Attivazione di T3

- La condizione del trigger è falsa
  - Il salario è stato decrementato per meno del 3%
- L'attivazione dei trigger ha raggiunto lo stato di terminazione

Progettazione

Proprietà dei trigger

È importante avere garanzie sul fatto che l'interferenza tra diversi trigger e l'attivazione a catena non generi anomalie nel comportamento del sistema

- 3 proprietà classiche
  - Terminazione
    - Per qualunque stato iniziale e qualunque sequenza di modifiche, i trigger producono uno stato finale (non vi sono cicli infiniti di attivazione)
  - Confluenza
    - I trigger terminano e producono un unico stato finale, indipendente dall'ordine in cui i trigger vengono eseguiti
    - La proprietà è significativa solo quando il sistema presenta del non-determinismo nella scelta dei trigger da eseguire
  - Determinismo delle osservazioni
    - I trigger sono confluenti e in più producono la stessa sequenza di messaggi
- La proprietà più importante è di gran lunga la terminazione

Analisi di terminazione

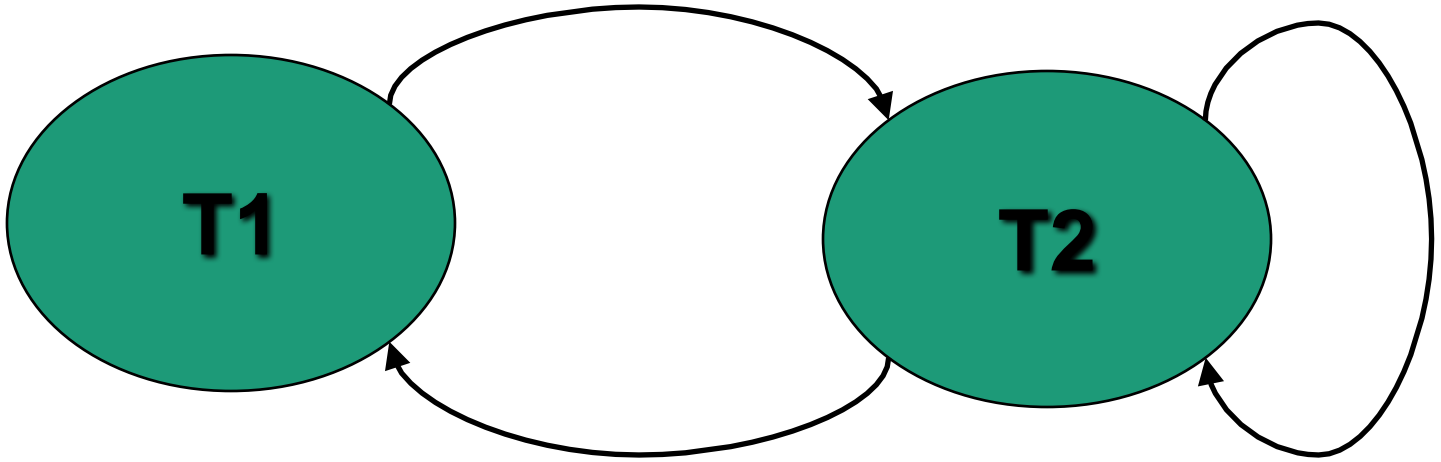
- Vi sono diversi strumenti concettuali, quasi tutti basati su grafi
- Il più semplice è il grafo di attivazione (*triggering graph*)
  - Un nodo per ogni trigger
  - Un arco da un nodo *ti* a un nodo *tj* se l'esecuzione dell'azione di *ti* può attivare il trigger *tj* (si può fare con una semplice analisi sintattica)
- Se il grafo è aciclico, si ha la garanzia che il sistema è terminante
  - non vi possono essere sequenze infinite di trigger
- Se il grafo ha dei cicli, c'è la possibilità che il sistema sia non-terminante (ma non è detto)

Esempio con due trigger

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

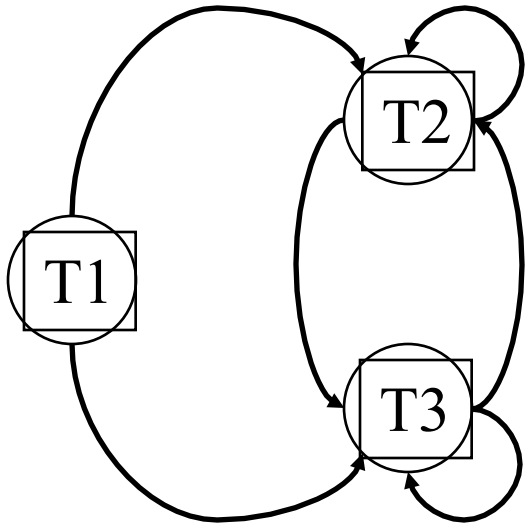
Triggering graph per i trigger precedenti



Vi sono 2 cicli. Il sistema è però terminante (se T1 ha priorità maggiore di T2; update Impiegato set Contributi=100000 where Matr=1 ….)

