Sviluppo di Applicazioni per Basi di Dati Accoppiamento esterno

Integrazione di due linguaggi

- SQL
- linguaggio di programmazione generico

Flusso di esecuzione

Passi fondamentali

1. **Connessione alla base di dati**

   per specificare il server a cui connettersi servono

   - stringa di locazione
   - credenziali di accesso (nome utente e password)

   Privo di senso per accoppiamento interno, perché si esegue da dentro il DBMS

2. **Esecuzione dei comandi SQL**

   - discussa dopo

3. **Chiusura della connessione**

   - Da fare quando l'interazione con la base di dati è terminata
     - i DBMS hanno un massimo di connessioni contemporaneamente attive
     - chiudere una connessione libera risorse
     - in genere, la terminazione di un'applicazione comporta la chiusura automatica di tutte le connessioni aperte

Esecuzione di comandi SQL

Tre passi

- Preparazione del comando
  - generazione delle strutture dati necessarie per la comunicazione con il DBMS
  - eventuale compilazione ed ottimizzazione del comando da parte del DBMS
- Esecuzione del comando
  - sul DBMS a seguito di una chiamata dell'applicazione
  - spesso disaccoppiata dalla fase di preparazione
- Manipolazione del risultato
  - manipolazione del risultato tradotto nelle strutture del linguaggio di programmazione

Disaccoppiamento preparazione/esecuzione

- Molto utile se si esegue lo stesso comando SQL più volte
  - eventualmente per diversi valori dei parametri
- Esempio: applicazione che vuole determinare più volte il numero medio dei noleggi per un particolare genere di film, preso in input
  - Il comando SQL da eseguire è sempre lo stesso
  - Nella fase di preparazione si determina il piano più efficiente per l'esecuzione dell'operazione
    - indipendente dal valore del parametro
  - Tutte le esecuzioni del comando nel contesto della stessa esecuzione dell'applicazione usano lo stesso piano

SQL statico e dinamico

- SQL statico
  - comandi SQL noti già a tempo di compilazione
  - permette preparazione del comando (⇒ efficienza)
  - l'applicazione ha pieno controllo del comando (⇒sicurezza/correttezza)
- SQL dinamico
  - comandi SQL noti solo a tempo di esecuzione
    - es. clausola WHERE che cambia in relazione a valori in input
    - caso limite: il comando è preso tutto o in parte da input
  - apre la porta a
    - errori
    - attacchi
  - facilita lo sviluppo di applicazioni on-line

SQL dinamico - Problematiche

- Maggiori problemi in termini di ottimizzazione, compilazione ed esecuzione
  - ⇒ Alcuni approcci di accoppiamento non ne permettono l'esecuzione
- Il risultato della preparazione di un comando SQL statico
  - si fa una volta e può essere utilizzato più volte in esecuzioni distinte
- Un comando SQL dinamico deve essere preparato **ogni volta**
  - ⇒ aumenta il tempo di risposta

Impedance mismatch

Dovuto alle differenze tra i due linguaggi

- Differenze nei tipi di dato

  necessità di binding tra tipi di dato del linguaggio e di SQL

  - es. date in SQL e C# possono avere granularità diversa ⇒ salvare e recuperare una data sul/dal DB può modificarla
- Differenze nella modalità di elaborazione
  - set-oriented per SQL
  - tuple-oriented per linguaggi di programmazione
- Scarsa integrazione in tecniche di progettazione e sviluppo
  - es. testing

IMPEDENCE MISMATCH - SOLUZIONI

- Tradizionali (recepite dallo standard SQL)
  - estensioni operazionali per armonizzare le differenze (es. cursori)
  - comandi SQL possono essere invocati da linguaggi general-purpose
    - execute
  - ma i comandi SQL sono *opachi* al linguaggio di programmazione
    - no verifiche a compile time ⇒ **integrazione debole**
- Innovative (esterne a SQL, a livello di linguaggio/piattaforma)
  - infrastruttura run-time per gestire dati relazionali come oggetti
    - possibilità di eseguire tipiche operazioni di interrogazione e manipolazione all'interno di linguaggi general purpose ⇒ **integrazione forte**
  - Esempio
    - LINQ in .NET
    - hibernate….

