8- Programmazione di una base di dati attraverso JDBC

Introduzione ai differenti approcci - JDBC: Architettura e Programmazione

Introduzione

- Nella pratica gli utenti finali accedono al contenuto di una base di dati non direttamente attraverso l'uso del linguaggio SQL, ma attraverso programmi applicativi che realizzano apposite interfacce di accesso alla base di dati stessa.

Cenni agli approcci per la programmazione di basi di dati

- *SQL-embedded*:
  - Un **precompilatore** o **preprocessore** esamina prima di tutto il codice sorgente del programma per identificare le istruzioni di interazione con la base di dati ed estrarle per l'elaborazione con il DBMS.
  - Queste vengono sostituite nel programma da chiamate di funzione al codice generato dal DBMS.
  - La comunicazione dei risultati elaborativi del DBMS all'applicazione avviene a mezzo di apposite **variabili condivise** (e.g., SQLCODE, SQLSTATE), mentre le variabili del programma possono essere usate come parametri nelle istruzioni SQL

ESEMPIO

```
EXEC SQL BEGIN DECLARE SECTION;
varchar nome stud [50];
varchar cognome stud [50];
varchar matr stud [10];
int SQLCODE;
EXEC SQL END DECLARE SECTION;
printf(''Immettere matricola: '');
scanf(''%s'',&matr stud);
EXEC SQL;
SELECT NOME, COGNOME into :nome stud, :cognome
    stud
FROM STUDENTI WHERE MATRICOLA=:matr stud;
if (SQLCODE==0) printf(''Studente: %s %s'', nome
    stud,cognome stud);
else printf(''Matricola inesistente'');
```

SQL/CLI (Call Level Interface):

- Apposite libreria di funzioni sono rese disponibili al linguaggio ospiste per l'interfacciamento alla base di dati.
- Vi sono funzioni per la connessione al database, per l'esecuzione di interrogazione, per l'aggiornamento dei dati, e cosi' via.
- Questo approccio fornisce delle apposite API

Linguaggi di programmazione per basi di dati:

- Definiti appositi linguaggi di programmazione "compatibili" con il modello logico della base di dati e con il linguaggio di interrogazione SQL.
- PL/SQL dell'Oracle che integra la potenza e la flessibilità di SQL con i costrutti tipici di un linguaggio procedurale.

Limiti del SQL-embedded

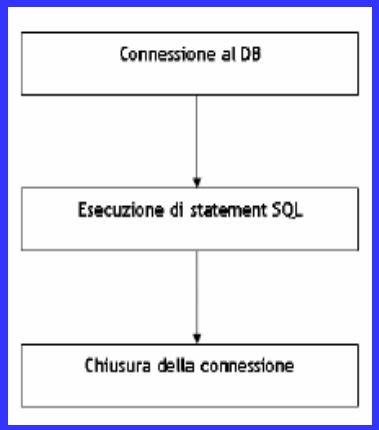
- Il linguaggio SQL-embedded risulta un approccio di programmazione delle basi di dati **statico** in quanto il testo dell'interrogazione è scritto all'interno del programma e non può essere cambiato senza ricompilare o rielaborare il codice sorgente
- Inoltre poiché ogni applicazione per interfacciarsi ad un DBMS fa riferimento a delle proprie librerie (di solito scritte C), se questa ultima deve utilizzare per qualche ragione un nuovo database, diverso dal precedente, occorre riscrivere tutto il codice di gestione dei dati.

Accesso mediante SQL/CLI

- standard a "livello di chiamata" per l'interfacciamento alle basi di dati detto appunto SQL/CLI proposto da X/Open
  - approccio **dinamico** per la programmazione delle basi di dati: per l'accesso alla base di dati viene usata una libreria di funzioni che da un lato fornisce maggiore flessibilità e non richiede la presenza di alcun preprocessore, dall'altro però, comporta che la verifica della sintassi e altri controlli sui comandi SQL avvenga solo al momento dell'esecuzione del programma.

