BASI DI DATI

SQL nei linguaggi di programmazione

Polese G. Caruccio L. Breve B.

a.a. 2023/2024

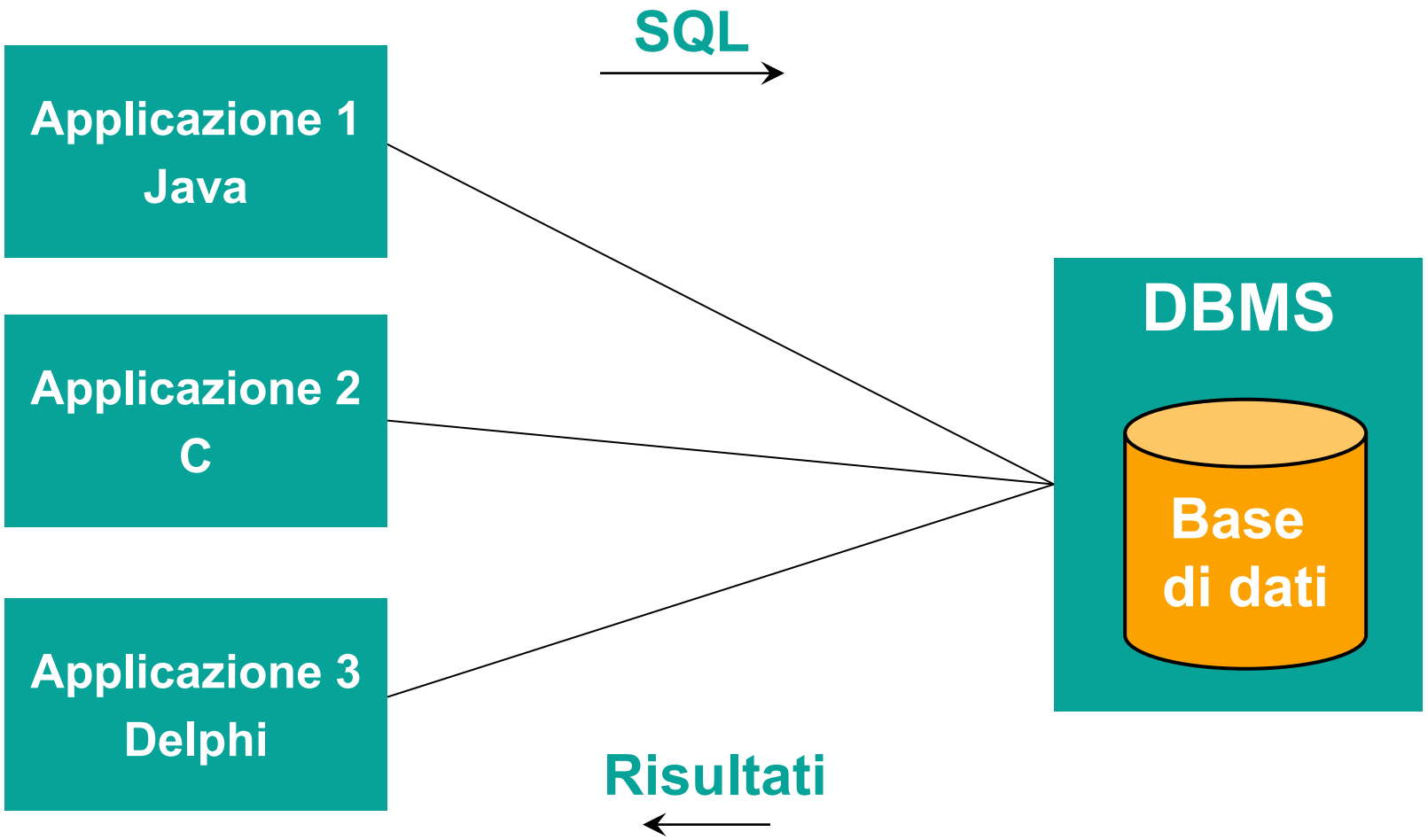
SQL e applicazioni

- In applicazioni complesse, l'utente non vuole eseguire comandi SQL, ma programmi, con poche scelte
- SQL non basta, sono necessarie 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 e linguaggi di programmazione

