SQL

Laboratorio di
Progettazione di Basi di Dati (CdS in Informatica e TPS)

a.a. 2013/2014

http://www.di.uniba.it/~lisi/courses/basi-dati/bd2013-14.htm

dott.ssa Francesca A. Lisi
lisi@di.uniba.it

Orario di ricevimento: giovedì ore 10-12

Sommario (VI parte)

- Procedure memorizzate (o *stored procedures*)
- SQL immerso (o *embedded SQL*)
- Call Level Interface

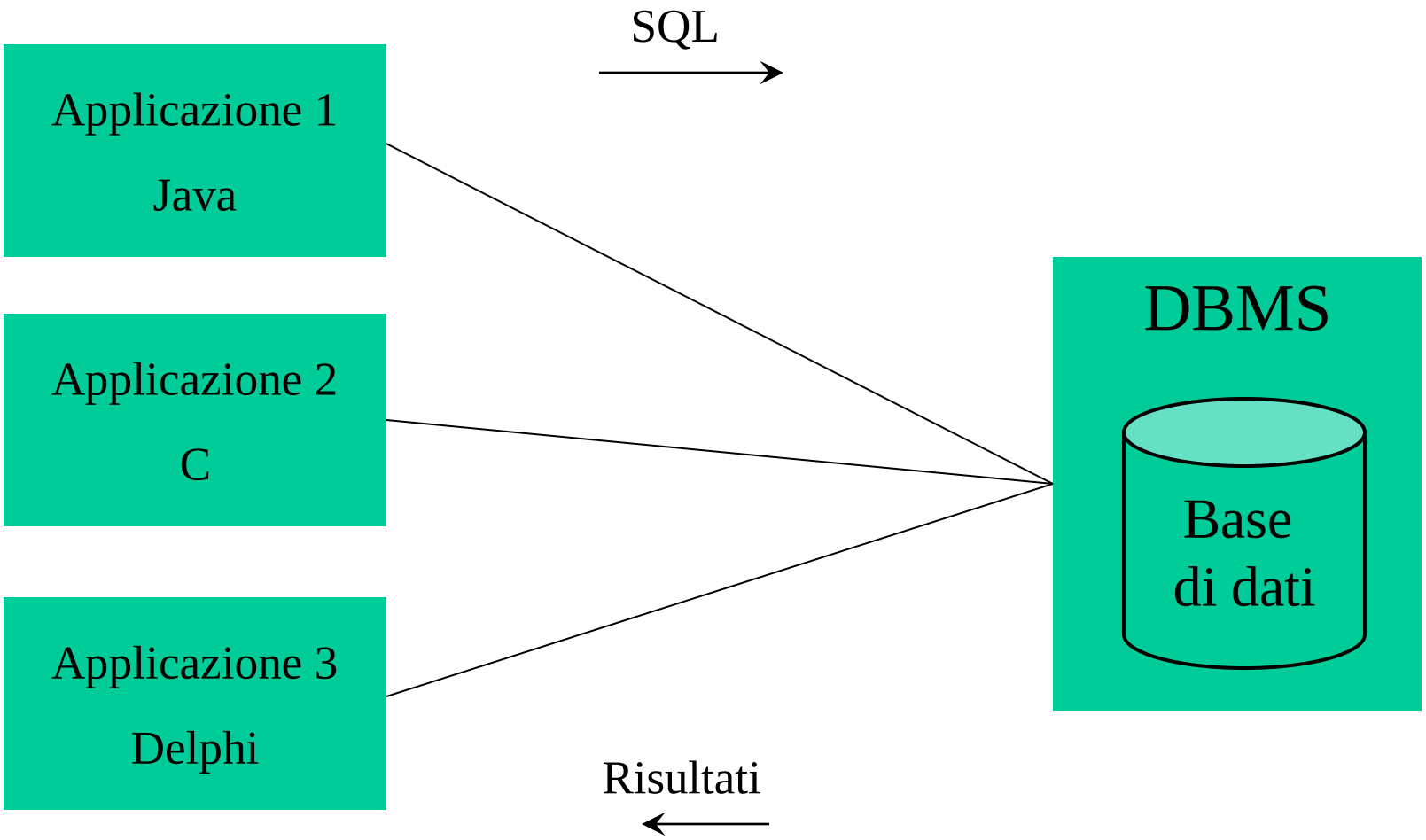
Riferimenti

- cap. 8 di Pratt
- cap. 6, in particolare 6.1, 6.3 e 6.4, di Atzeni et al.
- cap. 9, in particolare 9.4-9.7 di Elmasri & Navathe

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

SQL ed applicazioni: Architettura



SQL ed applicazioni: Approcci

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

Procedure memorizzate

- Sequenza di istruzioni SQL con parametri
- Memorizzate nella base di dati come parte dello schema

```
CREATE PROCEDURE AssegnaCitta
(IN Dip VARCHAR(20), IN City VARCHAR(20))
UPDATE Dipartimento
SET Citta = City
WHERE Nome = Dip;
```