Per renderlo non terminante è sufficiente invertire il verso del confronto nella condizione del trigger T2

Grafo di Terminazione per i trigger di gestione dei salari



Il grafo è ciclico, ma l'esecuzione ripetuta dei trigger porta comunque allo stato di quiescenza

Problemi nel disegno di applicazioni dei trigger

- Il potenziale dei trigger è molto elevato
  - possono essere uno strumento che arricchisce la base di dati e porta all'interno di essa aspetti relativi alla gestione dei dati che altrimenti vengono distribuiti su tutte le applicazioni che usano i dati

  ... ma questo potenziale è poco sfruttato
- Realizzare applicazioni con i trigger è complicato
- L'ambiente di sviluppo offerto dai sistemi è inadeguato
- I trigger sono usati per realizzare servizi innovativi da parte dei produttori di DBMS, introducendo meccanismi per la generazione automatica di trigger.
- Ad esempio:
  - gestione di vincoli
  - replicazione dei dati
  - mantenimento di viste

Tecniche e metodologie per il disegno di trigger

- Proposte per il progetto su piccola scala e su scala più ampia
  - per il progetto su piccola scala, conviene sfruttare gli strumenti di analisi disponibili (triggering graph e altri)
  - per il progetto su scala più grande, conviene far riferimento a tecniche e metodologie apposite
- La modularizzazione è una tecnica che prevede di organizzare i trigger in moduli destinati a un obiettivo specifico
  - Se ciascun modulo realizza correttamente il proprio obiettivo e se l'interferenza con gli altri moduli è innocua (da dimostrare in modi diversi), si ha la garanzia che il sistema è corretto

Applicazioni delle basi di dati attive

Applicazioni delle basi di dati attive

- Applicazioni classiche: regole **interne** alla base di dati
  - Trigger generati dal sistema e non visibili all'utente
  - Principali funzionalità:
    - Gestione dei vincoli di integrità, la computazione di dati derivati, la gestione dei dati replicati;
  - Altre funzionalità:
    - Gestione di versioni, privatezza, sicurezza, logging delle azioni, registrazione degli eventi, …

- Regole **esterne** (o regole aziendali)
  - Esprimono conoscenza di tipo applicativo

Gestione dell'integrità referenziale

- *Strategie di riparazione* per le violazioni dei vincoli di integrità referenziale
  - Il vincolo è espresso come predicato nella parte condizione

Es:
```
CREATE TABLE Impiegato(
    … …
    FOREIGN KEY(NDip) REFERENCES Dipartimento(NroDip)
      ON DELETE SET NULL,
    … …);
```

- Operazioni che possono violare questo vincolo:
  - `INSERT` in Impiegato
  - `UPDATE` di Impiegato.NDip
  - `UPDATE` di Dipartimento.NroDip
  - `DELETE` in Dipartimento

https://goo.gl/Zx7EQO

Azioni nella tabella Impiegato

Evento: **inserimento** in `Impiegato`

Condizione: il nuovo valore di `Ndip` non è tra quelli contenuti nella tabella `Dipartimento`

Azione: si inibisce l'inserimento, segnalando un errore

```
CREATE TRIGGER ControllaDipImpiegato
BEFORE INSERT ON Impiegato
FOR EACH ROW
WHEN (NOT EXISTS (SELECT * FROM Dipartimento
     WHERE NroDip = NEW.NDip))
BEGIN
  SELECT RAISE(ABORT, "Dipartimento non
  valido");
END;
```

Per la modifica di `NDip` in `Impiegato` il trigger cambia solo nella parte evento

https://goo.gl/YGMWrs

Cancellazione nella tabella Dipartimento

Evento: **cancellazione** in `Dipartimento`

Condizione: il valore di `NroDip` che si intende cancellare è tra quelli contenuti nella tabella `Impiegato`

Azione: si assegna il valore nullo a `NDip` in `Impiegato`

```
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;
```

https://goo.gl/gs95rZ

Modifiche nella tabella Dipartimento

Evento: modifica dell'attributo **`NroDip`** in **`Dipartimento`**

Condizione: il vecchio valore di **`NroDip`** è tra quelli contenuti nella tabella **`Impiegato`**