Accoppiamento esterno
Gestione dei risultati

- Il risultato di un'interrogazione SQL (relazione) deve essere inserito in una struttura dati del linguaggio di programmazione
- Se l'interrogazione restituisce solo una tupla
  - è possibile definire variabili di comunicazione in cui inserire i valori degli attributi della tupla restituita
- Se l'interrogazione restituisce più tuple
  - la dimensione del risultato non è nota a compile-time
  - può essere troppo grande per essere caricata in memoria
  - è necessario un meccanismo **(cursore)** per caricare dal risultato residente sul DB alla memoria del programma le tuple una ad una

Librerie di funzioni

- Direttamente o indirettamente l'accesso alla base di dati avviene tramite un'interfaccia chiamata Call Level Interface (CLI)
- La maggioranza dei DBMS offre soluzioni di questo tipo, che però sono proprietarie e quindi vanno bene solo per uno specifico DBMS
- Lo standard SQL prevede SQL/CLI
- Standard de facto: ODBC, JDBC

JDBC/ODBC

- La portabilità a livello di eseguibile è ottenuta attraverso driver
  - del tutto analogo alla gestione delle periferiche da parte del sistema operativo
- Un driver è un programma che traduce tutte le chiamate ODBC/JDBC in chiamate specifiche per un DBMS
  - come un driver per una stampante traduce le chiamate generiche del sistema operativo in quelle specifiche della particolare stampante
- Nel seguito ci concentreremo su JDBC

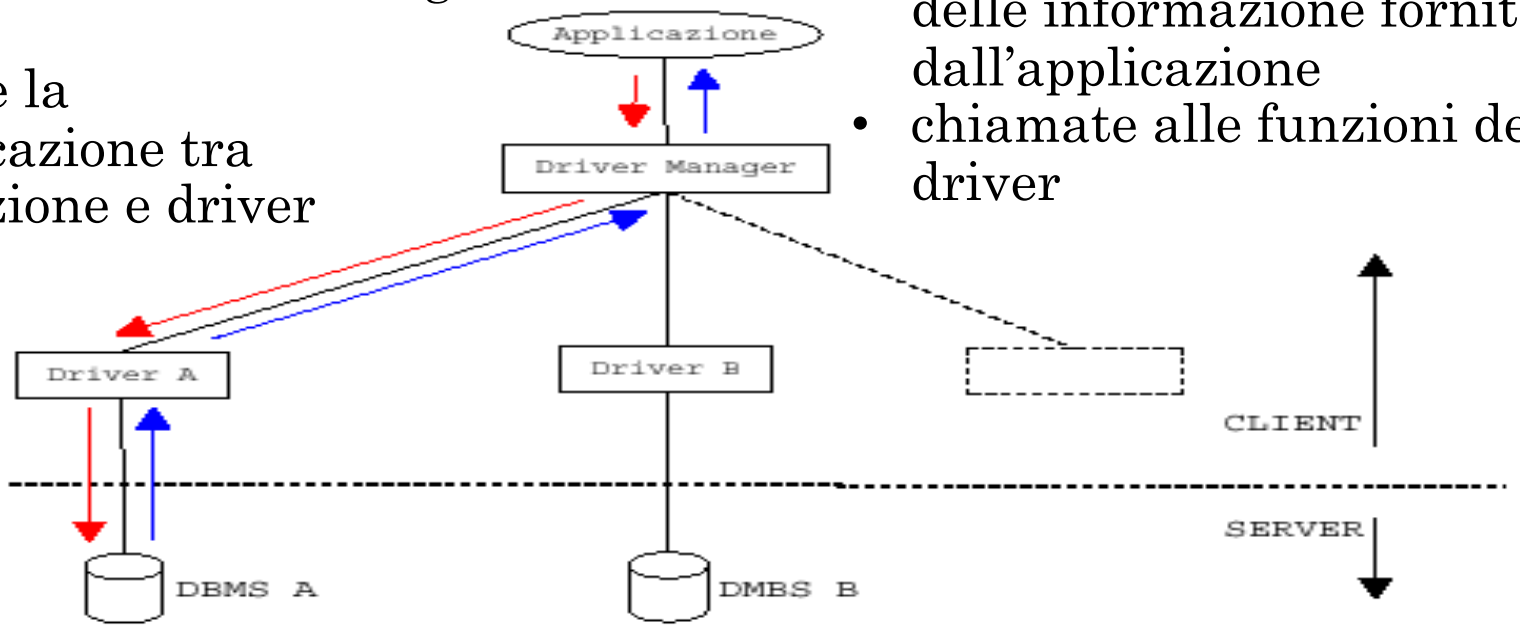
JDBC: Architettura di riferimento

L'applicazione interagisce solo con il driver manager

Risolve problematiche comuni a tutte le applicazioni

- scelta del driver a seconda delle informazione fornite dall'applicazione
- chiamate alle funzioni dei driver

