

SQL

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

a.a. 2012/2013

<http://www.di.uniba.it/~lisi/courses/basi-dati/bd2012-13.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

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

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

Procedure memorizzate: invocazione

- Possono essere invocate

- Internamente

EXECUTE PROCEDURE

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

- Esternamente

...

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

...

Procedure memorizzate: Estensioni SQL per il controllo

- Esistono diverse estensioni

```
PROCEDURE CambiaCittàADip (:NomeDip VARCHAR(20) ,
                           :NuovaCittà VARCHAR(20))

    IF      (      SELECT *
                   FROM Dipartimento
                   WHERE Nome = :NomeDip  ) = NULL
        INSERT INTO ErroriDip VALUES (:NomeDip)
    ELSE
        UPDATE Dipartimento
        SET Città = :NuovaCittà
        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

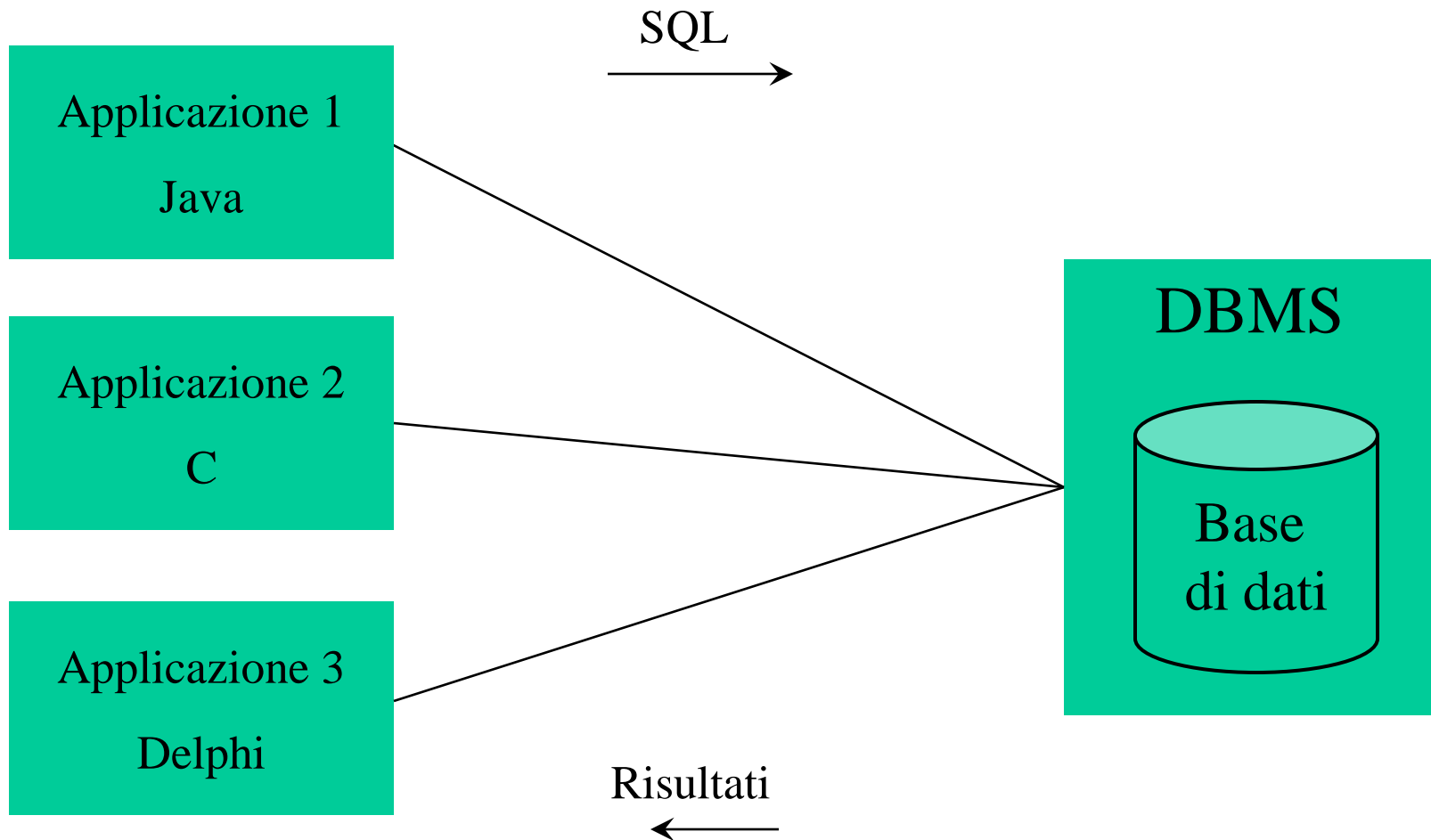
Procedure in Oracle PL/SQL