- Le applicazioni sono scritte in
  - Linguaggi di programmazione tradizionali:
    - ✓Cobol, C, Java, Fortran
  - Linguaggi “ad hoc”, proprietari e non:
    - ✓PL/SQL, Informix4GL, Delphi

Applicazioni ed SQL: architettura



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

- 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)
- Tipi di base:
  - Linguaggi: numeri, stringhe, booleani
  - SQL: CHAR, VARCHAR, DATE, ...
- Costruttori di tipo:
  - Linguaggio: dipende dal paradigma
  - SQL: relazioni e ennuple

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

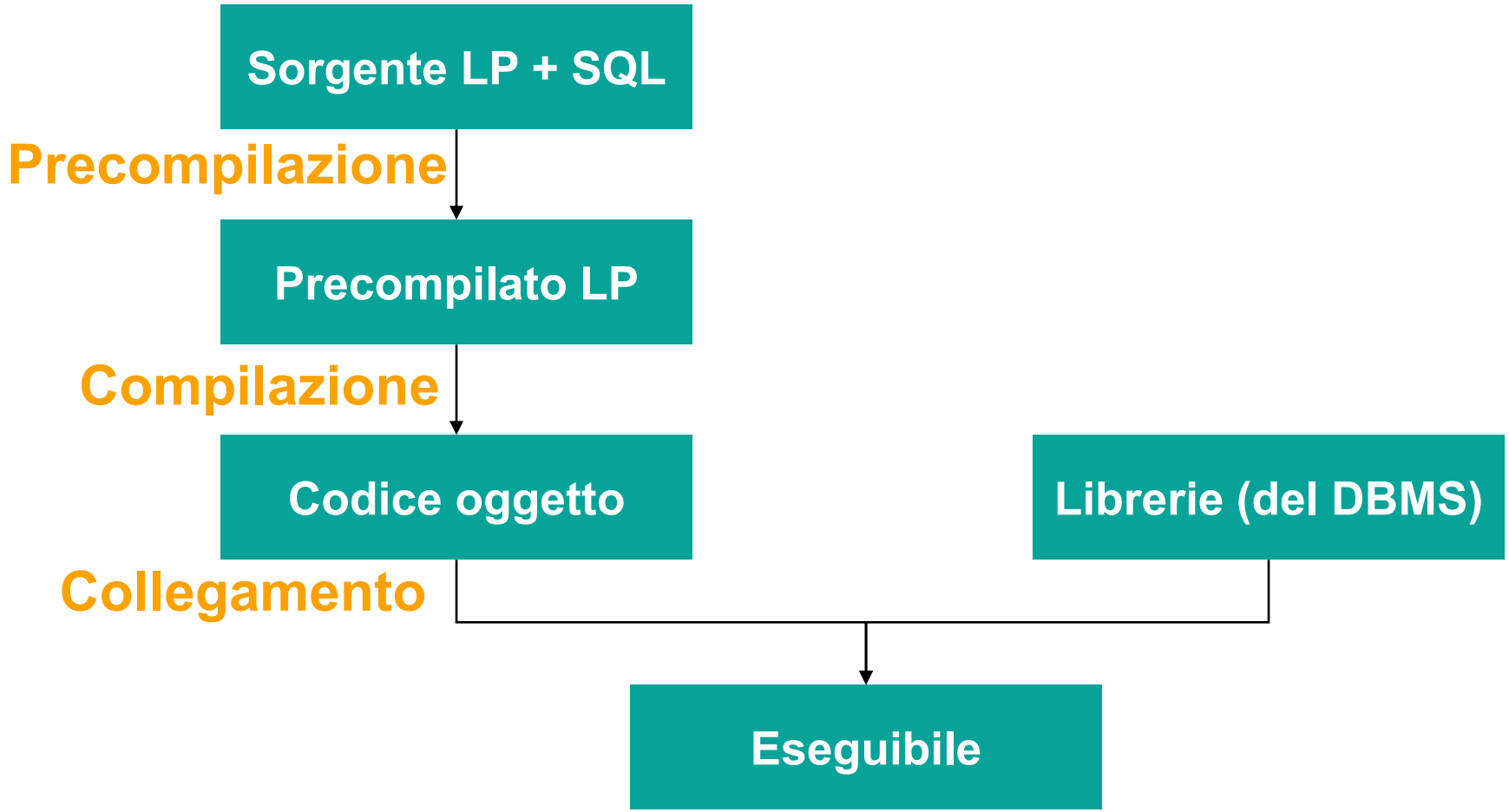
- Le istruzioni SQL sono “immerse” nel programma scritto in linguaggio “ospite”
- Un precompilatore del DBMS analizza il programma e lo traduce in un sorgente nel linguaggio ospite, sostituendo istruzioni SQL con chiamate a funzioni di un’API del DBMS

```
#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
- `sqlca` è una struttura dati per la comunicazione fra programma e DBMS
- `sqlcode` è un campo di `sqlca` che mantiene il codice d’errore dell’ultimo comando SQL eseguito:
  - Zero: successo
  - Altro valore: errore o anomalia

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;
}
```