Gestisce la comunicazione tra applicazione e driver



Ricevono sempre e solo richieste nel linguaggio supportato
Eseguono il comando SQL ricevuto dal driver e restituiscono i risultati

JDBC

- JDBC è una API Java standard per interagire con basi di dati
- Cerca di essere il più semplice possibile rimanendo al contempo flessibile
- Permette di ottenere una soluzione “pure Java” per l’interazione con DBMS
  - indipendenza dalla piattaforma

Driver JDBC

Per ciascun DBMS

- Esistono quattro tipi di driver
  - si differenziano per il livello a cui interagiscono con la base di dati
  - il più semplice è il JDBC-ODBC Bridge che utilizza ODBC per connettersi alla base di dati

ESEMPIO

apertura connessione

esecuzione (unica possibiltà per SQL dinamico)

risultato restituito una tupla alla volta, stile cursore

SQL statico

preparazione

esecuzione

chiusura connessione

```
import java.sql.*;
import java.io.*;
class exampleJDBC
{ public static void main (String args [])
{
   Connection con = null;
   try {
      String ilGenere = "comico";
      Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
      con = DriverManager.getConnection("jdbc:odbc:ilMioDB",
                                        "laMiaLogin",
                                        "laMiaPassword");
      Statement st = con.createStatement();
      ResultSet rs = st.executeQuery("SELECT AVG(valutaz)
                                      FROM Film
                                      WHERE genere = '"+ ilGenere + "'");
      rs.next();
      if (rs.getDouble(1) < 4)
         st.executeUpdate("UPDATE Film
                           SET valutaz = valutaz * 1.05");
      else
         st.executeUpdate("UPDATE Film
                           SET valutaz = valutaz * 0.95");
      PreparedStatement pst =
         con.prepareStatement("SELECT titolo, regista, valutaz
                               FROM Film
                               WHERE genere = ?");
      pst.setString(1,ilGenere);
      rs = pst.executeQuery();
      while (rs.next())
         System.out.println("Titolo:  "       + rs.getString(1)+
                            " Regista:  "     + rs.getString(2)+
                            " Valutazione:  " + rs.getDouble(3));
      con.close();
      }
      catch(java.lang.ClassNotFoundException e) {
         System.err.print("ClassNotFoundException:  ");
         System.err.println(e.getMessage());}
      catch (SQLException e) {
         while( e!=null){
            System.out.println("SQLState:  " + e.getSQLState());
            System.out.println("    Code:  " + e.getErrorCode());
            System.out.println(" Message:  " + e.getMessage());
            e = e.getNextException();
         }}}}
```

Estensioni del linguaggio applicativo per SQL

Si aggiunge zucchero sintattico al linguaggio applicativo

- per accedere alle librerie in maniera comoda
- lo *zucchero* viene eliminato da un (pre)compilatore
- due approcci possibili
  - integrazione forte: costrutti del linguaggio, integrati totalmente con esso, da tradursi in comandi SQL
    - LINQ ⇒ TAP
  - integrazione debole: comandi SQL iniettati nel codice ma ben separati mediante keyword/simboli/blocchi
    - SQL ospitato

SQL ospitato - Concetti di base

- Lo standard SQL definisce come SQL possa essere ospitato in un vari linguaggi di programmazione, tra cui
  - C, COBOL
  - Pascal, Fortran
  - ma non Java
- Dopo lo sviluppo di JDBC è stata proposta una specifica ANSI/ISO per incapsulare comandi SQL in programmi Java:
  - SQLJ (SQL Java) sviluppato dall'SQLJGroup
- I comandi SQL (embedded SQL) possono apparire in ogni punto in cui può comparire un'istruzione del linguaggio ospite
- Ogni istruzione SQL deve essere chiaramemente identificabile nel testo del programma:
  - Preceduta da un prefisso e seguita da un terminatore
  - Per molti linguaggi (C, Pascal, Fortran, Cobol):
    - Prefisso = EXEC SQL
    - Terminatore = ;
  - SQLJ:
    - Prefisso = #sql
    - Terminatore = ;

