Basi Dati

SQL e i linguaggi di programmazione

a.a. 2021/2022

Prof.ssa G. Tortora

Outline

1. SQL e applicazioni
2. Conflitto di impedenza
3. Approcci alla programmazione per basi di dati
4. SQL immerso, CLI e PL/SQL
5. Interrogazioni in C
6. Utilizzo del cursore
7. JDBC

SQL e applicazioni

- Nella pratica gli utenti finali accedono al contenuto di db, non direttamente attraverso l'uso del linguaggio SQL, ma attraverso programmi applicativi che realizzano apposite interfacce di accesso alla base di dati stessa.
  - **Es:** le interfacce web consentono in maniera del tutto *"grafica"* di prenotare un aereo, di acquistare un prodotto o di visualizzare gli ultimi movimenti sul proprio conto bancario.

SQL e applicazioni (2)

- La classica interazione SQL con la base di dati viene nascosta all'utente e racchiusa all'interno del programma, ed è quindi compito del programmatore dell'applicazione implementare le varie funzionalità previste per l'accesso ai dati.

- In tale ottica fondamentale importanza assumono quindi i meccanismi con cui le applicazioni accedono ai dati.

SQL e applicazioni (3)

- L'utente non vuole eseguire comandi SQL, ma programmi, con poche scelte.

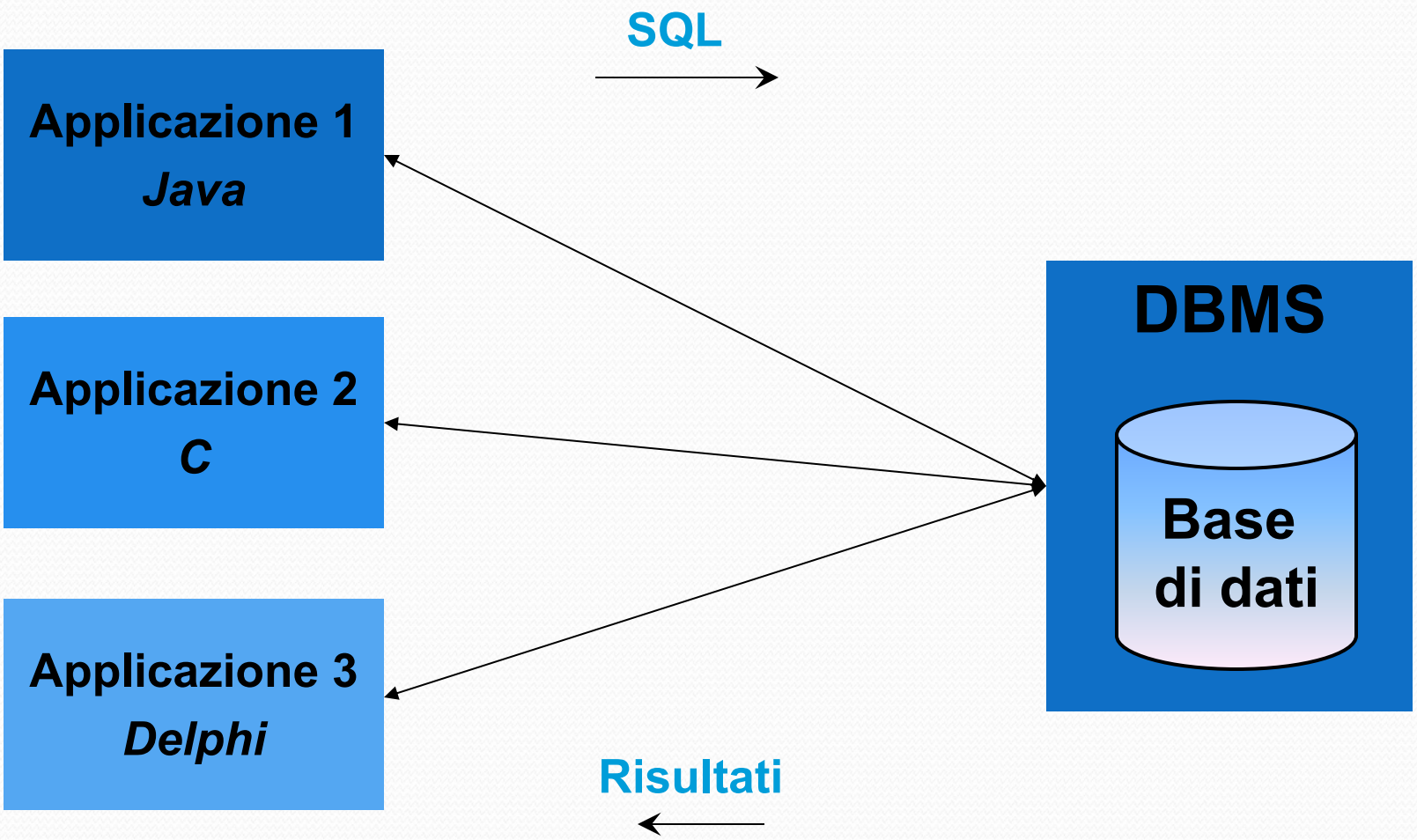
- SQL non basta, sono necessarie altre funzionalità, per gestire:
  - input (scelte dell'utente e parametri);
  - output (con dati che non sono relazioni o se si vuole una presentazione complessa);
  - il controllo.

