Atzeni, Ceri, Paraboschi, Torlone

Basi di dati

McGraw-Hill, 1996-2009

Capitolo 6:

SQL nei linguaggi di programmazione

27/07/2009

SQL e applicazioni

- In applicazioni complesse, l'utente non vuole eseguire comandi SQL, ma programmi, con poche scelte
- SQL non basta, sono necessarie altre funzionalità, per gestire:
  - input (scelte dell'utente e parametri)
  - output (con dati che non sono relazioni o se si vuole una presentazione complessa)
  - per gestire il controllo

Approcci

- Incremento delle funzionalità di SQL
  - Stored procedure
  - Trigger
  - Linguaggi 4GL
- SQL + linguaggi di programmazione

Stored procedure

- Sequenza di istruzioni SQL con parametri
- Memorizzate nella base di dati

```
procedure AssegnaCitta(:Dip varchar(20),
                       :Citta varchar(20))
update Dipartimento
set Città = :Citta
where Nome = :Dip;
```

Invocazione di stored procedure

- Possono essere invocate
  - Internamente

```
execute procedure
  AssegnaCitta('Produzione','Milano');
```

  - Esternamente

```
…
$ AssegnaCitta(:NomeDip,:NomeCitta);
…
```

Estensioni SQL per il controllo

- Esistono diverse estensioni

```
procedure CambiaCittaADip(:NomeDip varchar(20),
                          :NuovaCitta varchar(20))
      if    (     select *
                  from Dipartimento
                  where Nome = :NomeDip  ) = NULL
            insert into ErroriDip values (:NomeDip)
      else
            update Dipartimento
            set Città = :NuovaCitta
            where Nome = :NomeDip;
      end if;
  end;
```

Linguaggi 4GL

- Ogni sistema adotta, di fatto, una propria estensione
- Diventano veri e propri linguaggi di programmazione proprietari “ad hoc”:
  - PL/SQL,
  - Informix4GL,
  - PostgreSQL PL/pgsql,
  - DB2 SQL/PL