ODBC

- Basandosi sullo standard SQL/CLI, ogni DBMS poteva quindi mettere a disposizione apposite interfacce di programmazione (API) per i principali linguaggi (C/C++, VB, ...), che permettevano alle applicazioni scritte in quel linguaggio di interrogare



ODBC

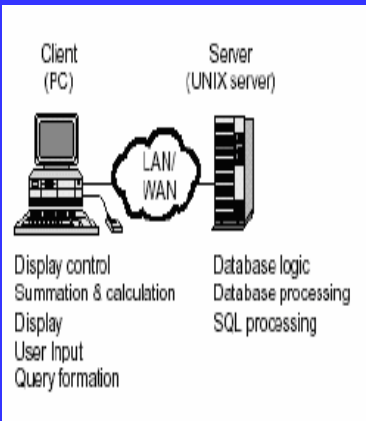
- Col passare degli anni nasce poi l'esigenza di standardizzare l'interfaccia attraverso
- la quale un'applicazione poteva accedere ad una qualsiasi base di dati relazionale (e.g.,
- Oracle, Access, DB2, MySQL, ...), garantendone l'interoperabilità . A tale proposito
- nel 1991 la Microsoft propone uno standard per la comunicazione con database
- remoti noto col nome di ODBC (Open Database Connectivity).

ODBC

- Nell'architettura ODBC il collegamento tra un'applicazione client e il server della base di dati, richiede l'uso di un *driver*, ovvero di una libreria che viene collegata dinamicamente all'applicazione e da essa invocata quando si vuole accedere alla base di dati.
  - Ogni driver maschera le differenza di interazione legate non solo al DBMS, ma anche al sistema operativo e ai protocolli di rete utilizzati.
  - i driver astraggono dai protocolli specifici dei produttori dei DBMS, fornendo un'interfaccia di programmazione delle applicazioni comune ai client del database

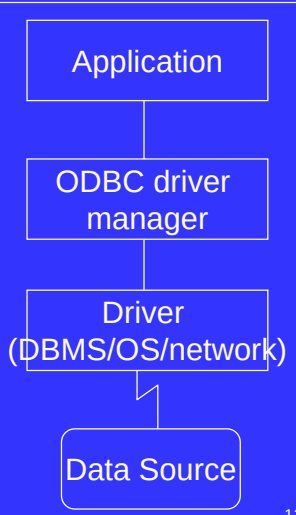
ODBC

- Usando nel proprio client le chiamate (API) all'interfaccia ODBC si rende il proprio programma capace di accedere quindi a più server di database, senza dovere conoscere le interfacce proprietarie di ogni singolo database



ODBC

- *l'applicazione*:
- *il driver manager*
- *i driver*
- *la sorgente dati o data source*



JDBC

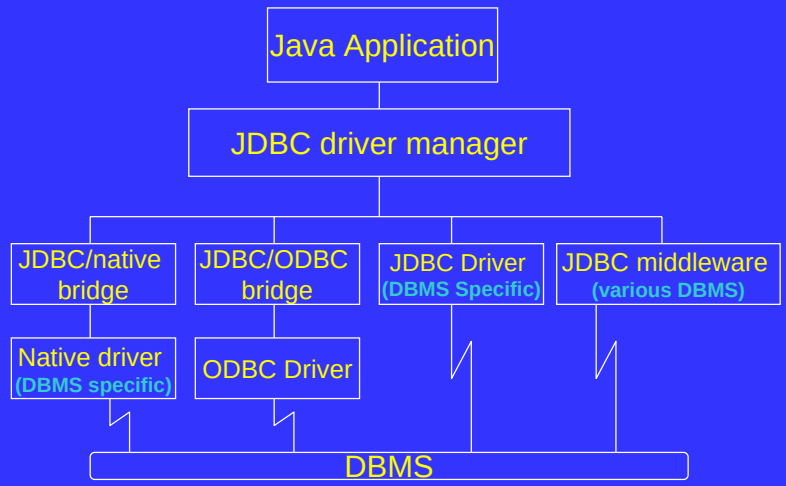
- JDBC é un interfaccia di programmazione (API) ad oggetti basata sullo standard CLI e definita dalla Sun Microsystems per l'accesso ad un database in maniera indipendente dalla piattaforma.

