

Basi di Dati:

Corso di laboratorio

Lezione 8

Raffaella Gentilini

Sommario

- 1 Introduzione: DB e Programmazione
- 2 Embedded SQL
- 3 LP + API
 - JDBC

Approcci alla Programmazione di un DB

Meccanismi Disponibili

- ① **Linguaggi di programmazione forniti dal DBMS** che estendono SQL (e.g. PL/pgSQL per PostgreSQL, PL/SQL per Oracle ...)
- ② **Embedded SQL**: Embedding di SQL nei principali linguaggi di programmazione
 - Un **precompilatore** identifica e traduce le chiamate al DB
- ③ **SQL + API**: Vengono fornite apposite **librerie (API)** per interfacciarsi al DB.

Embedded SQL

SQL Incapsulato: Principi Fondamentali

- Il **codice SQL** e' inserito nel **programma** scritto nel linguaggio ospite (e.g. C)
- le **istruzioni SQL** vengono **segnalate mediante le parole chiave EXEC SQL**
- Il codice viene precompilato utilizzando un **precompilatore** fornito dal DBMS
 - **ecpg** per **postgreSQL** e **C**

Embedded SQL: Esempio

A titolo di esempio, si consideri il seguente frammento di codice (incapsulato in un programma C)

Example

```
EXEC SQL BEGIN DECLARE SECTION;  
    char nome[200];  
    char cognome[200];  
    int nCorsi;  
EXEC SQL END DECLARE SECTION;  
EXEC SQL SELECT persona.nome, persona.cognome  
    INTO :nome,:cognome,:nCorsi FROM persona;
```

Embedded SQL: Esempio

A titolo di esempio, si consideri il seguente frammento di codice (incapsulato in un programma C)

Example

```
EXEC SQL BEGIN DECLARE SECTION;  
    char nome[200];  
    char cognome[200];  
    int nCorsi;  
EXEC SQL END DECLARE SECTION;  
EXEC SQL SELECT persona.nome, persona.cognome  
    INTO :nome, :cognome, :nCorsi FROM persona;
```

- E' possibile utilizzare le variabili del linguaggio nelle istruzioni SQL

Embedded SQL: Esempio

A titolo di esempio, si consideri il seguente frammento di codice (incapsulato in un programma C)

Example

```
EXEC SQL BEGIN DECLARE SECTION;  
    char nome[200];  
    char cognome[200];  
    int nCorsi;  
EXEC SQL END DECLARE SECTION;  
EXEC SQL SELECT persona.nome, persona.cognome  
    INTO :nome,:cognome,:nCorsi FROM persona;
```

- E' possibile utilizzare le variabili del linguaggio nelle istruzioni SQL
- Le variabili devono essere **annunciate al preprocessore** in una **DECLARE SECTION**

Embedded SQL: Esempio

Example

```
EXEC SQL CONNECT TO DEFAULT;
if(sqlca.sqlcode>=0) {
    EXEC SQL BEGIN DECLARE SECTION;
        char nome[200];
        char cognome[200];
        int nCorsi;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SELECT persona.nome, persona.cognome
        INTO :nome,:cognome,:nCorsi FROM persona
}
```

- Il preprocessore mette a disposizione del programmatore una struttura chiamata **sqlca** che contiene diverse **variabili di controllo** (e.g. variabile sqlcode verifica esito ultima istruzione)

Embedded SQL: Cursori

Cursori

- Il **cursore** e' uno strumento che permette di **leggere il risultato di un'interrogazione una riga alla volta**
- Le **operazioni di base** su un cursore sono:
 - ① **open**: apertura cursore
 - ② **fetch**: lettura riga
 - ③ **close**: chiusura cursore

Cursori: Esempio

Example

```
// dichiarazione cursore
EXEC SQL DECLARE cursore_persona CURSOR FOR SELECT nome
FROM persona;
// apertura cursore
EXEC SQL OPEN cursore_persona;
// utilizzo del cursore
EXEC SQL FETCH FROM cursore_persona INTO :nome;
while (sqlca.sqlcode >= 0) {
EXEC SQL FETCH FROM cursore_persona INTO :nome;
};
```

