

PL/SQL

Versione dei trigger e PSM di Oracle

Lucidi derivati da quelli di Jeffrey D. Ullman

1

PL/SQL

- ◆ Oracle usa una variante di SQL/PSM che si chiama PL/SQL.
- ◆ PL/SQL non consente solo di creare e memorizzare procedure e funzioni, ma può essere eseguito dalla interfaccia generica per le query, come una istruzione SQL.
- ◆ I triggers sono una parte di PL/SQL.

2

Differenze nei triggers

- ◆ In confronto con i trigger SQL standard, quelli di Oracle hanno le seguenti differenze:
 1. L'azione è una istruzione PL/SQL.
 2. Sono automaticamente disponibili riferimenti alle tuple nuove/vecchie.
 3. Ci sono vincoli forti sulle azioni previste per evitare catene infinite di attivazioni di triggers.

3

Ordine degli elementi dei triggers di Oracle

1. CREATE OR REPLACE TRIGGER
2. Evento, ad es. AFTER INSERT ...
3. FOR EACH ROW, se richiesto.
4. Condizione.
5. Azione.
6. Un punto e la parola "run". Questa fa sì che il trigger sia installato nel database.

4

Tuple vecchie/nuove

- ◆ Invece di una clausola REFERENCING, Oracle assume che alle nuove tuple si faccia riferimento con "new" e a quelle vecchie con "old."
- ◆ Inoltre, per tiggers a livello di istruzione: "newtable" e "oldtable".
- ◆ Nelle azioni, *ma non nelle condizioni*, bisogna far precedere "new," ecc., da un due punti.

5

Esempio: BeerTrig

- ◆ Si ricordi il trigger BeerTrig, che inserisce un nome di birra in Beers quando una tupla era inserita in Sells con una birra che non era menzionata in Beers.
- ◆ Ecco la versione Oracle dello stesso trigger.

6

BeerTrig nell'SQL di Oracle

```
CREATE OR REPLACE TRIGGER BeerTrig
AFTER INSERT ON Sells
FOR EACH ROW
WHEN (new.beer NOT IN
      (SELECT name FROM Beers))
BEGIN
  INSERT INTO BEERS(name) VALUES (:new.beer);
END;
```

run

Necessario per memorizzare il trigger come un elemento del database

Si noti che "new" e' capito. Inoltre, i due punti sono usati solo nell'azione.

7

Un altro esempio

- ◆ Si ricordi PriceTrig, che memorizza nella relazione Ripoffbars(bar) il nome di ogni bar che aumenta il prezzo di una qualunque birra di piu' di 1\$.
- ◆ Ecco la versione di Oracle.

8

PriceTrig in Oracle

```
CREATE OR REPLACE TRIGGER PriceTrig
  AFTER UPDATE OF price ON Sells
  FOR EACH ROW
  WHEN (new.price > old.price + 1.00)
  BEGIN
    INSERT INTO RipoffBars VALUES(:new.bar);
  END;
.
```

run

9

Limitazioni di Oracle sulle relazioni coinvolte

- ◆ Ogni trigger riguarda una relazione R , menzionata nell'evento.
- ◆ Lo standard SQL standard non mette vincoli su quali relazioni, incluso R , possono essere modificate nell'azione.
- ◆ Come risultato, sequenze infinite di eventi scatenati sono possibili.

10

Esempio: triggering infinito

- ◆ Sia $R(x)$ una relazione unaria che è un insieme di interi.
- ◆ È facile scrivere un trigger con evento INSERT ON R , che, come azione, inserisce $i+1$ se i era l'intero che ha scatenato il trigger.
- ◆ Risulta in una sequenza senza fine di inserimenti.

11

Limitazione di Oracle

- ◆ Oracle è molto conservativo riguardo alle relazioni che possono essere modificate quando l'evento è su R .
- ◆ R sicuramente non può essere il soggetto di una modifica nell'azione.
- ◆ Ma è più sottile: ogni relazione che è collegata ad R da una catena di vincoli di chiave esterna non può essere cambiata anch'essa.

12

Esempio: catene di chiavi esterne

- ◆ Si supponga che $R.a$ sia una chiave esterna, che fa riferimento a $S.b$.
- ◆ Inoltre, $T.c$ e' una chiave esterna che fa riferimento a $S.b$.
- ◆ Allora in un trigger sulla relazione R , ne' T ne' S possono essere modificate.

13

PL/SQL

- ◆ In aggiunta alla stored procedures, si possono scrivere istruzioni PL/SQL che assomigliano al corpo di una procedura, ma sono eseguite solo una volta, come una istruzione SQL scritta nell'interfaccia generica.
 - ◆ Oracle chiama l'interfaccia generica "sqlplus."
 - ◆ Il "plus" e' in realta il PL/SQL.

14

Forma delle istruzioni PL/SQL

```
DECLARE
  <dichiarazioni>
BEGIN
  <istruzioni>
END;
.
```

run

- ◆ La sezione DECLARE e' opzionale.

15

Forma di una procedura PL/SQL

```
CREATE OR REPLACE PROCEDURE
  <nome> (<argomenti>) AS
  <dichiarazioni opzionali>
BEGIN
  <istruzioni PL/SQL>
END;
```

Si noti AS e' richiesto qui

run

Richiesto per memorizzare la procedura nel database

16

Dichiarazioni e assegnamenti PL/SQL

- ◆ La parola DECLARE non deve apparire davanti a ogni dichiarazione locale.
 - ◆ Si usa semplicemente il nome della variabile e il suo tipo.
- ◆ Non si usa la parola SET negli assegnamenti e := e' usato al posto di =.
 - ◆ Esempio: `x := y;`

17

Parametri delle procedure PL/SQL

- ◆ Ci sono diverse differenze nella forma degli argomenti e delle variabili locali in PL/SQL rispetto allo standard SQL/PSM:
 1. L'ordine e' nome-modo-tipo, non modo-nome-tipo.
 2. INOUT e' sostituito da IN OUT in PL/SQL.
 3. Diversi nuovi tipi.

