Lezione 14

DBMS attivi

Seconda parte

Procedure e funzioni

Le procedure

- Una procedura è un codice composto
  - da un blocco con intestazione
  - da dichiarazioni SQL
  - da istruzioni procedurali
  - può avere parametri di input (argomenti) memorizzate nella base di dati
- Il codice deve essere attivato richiamandolo da
  - un programma esterno
  - un'altra procedura
  - un trigger
  - riga di comando di sql-developer

Le procedure

- Hanno una intestazione con il nome e i parametri di input
- la clausola IS sostituisce la DECLAIRE
- La definizione fino alla parola chiave IS è detta FIRMA della procedura

```
CREATE [or REPLACE] PROCEDURE
        nome_procedura [(parametri_formali)]
IS
    definizioni;
BEGIN
    corpo procedura;
END;
```

- La [or REPLACE], opzionale, serve per ricreare la procedura nel caso in cui essa sia già stata definita in precedenza

Attivazione procedure

- Mediante chiamata

nome_procedura**(**[parametri_effettivi]**);**

- La lista dei parametri_effettivi deve corrispondere in numero e tipo a quella parametri_formali
- Possono essere vuote nome_procedura()
- In alcune versioni di PL/SQL non Oracle la chiamata è esplicita con CALL nome

EXEC[UTE] nome_procedura**(**[parametri_effettivi]**);**

- Consente di attivare la procedura dalla shell di SQL Developer di Oracle

La lista dei parametri formali

<nome parametro> [IN | OUT | IN OUT] <tipo parametro>[,]

- La lista parametri_formali è una sequenza di

$nome\_var_1$ [mod_1] $TIPO_1$, $nome\_var_2$ [mod_2] $Tipo_2$, …, $nome\_var_n$ [mod_n] $Tipo_n$

  - in cui mod fissa la modalità di sostituzione IN, OUT e IN OUT
  - in cui TIPO è solo il qualificatore di un tipo ORACLE
  - le caratteristiche (lunghezza, precisione, etc.) sono derivate dai corrispondenti parametri_effettivi
  - ad esempio NUMBER va bene mentre NUMBER(3) no

- IN indica che il parametro è di input, OUT di uscita
- Di default i parametri sono utilizzati solo per il passaggio in ingresso delle informazioni (IN)
- Il passaggio IN OUT equivale a un passaggio di dati per riferimento in C.

A cosa servono

- Per operazioni di manutenzione della base dati non interattive (backup e restore particolari)
- Per aggiornamenti programmati di tipo batch
  - assegnazione di valori iniziali
  - aggiornamento/controllo dati ridondati
  - allineamento delle viste materializzate
  - sincronizzazione di basi dati distribuite
- Per migliorare le prestazioni delle applicazioni evitando molti collegamenti che impegnano le reti per operazioni complesse

Funzioni

- Come le procedure ma restituiscono un valore con RETURN

```
CREATE [or REPLACE] FUNCTION
            nome_funzione [(parametri_formali)]
            RETURN <tipo_dato>
IS
   definizioni;
BEGIN
    corpo_funzione;
    RETURN espressione;
END;
```

I Package

Cosa sono

- I package sono librerie di procedure e funzioni
- Sono strutturati in due parti
  - la specification ed il body
  - solo le procedure, funzioni e variabili incluse nella specification possono essere accedute dagli altri programmi PL/SQL presenti nel DB;
  - le procedure, funzioni e variabili presenti solo nel body sono “private” ed accessibili solo dagli altri programmi del package.

Vantaggi

- Modularità dell'architettura applicativa
  - pochi pacchetti di programmi omogenei anziché molti programmi sciolti
- Riusabilità in progetti diversi delle stesse funzionalità
- Visibilità delle funzionalità e occultamento dei dettagli con la struttura a due livelli
- Gestione delle variabili di sessione
  - una variabile globale (definita nel package al di fuori di tutte le procedure e funzioni) è istanziata la prima volta che il package viene richiamato e resta valida fino a fine sessione
  - possono essere quindi utilizzate da altri programmi che gireranno in seguito nella stessa sessione

La SPECIFICATION

CREATE OR REPLACE PACKAGE <nome package> IS
PROCEDURE/FUNCTION <nome procedura o funzione$_1$>
<parametri> <eventuale clausola return>;
PROCEDURE/FUNCTION <nome procedura o funzione$_2$>
<parametri> <eventuale clausola return>;

……

PROCEDURE/FUNCTION <nome procedura o funzione$_n$>
<parametri> <eventuale clausola return>;
<dichiarazione delle variabili di package>
END <nome package> ;

- Nella specification si dichiarano solo le firme delle procedure o funzioni accessibili dall'esterno del package

Il BODY

CREATE OR REPLACE PACKAGE BODY <nome package>
IS
<dichiarazione delle variabili private di package>