Azione: si modifica anche **`NDip`** in **`Impiegato`**

```
CREATE TRIGGER ControllaModificaDipartimento
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;
```

https://goo.gl/WHrMNu

Trigger per il mantenimento di viste materializzate

- Consistenza delle viste rispetto alle tabelle sulle quali sono state definite
  - Le modifiche sulle tabelle di base devono essere propagate sulle viste
- Gestione della replicazione:

```
CREATE MATERIALIZED VIEW ReplicaImpiegato
REFRESH FAST AS
SELECT * FROM
DBMaster.Impiegato@sitomaster.world;
```

- Il mantenimento delle viste materializzate è gestito tramite trigger

Gestione della ricorsione

- Trigger per la gestione della ricorsione
  - Ricorsione non ancora supportata da tutti i DMBS correnti
- Es.: rappresentazione di una gerarchia di prodotti
  - Ogni prodotto è caratterizzato da un **super-prodotto** e da un **livello** di profondità nella gerarchia
  - Rappresentabile tramite una vista ricorsiva (costrutto **`with recursive`** in SQL:1999)
  - In alternativa: uso dei trigger per la costruzione ed il mantenimento della gerarchia

Prodotto(Codice, Nome, Descrizione, SuperProdotto, Livello)

- Gerarchia rappresentata tramite **`SuperProdotto`** e **`Livello`**
- Prodotti non contenuti in altri prodotti: **`SuperProdotto = NULL`** e **`Livello = 0`**

https://goo.gl/HMpKEw

Cancellazione di un prodotto

In caso di cancellazione di un prodotto è necessario cancellare anche tutti i sottoprodotti che lo compongono

```
CREATE TRIGGER CancellaProdotto
 AFTER DELETE ON Prodotto
 FOR EACH ROW
 BEGIN
  delete from Prodotto
  where SuperProdotto = OLD.Codice;
 END;
```

https://goo.gl/JS9Gzt

Inserimento di un nuovo prodotto

In caso di inserimento è necessario calcolare il valore appropriato per l'attributo **`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)
  ELSE
       UPDATE Prodotto
       SET Livello = 0
       WHERE Codice = NEW.Codice;
  ENDIF;
END;
```

https://goo.gl/Vh7yLT

Set di trigger non terminanti

- https://goo.gl/bvyEh3

- In SQLite è importante specificare che i trigger possono essere richiamati ricorsivamente con:

- **PRAGMA recursive_triggers = 1;**

Controllo degli accessi

- I trigger possono essere utilizzati per rinforzare il controllo sugli accessi
- E' conveniente definire solo quei trigger che corrispondono a condizioni che non possono essere verificate direttamente dal DBMS
- Uso del **`BEFORE`** per i seguenti vantaggi:
  - Il controllo dell'accesso è eseguito prima che l'evento del trigger sia eseguito
  - Il controllo dell'accesso è eseguito una sola volta e non per ogni tupla su cui si verifica l'evento del trigger

Trigger InibisciModificaSalario

```
CREATE TRIGGER InibisciModificaSalario
BEFORE INSERT ON Impiegato
DECLARE
  non_nel_weekend EXCEPTION;
  non_in_extraOreLavorative EXCEPTION;
BEGIN
/*se weekend*/
   IF (to_char(sysdate, 'dy') = 'SAT'
    OR to_char(sysdate, 'dy') = 'SUN')
  THEN RAISE non_nel_weekend;
  END IF;
/* se al di fuori dell'orario di lavoro(8-18) */
  IF (to_char(sysdate, 'HH24') < 8
       OR to_char(sysdate, 'HH24') > 18)
  THEN RAISE non_in_extraOreLavorative;
  END IF;
```

Trigger InibisciModificaSalario (cont.)