```
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



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)

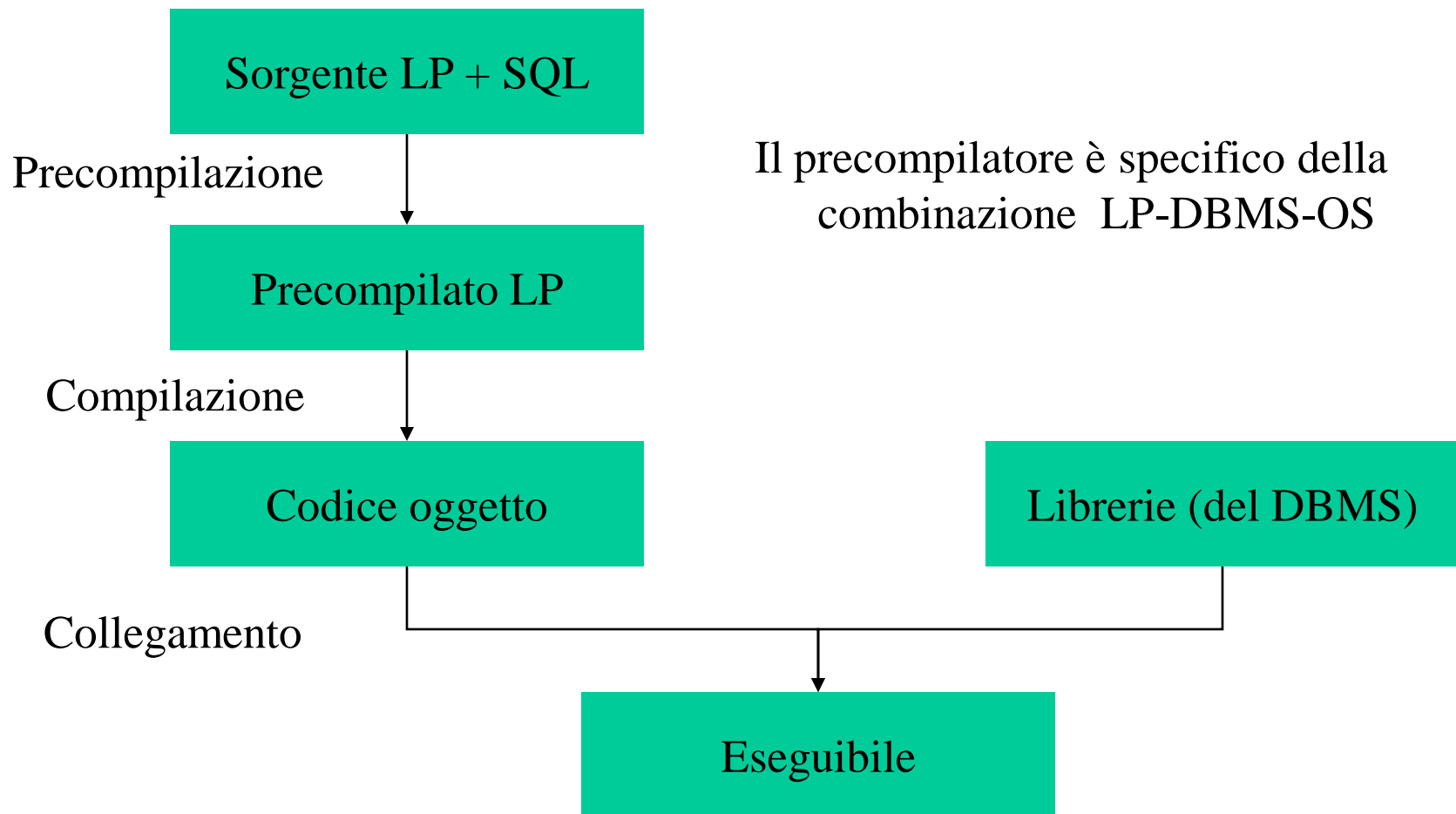
SQL immerso: un esempio in C

```
#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



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;  
}
```