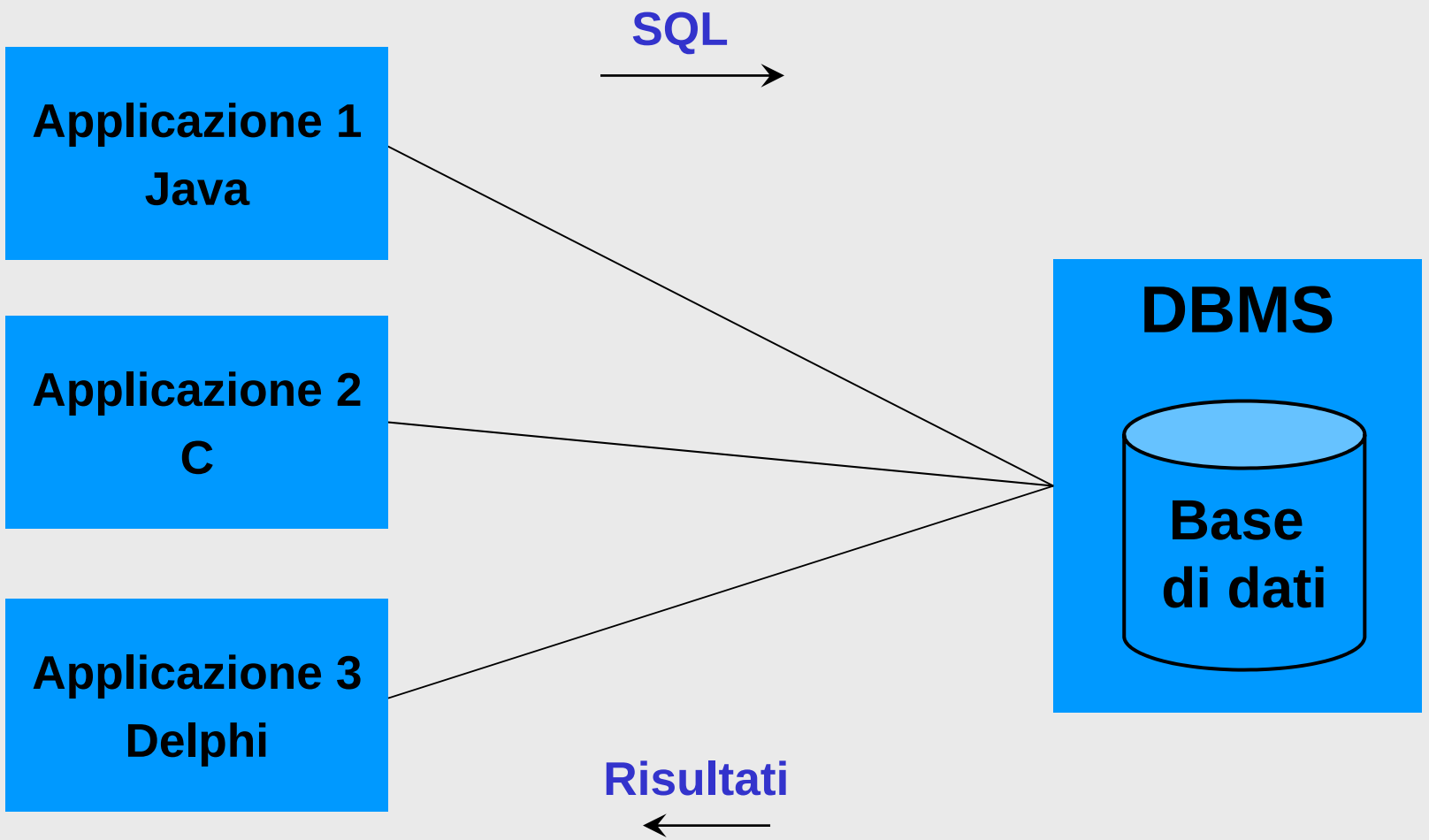
Procedure in Oracle PL/SQL

```
Procedure Debit(ClientAccount char(5),Withdrawal
integer) is
  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 e linguaggi di programmazione

- Le applicazioni sono scritte in
  - linguaggi di programmazione tradizionali:
    - Cobol, C, Java, Fortran
  - linguaggi “ad hoc”, proprietari e non:
    - vedi lucidi precedenti
- Vediamo solo l’approccio “tradizionale”, perché più generale

Applicazioni ed SQL: architettura



Una difficoltà importante

- 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)

Altre differenze

- Tipi di base:
  - linguaggi: numeri, stringhe, booleani
  - SQL: CHAR, VARCHAR, DATE, ...
- 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
  - SQL: join (ottimizzabile)

SQL e linguaggi di programmazione: tecniche principali

- SQL immerso (“Embedded SQL”)
  - sviluppata sin dagli anni ’70
  - “SQL statico”
- SQL dinamico
- Call Level Interface (CLI)
  - più recente
  - SQL/CLI, ODBC, JDBC

SQL immerso

- le istruzioni SQL sono “immerse” nel programma redatto nel linguaggio “ospite”
- un precompilatore (legato al DBMS) viene usato per analizzare il programma e tradurlo in un programma nel linguaggio ospite (sostituendo le istruzioni SQL con chiamate alle funzioni di una API del DBMS)

SQL immerso, un esempio