SQL e linguaggi di programmazione

- Le applicazioni sono scritte in
  - linguaggi di programmazione tradizionali:
    - Cobol, C, Java, Delphi, …
  - linguaggi *"ad hoc"*, proprietari e non:
    - PL/SQL, …

- Utilizzeremo l'approccio *"tradizionale"*, perché più generale.

Architettura



Una difficoltà importante

- Conflitto di impedenza (*"disaccoppiamento di impedenza"*) fra base di dati e linguaggio di programmazione:
  - Le tecniche di accesso al db devono essere in grado di colmare le differenze esistenti tra il modello della base di dati ed il modello del linguaggio di programmazione.
    - I linguaggi di programmazione accedono agli elementi di una tabella scandendo le righe una ad una (approccio **tuple-oriented**).
    - SQL è un linguaggio di tipo **set-oriented** che opera su intere tabelle che restituisce come risultato.

Differenze

- Interrogazioni:
  - **Linguaggi:** operazioni su singole variabili o oggetti.
  - **SQL:** operazioni su relazioni (insiemi di ennuple).
- Accesso ai dati e correlazione:
  - **Linguaggi:** dipende dal paradigma e dai tipi disponibili;

    Es: scansione di liste o *"navigazione"* tra oggetti.
  - **SQL:** join.
- Tipi di base:
  - **Linguaggi:** tipi per numeri, stringhe, booleani;
  - **SQL:** CHAR, VARCHAR, DATE, …
- Costruttori di tipo:
  - **Linguaggi:** dipende dal paradigma.
  - **SQL:** relazioni e ennuple.

SQL e linguaggi di programmazione: tecniche principali

1. SQL immerso ("Embedded SQL"):
   - Sviluppata sin dagli anni '70.
   - SQL statico.
   - SQL dinamico.

2. Call Level Interface (CLI):
   - Più recente.
   - ODBC (Open DataBase Connectivity), JDBC.

3. Linguaggi di programmazione per basi di dati:
   - PL/SQL.

Tecniche principali

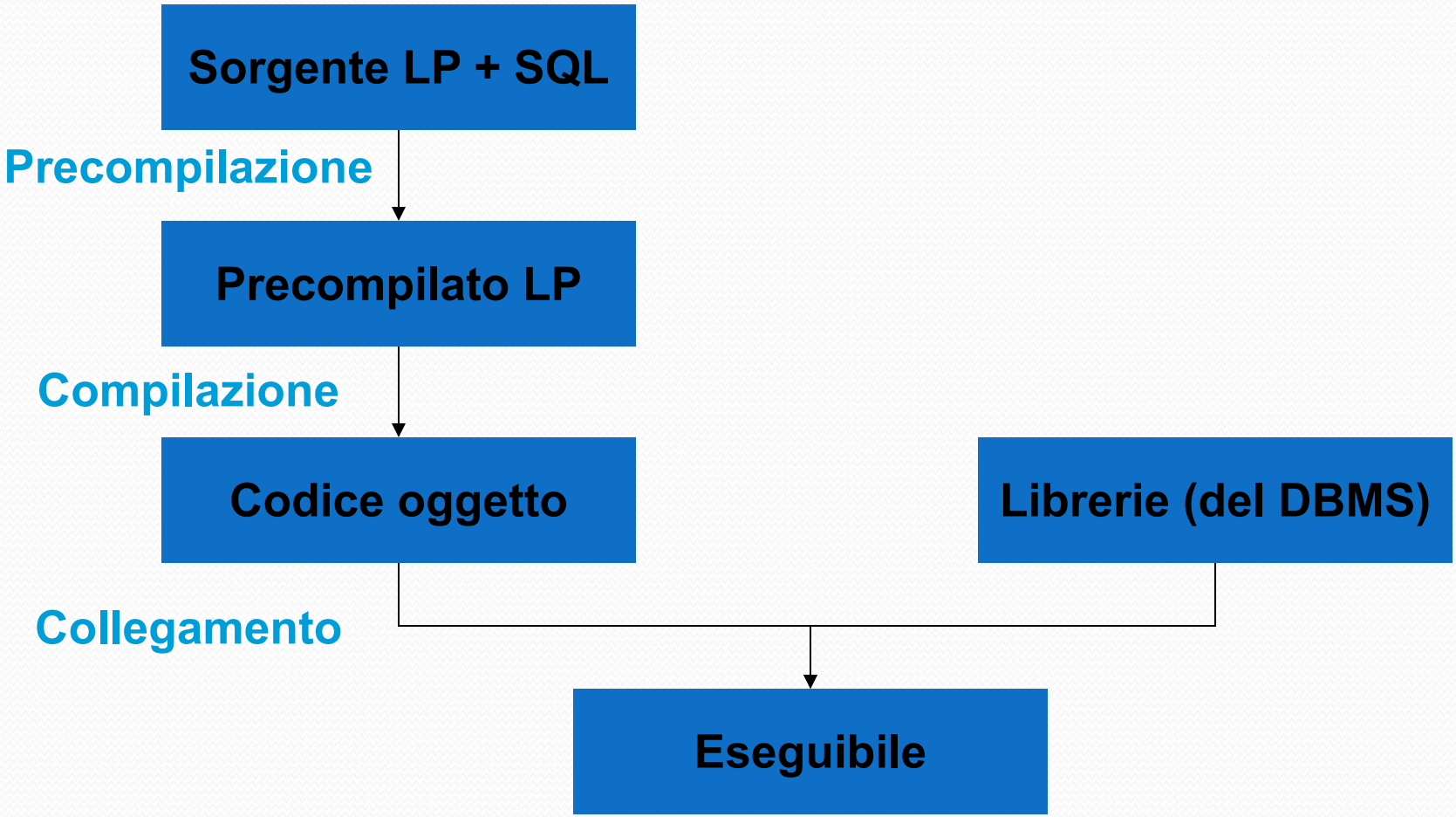
- I primi due approcci sono sicuramente i più comuni, in quanto molte applicazioni scritte in linguaggi di tipo general-purpose richiedono l'accesso ad una base di dati.
  - Esiste il *conflitto di impedenza*.

