

SQL nei Linguaggi di Programmazione

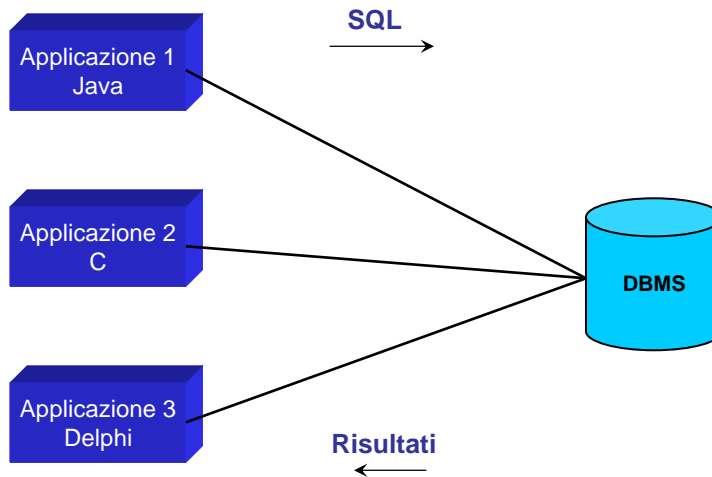
2007

Gianluigi Ciocca

SQL e Applicazioni (1)

- Interrogazione e manipolazione dei DB non basta
 - Ambiente di gestione più user-friendly o potente
 - Creazione di applicazioni complesse
 - Interazione con l'utente
 - Integrazione di un DBMS in applicazioni esistenti

SQL e Applicazioni (2)



SQL nei Linguaggi di Programmazione

3

I Problemi (1)

- Conflitto di impedenza (“disaccoppiamento di impedenza”) fra base di dati e linguaggio
 - Linguaggi: operazioni su singole variabili o oggetti
 - SQL: operazioni su relazioni (insiemi di ennuple)
- Tipi di base:
 - linguaggi: numeri, stringhe, booleani, ...
 - SQL: CHAR, VARCHAR, DATE, ...

SQL nei Linguaggi di Programmazione

4

I Problemi (2)

- Tipi “strutturati” disponibili:
 - linguaggio: dipende dal paradigma
 - SQL: relazioni e ennuple
- Accesso ai dati e correlazione:
 - linguaggio: dipende dal paradigma e dai tipi disponibili; ad esempio scansione di liste o “navigazione” tra oggetti

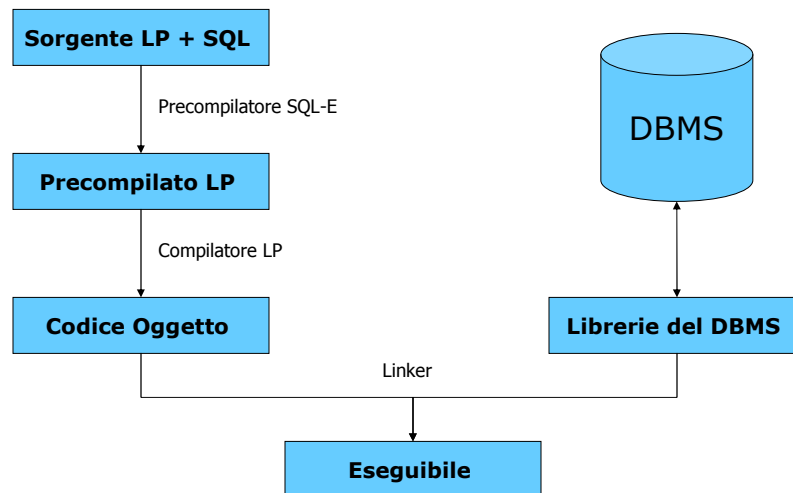
Tecniche principali

- SQL incapsulato (“Embedded SQL”)
 - sviluppata sin dagli anni '70
 - “SQL statico”
 - “SQL dinamico” (Dynamic SQL)
- Call Level Interface (CLI)
 - più recente
 - SQL/CLI, ODBC, JDBC
- Stored Procedures

Embedded SQL (1)

- Le istruzioni SQL sono “immerse” all’interno del programma ospite.
- Le istruzioni SQL sono separate da quelle del linguaggio di programmazione usato.
 - Devono essere “tradotte” nel linguaggio ospite
 - Necessario una fase di precompilazione per la traduzione
 - Sostituzione istruzioni SQL con chiamate specifiche al DBMS

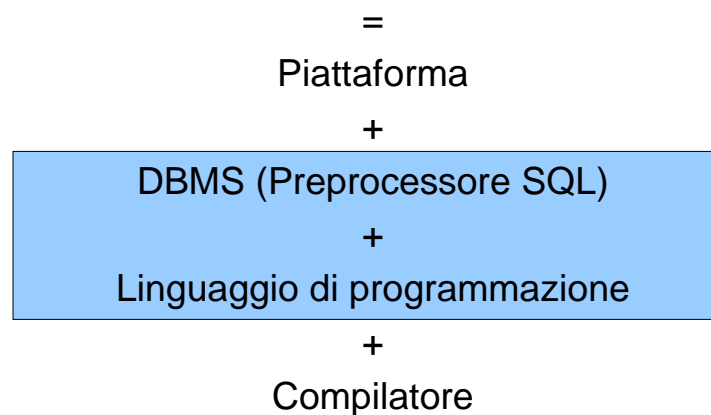
Embedded SQL (2)



Embedded SQL (3)

- Istruzioni SQL delimitate
 - Iniziano con “exec sql”
 - Terminano con “;” o “end-exec”
- Il preprocessore SQL riconosce le istruzioni SQL dai delimitatori, le decodifica e le traduce in istruzioni del linguaggio ospite con chiamate a librerie di funzioni del DBMS usato

Embedded SQL (4)



Embedded SQL: Struttura (1)

```
#include<stdlib.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;
    }
}
```

ECPG: Preprocessore SQL per il C e PostgreSQL

Embedded SQL: Struttura (2)

```
#include<stdlib.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;
    }
}
```

Definizione delle variabili condivise tra C e SQL
Accessibili da SQL con “:NomeVariabile”

Embedded SQL: Struttura (3)

```
#include<stdlib.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;
    }
}
```

Attivazione della connessione al DBMS

Embedded SQL: Struttura (4)

```
#include<stdlib.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;
    }
}
```

Controllo sulla riuscita di un comando

sqlca: struttura predefinita di comunicazione con DBMS

Embedded SQL: Struttura (5)

```
#include<stdlib.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;
    }
}
```

Esecuzione comando SQL

Embedded SQL: Struttura (6)

```
#include<stdlib.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;
    }
}
```

Disconnessione

Embedded SQL: SQLCA (1)

- SQLCA : SQL Communication Area
 - Definita dallo standard SQL
 - Predefinita dal preprocessore o da includere
 - (Es. In Oracle) : “exec sql include SQLCA;”
 - SQLCA.SQLCODE (intero)
 - 0: successo
 - <0: errore
 - >0: fine dati

Embedded SQL: SQLCA (2)

- SQLCA : SQL Communication Area
 - SQLCA.SQLSTATE (stringa)
 - 00000: assenza di errori o eccezioni
 - 00001-99999: errori ed eccezioni
 - Codifica standardizzata per tutti i fornitori e piattaforme SQL.
 - I valori di SQLCODE dipendono dalla piattaforma.