ESEMPIO

apertura connessione

SQL statement
uso simile a
plpg/SQL

sql iterator ~
cursore

```
import java.sql.*;
import java.io.*;
import java.math.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;
import oracle.sqlj.runtime.*;
class exampleSQLj
{
   #sql iterator FilmIter(String titolo, String regista);
   public static void main (String args [])
   {
   try{
      Double valMedia;
      String ilGenere = "comico";
      oracle.connect("jdbc:odbc:ilMioDB",
                     "laMiaLogin",
                     "laMiaPassword");
      #sql{SELECT AVG(valutaz) INTO :valMedia
           FROM Film
           WHERE genere = :ilGenere};
      if (valMedia > 4)
         #sql{UPDATE Film SET valutaz = valutaz * 1.05};
      else
         #sql{UPDATE Video SET valutaz = valutaz * 0.95};
      FilmIter ilFilmIter = null;
      #sql ilFilmIter ={SELECT titolo, regista, valutaz
                        FROM Film
                        WHERE genere = :ilGenere};
      while (ilFilmIter.next())
         System.out.println("Titolo:  "       + ilFilmIter.titolo() +
                            " Regista:  "     + ilFilmIter.regista() +
                            " Valutazione:  " + ilFilmIter.valutaz());
   }
   catch(java.lang.ClassNotFoundException e) {
      System.err.print("ClassNotFoundException:  ");
      System.err.println(e.getMessage());}
   catch (SQLException e) {
      while(e!=null){
         System.out.println("SQLState:  " + e.getSQLState());
         System.out.println("    Code:  " + e.getErrorCode());
         System.out.println(" Message:  " + e.getMessage());
         e = e.getNextException();
      }}}}
```

Transazioni

- una transazione è una sequenza di operazioni di lettura e scrittura
- costituisce l'effetto dell'esecuzione di programmi che effettuano le funzioni desiderate dagli utenti
- ogni transazione è eseguita
  - o completamente (cioè effettua ***commit***)
  - oppure per nulla (cioè effettua ***abort***) se si verifica un qualche errore (hardware o software) durante l'esecuzione
- **ACID**ity property, da
  - **Atomicità**
  - **Consistenza**
  - **Isolamento**
  - **Durability - Persistenza**

Transazioni e Programmi Applicativi

- A seconda di come è settato l'autocommit
  - Ogni comando costituisce implicitamente una transazione a sé stante
  - Tutti i comandi eseguiti all'interno di una sessione costituiscono implicitamente una transazione

  - Comando SET AUTOCOMMIT
  - Comandi START TRANSACTION; COMMIT; ROLLBACK

  - In PostgreSQL il default è FALSE

TRANSAZIONI - JDBC

- Se non altrimenti specificato, in JDBC autocommit è true
- Ogni comando SQL viene eseguito come una transazione.
- Questo comportamento può essere disabilitato utilizzando il metodo **`setAutoCommit`** della classe **`Connection`**, a cui deve essere passato un valore booleano (false se vogliamo disabilitare l'autocommit)
- Se l'autocommit è disabilitato, una transazione viene creata all'inizio dell'esecuzione dell'applicazione
- Tale transazione può essere terminata esplicitamente, invocando il metodo **`commit`** o **`rollback`** della classe **`Connection`**, o implicitamente, al termine dell'esecuzione
- Tutti i comandi SQL compresi tra le istruzioni **`con.setAutoCommit(false)`** e **`con.close()`** costituiscono un'unica transazione e questo implica che o vengono eseguiti tutti oppure non ne viene eseguito alcuno

TRANSAZIONI - SQLJ

- SQLJ, invece, per default disabilita l'autocommit
- Quindi, una transazione viene creata all'inizio dell'esecuzione dell'applicazione
- Tale transazione può essere esplicitamente terminata usando i comandi **`COMMIT`** e **`ROLLBACK`**, con la seguente sintassi:
  - **`#sql {COMMIT};`**
  - **`#sql {ROLLBACK};`**
- Dopo l'esecuzione di uno di questi due comandi, viene automaticamente creata una nuova transazione
- L'autocommit può comunque essere richiesto al momento della connessione con il DBMS

Meditazione 1.

```
SELECT balance INTO e FROM Account WHERE number=i
UPDATE Account SET balance=e + 1 WHERE number=i

UPDATE Account SET balance=balance+1 WHERE number=i
```

Meditazione 2. (SAW)

- Se per l'inserimento di un utente in tabella scrivete:

```
SELECT ... FROM users WHERE (...)
INSERT INTO users VALUES (...)
```

  - Cosa succede se due utenti con lo stesso username cercano di registrarsi al sito contemporaneamente?
  - Vi sembra che sia ragionevole fare sempre due query SQL per ogni operazione di INSERT?