- Il terzo approccio è invece più diffuso per applicazioni che hanno un'interazione molto forte con la base di dati e non presenta problemi di conflitti di impedenza, in quanto è il linguaggio stesso che colma il gap tra la tecnologia dei database e le caratteristiche di un linguaggio procedurale.

SQL immerso

- Istruzioni SQL *"immerse"* nel programma scritto nel linguaggio *"ospite"*.
- Un'istruzione SQL incapsulata si distingue dalle altre istruzioni premettendo le parole chiave **EXEC SQL** e termina con ";" oppure **END-EXEC**.
- Un **precompilatore o preprocessore** (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 **A**PPLICATION **P**ROGRAMMING **I**NTERFACE (API) del DBMS).

SQL immerso, fasi



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 personale@unisa;
  if (SQLCODE != 0) {
    printf("Connessione al DB non riuscita\n"); }
  else {
    EXEC SQL insert into Dipartimento
           values(:NomeDip,:CittaDip,:NumeroDip);
    EXEC SQL DISCONNECT ALL;
  }
}
```

Costrutti del linguaggio

- `EXEC SQL CONNECT`
  - Connessione alla base di dati.
- `EXEC SQL CONNECT TO <nome server>`
  `AS <nome connessione>`
  - Stabilisce la connessione al server.
- `EXEC SQL DISCONNECT <nome connessione>`
  - Per terminare la connessione (ALL per tutte).
- `EXEC SQL BEGIN/END DECLARE SECTION`
  - Sezione per la dichiarazione di variabili.
- `EXEC SQL`
  - Comando SQL.

Costrutti del linguaggio (2)

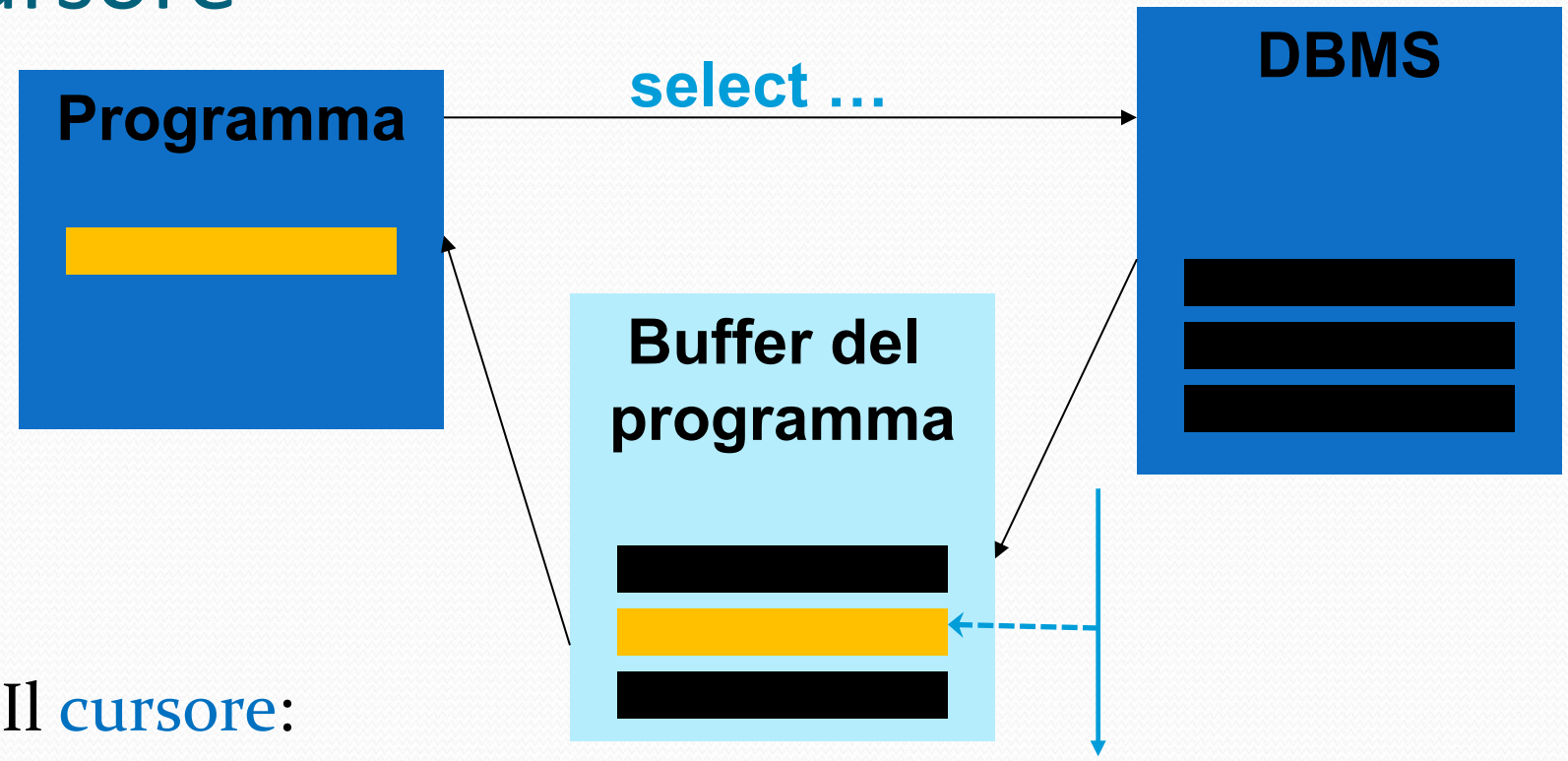
- La comunicazione dei risultati elaborativi dal DBMS all'applicazione avviene a mezzo di apposite **variabili condivise** (e.g., SQLCODE, SQLSTATE).
- **`SQLCODE`** è una variabile che mantiene il codice di errore dell'ultimo comando SQL eseguito:
  - zero: successo;
  - altro valore: errore o anomalia;
- **`SQLSTATE`** fornisce alla applicazione uno schema che rappresenta le condizioni di successo, di warning e di errore dell' ultimo comando SQL eseguito.
  - È un array di cinque caratteri.
- Le variabili del programma possono essere usate come "parametri" nelle istruzioni SQL (precedute da "**:**") dove sintatticamente sono ammesse costanti.

Interrogazioni in SQL immerso: conflitto di impedenza

- Il risultato di una SELECT è costituito da zero o piú *n*-ple:
  - **Zero o una:** *ok* (l'eventuale risultato puó essere gestito in un record).
  - **Piú *n*-ple:** *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



- Il cursore:
  - Accede a tutte le *n*-ple di una interrogazione in modo globale (tutte insieme o a blocchi – è il DBMS che sceglie la strategia efficiente).
  - Trasmette le *n*-ple al programma una alla volta.

Operazioni sui cursori

- Definizione del cursore:

```
DECLARE NomeCursore [scroll] CURSOR FOR Select …
```

- Preleva il risultato dell'interrogazione:

```
OPEN NomeCursore
```

- Utilizzo dei risultati (una *n*-pla alla volta):

```
FETCH [Posizione FROM] NomeCursore INTO Variabili
```

- Disabilita il cursore:

```
CLOSE CURSOR NomeCursore
```

- Accesso alla *n*-pla corrente (di un cursore su singola relazione a fini di aggiornamento):

```
CURRENT OF NomeCursore
```

  nella clausola **where** dello statement SQL:

```
EXEC SQL UPDATE emp SET sal = :new_salary WHERE
CURRENT OF NomeCursore;
```

SELECT senza cursori

```
EXEC SQL SELECT V.vnome, V.età
INTO :c_vnome, :c_età
FROM Velisti V
WHERE V.vid = :c_vid;
```

- Questa select restituisce al più una sola tupla:
  - Non è necessario definire un cursore.

Un *esempio* di utilizzo del cursore

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION;
char c_vnome[20];
short c_minesperienza;
float 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;
EXEC SQL OPEN vinfo;
do {
  EXEC SQL FETCH vinfo INTO :c_vnome, :c_età;
  printf("%s ha %d anni\n", c_vnome, c_età);
} while (SQLSTATE != "02000");
EXEC SQL CLOSE vinfo;
```

Un *esempio* di utilizzo del cursore (2)

```
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;
```

Il comando *fetch*

- Il parametro *Posizione* permette di specificare quale riga dovrà essere oggetto dell'operazione di fetch:
  - **`NEXT`** (la riga successiva alla corrente);
  - **`PRIOR`** (la riga precedente alla corrente);
  - **`LAST`** (l'ultima riga del risultato);
  - **`ABSOLUTE <espressione intera>`** (la riga che compare in posizione *i*-esima nella tabella, se *i* è il risultato dell'espressione);
  - **`RELATIVE <espressione intera>`** (come absolute, solo che viene preso come punto di riferimento la posizione corrente);
- Queste opzioni sono utilizzabili se nella definizione del cursore è specificata l'opzione **`scroll`**.

Variabili

- Per riconoscere quando un cursore ha terminato di estrarre tutte le righe, si fa uso del campo **`sqlcode`** della variabile predefinita **`sqlca`**.

Un *esempio* di utilizzo del cursore (3)

```
void VisualizzaStipendiDipart(char NomeDip[])
{
   EXEC SQL BEGIN DECLARE SECTION;
   char Nome[20], Cognome[20];
   long int Stipendio;
   EXEC SQL END DECLARE SECTION;

   EXEC SQL DECLARE ImpDip CURSOR FOR
      SELECT Nome, Cognome, Stipendio
      FROM Impiegato
      WHERE Dipart = :NomeDip;
   EXEC SQL OPEN ImpDip;
   EXEC SQL FETCH ImpDip INTO :Nome, :Cognome, :Stipendio;
   printf("Dipartimento %s\n",NomeDip);
   while (sqlca.sqlcode == 0) {
         printf("Nome e cognome dell'impiegato: %s %s\n",Nome,Cognome);
         printf("Attuale stipendio: %d\n",Stipendio);
         EXEC SQL FETCH ImpDip INTO :Nome, :Cognome, :Stipendio;
   }
   EXEC SQL CLOSE CURSOR ImpDip;
}
```

SQL dinamico

- Quando le applicazioni non conoscono a tempo di compilazione gli statemente SQL da eseguire, è necessario utilizzare l' **SQL dinamico**.
  - Il linguaggio *"SQL embedded"* risulta un approccio di programmazione delle basi di dati **statico** in quanto il testo dell'interrogazione è scritto all'interno del programma e non può essere cambiato senza ricompilare o rielaborare il codice sorgente.