Embedded SQL: Traduzione (1)

```
int main() {
    exec sql connect to universita user pguser identified by pguser;
    exec sql create table studente(matricola integer primary key,
        nome varchar(20), annodicorso integer);
    exec sql disconnect;
    return sqlca.sqlcode;
}
```

```
/* These include files are added by the preprocessor */
#include <ecpgtype.h>
#include <ecpglib.h>
#include <ecpgerrno.h>
#include <sqlca.h>
int main() {
    ECPGconnect(__LINE__,"universita","pguser","pguser",NULL,0);
    ECPGdo(__LINE__, NULL, "create table studente ( matricola integer primary key,
        nome varchar ( 20 ),annodicorso integer )", ECPGt_EOIT, ECPGt_EORT);
    ECPGdisconnect(__LINE__, "CURRENT");
    return sqlca.sqlcode;
}
```

SQL nei Linguaggi di Programmazione

19

Embedded SQL: Traduzione (2)

```
int main() {
    exec sql connect to universita user pguser identified by pguser;
    exec sql create table studente(matricola integer primary key,
        nome varchar(20), annodicorso integer);
    exec sql disconnect;
    return sqlca.sqlcode;
}
```

```
/* These include files are added by the preprocessor */
#include <ecpgtype.h>
#include <ecpglib.h>
#include <ecpgerrno.h>
#include <sqlca.h>
int main() {
    ECPGconnect(__LINE__,"universita","pguser","pguser",NULL,0);
    ECPGdo(__LINE__, NULL, "create table studente ( matricola integer primary key,
        nome varchar ( 20 ),annodicorso integer )", ECPGt_EOIT, ECPGt_EORT);
    ECPGdisconnect(__LINE__, "CURRENT");
    return sqlca.sqlcode;
}
```

SQL nei Linguaggi di Programmazione

20

Embedded SQL: Traduzione (3)

```
int main() {
    exec sql connect to universita user pguser identified by pguser;
    exec sql create table studente(matricola integer primary key,
        nome varchar(20), annodicorso integer);
    exec sql disconnect;
    return sqlca.sqlcode;
}
```

```
/* These include files are added by the preprocessor */
#include <ecpgtype.h>
#include <ecpglib.h>
#include <ecpgerrno.h>
#include <sqlca.h>
int main() {
    ECPGconnect(__LINE__, "universita", "pguser", "pguser", NULL, 0);
    ECPGdo(__LINE__, NULL, "create table studente ( matricola integer primary key,
        nome varchar ( 20 ),annodicorso integer )", ECPGt_EOIT, ECPGt_EORT);
    ECPGdisconnect(__LINE__, "CURRENT");
    return sqlca.sqlcode;
}
```

SQL nei Linguaggi di Programmazione

21

Embedded SQL: Traduzione (4)

```
int main() {
    exec sql connect to universita user pguser identified by pguser;
    exec sql create table studente(matricola integer primary key,
        nome varchar(20), annodicorso integer);
    exec sql disconnect;
    return sqlca.sqlcode;
}
```

```
/* These include files are added by the preprocessor */
#include <ecpgtype.h>
#include <ecpglib.h>
#include <ecpgerrno.h>
#include <sqlca.h>
int main() {
    ECPGconnect(__LINE__, "universita", "pguser", "pguser", NULL, 0);
    ECPGdo(__LINE__, NULL, "create table studente ( matricola integer primary key,
        nome varchar ( 20 ),annodicorso integer )", ECPGt_EOIT, ECPGt_EORT);
    ECPGdisconnect(__LINE__, "CURRENT");
    return sqlca.sqlcode;
}
```

SQL nei Linguaggi di Programmazione

22

Embedded SQL: SQLJ (1)

- Analogo di ECPG ma per Java



- Dichiarazioni tra “#sql { ... };”
- Sfrutta le API JDBC per la connessione ai DB
- Diversi livelli di integrazione
 - Part 0: Basic
 - Part 1: Chiamate esterne
 - Part 2: Oggetti Java come dati DB

Embedded SQL: SQLJ (2)

```
...
#sql {create table N_Course (cid char(6), Ctitle varchar2(10),Pcode char(4),
    primary key(cid))};

System.out.println ("Table N_Course is created");

String[] cn= new String[3];
cn[0]="IT1010";cn[1]="CS3421";cn[2]="IT3220";
String[] tc=new String[3];
tc[0]="Inf_Org";tc[1]="DBMS";tc[2]="Using DB";

for (int i=0;i<cn.length;i++)
{ #sql {insert into N_Course values (:cn[i],:(tc[i]),:(pc[i]))}; };

System.out.println ("Table N_Course is populated");

#sql {update N_Transcript T set T.mark=T.mark+1 where T.cid IN
    (select C.cid from N_Course C where C.Pcode='COSC')};

System.out.println ("Table N_Transcript is updated");

float Av_mark;

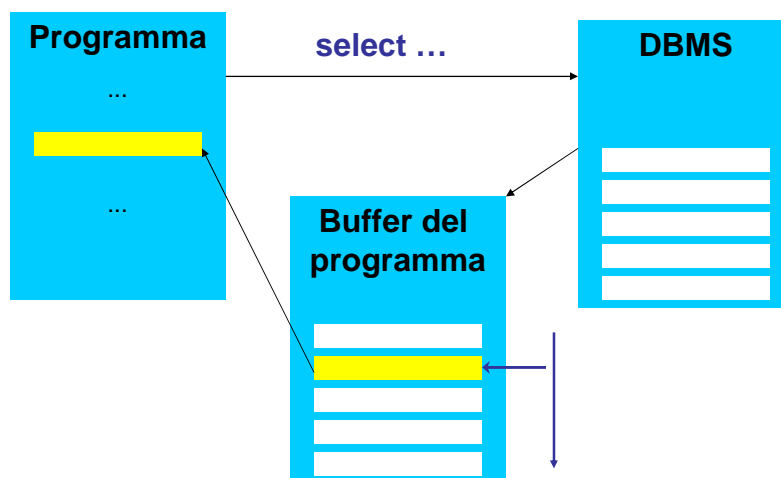
#sql {select avg(T.mark) into :Av_mark from N_Transcript T where T.cid=:cn[0]};

System.out.println ("Average mark for IT1010 is " + Av_mark);
```

