Appunti Basi di Dati

Luca Seggiani

2 Maggio 2024

1 SQL Procedurale

Introduciamo adesso il PL/SQL, un linguaggio di programmazione procedurale che estende l'SQL.

Stored procedure

Le stored procedure (procedure "stoccate") sono dei programmi dichiarativo-procedurali che possono essere memorizzati nel DBMS e invocati proprio come funzioni nei linguaggi di programmazione tradizionali, restituendo valori. Procedure possono essere dichiarate e poi chiamate all'interno di query, o altre procedure.

Interfacce con stored procedure

Il meccanismo delle stored procedure può essere usato per implementare delle interfacce: all'utente non si mette a disposizione l'SQL, ma soltanto le procedure definite su quel DBMS. In questo modo l'utente non ha mai effettivamente accesso al database stesso, ma può comunque interagirci con procedure già definite. Questo sistema assicura il mascheramento dei dati e del codice, aumentando la sicurezza e la protezione da eventuali attacchi. Meccanismi di questo tipo sono alla base di architetture **multilivello** (*multi-tier*) usati ad esempio nei siti internet: l'uso di procedure predefinite per l'interazione fra server e database, e quindi fra utente e server, permette di aumentare la sicurezza del sistema. L'invocazione di procedure richiede, da parte del chiamante, il possesso di un particolare **grant** ("permesso"), che garantisce l'accesso del suddetto a tali procedure. E' ad esempio possibile che un'applicazione abbia accesso alle procedure ma non alle tabelle del database. Vediamo un'esempio. Vogliamo scrivere una stored procedure che restituisca tuttte le specializzazioni mediche offerte dalla clinica. Dovremo usare la seguente sintassi:

```
DROP PROCEDURE IF EXISTS mostra_specializzazioni
DELIMITER $$
```

```
CREATE PROCEDURE mostra_specializzazioni()
  BEGIN
    SELECT DISTINCT Specializzazione
    FROM Medico
  END $$
DELIMITER;
```

Il pasticcio di delimitatori serve a far ignorare all'SQL l'uso del ; all'interno della procedura, che altrimenti significherebbe "compila quanto hai letto fino ad ora". Alla fine della dichiarazione reimpostiamo il delimitatore a quello di default.
Potremo adesso chiamare la procedure con:

```
CALL mostra_specializzazioni()
```

Variabili locali

All'interne delle stored procedure è possibile memorizzante informazioni intermedie di ausilio, attraverso variabili locali. La dichiarazione di variabili locali va fatta contestualmente alla dichiarazione della procedura (cioè subito) con la sintassi:

```
DECLARE nome_variabile tipo(size) DEFAULT valore_default;
```

La tipizzazione è forte, e sono presenti i classici tipi (int, double, char, date...). Il valore size rappresenta la dimensione della variabile: ad esempio, per il tipo char, potremo specificare char(50) per indicare una stringa di 50 caratteri. Il tipo varchar è un tipo particolare che può variare la sua dimensione, occupando *al massimo* il valore fornito alla dichiarazione. Si può fare assegnamento sulle variabili attraverso la parola chiave SET:

```
DECLARE min_visite_mensili INT DEFAULT 0;

% il corpo della procedura...

SET min_visite_mensili = 20;
```

Facciamo un'esempio: supponiamo di essere nel body di una stored procedure e creare una variabile contenente il numero minimo di visite effettuate questo mese, avremo:

```
DECLARE visite_mese_attuale INT DEFAULT 0;

% il corpo della procedura...

SET visite_mese_attuale = (
  SELECT COUNT(*)
  FROM Visita V
  WHERE MONTH(V.Data) = MONTH(CURRENT_DATE)
    AND YEAR(V.Data) = YEAR(CURRENT_DATE)
)
```

oppure, analogamente:

```
DECLARE visite_mese_attuale INT DEFAULT 0;

% il corpo della procedura...

SELECT COUNT(*)
FROM Visita V
WHERE MONTH(V.Data) = MONTH(CURRENT_DATE)
  AND YEAR(V.Data) = YEAR(CURRENT_DATE)
INTO visite_mese_attuale
```

Notiamo l'errore (di impedenza) in cui potremmo incorrere facendo qualcosa del tipo:

```
DECLARE visite_mese_attuale VARCHAR(255)
SELECT * INTO visite_mese_attuale
FROM Visita V
WHERE MONTH(V.Data) = MONTH(CURRENT_DATE)
  AND YEAR(V.Data) = YEAR(CURRENT_DATE)
```

Non possiamo infatti immagazzinare (anzi, "collassare") un'intero record in una sola variabile di tipo VARCHAR.

Variabili user-defined

Le variabili user-defined sono variabili definite dall'utente senza necessità di dichiarazione, che hanno tempo di vita uguale alla durata della connessione dell'utente al server. A differenza delle variabili locali, le variabili user-defined hanno tipizzazione debole e dinamica: potremo memorizzarvici qualsiasi tipo di valore, e anche valori di tipi diversi in momenti diversi. Non possono contenere result set, ma solo valori scalari. Si dichiarano con @.

