

Estensioni procedurali di SQL

Laboratorio di basi di dati
Marco Frasca, Stefano Montanelli
Dipartimento di Informatica
Università degli Studi di Milano
<http://islab.di.unimi.it/bdlab1>
{marco.frasca,stefano.montanelli}@unimi.it



Obiettivo

- Disporre di un linguaggio di programmazione in grado di realizzare applicazioni procedurali all'interno di un DBMS
 - Esempio: data la matricola di uno studente, calcolare la media dei voti conseguiti nell'ultimo anno
 - Esempio: consentire ad utenti di categoria 'business' la visualizzazione di dati relativi alle vendite in area metropolitana
 - Esempio: calcolare il percorso più breve fra due città

Soluzione 1

- **Usare un linguaggio procedurale per DBMS**
- Si tratta di linguaggi di programmazione appositamente concepiti per essere utilizzati in ambiente DBMS (e.g., PL/SQL per Oracle, SQL PL per IBM, T-SQL per SQL Server, PL/pgSQL per PostgreSQL)
- Sono linguaggi di programmazione completi
- Sono linguaggi compatibili con il modello dei dati supportato dal DBMS

Soluzione 2

- **Usare un linguaggio ospite con precompilatore**
- Si tratta di linguaggi di programmazione tradizionali (tipicamente C e C++) utilizzati in ambiente DBMS (e.g., ECPG per PostgreSQL)
- Le istruzioni SQL sono “incapsulate” nel linguaggio ospite e identificate da un prefisso speciale (e.g., EXEC SQL)
- Prima della compilazione, il codice SQL viene passato a un precompilatore e sostituito con codice generato dal DBMS

Soluzione 3

- **Usare un linguaggio esterno al DBMS**
- Si tratta di linguaggi di programmazione tradizionali utilizzati all'esterno dell'ambiente DBMS (e.g., PHP, Java, C#)
- Ogni linguaggio dispone di apposite librerie di funzioni per interagire con i vari DBMS
- La libreria consente di contattare il DBMS, inviare comandi SQL e manipolare i risultati

Soluzioni a confronto

- **Linguaggio procedurale per DBMS**
 - permette di ottenere buone performance
 - è difficilmente portabile
 - richiede la conoscenza di linguaggi di programmazione specifici per i vari DBMS
- **Linguaggio ospite con precompilatore**
 - permette di ottenere buone performance
 - favorisce la portabilità del codice (ma con precisazione)
 - utilizza linguaggi di programmazione tradizionali
- **Linguaggio esterno al DBMS**
 - performance condizionate dall'overhead di connessione
 - favorisce la portabilità del codice
 - utilizza linguaggi di programmazione tradizionali

Linguaggi procedurali per DBMS

- I principali linguaggi procedurali per DBMS (e.g., PL/SQL e PL/pgSQL) recepiscono lo standard differenziandosi nella sintassi
- Ne consegue che il codice di un linguaggio è scarsamente portabile su un DBMS diverso da quello per cui il linguaggio è stato concepito
- In questo corso, vedremo:
 - esempi di embedded SQL con la sintassi prevista dallo standard
 - esempi di linguaggio procedurale per DBMS con PL/pgSQL
 - esempi di linguaggio esterno ai DBMS con il linguaggio PHP e le corrispondenti librerie di funzioni per l'interazione con il DBMS PostgreSQL

Embedded SQL - definizione

- Nel caso di linguaggi procedurali per DBMS e di linguaggi ospite con precompilatore (soluzioni 1 e 2), si parla di **embedded SQL** per indicare il fatto che le istruzioni SQL sono “incapsulate” all'interno di un linguaggio di programmazione
- Lo standard SQL-3 stabilisce
 - la sintassi dei comandi per l'incapsulamento di istruzioni SQL in un linguaggio ospite (embedded SQL)
 - le linee guida per la definizione di linguaggi procedurali in ambito DBMS

Embedded SQL - requisiti

- Per inserire statement SQL in un programma scritto in linguaggio ospite sono necessari
 - A. **Strumenti per descrivere lo stato dei comandi SQL**
 - B. **Strumenti per scambiare valori tra variabili di programma e SQL**