Embedded SQL: I Risultati

- Come si accede ai risultati?
 - SELECT ritorna 0 o N t-ple ($N \geq 1$)
 - 0 o 1 t-ple → una struttura tipo record
 - N t-ple → ?
- Soluzione
 - Generale (per diversi linguaggi)
 - Accesso sequenziale alle t-ple

Embedded SQL: Cursori



Embedded SQL: Cursori (1)

- Si può dichiarare un cursore
 - su una relazione
 - Su un comando di interrogazione (che genera una relazione)
- Si può
 - **aprire** un cursore,
 - **prelevare** una tupla,
 - **muovere** il cursore fino a quando tutte le tuple siano state lette.

Embedded SQL: Cursori (2)

- Si può usare ORDER BY, nelle interrogazioni cui si accede tramite cursore, per controllare l'ordine in cui le tuple sono restituite
 - I campi nella clausola ORDER BY devono apparire anche nella clausola SELECT
 - La clausola ORDER BY, che ordina le tuple della risposta, è permessa *solo* nel contesto di un cursore
- Si può anche modificare/cancellare la tupla puntata dal cursore

Embedded SQL: Cursori (3)

- Dichiarazione di un cursore

```
exec sql declare NomeCursore cursor for Select ...
```

- Esecuzione di una interrogazione

```
exec sql open NomeCursore
```

- Utilizzo dei risultati

```
exec fetch NomeCursore into :var1, :var2, :var3,...
```

- Disabilitazione del cursore

```
exec sql close NomeCursore
```

- Accesso alla t-pla corrente (nella clausola where)

```
... where current of NomeCursore ...
```

SQL nei Linguaggi di Programmazione

29

Embedded SQL: Cursori (4)

```
...  
exec sql declare C cursor for  
select p.nome, p.reddito  
from persone as p  
where p.citta=:citta;
```

→ Associa al cursore una query

```
exec sql open C;
```

→ Esegue la query

```
exec sql fetch C into :nome, :reddito;
```

→ Recupera nome e reddito
e li mette nelle variabili

```
while (sqlca.sqlcode==0)
```

```
{  
  exec sql update persone  
  set reddito=reddito + :aumento  
  where current of C  
  exec sql fetch C into :nome, :reddito;  
}
```

→ Finchè ci sono dati aggiorna
il reddito "puntato" dal
cursore e recupera i dati
successivi

SQL nei Linguaggi di Programmazione

30

Embedded SQL: Cursori (5)

- Scrivere un programma che stampi, in ordine alfabetico, nome ed età di tutti i velisti che hanno almeno X anni di esperienza di vela:

```
Velisti(nome, cognome, eta, esperienza)
```

- Uso dei cursori
- Scelta di X random

Embedded SQL: Cursori (6)

```
EXEC SQL BEGIN DECLARE SECTION
char c_vnome[20]; int c_minesperienza; int c_età;
EXEC SQL END DECLARE SECTION

C_minesperienza = random();

EXEC SQL DECLARE vinfo CURSOR FOR
  SELECT V.vnome, V.età FROM Velisti V
  WHERE V.esperienza > :c_minesperienza
  ORDER BY V.vnome;

do{
  EXEC SQL FETCH vinfo INTO :c_vnome, :v_età;
  printf("%s ha %d anni\n", c_vnome, c_età);
} WHILE (SQLCA.SQLSTATE != 0x02000);

EXEC SQL CLOSE vinfo;
```


Embedded SQL: Cursori (7)

- Nel caso in cui il comando ritorna un solo risultato il cursore non serve:

```
...  
SELECT nome, cognome INTO :nome_studente, :cognome_studente  
FROM Studenti WHERE num_matricola = :matricola_studente  
...
```

Embedded SQL: Cursori (8)

- Nell'operazione di fetch è possibile definire la posizione di elaborazione:

FETCH [Posizione FROM] nomecursore INTO listafetch

– Posizione può essere

- next / prior
- first / last
- absolute / relative

Dynamic SQL (1)

- Static Embedded SQL
 - Nessuna possibilità di intervento sulla struttura della query
 - Error-Check: fatta dal precompilatore
 - Alte performance
- Dynamic Embedded SQL
 - Dinamico : query definite a run-time
 - La struttura della query può cambiare
 - Digitata dall'utente
 - Scelta interattiva dei campi da ritornare
 - ...
 - Error-Check : eccezioni a run-time

SQL nei Linguaggi di Programmazione

35

Dynamic SQL (2)

- Esecuzione immediata
 - Senza parametri
- ... EXECUTE IMMEDIATE :*sqlstatement*
- Esempio

```
Char sqlstring[256]="delete from impiegato where ID=123456"  
  
EXEC SQL EXECUTE IMMEDIATE :sqlstring
```

SQL nei Linguaggi di Programmazione

36

Dynamic SQL (3)

- Esecuzione differita (o ripetibile)
 - Con o senza parametri

```
... PREPARE cmd FROM sqlstatement  
... EXECUTE cmd [ INTO targetlist ] [ USING parameters ]  
... DEALLOCATE cmd
```

- Esempio

```
EXEC SQL PREPARE command FROM  
    "select nome from impiegato where id=:var";  
  
EXEC SQL EXECUTE command INTO :nome_imp USING :id_imp;  
  
EXEC SQL DEALLOCATE command;
```

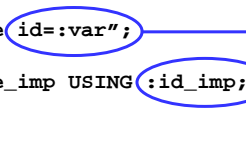
SQL nei Linguaggi di Programmazione

37

Dynamic SQL (4)

- Esempio

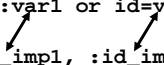
```
EXEC SQL PREPARE command FROM  
    "select nome from impiegato where id=:var";  
  
EXEC SQL EXECUTE command INTO :nome_imp USING :id_imp;  
  
EXEC SQL DEALLOCATE command;
```