SQL Embedded Dinamico

SQL Dinamico

- Permette di eseguire interrogazioni il cui schema non è noto a priori
- In particolare, il comando **EXECUTE IMMEDIATE** permette di eseguire lo statement SQL contenuto in una riga

Example

```
EXEC SQL BEGIN DECLARE SECTION;  
char query[100];  
EXEC SQL END DECLARE SECTION;  
// leggere query opportunamente ...  
EXEC SQL EXECUTE IMMEDIATE :query;
```

SQL Embedded e portabilit 

Embedded SQL: Vantaggi e Svantaggi

- **Vantaggi** SQL Immerso
 - prestazioni (precompilazione ottimizzata)
- **Problemi** SQL Immerso
 - Limiti di portabilit :
 - applicazione dipende dal precompilatore
 - difficile sviluppare su DBMS diversi

Verso una maggior indipendenza dal DBMS: CLI (I)

Tale limite viene superato con l'uso di interfacce (API) standard per la comunicazione con il DBMS:

Call Level Interface (CLI)

- la comunicazione client/server e' basata su una **API** standardizzata
- il codice SQL non e' immerso nel codice sorgente, ma viene inviato al DBMS sotto forma di stringhe
- il client deve disporre di un opportuno modulo, detto **driver** che implementa l'API e gestisce la comunicazione con il DBMS

CLI (II)

CLI: Concetti Fondamentali

- **Driver**: modulo sul lato del client che **implementa lo standard (API di comunicazione)**
- dal momento che il codice non viene precompilato, il client puo' utilizzare diversi driver contemporaneamente per accedere a diversi DBMS
- **Driver manager** modulo sul lato del client che **gestisce i diversi driver disponibili**

CLI (III)

Esempi di CLI

- ODBC
- JDBC

ODBC

- Acronimo di Open Database Connectivity
- API standard definito da Microsoft nel 1992
- Diffuso per l'interfacciamento di applicazioni eterogenee

JDBC

JDBC

- **API Java**: Insieme di classi ed interfacce scritte in Java (package java.sql) **per connettersi ad un DBMS, fare interrogazioni e manipolare i risultati delle interrogazioni**
- JDBC non e' un acronimo (**J**ava **D**atabase **C**onnectivity) ma un marchio depositato di Sun Microsystems
- **Ciascun DBMS mette a disposizione un driver specifico** che:
 - Viene **caricato a runtime**
 - **Traduce le chiamate alle funzioni JDBC in chiamate a funzioni del DBMS**

Architettura JDBC

Un sistema che usa JDBC ha quattro componenti principali

- 1 **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)
- 2 **Gestore di driver**: carica i driver, passa le chiamate al driver corrente, esegue controlli sugli errori
- 3 **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
- 4 **Sorgente di dati**: elabora i comandi provenienti dal driver restituisce i risultati

Tipi di Driver JDBC

- 1 **JDBC-ODBC** Utilizza un bridge creato dalla Sun che permette l'utilizzo di driver ODBC all'interno di applicazioni Java
- 2 **JDBC basato su funzioni scritte in linguaggio "nativo"** (cio non in Java) e poi richiamate da Java
- 3 **JDBC-Net**: non comunica direttamente con il DBMS, ma con un middleware, in grado di prendere le chiamate e convertirle in qualcosa che il DB in grado di capire, rispondendo poi di conseguenza
- 4 **JDBC, pure Java, con accesso diretto**. Non occorre nient'altro se non il driver e il DBMS a cui collegarsi

Applicazioni Java con JDBC

Le **fasi fondamentali** di un programma applicativo Java che accede ad un DB mediante chiamate di funzione JDBC:

1. Importazione libreria di classi JDBC
2. Registrazione driver JDBC
3. Apertura connessione con il DBMS (Creazione oggetto **Connection**)
4. Creazione di un oggetto **Statement**
5. Esecuzione query e restituzione oggetto **ResultSet**
6. Utilizzo Risultati
7. Chiusura oggetto/i **ResultSet** e oggetto **Statement**
8. Chiusura connessione.