Procedure memorizzate: invocazione

- Possono essere invocate
  - Internamente

```
CALL PROCEDURE
  AssegnaCitta('Produzione','Milano')
  ;
```

  - Esternamente (per es., da un programma in C)

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

Procedure memorizzate: 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

SQL e linguaggi di programmazione: 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)

SQL e linguaggi di programmazione: 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/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)

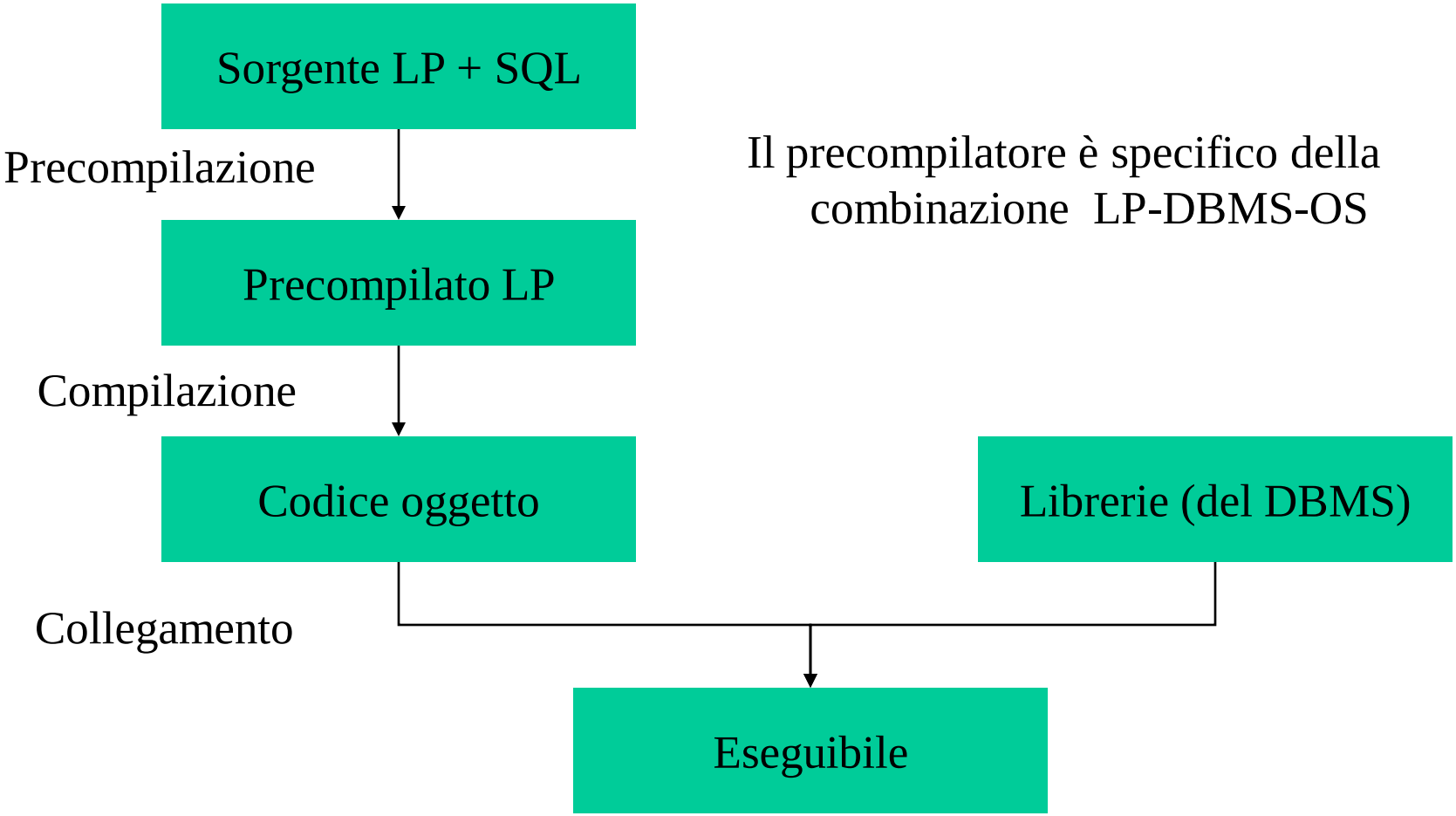
SQL immerso:
un esempio in C

```
#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:
un esempio in C (2)

- **`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
- **`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: fasi



Il precompilatore è specifico della combinazione LP-DBMS-OS

SQL immerso:
un altro esempio in C (cod. sorgente)