Il placeholder ":var" viene sostituito con il valore in ":id_imp"

La regola è che ogni placeholder viene sostituito con il valore nella variabile in posizione corrispondente:

```
... where id=:var1 or id=:var2";  
... USING :id_imp1, :id_imp2;
```



SQL nei Linguaggi di Programmazione

38

Dynamic SQL (5)

- Aggiornamenti
 - Nessun problema se fatti con SQL dinamico
- Interrogazioni
 - Problemi sulla struttura dati che riceve i risultati
 - Struttura complessa e dinamica

Dynamic SQL (6)

- Dynamic SQL e cursori

```
...
strcpy(prepare_string,
       "SELECT tablename FROM tables WHERE tabschema=?");

EXEC SQL PREPARE s1 FROM :prepare_string;

EXEC SQL DECLARE c1 CURSOR FOR s1;

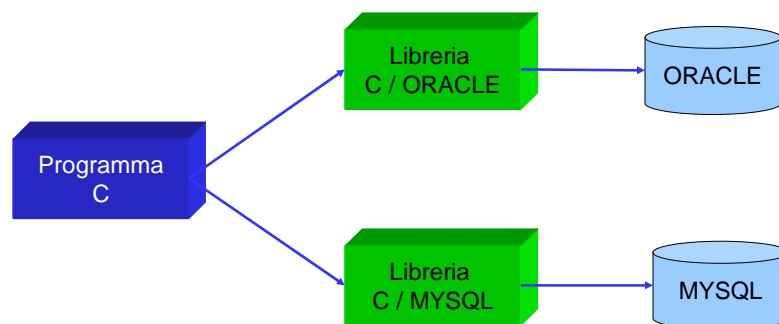
EXEC SQL OPEN c1 USING :variable;
```

- “?” è un placeholder generico

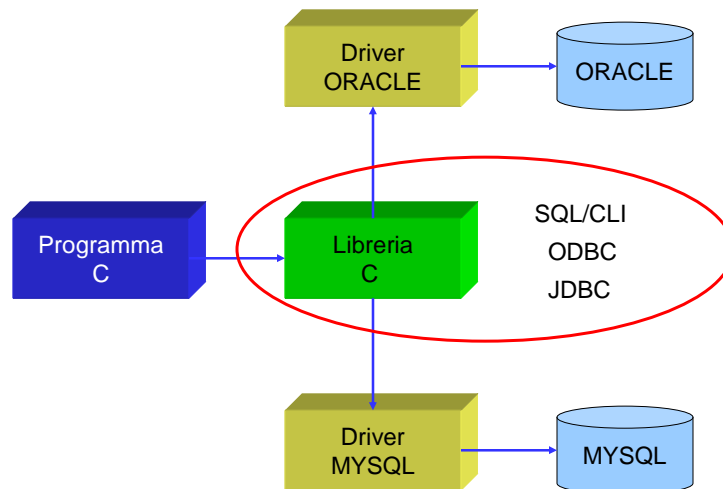
CLI : Call Level Interface

- Utilizzo di una libreria di funzioni
 - API: Application Programming Interface
- Approccio dinamico
 - Flessibile
- Verifica sintassi e controlli a run-time
 - Gestione degli errori mediante eccezioni
- Programmazione più complessa

CLI : Approccio 1



CLI : Approccio 2



SQL/CLI (1)

- Standard SQL per l'interfacciamento ad un DBMS in un programma.
 - Le istruzioni SQL vengono create dinamicamente e passate a chiamate di funzione.
 - Si tengono traccia delle interazioni tra programma e base di dati in strutture record
 - Record di ambiente
 - Record di connessione
 - Record di istruzione
 - Record di descrizione
 - Ai record si accede attraverso handle

SQL/CLI (2)

- Record d'ambiente
 - Tiene traccia di una o più connessioni alla base di dati
- Record di connessione
 - Tiene traccia delle informazioni relative ad una particolare connessione
- Record di istruzione
 - Tiene traccia delle informazioni necessarie per l'istruzione SQL da eseguire
- Record di descrizione
 - Tiene traccia delle informazioni sulle tuple o sui parametri

SQL/CLI (3)

```
#include "sqlcli.h"

SQLHSTMT stmt1; // Handle record istruzioni
SQLHDBC con1; // Handle record connessione
SQLHENV env1; // Handle record ambiente

SQLAllocHandle(SQL_HANDLE_ENV,0,&env1);
SQLAllocHandle(SQL_HANDLE_DBC,env1,&con1);
SQLConnect(con1,"dbms",SQL_NTS,"user",SQL_NTS,"pwd",SQL_NTS);
SQLAllocHandle(SQL_HANDLE_STMT,con1,&stmt1);

SQLPrepare(stmt1,"select COGNOME, STIPENDIO from IMPIEGATO
where SSN=?",SQL_NTS);

SQLBindParameter(stmt1,1,SQL_CHAR,&ssn,9,&length);
SQLExecute(stmt1);
SQLFetch(stmt1);

SQLGetData(stmt1,1,SQL_CHAR,&cognome,15,&length);
SQLGetData(stmt1,2,SQL_FLOAT,&stipendio,4,&length);
```

Inizializzazione e connessione

Preparazione della Query

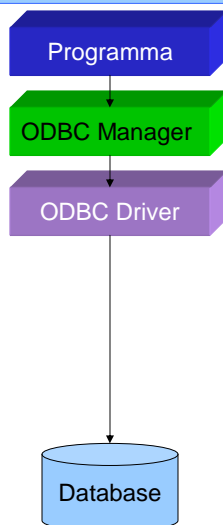
Settaggio Parametri ed Esecuzione

Recupero Risultati

ODBC

- Open Database Connection
 - Proposto da Microsoft e poi standard
 - Definito per funzionare in qualunque ambiente di programmazione e piattaforma.
 - I programmi inviano comandi sottoforma di stringhe.
 - La conoscenza delle API proprietarie è nascosta.
 - Genera (attraverso un driver) richieste che il sistema di database utilizzato è in grado di capire.
- 3 diversi tipi di implementazione

ODBC: Tipo 1

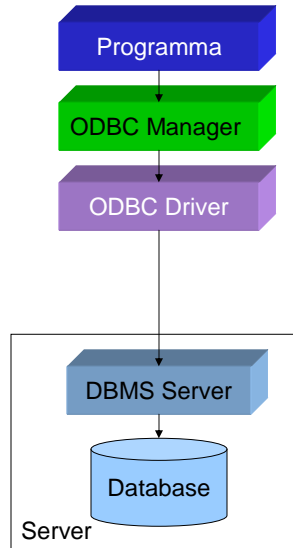


Il programma chiama una funzione ODBC.