PROCEDURE/FUNCTION <nome procedura o funzione$_1$>
<parametri> <eventuale clausola return>
IS
<variabili di procedura o funzione>
BEGIN
<corpo della procedura o funzione>
END;
PROCEDURE/FUNCTION <nome procedura o funzione$_2$>
<parametri> <eventuale clausola return>
IS
<variabili di procedura o funzione>
BEGIN
<corpo della procedura o funzione>
END;
……..
PROCEDURE/FUNCTION <nome procedura o funzione$_n$>
<parametri> <eventuale clausola return> IS
<variabili di procedura o funzione>
BEGIN
<corpo della procedura o funzione>
END;
END <nome package> ;

- Nel body si dettaglia il corpo di tutte le procedure e funzioni
  - sia di quelle accessibili solo dagli altri programmi del package sia di quelle accessibili esternamente anche dagli altri programmi presenti nel DB.

Considerazioni

- Il file SPECIFICATION separato dal BODY ci consente di:
  - risolvere il problema delle dichiarazioni delle procedure/funzioni nel rispetto della regola di visibilità
    - un identificatore può essere usato se in precedenza dichiarato
    - tipico della dichiarazione di un insieme di procedure/funzioni in cui una di esse richiama un'altra disposta dopo nell'elenco

I trigger

Cosa sono

- Un *trigger* è una specifica per la quale
  - una data azione o funzione deve attivarsi in maniera automatica
  - ogni qual volta un'operazione specifica viene eseguita su un oggetto della base di dati
- I trigger sono applicazioni memorizzate nel DBMS che li esegue automaticamente quando accade un dato evento
  - si dice che i trigger si "accendono" (*fire*) da soli al verificarsi dell'evento
- Si parla pertanto di **DBMS attivi (ADBMS)**

Esempio

- Regola su un Data Base di magazzino
  - se la quantità di un prodotto si modifica per la vendita e il valore scende al di sotto di 4 pezzi, allora si deve fare un ordine di 100 pezzi
- La si può implementare a livello applicativo
  - o si chiede al DBMS di intervenire ogni volta che la quantità di un prodotto in magazzino viene modificata

Lo standard

- La semantica generale di un trigger è
  - indipendente dal DBMS di riferimento
  - la sua specifica appartiene al DDL (Data Definition Language)
  - obbedisce allo standard SQL-3
- Due tipologie
  - *Trigger DML*
    - l'evento di innesco è una primitiva per la manipolazione dei dati: INSERT UPDATE o DELETE
  - *Trigger DDL o* trigger di sistema
    - reagiscono ad eventi di sistema, quali l'avvio e la chiusura di un database, la creazione o cancellazione di tabelle, ecc.

A cosa servono

- Gli usi possibili sono molteplici:
  - implementare *controlli* non esprimibili attraverso i costrutti dichiarativi del linguaggio SQL
  - segnalare automaticamente ad altri programmi che devono essere effettuate azioni quando vengono apportate modifiche a una tabella;
  - inserire valori opportuni in particolari colonne al momento di un inserimento o modifica
    - ricalcolo dei dati derivati;
  - rendere globali alcuni controlli affidandone l'esecuzione al DBMS
- I trigger permettono di effettuare controlli in modo più ampio di quanto consentito dai vincoli di integrità referenziali o di tipo check
  - ne rappresentano una estensione

Tre concetti portanti

- l'evento
  - ha il compito di accendere il trigger
- la condizione
  - che deve essere soddisfatta affinchè il trigger sia eseguito
- l'azione
  - che viene eseguita se la condizione sul trigger acceso è soddisfatta.

Paradigma E-C-A

- *Evento*
  - lo standard prevede che la codifica dell'evento sia un comando del SQL
    - in base all'esecuzione di detta primitiva, avviene l'innesco della regola (*attivazione*)
- *Condizione*
  - la condizione è un predicato booleano atto a valutare il verificarsi dell'evento (*valutazione o considerazione*)
  - se sottintesa, assume sempre il valore TRUE
- *Azione*
  - sequenza di operazioni che devono essere eseguite quando i due elementi precedenti hanno esito positivo (*esecuzione*)

Regola attiva

- La forma più comune di trigger è quindi:

  ON *evento* IF *condizione* THEN *azione*

  - se si verifica l'evento, la condizione è valutata
  - se la condizione è soddisfatta l'azione viene eseguita
- **Che cos'è un evento?**

  "Un evento è qualcosa che accade, o si verifica, che è di interesse e che può essere mappato in un istante di tempo"
- Le regole attive hanno origine dalle regole dell' Intelligenza Artificiale