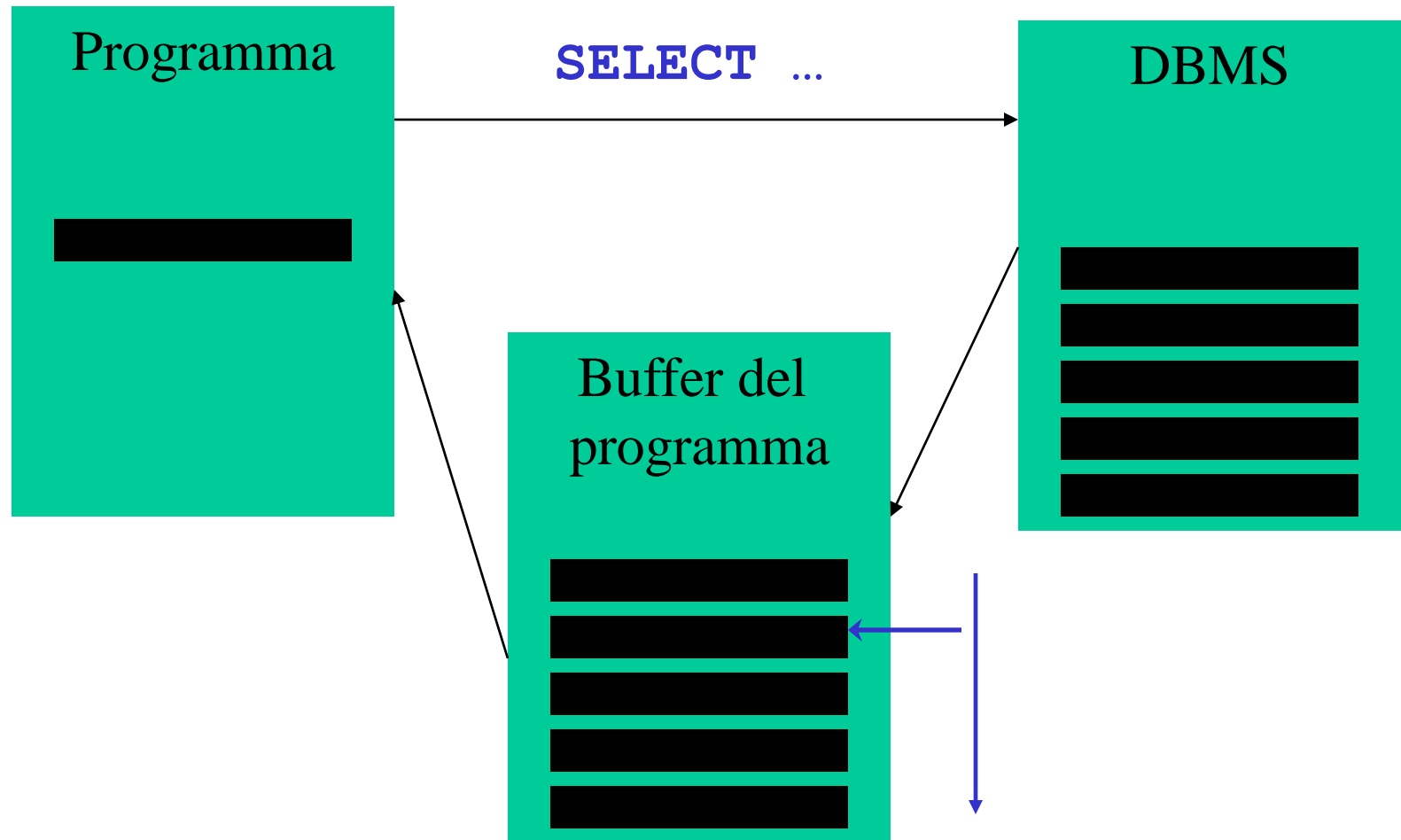
SQL immerso: un altro esempio in C (cod. precomp.)

```
/* 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 più ennuple:
 - zero o una: ok -- l'eventuale risultato può essere gestito in un record
 - più 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();
            ...
        }
    }
}
```