Servizi per l'accesso a dati

- Connessione al database;
- Invio di statement SQL;
- Processing dei risultati;
- Gestione delle Transazioni.

JDBC è divenuto oramai uno "standard de facto" per lo sviluppo di applicazioni JAVA "DB-oriented" e fa parte del pacchetto software JDK dalla versione 1.1.

Architettura



Architettura

- *JAVA application*:
  - l'applicazione JAVA che richiede l'accesso ad una base di dati.
- *JDBC Driver Manager*
  - rappresenta il livello di gestione di JDBC e opera tra l'utente e i driver. Tiene traccia dei driver disponibili e gestisce la connessione tra un DB ed il driver appropriato
- *JDBC Driver*
  - Realizzano la vera comunicazione fisica con il database. Essi permettono di creare la connessione con la sorgente dati, inviare istruzione di interrogazione o di aggiornamento alla sorgente dati e di processare i risultati.
- *DBMS*
  - a questo livello troviamo il DBMS relazionale a cui l'applicazione si deve interfacciare.

il package java.sql.

- *Interfaccia Driver*
  - rappresenta il punto di partenza per ottenere una connessione ad un DBMS. I produttori di driver JDBC devono implementare un'interfaccia Driver (mediante un'opportuna classe) affinchè un'applicazione possa interfacciarsi con un tipo particolare di DBMS.
- *Classe DriverManager*
  - è la classe che contiene tutti i driver di database registrati
- *Interfaccia Connection*
  - un oggetto di tipo Connection rappresenta una connessione attiva ad un database

il package java.sql.

- *Interfaccia Statement*:
  - un oggetto di tipo Statement viene utilizzato per inviare query semplici SQL, che non fanno uso di parametri
- *Interfaccia PreparedStatement*
  - un oggetto di tipo PreparedStatement viene utilizzato per inviare query che presentano parametri di input.
- *Interfaccia CallableStatement*
  - usato per costruire query parametriche con parametri di input e output. Consente di eseguire anche **stored procedure** memorizzate sul server.
- *Interfaccia ResultSet*
  - un oggetto ResultSet rappresenta il risultato di una query di selezione (insieme di record o tuple).

Stringa di connessione

- In JDBC ogni database viene univocamente identificato da una particolare stringa di connessione detta JDBC URL. Tale stringa adotta un formalismo simile alla definizione degli URL e serve a definire dove si trova e come accedere al database. In particolare, essa ha una struttura del tipo:

jdbc:<driver>:<database>

Stringa di connessione

- In connessioni di tipo JDBC-ODBC il terzo termine coincide con il nome della sorgente dati utente o di sistema (Data Source Name o DSN), ed, un possibile JDBC URL potrebbe essere il seguente

jdbc:odbc:dbsegreteria

Stringa di connessione

- Nel caso di connessioni dirette con driver java puri, il terzo termine invece definisce i parametri di accesso dell'istanza di database. Un esempio più elaborato può essere quello per l'accesso ad un database server Oracle mediante driver di tipo **thin**:

jdbc:oracle:thin:@db.unina.it:1521:dbsegreteria

Programmare un'applicazione con JDBC

1. *Importazione dei package JDBC.*
2. *Registrazione dei driver JDBC.*
3. *Apertura della connessione al database.*
4. *Creazione dell'oggetto Statement.*
5. *Esecuzione della query e restituzione dell'oggetto ResultSet*
6. *Utilizzo dei risultati.*
7. *Chiusura degli oggetti Statement e ResultSet.*
8. *Chiusura della connessione.*

Importazione dei package JDBC

- Il primo passo da compiere per la creazione di un'applicazione Java che si connette ad una base di dati mediante è l'importazione dei package JDBC che avviene attraverso l'istruzione:

import java.sql.*;

Registrazione o caricamento del driver JDBC

registrazione del driver:

Class.forName(class driver)

dove class driver rappresenta il driver che gestisce la nostra base di dati.