- **Problema:** gestire e trasferire i parametri tra l'applicazione e l'environment SQL.

Esecuzione dell'SQL dinamico

- Riguardo l'SQL statico un comando viene preprocessato e quindi non deve essere analizzato ed ottimizzato ogni volta:
  - Si ottengono ottime prestazioni.

- L'SQL dinamico mette a disposizione due diverse modalità di interazione:
  - Si può immediatamente eseguire l'interrogazione.
  - L'esecuzione dell'interrogazione può avvenire in due fasi:
    1. Analisi.
    2. Esecuzione dell'interrogazione.

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***]

Esempio esecuzione immediata

Istruzione_sql = "delete from impiegato
where nome='Mario' ";

....

EXEC SQL execute immediate **:Istruzione_sql**;

Esempio di istruzione preparata

- Preparazione:
  EXEC SQL prepare **:comando**
  FROM "select Città from Dipartimento where Nome= **?**";

- Esecuzione:
  EXEC SQL execute **:comando** into **:citta** using **:dipartimento**;

  - Se la variabile *dipartimento* contiene la stringa *'produzione'*, l'effetto di questo comando è di eseguire la query:

    SELECT Città from Dipartimento
    WHERE Nome ='produzione';

Call Level Interface

- Indica genericamente interfacce che permettono di inviare richieste a DBMS per mezzo di parametri trasmessi a funzioni.
- Questo approccio fornisce delle apposite API per l'accesso ad una base di dati da programmi applicativi.
  - Vi sono funzioni per la connessione al database, per l'esecuzione dell'interrogazione, per l'aggiornamento dei dati, …
  - I comandi effettivi di interrogazione ed aggiornamento del db e, qualsiasi altra informazione necessaria, vengono inclusi come parametri nelle chiamate a funzione.

- Standard SQL/CLI ('95 e poi parte di SQL:1999).
- ODBC: implementazione proprietaria di SQL/CLI.
- JDBC: una CLI per il mondo Java.

Uso generale

1. Si utilizza un servizio della CLI per creare una connessione con il DBMS.
2. Si invia sulla connessione un comando SQL che rappresenta la richiesta.
3. Si riceve come risposta del comando una struttura relazionale in un opportuno formato:
   a. La CLI dispone di un insieme di primitive che permettono di descrivere e analizzare la struttura del risultato del comando.
4. Al termine della sessione di lavoro, si chiude la connessione e si rilasciano le risorse usate per la gestione della comunicazione.

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.

Linguaggi di programmazione per basi di dati

- In tale approccio, vengono definiti appositi linguaggi di programmazione “compatibili” con il modello logico della base di dati e con il linguaggio di interrogazione SQL.
  - Le istruzioni classiche SQL di interazione col database sono arricchite ed estese dalle classiche istruzioni tipiche di un linguaggio di programmazione (definzione di tipi e variabili, strutture di controllo, statement e costrutti iterativi, definizione di procedure e funzioni, etc..), al fine di ottenere un vero e proprio linguaggio di programmazione completo per le basi di dati.
  - Un esempio di linguaggio di questo tipo è il PL/SQL dell’Oracle che integra la potenza e la flessibilità di SQL con i costrutti tipici di un linguaggio procedurale.

Esempio di PL/SQL

```
DECLARE
   numero NUMBER(1);

BEGIN
   SELECT COUNT(numero_civico) INTO numero FROM indirizzi WHERE
   nome='Mario';
   IF numero > 1 THEN
      dbms_output.put_line('Ci sono' || numero || 'indirizzi.');
   ELSIF numero = 1 THEN
      dbms_output.put_line('Ci sta 1 indirizzo.');
   ELSE
        dbms_output.put_line('Non ci sono indirizzi.');
   END IF;
END;
```

JDBC

- Una API 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

- Esistono quattro tipi di driver (chiamati, in modo molto anonimo, *tipo 1, tipo 2, tipo 3, tipo 4*):

  (A titolo di curiosità: può bastare il primo tipo)