Parametri di una stored procedure

Una stored procedure MySQL accetta parametri di tipo:

- **Ingresso**
  Un parametro di ingresso può essere letto, ma non modificato. Equivale al passaggio per valore del C++. Si dichiara con la parola chiave IN. Ad esempio, poniamo di voler scrivere una stored procedure che stampi la parcella media di una specializzazione specificata come parametro:

  ```
  DROP PROCEDURE IF EXISTS parcella_media_spec;
  DELIMITER $$
  CREATE PROCEDURE parcella_media_spec(IN _specializzazione
      VARCHAR(100))
  % resto del codice...
  DELIMITER;
  ```

  a questo punto potremo chiamare la procedura con:

```
CALL parcella_media_spec('Ortopedia');
```

- **Uscita**
  Un parametro di uscita può essere modificato dalla procedura, e viene fornito in fase di chiamata dal chiamante. Nella chiamata si possono usare, nei parametri di uscita, variabili user defined (@variabile).
  Facciamo un'esempio: vogliamo scrivere una stored porcedure che restituisca *come parametro di uscita* il numero di pazienti visitati da medici di una data specializzazione, ricevuta come parametro di ingresso:

```
DROP PROCEDURE IF EXISTS parcella_media_spec;
DELIMITER $$
CREATE PROCEDURE parcella_media_spec(IN _specializzazione
    VARCHAR(100), OUT totale_pazienti INT)
% resto del codice...
DELIMITER;
```

  come prima, potremo adesso chiamare la procedura con la sintassi:

```
CALL tot_pazienti_visitati_spec('Neurologia',
   @quantiPazienti);
SELECT @quantiPazienti
```

  dove @quantiPazienti è una variabile user-defined.
  Si noti che si possono avere più parametri di uscita in una procedura SQL.

- **Ingresso-uscita**
  Non vengono trattate da questo corso. In altre parole t'attacchi.

Istruzioni condizionali

Le espressioni condizionali permettono di esprimere condizioni, modificando il flusso di esecuzioni. Possono contenere letterali, variabili e funzioni. Nell'SQL si riducono alle due parole chiave: IF e CASE.

- **Istruzione IF**
  L'IF è analogo a quello di altri linguaggi, con l'inclusione del THEN:

```
IF if_condition THEN
  -- blocco di istruzioni
ELSEIF elseif_1_condition THEN
 ...
ELSEIF elseif_N_condition THEN
ELSE
  -- blocco di else
END IF;

```

- **Istruzione CASE**
  Vediamo il CASE:

```
CASE
WHEN condition_1 THEN
  -- blocco 1
 ...
WHEN condition_n THEN
  --blocco n
END CASE
```

Istruzioni iterative

Le struzioni iterative permettono di ripetere blocchi di codice. L'SQL fornisce il WHILE, il REPAT, e il LOOP.

- **Istruzione WHILE**
  Corrisponde al semplice while.

```
WHILE condition DO
  -- blocco istruzioni
END WHIILE
```

- **Istruzione REPEAT**

```
REPEAT
  -- blocco istruzioni
UNTIL condition
END REPEAT
```

- **Istruzione LOOP**
  Definisce un LOOP, che andrà chiuso con un'istruzione LEAVE.

```
loop_label: LOOP
  -- blocco di istruzioni, check di condizioni
END LOOP
```

Istruzioni di salto

Le istruzioni di salto permettono di interrompere cicli o passare a iterazioni successive. Nell'SQL sono rispettivamente la LEAVE e ITERATE.

Cursore

Un cursore scorre i record su un result set, solo in avanti, per effettuare delle azioni all'interno di istruzioni iterative. Vediamo la sintessi:

```
DECLARE NomeCursore CURSOR FOR
SQL query;
```

I cursori si possono dichiarare immediatamente dopo la dichiarazione di tutte le variabili, contestualmente alla dichiarazione della procedura. Su un cursore valgono le seguenti operazioni:

```
OPEN NomeCursore;
FETCH NomeCursore INTO ListaVariabili;
CLOSE NomeCursore;
```

La prima e l'ultima operazione sono banali. Il FETCH si limita a restituire il prossimo record e avanzare il cursore.

Handler

Gli handler sono gestori di situazioni, utili per eseguire codice quando l'istruzione fetch porta il cursore in fondo alla tabella. Possono essere definiti dopo le dichiarazioni di variabili e cursori, sempre contestualmente alla dichiarazione della procedura. Ad esempio, esiste il NOT FOUND HANDLER:

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET finito = 1;
```

Questo handler viene eseguito quando si arriva a "fine corsa", ovvero in fondo alla tabella che si stava scorrendo. La parola chiave CONTINUE rappresenta il fatto che l'handler non interrompe l'esecuzione, a differenza di un handler EXIT.

Ciclo di fetch

Il meccanismo del FETCH e degli HANDLER ci permette di stabilire un ciclo di fetch:

```
DECLARE cur CURSOR FOR tabella
DECLARE CONTINUE HANDLER FOR NOT FOUND
SET finito = 1;

scan: LOOP
FETCH cur -- prelieva il record
IF finito = 0
  -- processo il record
ELSE LEAVE scan
END LOOP scan;
```