ESEMPIO

```
Class.forName(oracle.jdbc.driver.OracleDriver);
```


Apertura della connessione al database

```
Connection conn
  =DriverManager.getConnection(url
  connect,User,Password);
```

dove url connect rappresenta la stringa di connessione JDBC URL e User e Password le credenziali d'accesso al database.

Creazione dell'oggetto Statement

- Interfaccie utilizzabili:
  - Statement, PreparedStatement, Callable Statement,
- L'istruzione per creare un oggetto di tipo Statement è la seguente:

```
Statement
stmt=conn.createStatement();
```


Esecuzione della query e restituzione dell'oggetto ResultSet

```
ResultSet
  rs=stmt.executeQuery(''SELECT *
  FROM STUDENTI '');
```

Il metodo executeQuery restituisce un oggetto di tipo ResultSet contenente i dati recuperati sotto forma di array di record.

OGGETTO STATEMENT

(StatementObj.)executeQuery(query):
- per statement che generano un unico oggetto di tipo ResultSet (select).

(StatementObj.)executeUpdate(stmt):
- per statement di modifica (insert, update, delete) o DDL (create table o drop table). In questo caso viene restituito un numero intero (contatore di aggiornamento) rappresentante il numero di righe che sono state nserite/aggiornate/cancellate, o il valore 0 per statement di tipo DDL.

(StatementObj.)execute(stmt):
- per statement che possono includere più ResultSet o contatori di aggiornamento

Chiusura degli oggetti Statement e ResultSet e della connessione

- Completate le operazioni sul database occorre rilasciare le risorse acquisite:
  - Ciò avviene attraverso la chiusura degli oggetti Statement, ResultSet Connection con istruzioni del tipo:
    - rs.close();
    - stmt.close();
    - conn.close();

Osservazioni sull'oggetto PreparedStatement

- L'interfaccia PreparedStatement estende l'interfaccia Statement ereditandone tutte le funzionalità con dei metodi aggiuntivi per la gestione dei parametri.

```
Prepared Statament
   pstmt=conn.prepareStatement
(''INSERT INTO STUDENTI VALUES (?,?,?)'');
pstmt.setString(1,''010/000010'')
pstmt.setString(2,''Diego Armando'')
pstmt.setString(3,''Maradona'')
rs=pstmt.executeUpdate();
```


Osservazioni sull'oggetto Callable Statement

- **stored procedure** o **funzioni**
  - conn.PrepareCall(String sql)
- dove la string sql segue una sintassi del tipo: {call nome-procedura(?,?,..)},{call nome-funzione(?,?,..)}
- A differenza degli altri casi oltre a specificare i parametri di input, `e necessario anche indicare a Java il tipo e valore di eventuali parametri di output ritornati dalle procedure e delle funzioni. Ciò avviene utilizzando il metodo:

registerOutParameter(int indice, int tipo)

Esempio

```
CallableStatament cs=conn.prepareCall(''fcall inserisci-
    studente(?,?,?)g''
cs.setString(1,''010/000010'')
cs.setString(2,''Diego Armando'')
cs.setString(3,''Maradona'')
cs.execute();
CallableStatament
cs=conn.prepareCall(''fcall ? = conta-studente-per-
    cognome(?)g''
cs.registerOutParameter(1,types.INTEGER)
cs.setString(2,''Maradona'')
cs.execute();
int numstudenti=cs.getInt(1);
```


Esempio JDBC

- L'esempio seguente mostra un corretto utilizzo delle API JDBC per l'accesso ad un
- DBMS ORACLE per la lettura della tabella STUDENTI con una gestione completa
- delle eccezioni (ricordiamo che il metodo System.out.println(String s)
- ha l'effetto di visualizzare a video il contenuto della stringa s).