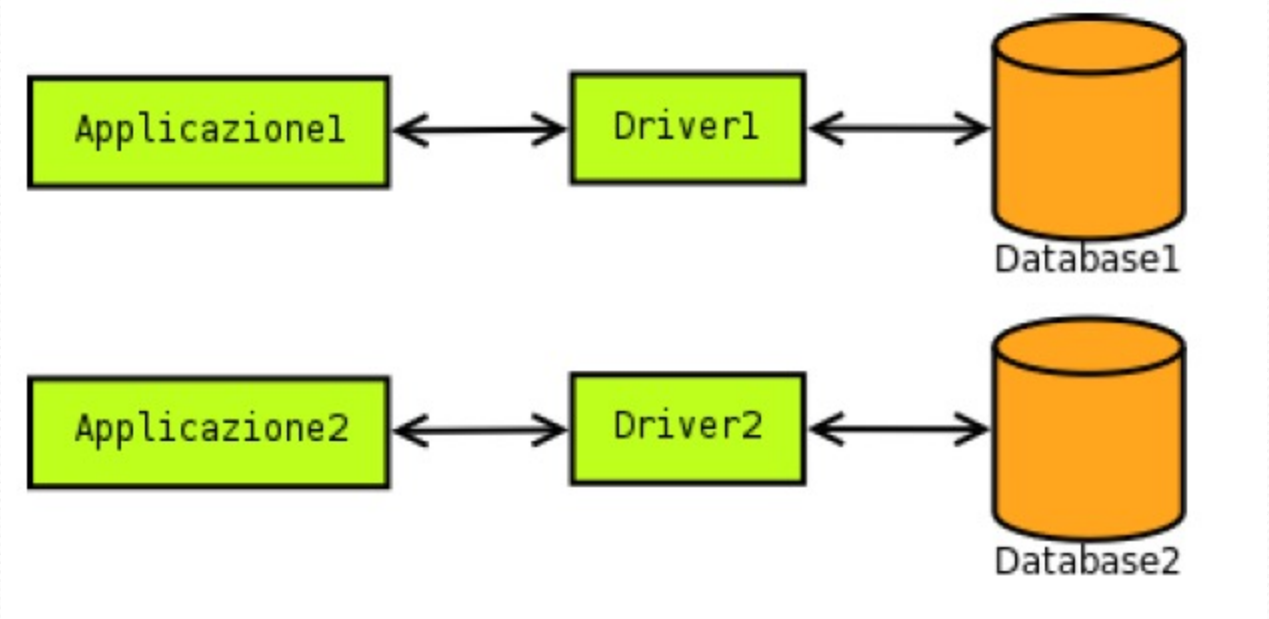
  1. **Bridge JDBC-ODBC:** richiama un driver ODBC, che deve essere disponibile sul client; è comodo ma potenzialmente inefficiente.
  2. **Driver nativo sul client:** richiama un componente proprietario (non necessariamente Java) sul client.

  (Queste due soluzioni non sono portabili)

  3. **Driver puro Java con server intermedio ("middleware server"):** comunica via protocollo di rete con il server intermedio, che non deve risiedere sul client.
  4. **Driver puro Java, con connessione al DBMS:** interagisce direttamente con il DBMS.