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 “prepare”:

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]*)

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

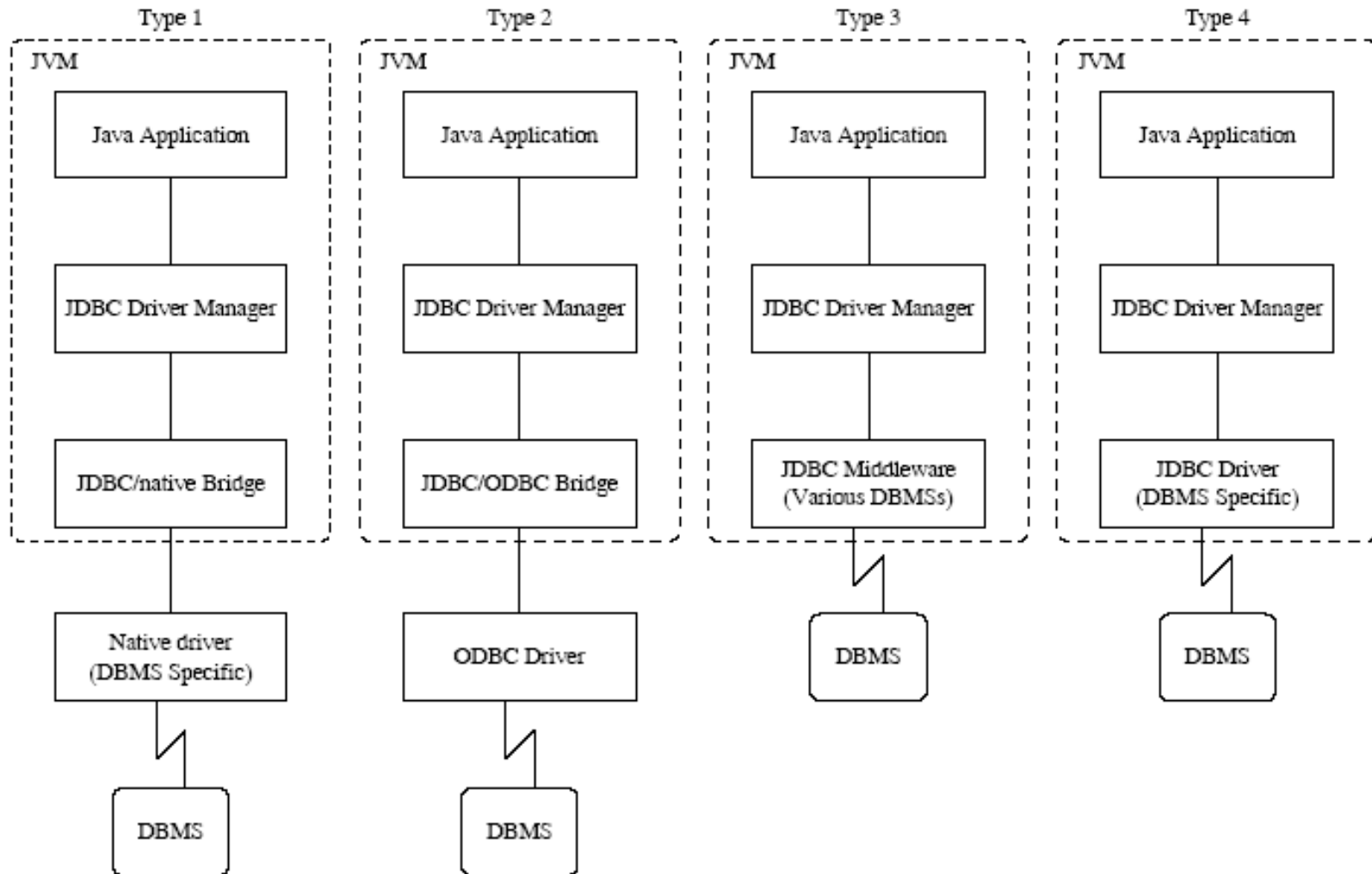
SQL immerso vs CLI

- 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)