18

Tipi PL/SQL

- ◆ In aggiunta ai tipi SQL, NUMBER puo' essere usato per indicare un INT or un REAL.
- ◆ Ci si puo' riferire al tipo di un attributo x della relazione R con `R.x%TYPE`.
 - ◆ Utile per evitare discordanze nel tipo.
 - ◆ Inoltre, `R%ROWTYPE` e' una tupla i cui componenti hanno il tipo degli attributi di R .

19

Esempio: JoeMenu

- ◆ Si ricordi la procedure `JoeMenu(b,p)` che aggiunge la birra b al prezzo p alle birre vendute da Joe's Bar (nella relazione `Sells`).
- ◆ Ecco la versione PL/SQL.

20

Procedura JoeMenu in PL/SQL

```
CREATE OR REPLACE PROCEDURE JoeMenu (  
  b IN Sells.beer%TYPE,  
  p IN Sells.price%TYPE  
) AS  
  BEGIN  
    INSERT INTO Sells  
      VALUES ('Joe''s Bar', b, p);  
  END;  
.  
run
```

21

Istruzioni di branching in PL/SQL

- ◆ Come IF ... in SQL/PSM, ma:
- ◆ Si usa ELSIF al posto di ELSEIF.
- ◆ Ad es.: IF ... THEN ... ELSIF ... ELSIF ... ELSE ... END IF;

22

Cicli in PL/SQL

- ◆ LOOP ... END LOOP come in SQL/PSM.
- ◆ Al posto di LEAVE ... , PL/SQL usa EXIT WHEN <condizione>
- ◆ e quando la condizione e' che il cursore c non ha trovato tuple, possiamo scrivere c%NOTFOUND come condizione.

23

Cursori in PL/SQL

- ◆ La forma di una dichiarazione di cursore PL/SQL e' :
CURSOR <nome> IS <query>;
- ◆ Per prelevare tuple dal cursore c, si usa:
FETCH c INTO <variabile(i)>;

24

Esempio: JoeIncrease () in PL/SQL

- ◆ Si ricordi che JoeIncrease () manda un cursore attraverso la porzione di Sells riferita a Joe's-Bar, e aumenta di 1\$ il prezzo di ogni birra venduta da Joe's Bar, se quel prezzo era inizialmente meno di 3\$.

25

Esempio: dichiarazioni di JoeIncrease ()

```
CREATE OR REPLACE PROCEDURE
    JoeIncrease() AS
    theBeer Sells.beer%TYPE;
    thePrice Sells.price%TYPE;
    CURSOR c IS
        SELECT beer, price FROM Sells
        WHERE bar = 'Joe''s Bar';
```

26

Esempio: corpo di JoeIncrease

```
BEGIN
    OPEN c;
    LOOP
        FETCH c INTO theBeer, thePrice;
        EXIT WHEN c%NOTFOUND;
        IF thePrice < 3.00 THEN
            UPDATE Sells SET price = thePrice + 1.00;
            WHERE bar = 'Joe''s Bar' AND beer = theBeer;
        END IF;
    END LOOP;
    CLOSE c;
END;
```

Il modo di PL/SQL di interrompere un ciclo di un cursore

Si noti che questa e' una clausola SET in una UPDATE, non un assegnamento. PL/SQL usa := per gli assegnamenti.

27

Variabile con valore tupla

- ◆ PL/SQL consente di avere una variabile x con un tipo tupla.
- ◆ x R%ROWTYPE assegna a x il tipo delle tuple di R.
- ◆ R puo' essere una relazione o un cursore.
- ◆ $x.a$ fornisce il valore dell'attributo a nella tupla x .

28

Esempio: tipo tupla

- ◆ Ecco le dichiarazioni di JoeIncrease(), usando una variabile *bp* il cui tipo e' coppia birra-prezzo, come restituito dal cursore *c*.

```
CREATE OR REPLACE PROCEDURE
  IncreaseJoe() AS
  CURSOR c IS
  SELECT beer, price FROM Sells
  WHERE bar = 'Joe's Bar';
  bp c%ROWTYPE;
```

29

Corpo di JoeIncrease usando *bp*

```
BEGIN
  OPEN c;
  LOOP
    FETCH c INTO bp;
    EXIT WHEN c%NOTFOUND;
    IF bp.price < 3.00 THEN
      UPDATE Sells SET price = bp.price + 1.00;
      WHERE bar = 'Joe's Bar' AND beer = bp.beer;
    END IF;
  END LOOP;
  CLOSE c;
END;
```

I componenti di bp sono ottenuti con un punto e il nome dell'attributo.

30

Esempio

```
Create Or Replace Procedure Debit(ClientAccount
char(5),Withdrawal integer) as
  OldAmount integer;
  NewAmount integer;
  Threshold integer;
begin
  select Amount, Overdraft into OldAmount,
  Threshold
  from BankAccount
  where AccountNo = ClientAccount;
  NewAmount := OldAmount - Withdrawal;
```

31

Esempio

```
if NewAmount > Threshold
  then update BankAccount
  set Amount = NewAmount
  where AccountNo = ClientAccount;
  else
  insert into OverDraftExceeded
  values(ClientAccount,Withdrawal,sysdate);
  end if;
end;
```

32