ODBC Manager determina le azioni da prendere.

Il driver ODBC esegue direttamente la richiesta.

ODBC: Tipo 2



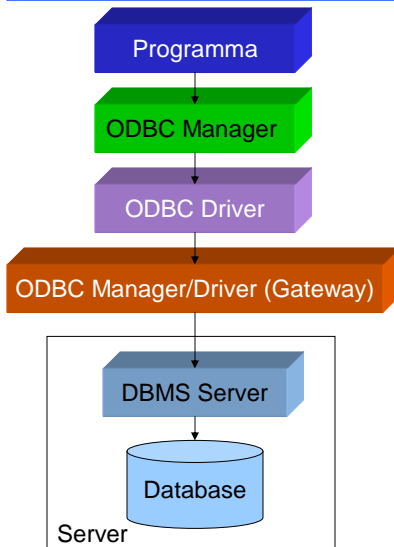
Il programma chiama una funzione ODBC.

ODBC Manager determina le azioni da prendere.

Il driver ODBC prepara la richiesta e la inoltra al DBMS.

Il DBMS esegue la richiesta.

ODBC: Tipo 3



Il programma chiama una funzione ODBC.

ODBC Manager determina le azioni da prendere.

Il driver ODBC prepara la richiesta e la inoltra al DBMS.

Il Gateway "traduce" la richiesta in un altro formato/ DBMS

Il DBMS esegue la richiesta.

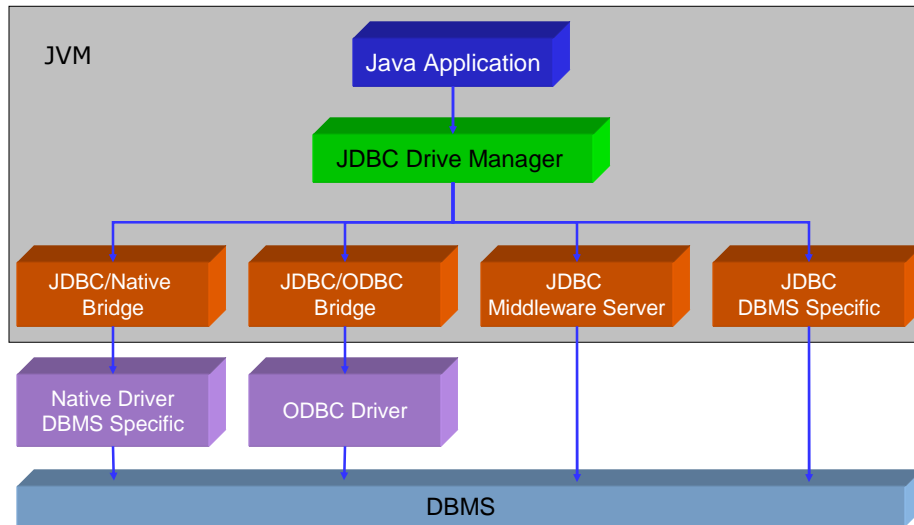
ODBC

- Altre implementazioni / estensioni proprietarie realizzate da Microsoft
 - DAO: Data Access Object
 - Orientata agli oggetti
 - RDO: Remote Data Object
 - ActiveX / COM
 - ADO: Active Data Object
 - Basata sulla tecnologia OLE DB (Object Linking and Embedding for DataBase)

JDBC (1)

- Java Database Connectivity
 - Stessa filosofia di ODBC ma realizzato da Sun Microsystems per il linguaggio Java.
 - Si basa su un insieme di classi e interfacce
 - Può essere usato per connettersi ad un DBMS direttamente
 - Può essere usato per interfacciarsi ad ODBC mediante un modulo detto “bridge”
 - 4 diversi modi di utilizzo

JDBC (2)



SQL nei Linguaggi di Programmazione

53

JDBC (3)

- Fasi di esecuzione
 - Importazione dei Packages necessari
 - Caricamento dei Driver JDBC
 - Aprire una connessione al Database
 - Creare un Statement Object
 - Eseguire la Query
 - I risultati sono ritornati in un Result Set Object
 - Processare il Result Set
 - Chiudere il Result Set e lo Statement Object
 - Chiudere la connessione

SQL nei Linguaggi di Programmazione

54

JDBC (4)

```
import java.io.*
import java.sql.*; //JDBC packages
import oracle.jdbc.driver.*;
class ExampleJDBC {
    public static void main (String args []) throws SQLException {
        // Load Oracle driver
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // Connect to the local database
        Connection conn = DriverManager.getConnection("jdbc:oracle:corisi");

        // Query execution
        Statement query=conn.createStatement();
        ResultSet risultato=query.executeQuery("select * from corisi");

        // Results processing
        while (risultato.next()) {
            String nomeCorso=risultato.getString("NomeCorso");
            System.out.println(nomeCorso);
        }

        // Close all the objects
        risultato.close();query.close();conn.close();
    }
}
```

JDBC (5)

```
import java.io.*
import java.sql.*; //JDBC packages
import oracle.jdbc.driver.*;
class ExampleJDBC {
    public static void main (String args []) throws SQLException {
        // Load Oracle driver
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // Connect to the local database
        Connection conn = DriverManager.getConnection("jdbc:oracle:corisi");

        // Query execution
        Statement query=conn.createStatement();
        ResultSet risultato=query.executeQuery("select * from corisi");

        // Results processing
        while (risultato.next()) {
            String nomeCorso=risultato.getString("NomeCorso");
            System.out.println(nomeCorso);
        }

        // Close all the objects
        risultato.close();query.close();conn.close();
    }
}
```

JDBC (6)

```
import java.io.*
import java.sql.*; //JDBC packages
import oracle.jdbc.driver.*;
class ExampleJDBC {
    public static void main (String args []) throws SQLException {
        // Load Oracle driver
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // Connect to the local database
        Connection conn = DriverManager.getConnection("jdbc:oracle:corisi");

        // Query execution
        Statement query=conn.createStatement();
        ResultSet risultato=query.executeQuery("select * from corisi");

        // Results processing
        while (risultato.next()) {
            String nomeCorso=risultato.getString("NomeCorso");
            System.out.println(nomeCorso);
        }

        // Close all the objects
        risultato.close();query.close();conn.close();
    }
}
```