```
#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;
  }
}
```

SQL immerso, commenti al codice

- EXEC SQL denota le porzioni di interesse del precompilatore:
  - definizioni dei dati
  - istruzioni SQL
- le variabili del programma possono essere usate come “parametri” nelle istruzioni SQL (precedute da “:”) dove sintatticamente sono ammesse costanti

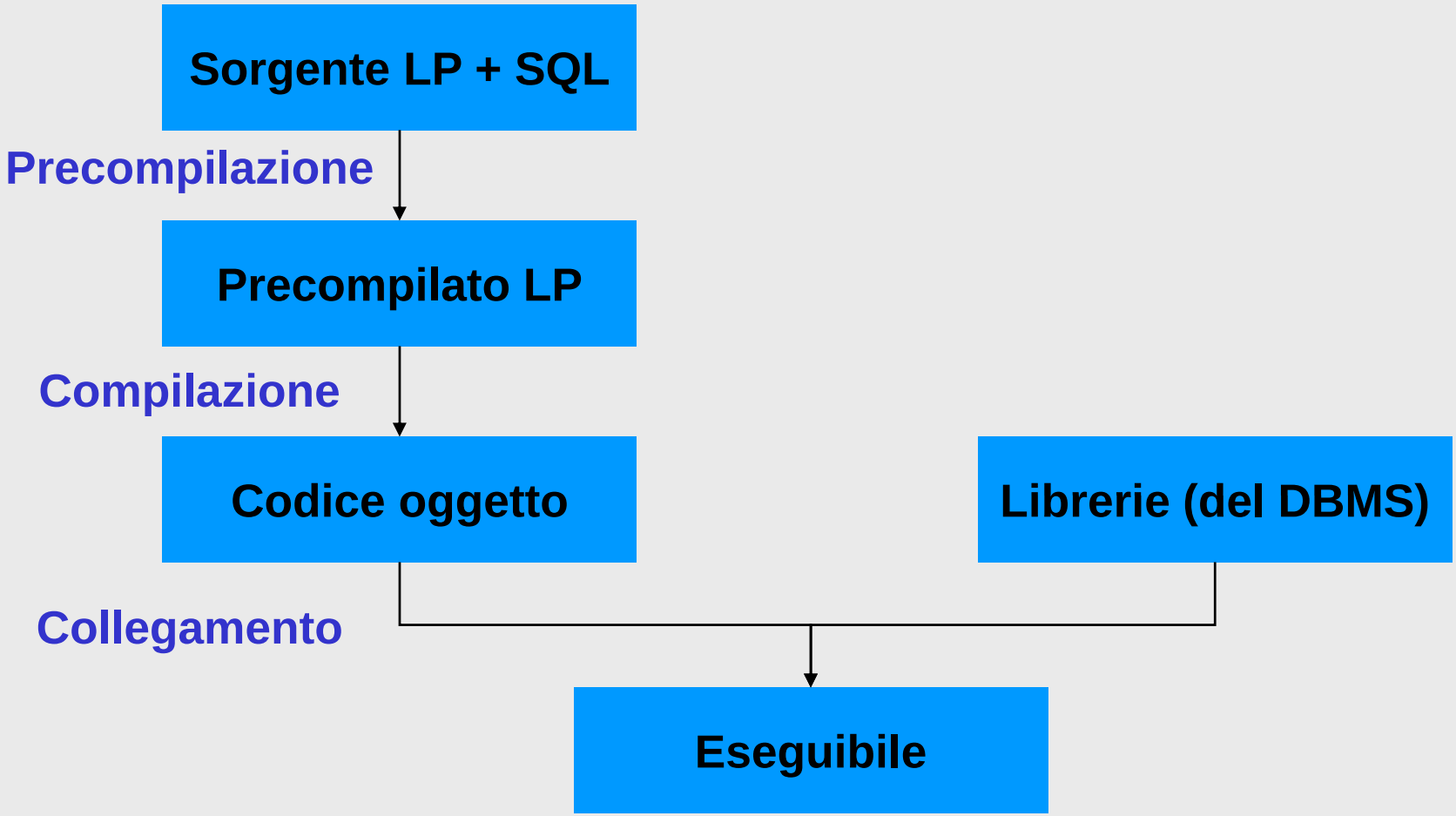
SQL immerso, commenti al codice, 2

- **`sqlca`** è una struttura dati per la comunicazione fra programma e DBMS
- **`sqlcode`** è un campo di **`sqlca`** che mantiene il codice di errore dell'ultimo comando SQL eseguito:
  - zero: successo
  - altro valore: errore o anomalia

SQL immerso, un esempio

```
#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;
  }
}
```

SQL immerso, fasi



Un altro esempio

```
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 0;
}
```

L’esempio “precompilato”