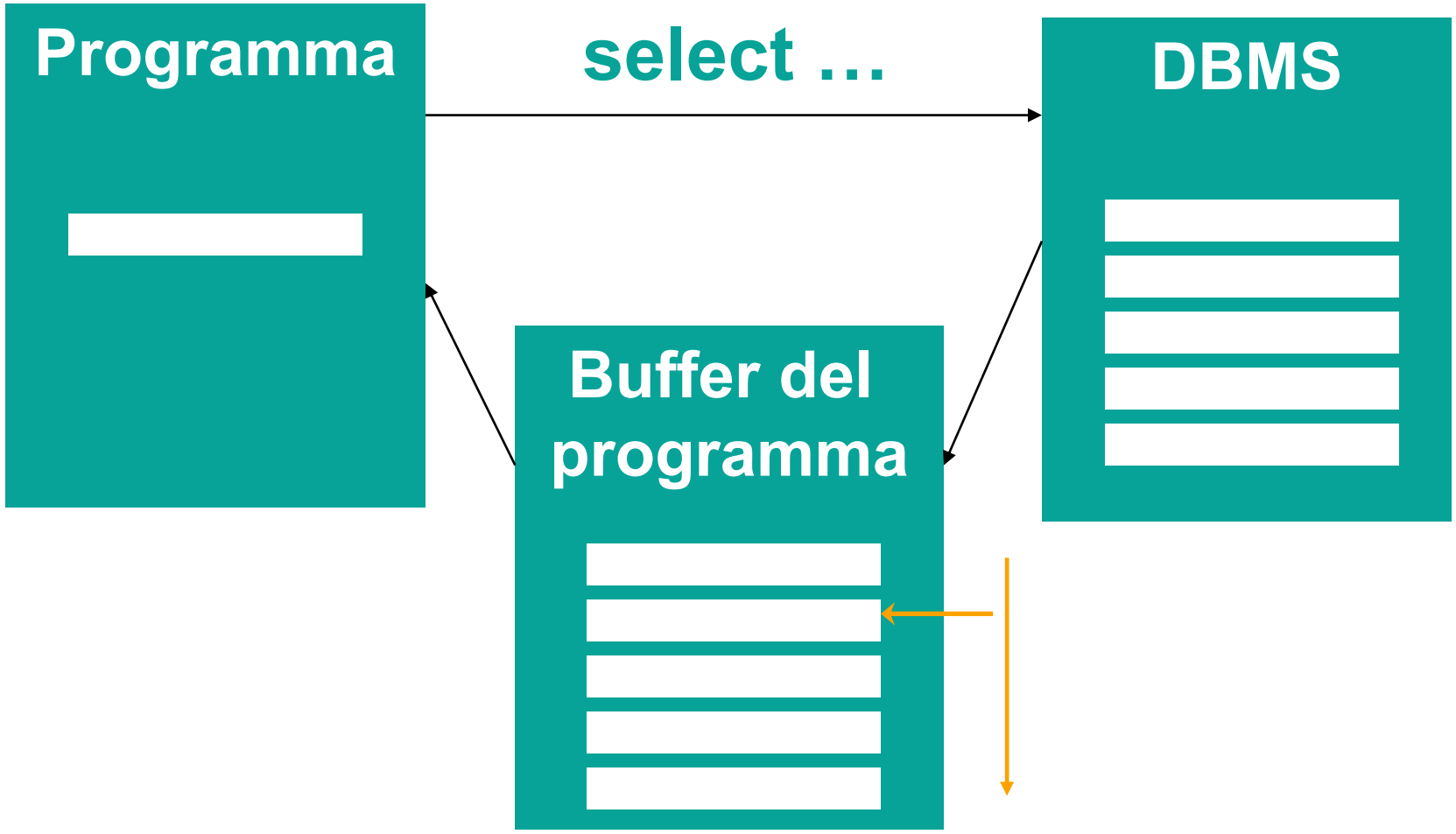
Note

Il precompilatore è specifico della seguente combinazione: linguaggio-DBMS-sistema operativo

Conflitto d'impedenza in SQL immerso

- 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 – il DBMS 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

```
char citta[20], nome[20];
long reddito, aumento;
printf(“nome della città?”);
scanf(“%s”,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)
{
  printf("nome della persona: %s, aumento?", nome);
  scanf(“%l”, &aumento);
  EXEC SQL UPDATE PERSONE
     SET REDDITO = REDDITO + :aumento
     WHERE CURRENT OF P
  EXEC SQL FETCH P INTO :nome, :reddito
}
EXEC SQL CLOSE CURSOR P
```

```
void VisualizzaStipendiDipart(char NomeDip[])
{
  char Nome[20], Cognome[20];
  long int Stipendio;
  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 (sqlcode == 0)
  {
      printf("Nome e cognome dell'impiegato: %s
                  %s",Nome,Cognome);
      printf("Attuale stipendio: %d\n",Stipendio);
      EXEC SQL fetch ImpDip into :Nome, :Cognome,
                                          :Stipendio;
  }
  EXEC SQL close cursor ImpDip;
}
```

Cursori, commenti

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

```
SELECT Nome, Cognome
    INTO :nomeDip, :cognomeDip
FROM Dipendente
WHERE Matricola = :matrDip;
```

- I cursori possono ricondurre 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 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)

SQL dinamico

- Le operazioni SQL possono essere:
  - eseguite immediatamente

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

    `prepare` *CommandName* `from` SQL*Statement*

    e poi eseguite (anche più volte):

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

Call Level Interface

- Interfacce che permettono di inviare richieste a DBMS per mezzo di parametri trasmessi a funzioni
- Standard SQL/CLI (’95 e poi parte di SQL:1999)
- ODBC: implementazione proprietaria di SQL/CLI
- JDBC (Java Database Connectivity): 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