I driver JDBC



Il funzionamento di JDBC, in breve

- Caricamento del driver
- Apertura della connessione alla base di dati
- Richiesta di esecuzione di istruzioni SQL
- Elaborazione dei risultati delle istruzioni SQL

Un programma con JDBC

```
import java.sql.*;
public class PrimoJDBC {
    public static void main(String[] arg){
        Connection con = null ;
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "jdbc:odbc:Corsi";
            con = DriverManager.getConnection(url);
        }
        catch(Exception e){
            System.out.println("Connessione fallita");
        }
        try {
            Statement query = con.createStatement();
            ResultSet result =
                query.executeQuery("select * from Corsi");
            while (result.next()){
                String nomeCorso = result.getString("NomeCorso");
                System.out.println(nomeCorso);
            }
        }
        catch (Exception e){
            System.out.println("Errore nell'interrogazione");
        }
    }
}
```

Un altro programma con JDBC (1)

```
import java.lang.*;
import java.sql.*;
class ProvaSelectJDBC
{
    public static void main(String argv[])
    {
        Connection con = null;
        try { Class.forName("com.ibm.db2.jcc.DB2Driver");
        }
        catch (ClassNotFoundException exClass) {
            System.err.println("Fallita connessione al database. Errore 1");
        }
        try {
            String url = "jdbc:db2:db04";
            con = DriverManager.getConnection(url);
        }
        catch (SQLException exSQL) {
            System.err.println("Fallita connessione al database. "+
                exSQL.getErrorCode() + " " +
                exSQL.getSQLState() + exSQL.getMessage() );
        }
    }
}
```

Un altro programma con JDBC (2)

```
try{ String padre = ""; String figlio = "" ; String padrePrec = "";
Statement query = con.createStatement();
String queryString =
    "SELECT Padre, Figlio FROM Paternita ORDER BY Padre";
ResultSet result = query.executeQuery(queryString);
while (result.next()){
    padre = result.getString("Padre");
    figlio = result.getString("Figlio");
    if (!(padre.equals(padrePrec))){
        System.out.println("Padre: " + padre +
            "\n Figli:  " + figlio);}
    else System.out.println( "          " + figlio ) ;
    padrePrec = padre ;
}
}
catch (SQLException exSQL) {
System.err.println("Errore nell'interrogazione. "+
    exSQL.getErrorCode() + " " + exSQL.getMessage() );
}
}
```


Preliminari

- L'interfaccia JDBC è contenuta nel package `java.sql`

```
import java.sql.*;
```

- Il driver deve essere caricato (trascuriamo i dettagli)

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

- Connessione: oggetto di tipo **Connection** che costituisce un collegamento attivo fra programma Java e base di dati; viene creato da

```
String url = "jdbc:odbc:Corsi";  
con = DriverManager.getConnection(url);
```

Preliminari dei preliminari: origine dati ODBC

- Per utilizzare un driver JDBC-ODBC, la base di dati (o altro) deve essere definita come "origine dati ODBC"
- In Windows (con **YYY**, avendo già definito la base di dati **xxx.yyy** da collegare):
 - Pannello di controllo
 - Strumenti di amministrazione
 - Opzione "Origini dati ODBC"
 - Bottone "Aggiungi" ("Add")
 - Nella finestra di dialogo "Crea Nuova origine dati" selezionare "**YYY Driver**" e nella successiva
 - selezionare il file **xxx.yyy**
 - attribuirgli un nome (che sarà usato da ODBC e quindi da JDBC)

Esecuzione dell'interrogazione ed elaborazione del risultato

Esecuzione dell'interrogazione

```
Statement query = con.createStatement();  
ResultSet result =  
    query.executeQuery("select * from Corsi");
```