JDBC (7)

```
import java.io.*
import java.sql.*; //JDBC packages
import oracle.jdbc.driver.*;
class ExampleJDBC {
    public static void main (String args []) throws SQLException {
        // Load Oracle driver
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // Connect to the local database
        Connection conn = DriverManager.getConnection("jdbc:oracle:corisi");

        // Query execution
        Statement query=conn.createStatement();
        ResultSet risultato=query.executeQuery("select * from corisi");

        // Results processing
        while (risultato.next()) {
            String nomeCorso=risultato.getString("NomeCorso");
            System.out.println(nomeCorso);
        }

        // Close all the objects
        risultato.close();query.close();conn.close();
    }
}
```

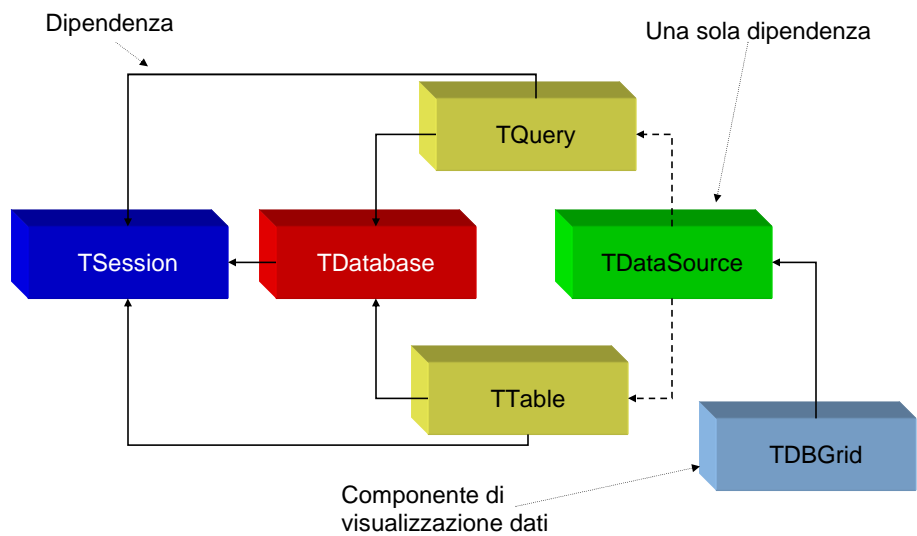
Programmazione ad oggetti

- Borland Database Engine (BDE)
 - Framework per la gestione dei database
 - C++ Builder
 - Turbo C++ 2006
 - Delphi
 - Libreria di diversi oggetti
 - TSession
 - TDatabase
 - TTable
 - TQuery
 - TDataSource
 - ...