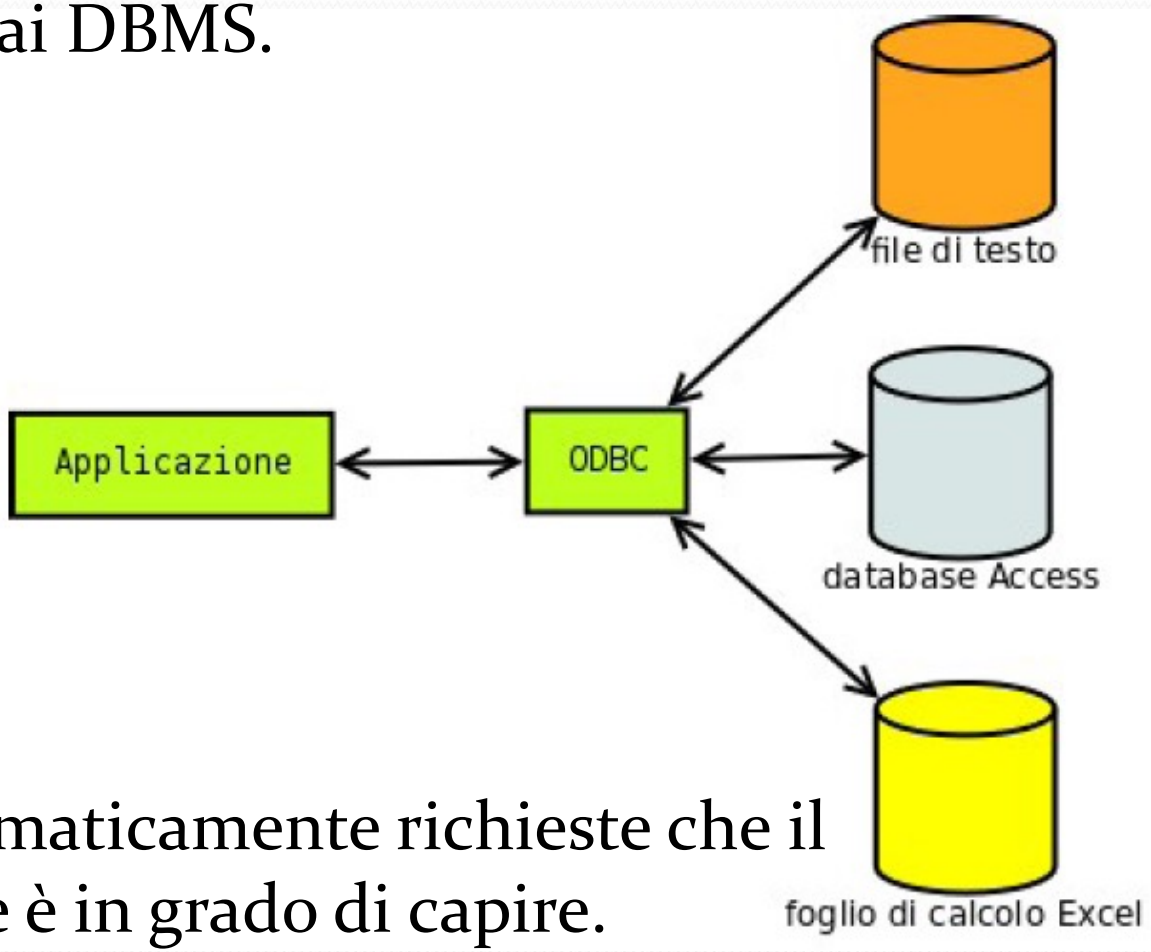
Origini

- In “origine” la comunicazione fra applicativo e database era piuttosto difficoltosa:
  - c'era infatti bisogno di un driver specifico, cioè di una API specifica e proprietaria, per poter accedere ad ogni diverso database.



ODBC (Open Database Connectivity) Driver

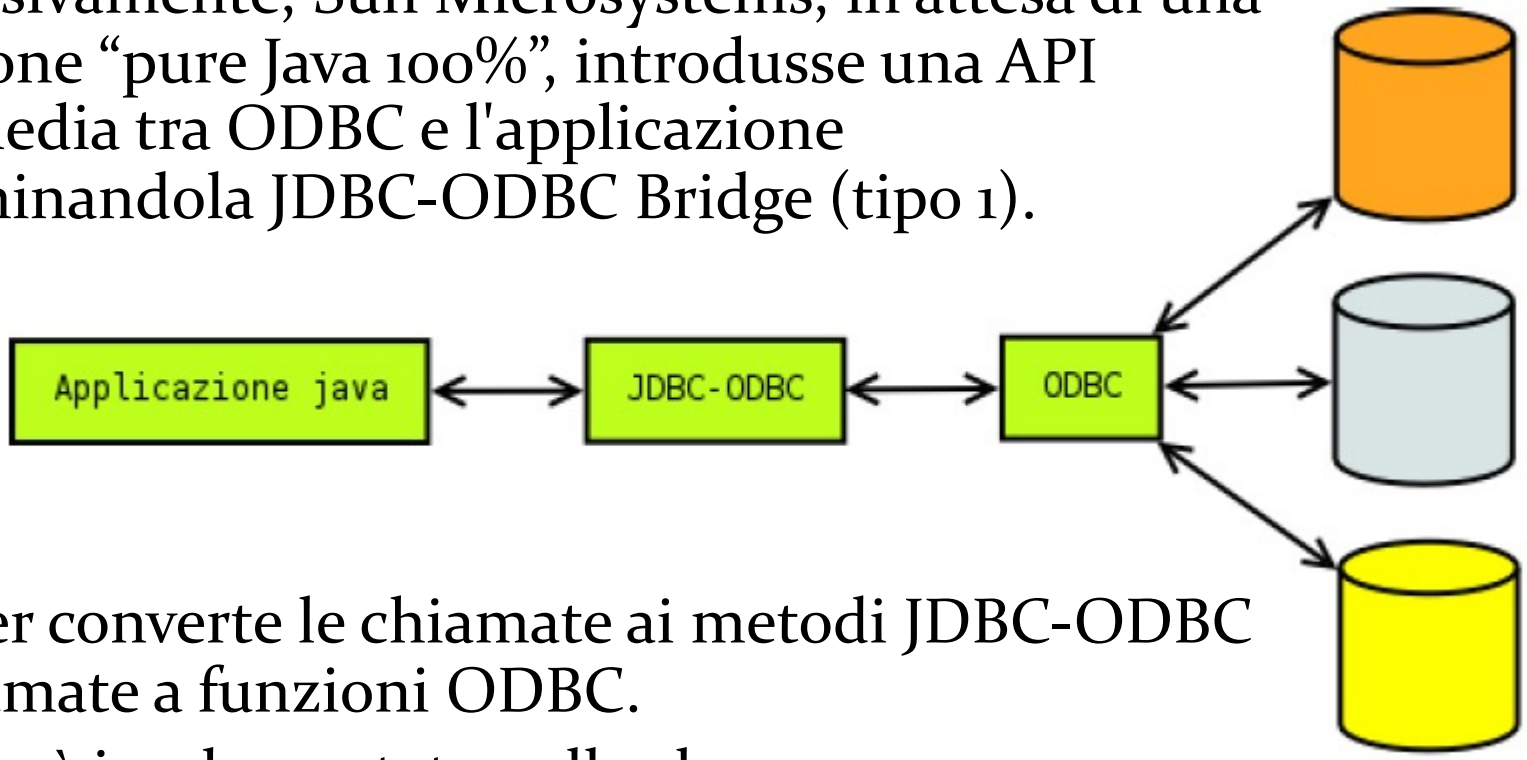
- Nel 1991, Microsoft progettò ODBC, una API standard per la connessione ai DBMS.



- ODBC genera automaticamente richieste che il sistema di database è in grado di capire.

JDBC-ODBC Bridge

- Successivamente, Sun Microsystems, in attesa di una soluzione “pure Java 100%”, introdusse una API intermedia tra ODBC e l'applicazione denominandola JDBC-ODBC Bridge (tipo 1).



- Il driver converte le chiamate ai metodi JDBC-ODBC in chiamate a funzioni ODBC.
- Il driver è implementato nella classe *jdbc.odbc.JdbcOdbcDriver ed è fornito con Java 2 SDK* Standard Edition.