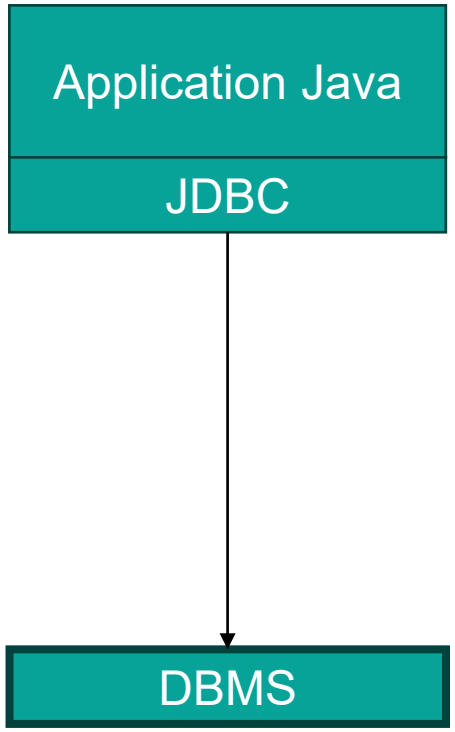
JDBC

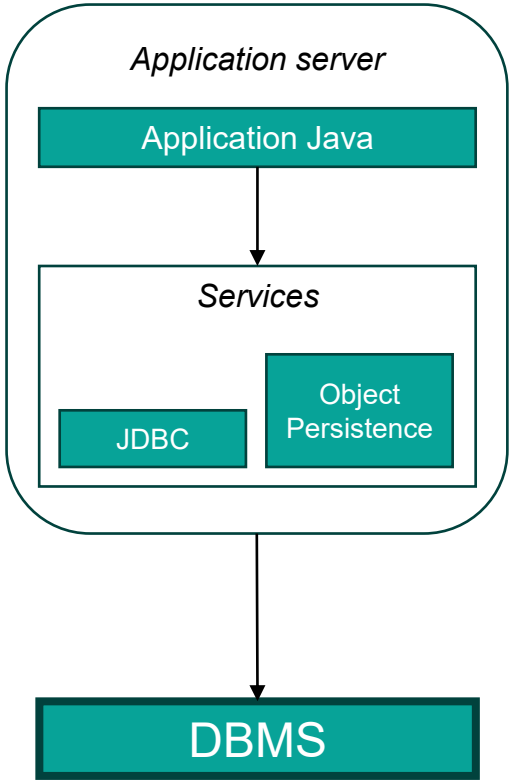
- Un’API (Application Programming Interface) di Java (intuitivamente: una libreria) per l'accesso a basi di dati, in modo indipendente dalla specifica tecnologia
- Ciascun DBMS fornisce un driver specifico che:
  - Viene caricato a run-time
  - Traduce le chiamate alle funzioni JDBC in chiamate alle funzioni del DBMS

Modalità di accesso al database

- JDBC è generalmente usato in due architetture di sistema:
  - L'applicazione Java "parla" direttamente col database
  - Un livello intermedio invia i comandi SQL al database, eseguendo controlli su accessi e aggiornamenti
    - ✓ Tutto il controllo sull'accesso ai dati è affidato allo strato centrale (middle-tier), che può avvalersi di un middleware o un application server
    - ✓ Può avere vantaggi in termini di scalabilità

Modalità di accesso al database





Architettura di JDBC

- Un sistema che usa JDBC ha quattro componenti principali
  - Applicazione: inizia e termina la connessione, imposta le transazioni, invia comandi SQL, recepisce risultati. Tutto avviene tramite l'API JDBC (nei sistemi three tiers se ne occupa lo strato intermedio)
  - Gestore di driver: carica i driver, passa le chiamate al driver corrente, esegue controlli sugli errori
  - Driver: stabilisce la connessione, inoltra le richieste e restituisce i risultati, trasforma dati e formati di errore dalla forma dello specifico DBMS allo standard JDBC
  - Sorgente di dati: elabora i comandi provenienti dal driver e restituisce i risultati