SQL nei Linguaggi di Programmazione

59

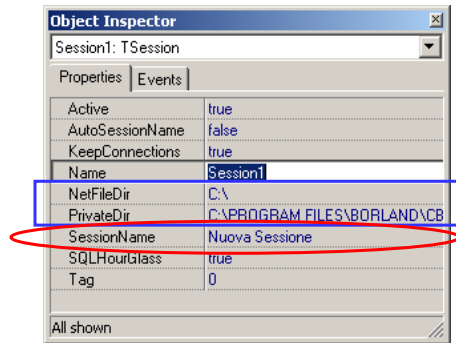
BDE



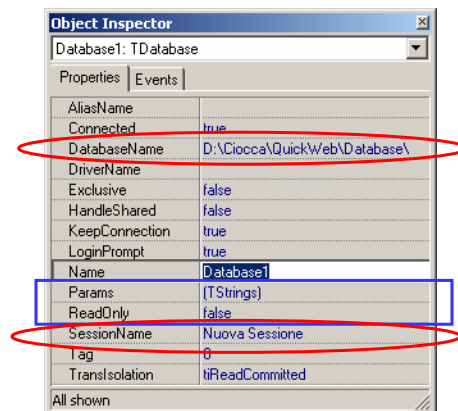
SQL nei Linguaggi di Programmazione

60

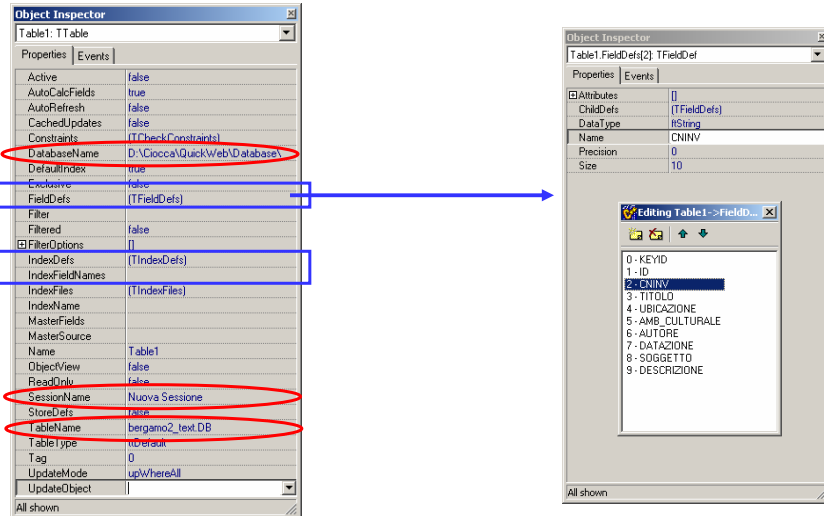
BDE: TSession



BDE: TDatabase



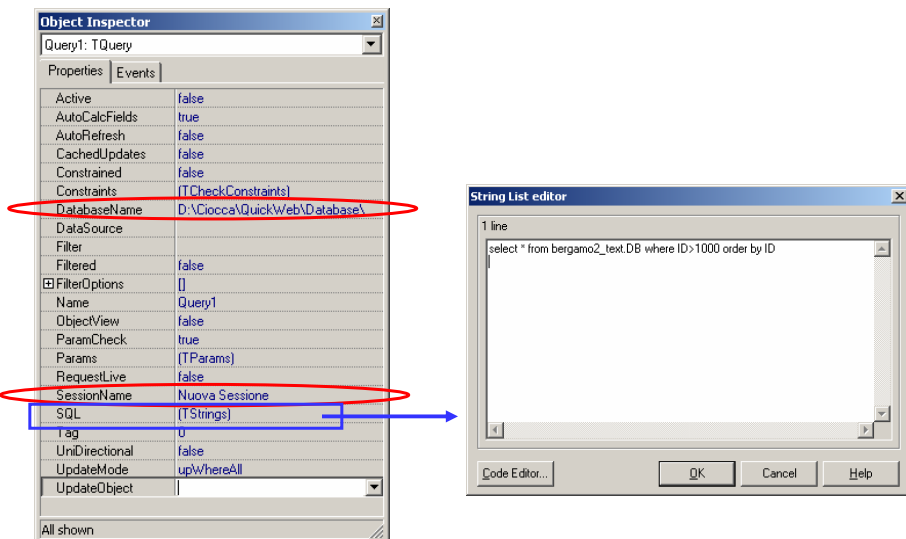
BDE: TTable



SQL nei Linguaggi di Programmazione

63

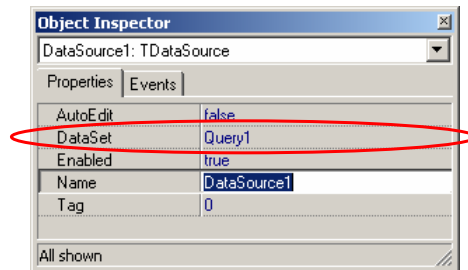
BDE: TQuery



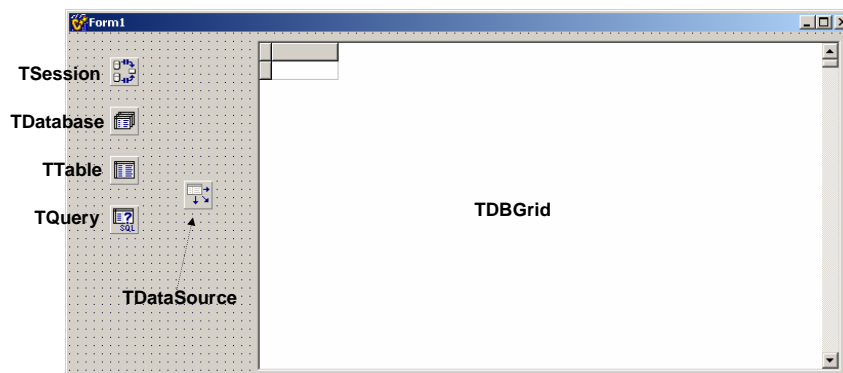
SQL nei Linguaggi di Programmazione

64

BDE: TDataSource

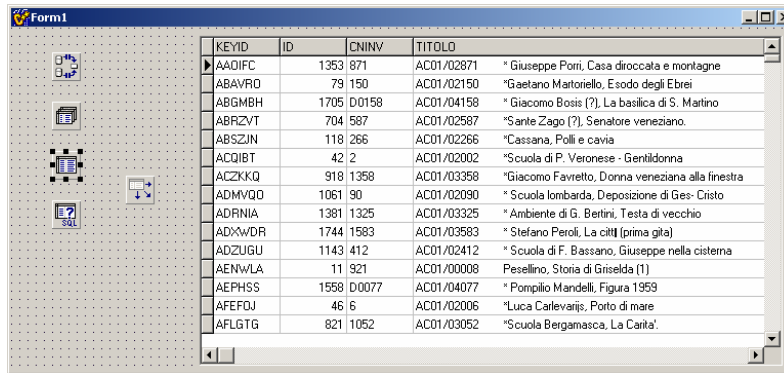


BDE: Esempio Applicazione



BDE: Esempio Applicazione

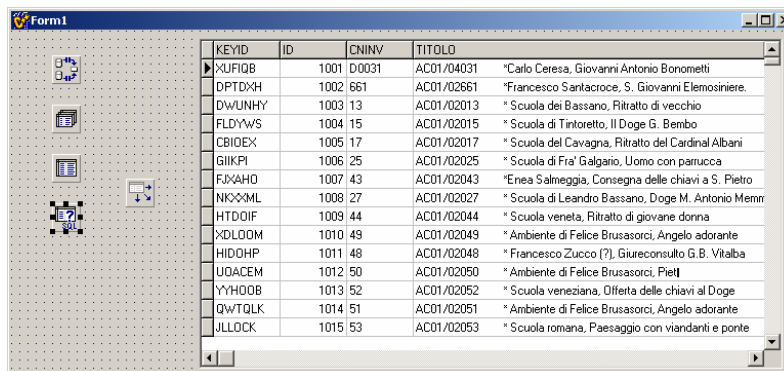
- TDataSource “connesso” a TTable



KEYID	ID	CNINV	TITOLO
AA0IFC	1353	871	AC01/02871 * Giuseppe Porri, Casa diroccata e montagne
ABAVRD	79	150	AC01/02150 *Gaetano Martoriello, Esodo degli Ebrei
ABGMBH	1705	D0158	AC01/04158 * Giacomo Bossi (?), La basilica di S. Martino
ABRZVT	704	587	AC01/02587 *Sante Zago (?), Senatore veneziano.
ABSZJN	118	266	AC01/02266 *Cassana, Polli e cavia
ACQIBT	42	2	AC01/02002 *Scuola di P. Veronese - Gentildonna
ACZKKQ	918	1358	AC01/03358 *Giacomo Favretto, Donna veneziana alla finestra
ADMVQO	1061	90	AC01/02090 * Scuola lombarda, Deposizione di Ges- Cristo
ADRNIA	1381	1325	AC01/03325 * Ambiente di G. Bertini, Testa di vecchio
ADXWDR	1744	1583	AC01/03583 * Stefano Peroli, La città (prima gila)
ADZUGU	1143	412	AC01/02412 * Scuola di F. Bassano, Giuseppe nella cisterna
AENwLA	11	921	AC01/00008 Pesellino, Storia di Griselda (1)
AEPHSS	1558	D0077	AC01/04077 * Pompilio Mandelli, Figura 1959
AFFFDJ	46	6	AC01/02006 *Luca Carlevarij, Porto di mare
AFLGTG	821	1052	AC01/03052 *Scuola Bergamasca, La Carità.