```
/* 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 0;
}
```

Note

- Il precompilatore è specifico della combinazione

  linguaggio-DBMS-sistema operativo

SQLJ, uno standard per SQL immerso in Java

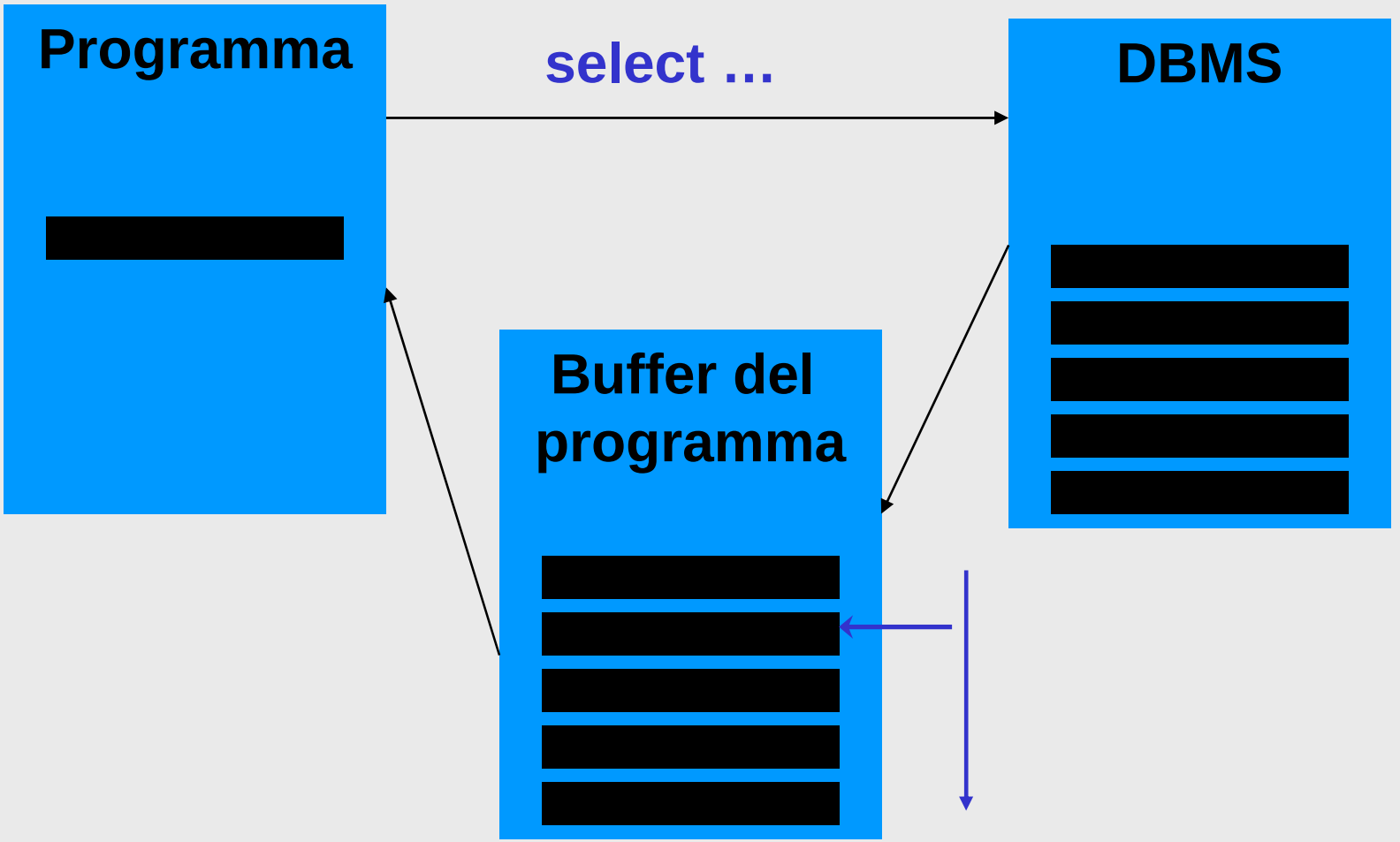
```
import …
#sql iterator CursoreProvaSelect(String, String);
class ProvaSelect
{
  public static void main(String argv[])
  {
    …
     Db db = new Db(argv[0]);
     db.getDefaultContext();
    …
     String padre = "";      String figlio = "" ;  String padrePrec = "";
     CursoreProvaSelect cursore;
     #sql cursore = {SELECT Padre, Figlio FROM Paternita ORDER BY Padre};
     #sql {FETCH :cursore INTO :padre, :figlio};
     while (!cursore.endFetch()){
     if (!(padre.equals(padrePrec))) { System.out.println("Padre: " + padre +  "\n  Figli:  " + figlio);}
     else System.out.println( "          " + figlio ) ;
     padrePrec = padre ;
     #sql {FETCH :cursore INTO :padre, :figlio};
     cursore.close();
      …
 }
```

vedi note

Interrogazioni in SQL immerso: conflitto di impedenza

- Il risultato di una select è costituito da zero o piú ennuple:
  - zero o una: ok -- l'eventuale risultato puó essere gestito in un record
  - piú ennuple: come facciamo?
    - l'insieme (in effetti, la lista) non è gestibile facilmente in molti linguaggi
- Cursore: tecnica per trasmettere al programma una ennupla alla volta

Cursore



Nota

- Il cursore
  - accede a tutte le ennuple di una interrogazione in modo globale (tutte insieme o a blocchi – è il DBMS che sceglie la strategia efficiente)
  - trasmette le ennuple al programma una alla volta

Operazioni sui cursori

Definizione del cursore

declare NomeCursore [**scroll**] **cursor for** Select …

Esecuzione dell'interrogazione

open NomeCursore

Utilizzo dei risultati (una ennupla alla volta)

fetch NomeCursore **into** ListaVariabili

Disabilitazione del cursore

close cursor NomeCursore

Accesso alla ennupla corrente (di un cursore su singola relazione a fini di aggiornamento)

current of NomeCursore

nella clausola **where**

```
write('nome della citta''?');
readln(citta);
EXEC SQL DECLARE P CURSOR FOR
   SELECT NOME, REDDITO
   FROM PERSONE
   WHERE CITTA = :citta ;
EXEC SQL OPEN P ;
EXEC SQL FETCH P INTO :nome, :reddito ;
while SQLCODE = 0
do begin
      write('nome della persona:', nome, 'aumento?');
      readln(aumento);
      EXEC SQL UPDATE PERSONE
                  SET REDDITO = REDDITO + :aumento
                  WHERE CURRENT OF P
      EXEC SQL FETCH P INTO :nome, :reddito
   end;
EXEC SQL CLOSE CURSOR P
```