Esempio JDBC completo

- try *f* Class.forName(oracle.jdbc.driver.OracleDriver)
- Connectio conn = DriverManager.getConnection
- (''jdbc:oracle:thin:@db.unina.it:1521:dbsegreteria'',''Vinni'','' Vinni'');
- Statement st=conn.createStatement();
- ResultSet rs=st.executeQuery(''select * from STUDENTI'');)
- *g* catch (ClassNotFoundException cnfe) *f* System.out.println(''Classe non trovata'' + cnfe.getMessage());*g*
- catch (SQLException sqle) *f* System.out.println(''SQL Error'');
- while sqle!=null *f* System.out.println(''Messaggio SQL: '' + sqle.getMessage());
- System.out.println(''SQLState: '' + sqle.getSQLState());
- System.out.println(''CODICE ERRORE SQL: '' + sqle.getErrorCode();
- sqle.getNextException();*gg*

I metadati

- La classe JDBC che consente l'interrogazione del catalogo e l'acquisione, ad esempio, dei nomi e del numero dei campi di una tabella è la classe ResultSetMetadata.

```
Statement st=conn.createStatement();
ResultSet rs=st.executeQuery(''select * from
    STUDENTI'');)
ResultSetMetadata rsmd = rs.getMetadata()
int cols=rsmd.getColumnCount();
for(int i=1;i<=cols;i++)
System.out.println(rsmd.getColumnName(i))
```

Cenni alla gestione delle transazioni in JDBC

- *uncommitted read (TRANSACTION READ UNCOMMITTED)*:
  - questo livello di isolamento permette alle transazioni di leggere, ma non scrivere, i dati che sono in uso da altre transazioni (il lock in lettura non `e esclusivo e non va in conflitto con quello in scrittura, quello in scrittura si), e quindi dati che stanno per essere modificati e potrebbero essere non integri.
- *committed read(TRANSACTION READ COMITTED)*:
  - questo livello di isolamento permette alle transazioni di leggere solo i dati che non sono in uso da altre transazioni, altrimenti queste si bloccano (solo due transazioni che richiedono un lock in lettura sono eseguite concorrentemente).
- *repeatable read(TRANSACTION REPEATABLE READ)*:
  - questo livello di isolamento non solo permette alle transazioni di leggere solo dati integri, ma anche, che se all'interno della stessa transazione questi saranno riletti, si riotterranno gli stessi risultati (cosa che non accadeva nel precedente caso).
- *serializable (TRANSACTION SERIALIZABLE)*:
  - coincide con il classico **Two-Phase Locking**, ovvero le transazioni in concorrenza su risorse comuni vengono oppurtanamente serializzate (solo due transazioni che richiedono un lock in lettura sono eseguite concorrentemente e transazioni che hanno gi`a rilasciato dei lock, non ne possono acquisirne degli altri) garantendo la piena consistenza dei dati ed evitando qualsiasi forma di anomalia quali: la lettura sporca (presente nel primo caso), lalettura non-ripetuta (presente nel primo e secondo caso) e la lettura fantasma di dati che prima non comparivano e poi compaiono(presente nel primo caso, secondo e terzo caso).

i batch update

- JDBC nella sua ultima versione supporta una nuova funzionalità nota col nome di **batch update**.
  Questa nuova possibilità permette di spedire una serie di aggiornamenti al database in blocco (batch) piuttosto che uno dietro l'altro, dando la possibilit`a al DBMS di effettuare delle ottimizzazioni sulla sequenza di modifiche da apportare. A tale proposito alle classi Statement, PreparedStatement, CallableStatement sono stati aggiunti dei metodi (addBatch(), clearBatch(), executeBatch()) in più per supportare tale tipologie di aggiornamenti , mentre `e stata introdotta una nuova eccezione BatcUpdateException, lanciata nel casodi anomalie nel processo di batch update.