BDE: Esempio Applicazione

- TDataSource “connesso” a TQuery



KEYID	ID	CNINV	TITOLO
XUFIQB	1001	D0031	AC01/04031 *Carlo Ceresa, Giovanni Antonio Bonometti
DPTDXH	1002	661	AC01/02661 *Francesco Santacroce, S. Giovanni Elemosiniere.
DWUNHY	1003	13	AC01/02013 * Scuola dei Bassano, Ritratto di vecchio
FLDYwS	1004	15	AC01/02015 * Scuola di Tintoretto, Il Doge G. Bembo
CBIOEX	1005	17	AC01/02017 * Scuola del Cavagna, Ritratto del Cardinal Albani
GIKPI	1006	25	AC01/02025 * Scuola di Fra' Galgario, Uomo con parrucca
FJXAOH	1007	43	AC01/02043 *Enea Salmeggia, Consegna delle chiavi a S. Pietro
NKXAML	1008	27	AC01/02027 * Scuola di Leandro Bassano, Doge M. Antonio Memm
HTDOIF	1009	44	AC01/02044 * Scuola veneta, Ritratto di giovane donna
XDLOOM	1010	49	AC01/02049 * Ambiente di Felice Brusasorci, Angelo adorante
HID0HP	1011	48	AC01/02048 * Francesco Zucco (?), Giureconsulto G.B. Vitalba
UOACEM	1012	50	AC01/02050 * Ambiente di Felice Brusasorci, Pietà
YYHO0B	1013	52	AC01/02052 * Scuola veneziana, Offerta delle chiavi al Doge
QWTQLK	1014	51	AC01/02051 * Ambiente di Felice Brusasorci, Angelo adorante
JLLOCK	1015	53	AC01/02053 * Scuola romana, Paesaggio con viandanti e ponte

(select * from bergamo2_text.DB where ID>1000 order by ID)

BDE: Codice

```
session->SessionName="Nuova Sessione";
session->AutoSessionName=true;
database->DatabaseName="d:\\ciocca\\quickweb\\database\\";
query->DatabaseName=table->DatabaseName=database->DatabaseName;

table->TableName="bergamo2_text.db";
table->Open();
for(table->First();!table->Eof;table->Next())
{
    std::cout<<table->FieldByName("KEYID")->AsString.c_str()<<" ";
    std::cout<<table->FieldByName("ID")->AsInteger<<" ";
    std::cout<<table->FieldByName("TITOLO")->AsString.c_str()<<std::endl;
}
table->Close();

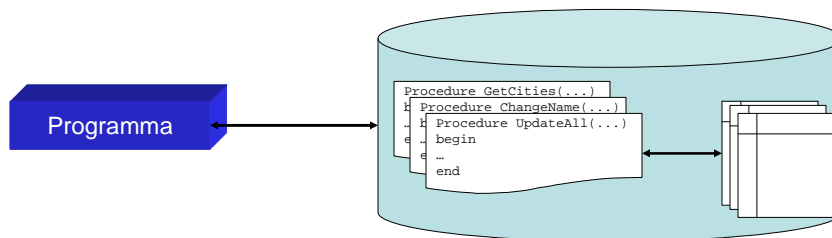
query->SQL->Add("select ID, KEYID from bergamo2_text.db where ID>1000 ORDER BY ID");
query->Open();
for(query->First();!query->Eof;query->Next())
{
    std::cout<<query->FieldByName("KEYID")->AsString.c_str()<<" ";
    std::cout<<query->FieldByName("ID")->AsInteger<<std::endl;
}
query->Close();
```

SQL nei Linguaggi di Programmazione

69

Stored Procedures (1)

- Sono funzioni di interazione con il DBMS
 - Memorizzate nel DB
 - Richiamabili all'esterno
 - Possono avere strutture di controllo
 - Simil-Linguaggi di programmazione



SQL nei Linguaggi di Programmazione

70

Stored Procedures (2)

- Vantaggi
 - Disponibilità di una libreria di comandi direttamente nel DB
 - Mascheramento dell'implementazione
 - Procedure già “digerite” dal DBMS
 - Prestazioni
- Svantaggi
 - DBMS dependent
 - Manutenibilità se la numerosità cresce troppo

Stored Procedures (3)

- MYSQL

```
CREATE PROCEDURE procedure1          /* name */
(IN parameter1 INTEGER                /* parameters */
OUT return_val INTEGER)
BEGIN                                /* start of block */
  DECLARE variable1 CHAR(10);        /* variables */
  IF parameter1 = 17 THEN             /* start of IF */
    SET variable1 = 'birds';          /* assignment */
  ELSE
    SET variable1 = 'beasts';         /* assignment */
  END IF;                             /* end of IF */
  INSERT INTO table1 VALUES (variable1); /* statement */
  SET return_val=0;                  /* return value */
END                                   /* end of block */
```

Stored Procedures (4)

- InterBase / Firebird (es. 1)

```
CREATE PROCEDURE DELETE_EMPLOYEE(EMP_NUM INTEGER) AS
  DECLARE VARIABLE any_sales INTEGER;
BEGIN
  any_sales = 0;

  /* If there are any sales records referencing this employee,
   can't delete the employee until the sales are re-assigned
   to another employee or changed to NULL. */

  SELECT count(po_number)
    FROM sales WHERE sales_rep = :emp_num INTO :any_sales;

  IF (any_sales > 0) THEN
  BEGIN
    EXCEPTION reassign_sales;
    SUSPEND;
  END

  /* -> continua */
```

Stored Procedures (5)

- InterBase / Firebird (es. 1)

```
/* If the employee is a manager, update the department. */
UPDATE department SET mngr_no = NULL WHERE mngr_no = :emp_num;

/* If the employee is a project leader, update project. */
UPDATE project SET team_leader = NULL WHERE team_leader = :emp_num;

/* Delete the employee from any projects.
DELETE FROM employee_project WHERE emp_no = :emp_num;

/* Delete old salary records.
DELETE FROM salary_history WHERE emp_no = :emp_num;

/* Delete the employee.
DELETE FROM employee WHERE emp_no = :emp_num;

SUSPEND;
END /* end of block */
```

Stored Procedures (6)

- InterBase / Firebird (es. 2)

```
CREATE PROCEDURE test1
  RETURNS (fullname char(100)) AS
BEGIN
  FOR
    SELECT first_name || ' ' || last_name FROM employee INTO :fullname
  DO
    BEGIN
      SUSPEND;
    END
  END
```

```
SELECT * FROM test1;
```

Stored Procedures (7)

- InterBase / Firebird (es. 3)

```
CREATE PROCEDURE test2 AS
BEGIN
  UPDATE salaries set salasy=salary+salary*0.05;
END
```

```
EXECUTE PROCEDURE test2;
```

- Da SQL Dinamico

```
CALL test2;
```

Stored Procedures (8)

- PL/SQL (Oracle)

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

SQL nei Linguaggi di Programmazione

2007