Embedded SQL – requisito A

- SQL possiede la variabile **SQLSTATE** che si riferisce al risultato dell'esecuzione di un comando e può essere utilizzata in fase di error handling:
 - $\text{SQLSTATE} = 0$ - esecuzione corretta
 - $\text{SQLSTATE} < 0$ - errore
 - $\text{SQLSTATE} = 100$ - risultato vuoto
- La variabile **SQLCODE** utilizzata in alcuni DBMS è un alias (deprecato) di **SQLSTATE**

Embedded SQL – requisito B

- Sono disponibili due meccanismi:
 - *Risultato costituito al più da una singola tupla*: si aggiunge alla selezione SQL la clausola **INTO** per trasferire il risultato nelle variabili del programma
 - *Risultato potenzialmente costituito da più di una tupla - recordset*: si utilizza un **cursore** per scorrere il risultato tupla per tupla. Si può definire un cursore come un puntatore ad una lista di elementi da scorrere in maniera sequenziale

Embedded SQL – dichiarare un cursore

- Un cursore viene dichiarato associandolo ad un comando SELECT:
*DECLARE <nomecursore> CURSOR
FOR <comandoselect>*
- L'esecuzione del comando avviene però solo all'esplicita apertura del cursore:
OPEN <nomecursore>

Embedded SQL – scorrere un cursore

- L'apertura del cursore posiziona il puntatore all'inizio del recordset restituito con il risultato del comando select
- Mediante un'istruzione di caricamento (normalmente associata ad un ciclo), si provoca l'avanzamento del cursore e la copia dei valori della tupla corrente nelle variabili di programma:

FETCH <nomecursore> INTO <varprogramma>

- Al termine del ciclo, il cursore si posiziona dopo l'ultima tupla del recordset e SQLSTATE = 02000 (SQLCODE = 100)
- E' quindi possibile disattivare il cursore:
CLOSE <nomecursore>

Linguaggi procedurali in PostgreSQL

- PostgreSQL supporta diverse estensioni procedurali di SQL
 - PL/pgSQL
 - PL/tcl
 - PL/python
 - PL/perl
- E' necessario che il linguaggio sia installato affinché esso sia disponibile all'interno di una base di dati del DBMS
- In una base di dati, è possibile installare (e utilizzare) più di un linguaggio

PL/pgSQL – installazione

- Per installare un linguaggio, aprire il prompt dei comandi, spostarsi nella cartella contenente i comandi eseguibili di PostgreSQL ed invocare
`createlang <nomelinguaggio> <nomedb>`
 - nomelinguaggio è il linguaggio da installare
 - nomedb è il database per il quale il nomelinguaggio verrà reso disponibile
- Seguendo le impostazioni predefinite per la configurazione di PostgreSQL, PL/pgSQL è installato automaticamente per ogni database del DBMS

PL/pgSQL – dichiarazione di funzioni

```
CREATE [OR REPLACE] FUNCTION
    <nomefunzione>([<param1, ..., paramn>])
    [RETURNS <tipodatorestituito>]
AS $$
    <corpofunzione>;
$$ LANGUAGE plpgsql
```

- Se la clausola *RETURNS* è specificata, l'istruzione *RETURN <espressione>* in *corpofunzione* viene utilizzata per terminare la funzione e restituire il risultato al chiamante

PL/pgSQL – uso delle funzioni

- Per invocare una funzione, utilizzare
*SELECT * FROM <nomefunzione>;*
- In psql, il codice di una funzione disponibile in un database è visualizzabile mediante il comando
\df [+] [nomefunzione]
- Per eliminare una funzione, utilizzare
DROP FUNCTION <nomefunzione>

PL/pgSQL – passaggio di parametri

- I parametri di una funzione sono specificati posizionalmente indicando il tipo di dato associato
(<tipoparam₁, ..., tipoparam_n>)
- I parametri sono riferiti tramite gli alias \$1...\$n
- In alternativa, è disponibile una sintassi per specificare alias con nome definito dall'utente
(<alias₁ tipoparam₁, ..., alias_n tipoparam_n>)