```
EXCEPTION
WHEN non_nel_weekend
    THEN raise_application_error(-20324,'non
    e' possibile modificare la tabella
    impiegato durante il weekend');
WHEN non_in_extraOreLavorative
    THEN raise_application_error(-20325,' non
    e' possibile modificare la tabella
    impiegato al di fuori dell'orario di
    lavoro');
END;
```

Evoluzione dei trigger

Evoluzione dei trigger: Eventi/1

- Eventi di sistema e comandi DDL
  - Sistema: *servererror*, *shutdown*, etc.
  - DDL: Modifiche di autorizzazioni
  - In entrambi i casi alcuni DBMS mettono già a disposizione questi servizi, che consentono la realizzazione di monitoraggi sofisticati
- Eventi temporali (anche periodici)
  - Esempi: "*il 23/7/04 alle ore 12:00*"; "*ogni giorno alle 4:00*"
  - Sono di interesse in diverse applicazioni
  - È difficile integrarli perché operano in un contesto transazionale autonomo
  - Si possono comunque simulare con componenti software esterne al DBMS che usano i servizi di gestione del tempo del sistema operativo per produrre un opportuno evento interno al DBMS

Evoluzione dei trigger: Eventi/2

- Eventi “definiti dall’utente”
  - Esempio: “TemperaturaTroppoAlta”
  - Sono di interesse in alcune applicazioni, ma non sono normalmente offerti
  - Sono anch’essi facilmente simulabili
- Interrogazioni
  - Esempio: chi legge gli stipendi
  - È di norma troppo pesante gestirli

Espressioni su eventi e modo **`instead of`**

- Combinazioni booleane di eventi
  - SQL:1999 consente di specificare più eventi per un trigger, in disgiunzione
    - è sufficiente un evento qualsiasi tra quelli elencati
  - Alcuni ricercatori hanno proposto modelli di composizione più sofisticati
    - Sono molto complicati da gestire
    - Non vi sono forti motivazioni che giustifichino il costo della loro introduzione
- Clausola **`instead of`**
  - è una modalità alternativa a BEFORE e AFTER
  - non si esegue l'operazione che ha attivato l'evento, ma un'altra azione
  - è implementata in diversi sistemi, spesso con forti limitazioni
    - in Oracle si può usare esclusivamente per eventi di modifica su viste, risolvendo il problema del view update per viste generiche

Priorità, attivazione e gruppi

- Definizione di priorità
  - Permette di specificare l'ordine di esecuzione dei trigger quando ve ne sono diversi attivati contemporaneamente
  - SQL:1999 specifica che prima un ordine che si basa sul modo di esecuzione e sulla granularità del trigger; a pari modo, la scelta dipende dall'implementazione
  - Nei sistemi spesso si usa l'ordine temporale di definizione
- Trigger attivabili e disattivabili
  - Non presente nello standard, ma spesso disponibile nei sistemi
- Trigger organizzati in gruppi
  - Qualche sistema offre meccanismi di raggruppamento dei trigger, per attivare e disattivare per gruppi

Modi di esecuzione

- Il modo di esecuzione descrive il legame tra l'attivazione (evento) e le fasi di considerazione ed esecuzione (condizione e azione)
  - condizione e azione sono sempre valutate assieme
- 3 alternative classiche
  - immediato (*immediate*)
    - Il trigger viene considerato ed eseguito con l'evento che lo ha attivato
    - Ad esempio: trigger che verificano immediatamente il rispetto di vincoli di integrità
  - differito (*deferred*)
    - Il trigger viene gestito al termine della transazione
    - Ad esempio: trigger che verificano il rispetto di vincoli di integrità che richiedono lo svolgimento di diverse operazioni
  - distaccato (*detached*)
    - Il trigger viene gestito in una transazione separata
    - Ad esempio: si vogliono gestire in modo efficiente le variazioni del valore di indici di borsa in seguito a numerosi scambi

Esercizi sulle Regole attive

Autori

Dato il seguente schema relazionale:

Libro (Isbn,Titolo,NumCopieVendute)
Scrittura (Isbn, Nome)
Autore (Nome, NumCopieVendute)

Definire un insieme di regole attive in grado di mantenere aggiornato l'attributo NumCopieVendute di *Autore* rispetto a:

- Modifiche sull'attributo *NumCopieVendute* di *Libro*
- Inserimenti su *Scrittura*.

https://goo.gl/S5F80J

```
create trigger AggiornaCopieAutoreDopoNuoveVendite
after update of NumCopieVendute on Libro
for each row
update Autore
  set NumCopieVendute = NumCopieVendute +
                        new.NumCopieVendute –
                        old.NumCopieVendute
where Nome in ( select Nome
                from Scrittura
                where Isbn = new.Isbn)
```

ATTENZIONE: NumCopieVendute di Autore (quello su cui agisce la regola) è diverso dal NumCopieVendute di Libro.

https://goo.gl/DVbg8r

```
create trigger AggiornaCopieAutoreDopoAggiuntaInScrittura
after insert on Scrittura
for each row
update Autore
  set NumCopieVendute = NumCopieVendute +
                        (select NumCopieVendute
                         from Libro
                         where Isbn=new.Isbn)
where Nome = new.Nome
```

https://goo.gl/PsM4dD

Potrebbe sembrare che dobbiamo anche reagire agli inserimenti in LIBRO, Tuttavia, perché la paternità del libro venga assegnata, deve essere comunque aggiunta una tupla in SCRITTURA, e abbiamo già definito un trigger che aggiorna le copie in quel caso. Sarebbe quindi sbagliato scrivere un trigger del tipo:

```
create trigger AggiornaCopiePerNuovoLibro
after insert on Libro
for each row
update Autore
  set NumCopieVendute = NumCopieVendute +
                        new.NumCopieVendute
where Nome in ( select Nome
                from Scrittura
                where Isbn = new.Isbn)
```