```
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;
}
```

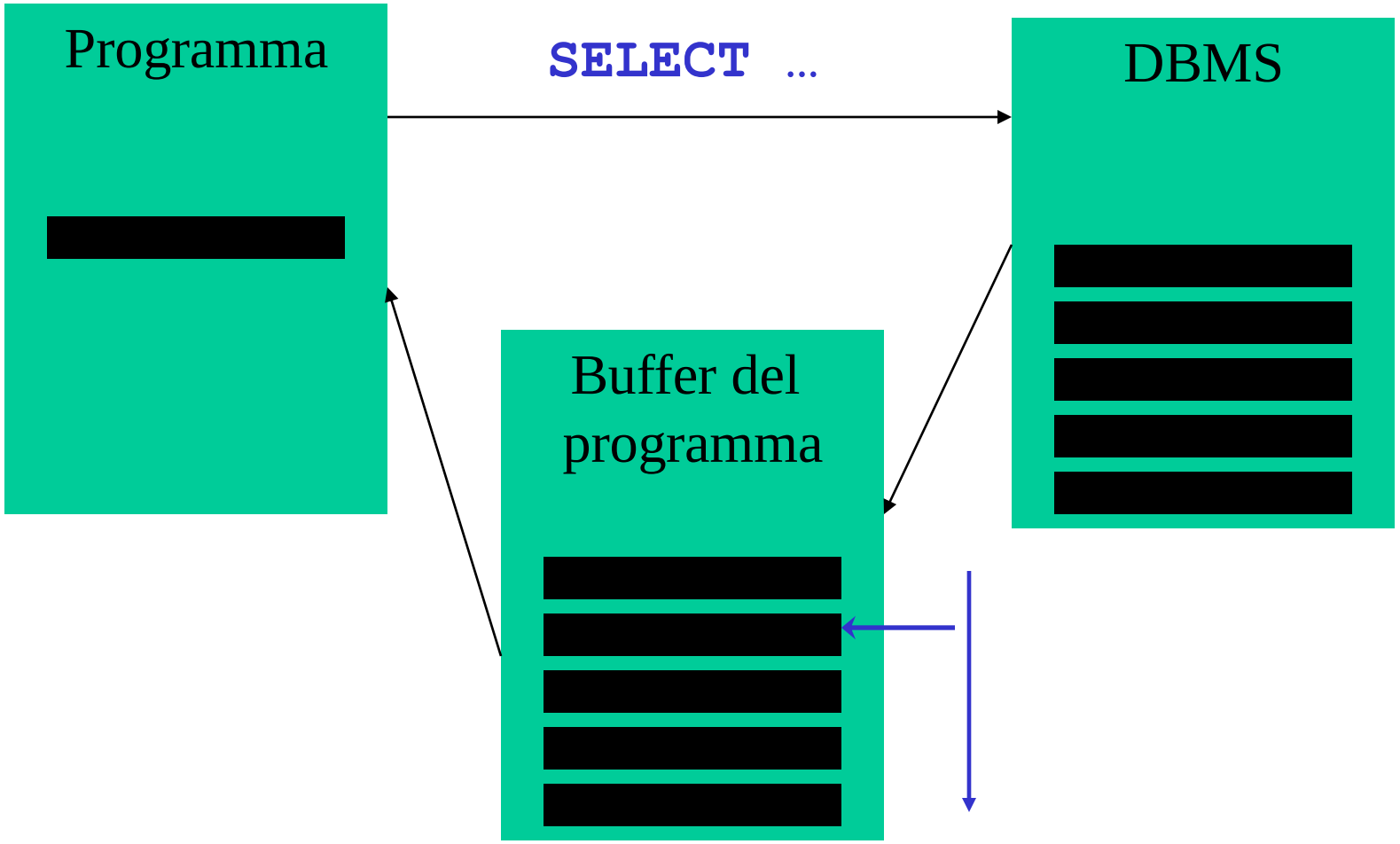
SQL immerso:
un altro esempio in C (cod. precomp.)

```
/* 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;
}
```

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
  - 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

SQL immerso: i cursori



SQL immerso:
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;
}
```

SQL immerso:
commenti sull'uso dei cursori

- 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;
```

- 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!

SQL immerso in Java: lo standard SQLJ

```
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

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

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

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 immerso:
uso di 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 immerso:
uso di SQL dinamico (2)

- 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*]

Esercitazione con MySQL

- Esercizi 1-3 sul database Prodotti Premiere da cap. 8, pag. 188-189 di Pratt (mediante procedure/funzioni memorizzate o mediante SQLJ);
- Esercizi 6.1, 6.2-6.3 (SQLJ), a pag. 202 di Atzeni et al.

- N.B. MySQL 5.5 supporta procedure e funzioni memorizzate
- **CREATE FUNCTION** *nomef*([*par*]) **RETURNS** *tipo*[*opzioni*] *codiceSQL*
- **CREATE PROCEDURE** *nomep*([*par*]) [*opzioni*]
- **DROP FUNCTION / PROCEDURE [IF EXISTS]** *nome*
- **CALL** *nomep*([*par*])