Importazione Package

Step 1. Importazione Package

```
// importazione package  
import java.sql.*;      // package JDBC
```

Caricamento e Registrazione Driver

Step 2. Caricamento e Registrazione Driver

```
// caricamento e registrazione driver  
string driver="org.postgresql.Driver";  
Class.forName(driver);
```

- Per **caricare un driver JDBC** esplicitamente si puo' usare la **funzione generica Java per il caricamento di una classe**
- Il **metodo statico forName della classe Class** restituisce un'istanza della classe Java specificata nella stringa passata come parametro
- Nel nostro caso, passiamo una stringa per la **creazione di un oggetto di classe Driver** (**specifico per il DBMS selezionato**), il quale automaticamente registra se' stesso con la classe DriverManager

Caricamento e Registrazione Driver

Step 2. Caricamento e Registrazione Driver

```
// caricamento e registrazione driver  
string driver="org.postgresql.Driver";  
Class.forName(driver);
```

- L'istruzione può sollevare un'eccezione di classe **ClassNotFoundException** nel caso in cui il **driver** (la classe) non sia **rintracciabile**
- L'eccezione deve essere gestita (pena la non compilazione) i.e. inserita in un blocco try—catch

Apertura Connessione

Una volta caricato il driver, il metodo statico `getConnection` della classe `DriverManager` permette di ottenere un oggetto di tipo `Connection`.

Step 3. Aprire connessione con DBMS

```
String url="jdbc:postgresql://localhost/my_DB";  
String user="my_userid";  
String pwd="my_passwd";  
Connection conn=DriverManager.getConnection(url,user,pwd);
```

Apertura Connessione

Una volta caricato il driver, il metodo statico `getConnection` della classe `DriverManager` permette di ottenere un oggetto di tipo `Connection`.

Step 3. Aprire connessione con DBMS

```
String url="jdbc:postgresql://localhost/my_DB";  
String user="my_userid";  
String pwd="my_passwd";  
Connection conn=DriverManager.getConnection(url,user,pwd);
```

- I parametri di `getConnection` sono:
 - 1 URL JDBC
 - 2 login
 - 3 password
- Il metodo `getConnection` puo' sollevare un'eccezione di tipo `SQLException` che deve essere gestita in un blocco `try—catch`

Apertura Connessione

Una volta caricato il driver, il metodo statico `getConnection` della classe `DriverManager` permette di ottenere un oggetto di tipo `Connection`.

Step 3. Aprire connessione con DBMS

```
String url="jdbc:postgresql://localhost/my_DB";  
String user="my_userid";  
String pwd="my_passwd";  
Connection con=DriverManager.getConnection(url,user,pwd);
```

- L'URL JDBC ha la forma:
 `jdbc:<sub-protocollo>:<altri-parametri>`
- `postgresql` e' il subprotocollo per il DBMS `postgreSQL`
- Nell'esempio sopra, `my_DB` e' il nome del DB cui ci si vuole connettere da `localhost`

Creazione Oggetto Statement

Una volta disponibile una istanza di connessione e' possibile definire un oggetto di tipo `Statement` (metodo `createStatement` di `Connection`) che consentira' di eseguire istruzioni SQL.

Step 4. Creazione Oggetto Statement

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

L'esecuzione del metodo puo' causare un'eccezione di tipo `SQLException` da gestire con un usuale blocco try—catch.

Esecuzione Istruzioni SQL

Step 5. Operare sul DB

```
stmt.executeUpdate("INSERT INTO PERSONA " +  
                  "VALUES (1,'Mia','Yi','1992/1/1')");
```

```
ResultSet rs=stmt.executeQuery("SELECT * FROM PERSONA");
```

Esecuzione Istruzioni SQL

Step 5. Operare sul DB

```
stmt.executeUpdate("INSERT INTO PERSONA " +  
                  "VALUES (1,'Mia','Yi','1992/1/1')");
```

```
ResultSet rs=stmt.executeQuery("SELECT * FROM PERSONA");
```

- Per impartire una **istruzione SQL** si usa una **stringa**
- La stringa e' concatenata con **+** perche' Java non ammette la **definizione di stringhe su piu' righe**
- Attenzione: A differenza di SQL Java e' case-sensitive. Gli apici che delimitano stringhe sono singoli in SQL e doppi in Java.

