7 - Sviluppo di applicazioni per Basi di Dati

Nell'utilizzare una base di dati, il dialogo diretto con l'interprete SQL è riservato a pochi utenti esperti. L'accesso di gran lunga più tipico a una base di dati avviene attraverso applicazioni integrate nel sistema informativo, le quali forniscono agli utenti un'interfaccia semplificata che favorisce l'interazione.
SQL supporta le applicazioni in due modi:

- Incrementando le funzionalità del DBMS per mezzo della definizione di **procedure** e **trigger** (visti nel capitolo 4 - SQL, più precisamente nelle caratteristiche evolute);
- Integrando comandi SQL con istruzioni di un linguaggio di programmazione (procedurale/oggetti) utilizzando SQL Embedded per i linguaggi datati, oppure il Cal Level Interface per linguaggi di programmazione più recenti

SQL Embedded

SQL Embedded prevede di introdurre direttamente nel programma sorgente scritto nel linguaggio di alto livello le istruzioni SQL, distinguendole dalle normali istruzioni tramite un opportuno separatore. Lo standard SQL prevede che il codice SQL sia preceduto dalla stringa exec sql e termini con il carattere ';'. Dal punto di vista dell'implementazione, è necessario far precedere la compilazione del linguaggio di alto livello all'esecuzione di un preprocessore che riconosca le istruzioni SQL e sostituisce a esse un insieme di chiamate ai servizi del DBMS, tramite una libreria specifica per ogni sistema.

Tutte le variabili usate per scambiare dati fra il programma e il DBMS sono dichiarate in un blocco di istruzioni compreso tra i due comandi:

$$\text{exec sql begin declare section}$$
$$\text{exec sql end declare section}$$

Ogni volta che si dichiara una variabile (x) in questo blocco nel linguaggio ospite, viene creata una variabile copia identica in SQL il cui nome è lo stesso ma preceduto dal carattere ' : '$(: x)$. Ogni modifica effettuata x viene effettuata anche se $: x$ e viceversa.

Per lavorare con un database da linguaggio ospite:

1. Si stabilisce una connessione con il sistema specificando il database su sui lavorare tramite il comando seguente:

   $$\text{CONNECTED TO} < IdUtente > \text{IDENTIFIED BY} < password > \text{USING} < Database >$$

2. Il preprocessore introduce implicitamente la dichiarazione di una struttura SQLCA (SQL communication area) per gestire la comunicazione tra programma e DBMS
3. Si può accedere al campo SQLCODE della struttura SQLCA, il cui valore è un intero che codifica l'effetto dell'ultima operazione SQL effettuata. Se il valore è uguale a zero indica

la corretta esecuzione del comando, se è diverso da zero indica che si è verificata una anomalia ed il comando non è andato a buon fine.

4. In caso di query scalary (unico risultato) il comando SELECT è esteso con la clausola

$$\text{into} < Variabile > \{, < Variabile >\}$$

per assegnare a delle variabili del programma il valore degli attributi dell'unica tupla del risultato. Se il comando SELECT può assegnare ad una variabile il valore nullo (non previsto nel linguaggio ospite) la variabile va dichiarata come:

$$: Variabile \ \mathrm{INDICATOR} < \mathrm{IndVariabile} >$$

Se dopo l'esecuzione $IndVariabile$ ha valore minore di zero allora $Variabile$ ha valore significativo (quindi non null).

Un esempio di implementazione di SQL Embedded tramite C è questo:

```c
#include <stdlib.h>
#include <stdio.h>

main()
{
    exec sql begin declare section;
        char *NomeDip = "Manutenzione";
        char *CittaDip = "Pisa";
        int NumeroDip = 20;
    exec sql end declare section;

    exec sql connect to utente@librobd;

    if (sqlca.sqlcode != 0) {
        printf("Connessione al DB non riuscita\n");
    }
    else {
        exec sql insert into Dipartimento
            values(:NomeDip, :CittaDip, :NumeroDip);

        exec sql disconnect all;
    }
}
```

Un importante problema che caratterizza l'integrazione tra SQL e i normali linguaggi di programmazione è il cosiddetto **conflitto di impedenza**. I linguaggi di programmazione

accedono agli elementi di una tabella scandendone le righe una a una (tuple-oriented). Al contrario SQL è un linguaggio di tipo set-oriented, che opera su intere tabelle e restituisce come risultato di un'interrogazione un'intera tabella. Le soluzioni a questo problema si ottengono con l'utilizzo dei **cursori** e l'utilizzo di linguaggi con costruttori di tipo in grado di gestire una struttura del tipo "insieme di righe" (Call Level Interface).

Cursori

Un cursore è una variabile speciale che permette ad un programma di accedere alle righe di una tabella una alla volta; il cursore viene definito su una generica interrogazione, con la seguente sintassi:$$
\begin{aligned}
&\text{DECLARE}\ NomeCursore \ [\ \text{SCROLL} \ ] \ \text{CURSOR FOR} \ SelectSQL \
&\quad [\ \text{FOR} \langle \text{READ ONLY} \ | \ \text{UPDATE} [\ \text{OF} Attributo \ {\ , Attributo \ } \ ] \ \rangle \ ]
\end{aligned}

$Al momento della sua dichiarazione, il cursore si riferisce a una struttura vuota. L'avvaloramento del c

\begin{aligned}
&\text{DELETE FROM } NomeTabella \ \text{WHERE CURRENT OF } NomeCursore
\end{aligned}

Infine, esiste il comando CLOSE che comunica al sistema che il risultato dell'interrogazione non s

\begin{aligned}
&\text{PREPARE } :comando \ \text{FROM } :istruzioneSQL \
&\text{DECLARE } Cursore \ \text{CURSOR FOR } :comando \
&\text{OPEN } Cursore \ \text{USING } :nome1
\end{aligned}

Quando un'istruzione SQL che era stata preparata non serve più, è possibile rilasciare la memoria occupata

\begin{aligned}
&\text{SET TRANSACTION ISOLATION LEVEL [READ UNCOMMITTED | READ} \
&\quad \text{COMMITTED | REPEATABLE READ | SERIALIZABLE]}
\end{aligned}

Analizziamo i vari livelli di isolamento per grado di concorrenza decrescente: 1. **Read uncommitted (deg

\begin{aligned}
&\text{GRANT} \langle \ Privileges \ | \ \text{ALL PRIVILEGES} \ \rangle \ \text{ON }
Resource \ \text{TO } Users \ [\ \text{WITH GRANT OPTIONS} \ ]
\end{aligned}

$GRANT OPTIONS$ *specifica se il privilegio può essere trasmesso ad altri utenti. Chiaramente i privi*

$$
\begin{aligned}
&\text{REVOKE } \text{Privileges } \text{ON } \text{Resource } \text{FROM } \text{Users } [\text{RESTRICT} \mid \text{CASCADE}]
\end{aligned}
$$

$La gestione delle autorizzazioni deve nascondere gli elementi cui un utente non può accedere, senza sospe$