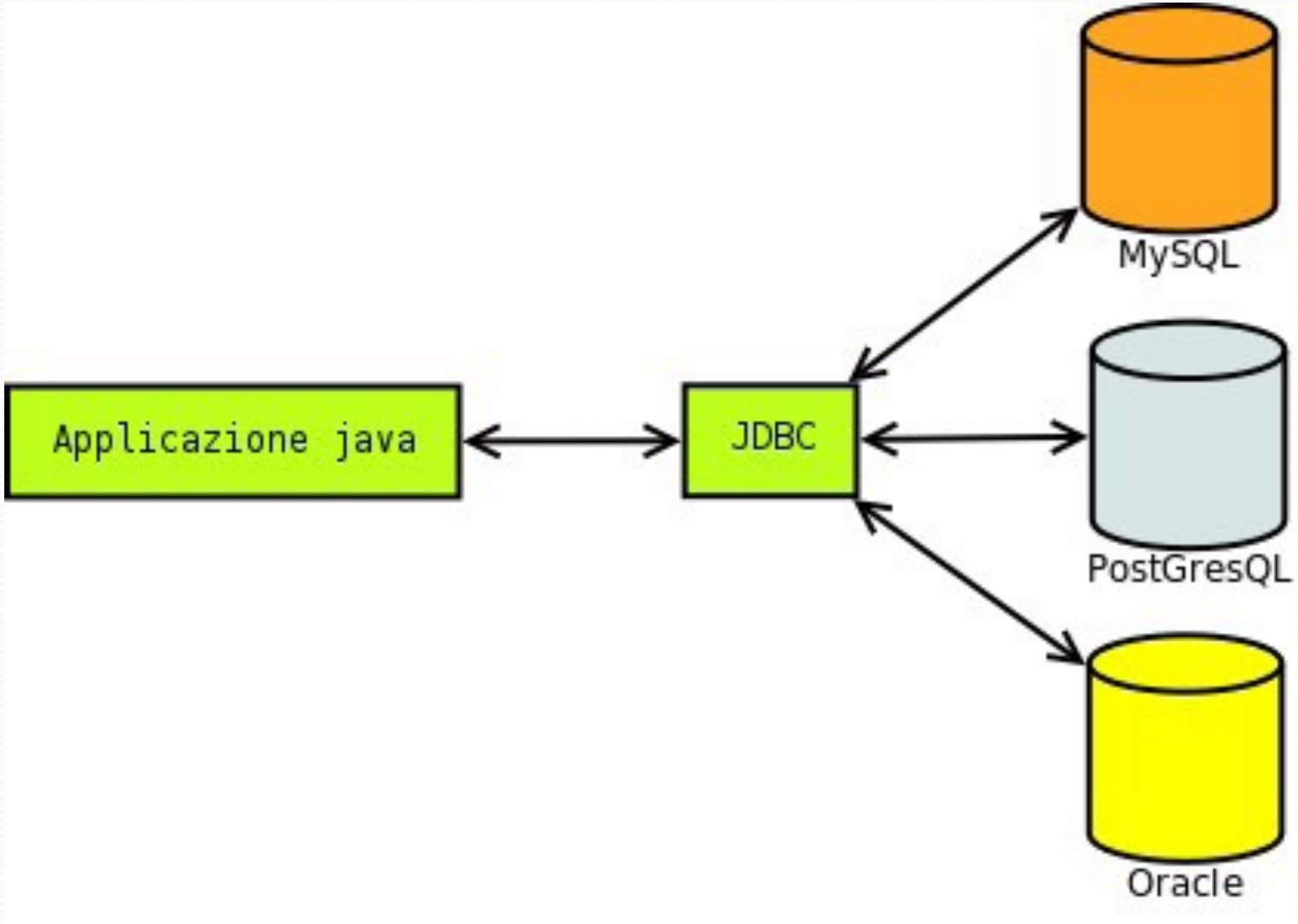
JDBC (Java DataBase Connectivity) Driver

- *JDBC (tipo 4) è una API per database interamente scritta in java* localizzata nel package ***java.sql***.

- *JDBC API fornisce metodi e interfacce per interrogare e* modificare i dati ed è Object Oriented.

- La piattaforma Java 2 Standard Edition contiene le API JDBC, insieme all'implementazione di un bridge JDBC-ODBC, che permette di connettersi a database relazionali che supportano ODBC (per esempio Access).

JDBC (Java DataBase Connectivity) Driver (2)



Il funzionamento di JDBC, in breve

1. Caricamento del driver.
2. Apertura della connessione alla base di dati.
3. Richiesta di esecuzione di istruzioni SQL.
4. Elaborazione dei risultati delle istruzioni SQL.

Un programma Java con JDBC (via ODBC)

```
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");
     }
   }
}
```

Connessione.java (MySQL)

```
import java.sql.*;

class Connessione {
   public static void main(String args[]) throws Exception {
      try {
          Class.forName("com.mysql.cj.jdbc.Driver");

          String url = "jdbc:mysql://localhost:3306/mysql";
          Connection con = DriverManager.getConnection(url, "root", "pwd");

          System.out.println("Connessione OK \n");
          con.close();
      }
      catch(Exception e) {
         System.out.println("Connessione Fallita \n");
         System.out.println(e);
      }
   }
}
```