```
void VisualizzaStipendiDipart(char NomeDip[])
{
  char Nome[20], Cognome[20];
  long int Stipendio;
  $ declare ImpDip cursor for
      select Nome, Cognome, Stipendio
      from Impiegato
      where Dipart = :NomeDip;
  printf("Dipartimento %s\n",NomeDip);
  $ open ImpDip;
  $ fetch ImpDip into :Nome, :Cognome, :Stipendio;
  while (sqlcode == 0)
  {
      printf("Nome e cognome dell'impiegato: %s
                  %s",Nome,Cognome);
      printf("Attuale stipendio: %d\n",Stipendio);
      $ fetch ImpDip into :Nome, :Cognome,
            :Stipendio;
  }
  $ close cursor ImpDip;
}
```

Cursori, commenti

- Per aggiornamenti e interrogazioni “scalari” (cioè che restituiscano una sola ennupla) il cursore non serve:

```
select Nome, Cognome
   into :nomeDip, :cognomeDip
from Dipendente
where Matricola = :matrDip;
```

Cursori, commenti, 2

- I cursori possono far scendere la programmazione ad un livello troppo basso, pregiudicando la capacità dei DBMS di ottimizzare le interrogazioni:
  - se “nidifichiamo” due o più cursori, rischiamo di reimplementare il join!

Esercizio

Studenti(Matricola, Cognome, Nome)
Esami(Studente,Materia,Voto,Data)
Corsi(Codice,Titolo)
con gli ovvî vincoli di integrità referenziale

- Stampare, per ogni studente, il certificato con gli esami e il voto medio

Output

Matricola Cognome Nome
Materia Data Voto
…
Materia Data Voto
VotoMedio
Matricola Cognome Nome
Materia Data Voto
…
Materia Data Voto
VotoMedio
…

Esercizio

Studenti(Matricola, Cognome, Nome)
Esami(Studente,Materia,Voto,Data)
Corsi(Codice,Titolo)
Iscrizioni(Studente,AA,Anno,Tipo)
con gli ovvî vincoli di integrità referenziale

- Stampare, per ogni studente, il certificato con gli esami e le iscrizioni ai vari anni accademici

Output

Matricola Cognome Nome
    AnnoAccademico AnnoDiCorso TipoIscrizione
    …
    AnnoAccademico AnnoDiCorso TipoIscrizione
        Materia Data Voto
        …
        Materia Data Voto
Matricola Cognome Nome
    AnnoAccademico AnnoDiCorso TipoIscrizione
    …
    AnnoAccademico AnnoDiCorso TipoIscrizione
        Materia Data Voto
        …
        Materia Data Voto

SQL dinamico

- Non sempre le istruzioni SQL sono note quando si scrive il programma
- Allo scopo, è stata definita una tecnica completamente diversa, chiamata *Dynamic SQL* che permette di eseguire istruzioni SQL costruite dal programma (o addirittura ricevute dal programma attraverso parametri o da input)
- Non è banale gestire i parametri e la struttura dei risultati (non noti a priori)

SQL dinamico

- Le operazioni SQL possono essere:
  - eseguite immediatamente

    `execute immediate` *SQLStatement*
  - prima “preparate”:

    `prepare` *CommandName* `from` *SQLStatement*

    e poi eseguite (anche più volte):

    `execute` *CommandName* `[into` *TargetList* `]`
    `[using` *ParameterList* `]`

Call Level Interface

- Indica genericamente interfacce che permettono di inviare richieste a DBMS per mezzo di parametri trasmessi a funzioni
- standard SQL/CLI (’95 e poi parte di SQL-3)
- ODBC: implementazione proprietaria di SQL/CLI
- JDBC: una CLI per il mondo Java

SQL immerso vs CLI

- SQL immerso permette
  - precompilazione (e quindi efficienza)
  - uso di SQL completo
- CLI
  - indipendente dal DBMS
  - permette di accedere a più basi di dati, anche eterogenee

JDBC

- Una API (Application Programming Interface) di Java (intuitivamente: una libreria) per l'accesso a basi di dati, in modo indipendente dalla specifica tecnologia
- JDBC è una interfaccia, realizzata da classi chiamate driver:
  - l'interfaccia è standard, mentre i driver contengono le specificità dei singoli DBMS (o di altre fonti informative)