Caratteristiche di JDBC

- Esecuzione di comandi SQL
  1. DDL (Data Definition Language)
  2. DML (Data Manipulation Language)
- Manipolazione dei risultati tramite result set (una forma di cursore)
- Reperimento di metadati
- Gestione di transazioni
- Gestione di errori ed eccezioni
- Definizione di stored procedure scritte in Java (supportate da alcuni DBMS)

Software: cosa occorre

1. Piattaforma Java
   - Java Standard Edition
   - Il package `java.sql`: funzionalità di base di JDBC
   - Il package `javax.sql`: funzionalità più avanzate (ad es. utili per la gestione di pooling di connessioni o per la programmazione a componenti)
2. Driver JDBC per il DBMS a cui ci si vuole connettere
3. DBMS

Driver JDBC

- Il driver JDBC per un certo DBMS viene distribuito in genere dalla casa produttrice del DBMS.
- È di fatto una libreria Java che va collegata in fase di esecuzione dell'applicativo. Questo va fatto:
  1) Settando opportunamente le variabili d'ambiente
  2) Configurando l'ambiente di sviluppo Java che state usando (caricare il .jar del driver nel progetto).
- Nel caso di MYSQL 5.0, il driver è il file `mysql-connector-java-5.0.4-bin.jar` scaricabile dal sito `www.mysql.com`

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 con JDBC

```
import java.sql.*;
public class PrimoJDBC {
   public static void main(String[] arg){
     Connection con = null ;
     try {
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://localhost:3306/corsi";//corsi nome schema
        String username = "<username>"; String pwd = "<pwd>"
        con = DriverManager.getConnection(url,username,pwd);
        }
     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");
     }
   }
}
```

Preliminari

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

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

- Il driver deve essere caricato (trascuriamo i dettagli)

```
Class.forName("com.mysql.jdbc.Driver");
```

Preliminari

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

```
String url =  "jdbc:mysql://localhost:3306/corsi";
String username = "<username>";
String pwd = "<pwd>";

con = DriverManager.getConnection(
                                    url,username, pwd);
```

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);
}
```

Statement

- Interfaccia i cui oggetti consentono di inviare, tramite una connessione, istruzioni SQL e di ricevere i risultati
- 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`

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()`
  - `get`*XXX*`(`*posizione*`)`
    - es: `getString(3); getInt(2)`
  - `get`*XXX*`(`*nomeAttributo*`)`
    - es: `getString("Cognome");getInt("Codice")`

Specializzazioni di Statement

- `PreparedStatement` premette 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 `set`*`XXX`*`( , )` permettono di definire i parametri
- `CallableStatement` premette 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("com.mysql.jdbc.Driver");
            String url = "jdbc:mysql://localhost:3306/corsi";
            String username = "<username>";
            String pwd = "<pwd>"
            con =
            DriverManager.getConnection(url,username,pwd);
        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("com.mysql.jdbc.Driver");
            String url = "jdbc:mysql://localhost:3306/corsi";
            String username = "<username>";
            String pwd = "<pwd>"
            con =
            DriverManager.getConnection(url,username,pwd);
            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
  - Aggiornamento dei ResultSet
  - Richiesta di metadati
  - Festione di transazioni

Transazioni in JDBC

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

  ```
  setAutoCommit(boolean autoCommit)
  ```

- `con.setAutoCommit(true)`
  - Ogni operazione è una transazione
- `con.setAutoCommit(false)`
  - Gestione delle transazioni da programma

    ```
    con.commit()
    con.rollback()
    ```

  - Non c'è `begin transaction`

Procedure

- SQL:1999 (come già SQL-2) permette la definizione di procedure e funzioni (chiamate genericamente “stored procedures”)
- Le stored procedures sono parte dello schema

```
procedure AssignCity(:Dep char(20), :City char(20))
update Department
set City = :City
where Name = :Dep
```

- Lo standard prevede funzionalità limitate e non è molto recepito
- Molti sistemi offrono estensioni ricche (ad esempio Oracle PL/SQL e Sybase-Microsoft Transact SQL)

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;
```