Struttura di una funzione PL/pgSQL

- Le procedure in PL/pgSQL presentano la seguente struttura a blocchi:

```
[DECLARE]
  /* dichiarazione variabili, tipi e strutture dati */
BEGIN
  /* istruzioni di flusso e comandi SQL */
  [ { EXCEPTION WHEN } ]
  /* procedure di handling delle eccezioni sollevate
  (gestione degli errori) */
END;
```

Dichiarazione di variabili

- Le variabili di programma devono essere dichiarate nella sezione DECLARE
 - ad eccezione delle variabili definite nel costrutto FOR

- Sintassi:

```
<nomevariabile> [CONSTANT] <tipodidato>
  [NOT NULL]
  [{ DEFAULT | := } espressione];
```

- L'opzione CONSTANT stabilisce che il valore della variabile non possa essere modificato a run-time
- L'opzione NOT NULL stabilisce che la variabile non possa assumere il valore nullo a run-time

Tipi di dato

- I tipi di dato previsti da PL/pgSQL sono i tipi di dato previsti da SQL-3
- E' possibile dichiarare variabili che siano dello stesso tipo assegnato ad un attributo

<nomevar> <nometab.nomeattr>%TYPE;

- E' possibile dichiarare variabili che siano del tipo di una tupla di una specifica tabella

<nomevar> <nometab>%ROWTYPE;

Commenti e istruzione di assegnamento

- Ogni istruzione termina con il carattere punto e virgola (;)
- Per la definizione di commenti è prevista una doppia sintassi:
 - riga singola: la riga inizia con un tratto doppio (--)
 - multi-riga: il commento è delimitato fra /* e */
- L'istruzione di assegnamento prevede la seguente sintassi

<identificatore> := <espressione>

Istruzione di controllo condizionale

Sintassi:

- *IF <condizione> THEN <istr₁;...; istr_n> END IF;*
- *IF <condizione> THEN <istr₁;...; istr_n>
ELSE <istr'₁;...; istr'_n> END IF;*
- *IF <condizione1> THEN <istr₁;...; istr_n>
ELSIF <condizione2> THEN <istr'₁;...; istr'_n>
ELSE <istr''₁;...; istr''_n> END IF;*
- Le strutture di controllo condizionali possono essere nidificate

Istruzione di controllo iterativa - 1

Istruzione LOOP. Sintassi:

*LOOP
 <istr₁;...; istr_n>
END LOOP;*

- La condizione di uscita dal loop viene impostata internamente all'iterazione
 - mediante un'istruzione condizionale combinata con l'istruzione EXIT
 - mediante un'istruzione di uscita condizionale
EXIT WHEN <condizione>;

Istruzione di controllo iterativa - 2

Istruzione WHILE. Sintassi:

```
WHILE <condizione> LOOP  
    <istr1;...; istrn>  
END LOOP;
```

Istruzione di controllo iterativa - 3

Istruzione FOR. Sintassi:

```
FOR <variabile> IN [REVERSE] <exprinizio>...  
    <exprfine> [BY <exprpasso>] LOOP  
    <istr1;...; istrn>  
END LOOP;
```

- <variabile> è definita automaticamente di tipo integer con scope limitato all'istruzione FOR
- L'opzione REVERSE permette iterazioni con indici decrescenti

Uso di query SQL

- All'interno di una funzione, è possibile invocare l'esecuzione di comandi SQL (i.e., SELECT)
- Come già detto, in embedded SQL, è necessario prevedere un meccanismo per passare il risultato di una query alle variabili del programma (i.e., clausola INTO)

Risultato di una query SQL

- Quando una funzione invoca l'esecuzione di una query (comando SQL SELECT) e il risultato della query contiene più di una tupla è necessario ricorrere all'uso di un **cursore**
- Per la gestione di cursori in PL/pgSQL, fare riferimento allo standard SQL-3 (discusso in precedenza nel contesto di embedded SQL)