Elaborazione del risultato

```
while (result.next()) {  
    String nomeCorso =  
        result.getString("NomeCorso");  
    System.out.println(nomeCorso);  
}
```

La classe Statement

- Un'interfaccia i cui oggetti consentono di inviare, tramite una connessione, istruzioni SQL e di ricevere i risultati forniti
- Un oggetto di tipo **Statement** viene creato con il metodo **createStatement** di **Connection**
- I metodi dell'interfaccia **Statement**:
 - **executeUpdate** per specificare aggiornamenti o istruzioni DDL
 - **executeQuery** per specificare interrogazioni e ottenere un risultato
 - **execute** per specificare istruzioni non note a priori
 - **executeBatch** per specificare sequenze di istruzioni
- Vediamo **executeQuery**

La classe `ResultSet`

- I risultati delle interrogazioni sono forniti in oggetti di tipo `ResultSet` (interfaccia definita in `java.sql`)
- In sostanza, un *result set* è una sequenza di ennuple su cui si può "navigare" (in avanti, indietro e anche con accesso diretto) e dalla cui ennupla "corrente" si possono estrarre i valori degli attributi
- Metodi principali:
 - `next()`
 - `getXXX(posizione)`
 - es: `getString(3) ; getInt(2)`
 - `getXXX(nomeAttributo)`
 - es: `getString("Cognome") ;
getInt("Codice")`

Specializzazioni di Statement

- **PreparedStatement** permette di utilizzare codice SQL già compilato, eventualmente parametrizzato rispetto alle costanti
 - in generale più efficiente di **Statement**
 - permette di distinguere più facilmente istruzioni e costanti (e apici nelle costanti)
- i metodi **setXXX(,)** permettono di definire i parametri
- **CallableStatement** permette di utilizzare "stored procedure", come quelle di Oracle PL/SQL o anche le query memorizzate (e parametriche) di Access

```

import java.sql.*;
import javax.swing.JOptionPane;

public class SecondoJDBCprep {
    public static void main(String[] arg){
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "jdbc:odbc:Corsi";
            Connection con = DriverManager.getConnection(url);
            PreparedStatement pquery = con.prepareStatement(
                "select * from Corsi where NomeCorso LIKE ?");
            String param = JOptionPane.showInputDialog(
                "Nome corso (anche parziale)?");
            param = "%" + param + "%";
            pquery.setString(1,param);
            ResultSet result = pquery.executeQuery();
            while (result.next()){
                String nomeCorso = result.getString("NomeCorso");
                System.out.println(nomeCorso);
            }
        } catch (Exception e){System.out.println("Errore");}
    }
}

```

```

import java.sql.*;
import javax.swing.JOptionPane;

public class TerzoJDBCcall {
    public static void main(String[] arg){
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "jdbc:odbc:Corsi";
            Connection con = DriverManager.getConnection(url);
            CallableStatement pquery =
                con.prepareCall("{call queryCorso(?)}");
            String param = JOptionPane.showInputDialog(
                "Nome corso (anche parziale)?");
            param = "*" + param + "*";
            pquery.setString(1,param);
            ResultSet result = pquery.executeQuery();
            while (result.next()){
                String nomeCorso =
                    result.getString("NomeCorso");
                System.out.println(nomeCorso);
            }
        } catch (Exception e){System.out.println("Errore");}
    }
}

```


Altre funzionalità

- Molte, fra cui
 - username e password
 - aggiornamento dei ResultSet
 - richiesta di metadati
 - gestione di transazioni

Transazioni in JDBC

- Scelta della modalità delle transazioni: un metodo definito nell'interfaccia `Connection`:

`setAutoCommit(boolean autoCommit)`

- `con.setAutoCommit(true)`
 - (default) "autocommit": ogni operazione è una transazione
- `con.setAutoCommit(false)`
 - gestione delle transazioni da programma
 - `con.commit()`
 - `con.rollback()`
 - non c'è `begin transaction`

Esercitazione con MySQL

- Esercizi 1-3 sul database Prodotti Premiere da cap. 8, pag. 188-189 di Pratt (mediante JDBC);
- Esercizi 6.5-6.13 a pagg. 202-204 di Atzeni et al. (mediante JDBC).