Accensione

- I trigger si accendono per effetto di uno dei seguenti eventi:
  - istruzioni DML
    - (INSERT UPDATE o DELETE) su tabelle o viste
  - istruzioni DDL
    - (CREATE, ALTER, DROP)
  - eventi di sistema sulla base di dati
    - (SERVER ERROR, LOGON, LOGOFF, STARTUP, SHUT DOWN)

Caratteristiche

- Si possono definire diverse tipologie di trigger sulla base di
  - granularità
  - modalità di esecuzione
  - tabelle e dati di transizione
- Ogni trigger fa riferimento ad una tabella (target)
  - risponde ad eventi relativi a tale tabella
  - se la tabella viene eliminata con DROP, anche i trigger a essa assaciata vengono eliminati
  - è una proprietà di una tabella

Struttura

- Ogni trigger è caratterizzato da
  - nome
  - nome della tabella che deve essere monitorata
  - modo di esecuzione (BEFORE o AFTER)
  - l'evento monitorato (INSERT, DELETE o UPDATE)
  - la granularità (statement-level o row-level)
  - alias per le transition values e le transition tables
  - l'azione
  - il timestamp di creazione

Sintassi SQL:1999 di creazione

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 (azione)

Sintassi SQL:1999 di eliminazione

DROP TRIGGER *Nome*

- *Si ricordi che anche il DROP della tabella dichiarata nel trigger ha come effetto l'eliminazione del trigger stesso*

Esecuzione di un singolo trigger

- Due modalità 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 della base dati, 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 interessate dall'evento attivante
  - 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 interessata dall'evento attivante
  - Consente di scrivere i trigger in modo più semplice
  - Può essere meno efficiente

Clausola referencing

- Il formato della clausola dipende dalla granularità
  - Per il modo row level, si hanno due transition variables
    - **old** rappresenta il valore precedente della tupla che si sta valutando
    - **new** rappresenta il valore successivo della tupla che si sta valutando
  - Per il modo statement level, si hanno due transition tables
    - **old table** contiene il valore vecchio di tutte le tuple valutate
    - **new table** contiene il valore nuovo di tutte le tuple valutate
- La variabile **old** e la tabella **old table** non sono utilizzabili in trigger il cui evento è **insert**
- La variabile **new** e e la tabella **new table** non sono utilizzabili in trigger il cui evento è **delete**
- Nel caso di **update** sono tutte utilizzabili, variabili e tabelle old e new
- Le variabili e le tabelle di transizione sono importanti per realizzare i trigger in modo efficiente
- Oracle non gestisce old table e new table

In altri termini

- INSERT:
  - le tuple inserite possono essere accedute usando la clausola REFERENCING NEW (a livello tupla o tabella)
- DELETE:
  - le tuple cancellate possono essere accedute usando la clausola REFERENCING OLD (a livello tupla o tabella)
- UPDATE:
  - i valori precedenti e correnti delle tuple possono essere acceduti usando le clausole REFERENCING OLD e NEW (a livello tupla o tabella)

Condizione e azione

- **Condizione**:
  - predicato SQL arbitrario (clausola analoga a quella del WHERE)
  - il trigger si chiude se la condizione restituisce FALSE o UNKNOWN
- **Azione:**
  - un singolo statement SQL
  - una sequenza di statement

```
BEGIN
    SQL statement 1; SQL statement 2;...
END
```

- Condizione e azione possono essere eseguite
  - FOR EACH ROW (per ogni tupla interessata dall'evento)
  - FOR EACH STATEMENT (una sola volta per tutto il comando che ha attivato il trigger)
    - e viene eseguito comunque, anche se il comando che attiva il trigger in realtà non ha prodotto effetti

Tipi di trigger

	STATEMENT	ROW
BEFORE	**Trigger before statement:** il trigger è eseguito un'unica volta prima dell'esecuzione del comando che lo attiva	**Trigger before row:** il trigger è eseguito prima di modificare ogni tupla coinvolta dall'esecuzione del comando che attiva il trigger
AFTER	**Trigger after statement:** il trigger è eseguito un'unica volta dopo l'esecuzione del comando che lo attiva	**Trigger after row:** il trigger è eseguito dopo aver modificato ogni tupla coinvolta dall'esecuzione del comando che attiva il trigger

Valutazioni

- Row level vs statement level
  - conviene usare trigger row level se l'azione del trigger dipende dal valore della tupla modificata
  - conviene usare trigger statement level se l'azione del trigger è globale per tutte le tuple modificate
- Before vs after
  - conviene usare trigger before se l'azione del trigger determina
    - se il comando verrà effettivamente eseguito (si evita di eseguire il comando e di farne eventualmente il rollback)
    - oppure per derivare valori di colonne da utilizzare in un INSERT o un UPDATE

Clausola REFERENCING

- La clausola REFERENCING “implementa” le transition table
  - a livello di tabella e di tupla
  - Il default è ROW

- È necessario specificare gli alias se la **condizione** e/o l’**azione** si riferiscono alla tabella sulla quale il trigger è definito

REFERNCING OLD comeera, NEW comesara

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
    - TEC_0 -> TEC_1 -> ... -> TEC_n
  - 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

Trigger Oracle

- Trigger statement level

  CREATE [OR REPLACE] TRIGGER <nome>
  {BEFORE | AFTER | INSTEAD OF} <eventi>
  ON <tabella>
  <blocco PL/SQL>

- Trigger row level

  CREATE [OR REPLACE] TRIGGER <nome>
  {BEFORE | AFTER | INSTEAD OF} <eventi>
  ON <tabella>
  [REFERENCING <riferimenti>]
  FOR EACH ROW
  [WHEN (<condizione>)]
  <blocco PL/SQL>

- Definizioni

<evento> ::= INSERT | DELETE | UPDATE [OF <colonne>]
<riferimento> ::= OLD AS <nome vecchio valore> | NEW AS <nome nuovo valore>

Esempio

```
CREATE OR REPLACE TRIGGER ordina_merce
            AFTER UPDATE OF quantità ON prodotti P
            FOR EACH ROW
            WHEN (NEW.quantita<NEW.sottoscorta)
DECLARE
     x NUMBER;
BEGIN
    SELECT COUNT(*) INTO x FROM ordini O
    WHERE O.codice_prodotto=:NEW.codice_prodotto;
    IF x=0 THEN
         INSERT INTO ordini
         VALUES (:NEW.codice_prodotto, :NEW.scorta, SYSDATE);
    END IF;
END;
```

Esempio

```
CREATE OR REPLACE TRIGGER tanto_per_provare
        AFTER UPDATE ON professori
        FOR EACH ROW
        WHEN (NEW.nome like 'P%' OR NEW.nome like ‘p%’)
begin
   dbms_output.put_line(‘Prof. ‘              || :old.codice ||
                        'vecchio nome ‘  || :old.nome ||
                        'nuovo   nome ‘  || :new.nome);
end;
```

- Da notare:
  - Per ogni colonna della tabella esistono i due valori :OLD.<nome colonna> e :NEW.<nome colonna> che rappresentano il valore di quella colonna prima e dopo l’istruzione DML che fa scattare il trigger
  - Da notare il fatto che prima di OLD e NEW è richiesto il carattere due punti nel corpo del trigger, ma non nella clausola WHEN

Uso del REFERENCING

```
CREATE OR REPLACE TRIGGER tanto_per_provare
            AFTER UPDATE ON professori
            REFERENCING OLD as vecchio NEW as nuovo
            FOR EACH ROW
            WHEN (NEW.nome like 'P%' OR NEW.nome like 'p%' )
begin
   dbms_output.put_line('Prof. '              || :vechio.codice ||
                        'vecchio nome ' || :vecchio.nome ||
                        'nuovo   nome '  || :nuovo.nome);
end;
```

Risultato

- se si usa il comando

update professori set nome='pippo’;

- Si ottiene nel caso di una tabella con 8 tuple

Prof. 1 vecchio nome MARCO nuovo pippo
Prof. 2 vecchio nome GIOVANNI nuovo pippo
Prof. 3 vecchio nome MATTEO nuovo pippo
Prof. 4 vecchio nome LUCA nuovo pippo
Prof. 5 vecchio nome AMBROGIO nuovo pippo
Prof. 6 vecchio nome GENNARO nuovo pippo
Prof. 7 vecchio nome PASQUALE nuovo pippo
Prof. 8 vecchio nome VINCENZO nuovo pippo

Trigger su update

- Solo sui trigger di UPDATE è possibile specificare le colonne che devono essere sotto controllo
  - Il trigger scatterà solo se viene modificata una di queste colonne

```
CREATE OR REPLACE TRIGGER tanto_per_provare
        AFTER UPDATE OF nome, cognome ON professori
        FOR EACH ROW
        WHEN (NEW.nome like 'P%')
begin
   dbms_output.put_line('Prof. ' || :old.codice || 'vecchio nome ' || :old.nome ||
                        'nuovo   nome ' || :new.nome);
end;
```

- se si lancia il comando

  UPDATE professori SET cod_fisc=null;

  il trigger non scatta

Alcuni vincoli sui trigger

- Tutti i trigger:
  - Non possono eseguire i comandi COMMIT e ROLLBACK perché sono parte di operazioni più ampie
- I before trigger:
  - Possono modificare i valori assegnati alle variabili new, ma non possono contenere comandi SQL che provochino una modifica allo stato della base di dati
  - Non possono essere specificati su una vista
- I trigger di tipo after
  - Non possono essere specificati su una vista
- I trigger di tipo instead of
  - Possono essere specificati solo su una vista