Esecuzione Istruzioni SQL

Step 5. Operare sul DB

```
stmt.executeUpdate("INSERT INTO PERSONA " +  
                  "VALUES (1,'Mia','Yi','1992/1/1')");  
ResultSet rs=stmt.executeQuery( "SELECT * FROM PERSONA");
```

Metodi Classe statement

- `executeUpdate` per **creazione, eliminazione, aggiornamento**:
Restituisce `int` pari al numero righe modificate. Per uno statement DDL, e.g. creazione tabella, viene restituito 0.
- `executeQuery` per **interrogazione**: Restituisce un oggetto di tipo `ResultSet` (struttura dati per insieme di risultati—simile ad un cursore).
- entrambi i metodi possono causare eccezioni di tipo `SQLException` da gestire con `try—catch`.

Esecuzione Istruzioni SQL

Step 5. Operare sul DB

```
stmt.executeUpdate("INSERT INTO PERSONA " +  
                  "VALUES (1,'Mia','Yi','1992/1/1')");
```

```
ResultSet rs=stmt.executeQuery( "SELECT * FROM PERSONA");
```

ResultSet

- Un ResultSet e' una tabella contenente il risultato di una query
- Come un cursore, un ResultSet e' inizialmente posizionato prima della prima riga.
- ResultSet avanza con il metodo next().

Utilizzo Risultati

Step 6. Utilizzo Oggetto ResultSet

```
while (rs.next()) {  
    String s=getString("Cognome")  
    String t=getString("Nome")  
    System.out.println(s+" "+t)  
}
```

Utilizzo oggetto ResultSet

- Una volta invocato, il metodo `next()` sposta il cursore di una riga in avanti e restituisce `false` se non ci sono piu' righe da analizzare, `true` altrimenti.

Utilizzo Risultati

Step 6. Utilizzo Oggetto ResultSet

```
while (rs.next()) {  
    String s=getString("Cognome")  
    String t=getString(3)  
    System.out.println(s+" "+t)  
}
```

Metodi getXXX

- **Metodi getXXX:** Restituiscono valore colonna specificata dal parametro (corrispondente alla riga corrente).
- Possono causare eccezioni di tipo `SQLException` da gestire con `try—catch`.
- Il parametro puo' essere il nome oppure il numero d'ordine della colonna

Utilizzo Risultati

Step 6. Utilizzo Oggetto ResultSet

```
while (rs.next()) {  
    String s=getString("Cognome")  
    String t=getString(3)  
    System.out.println(s+" "+t)  
}
```

Corrispondenza tipi di dato Java e SQL

Tipo SQL	Tipo o Classe Java	Metodo Lettura di ResultSet
BIT	boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Float	getFloat()
INTEGER	Int	getInt()

Utilizzo Risultati

Step 6. Utilizzo Oggetto ResultSet

```
while (rs.next()) {  
    String s=getString("Cognome")  
    String t=getString(3)  
    System.out.println(s+ " "+t)  
}
```

Corrispondenza tipi di dato Java e SQL

Tipo SQL	Tipo o Classe Java	Metodo Lettura di ResultSet
NUMERIC	Int	getInt()
REAL	Float	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp()

Chiusura Oggetti

Step 7. Chiusura Oggetti Statement e resultSet

```
rs.close();  
stmt.close();
```

Step 8. Chiusura Connessione

```
con.close();
```

L'invocazione di questi metodi deve essere inserita all'interno del blocco try-catch perche' possono produrre eccezioni `SQLException`.

Istruzioni SQL Parametriche

Query Parametriche

- Una **query parametrica** e' una **istruzione SQL** in cui una o piu' valori usati nelle clausole coinvolte non sono specificati
- E' compito dell'applicazione avere in input tali valori e fare in modo che l'istruzione del DDL/DML SQL li usi

JDBC: L'interfaccia PreparedStatement

L'interfaccia PreparedStatement

- Estende l'interfaccia Statement a cui aggiunge metodi per la gestione dei parametri
- Ogni oggetto è associato ad una singola query parametrica. L'associazione avviene nel momento in cui si costruisce l'oggetto
- Impostando i parametri si potrà eseguire la query quante volte si voglia
- La creazione dell'oggetto avviene attraverso uno dei metodi PreparedStatement dell'interfaccia Connection

PreparedStatement

Example

```
String query="SELECT * FROM corsi NATURAL JOIN frequenza"  
+ " WHERE id_insegnante= ? AND id_corso= ?"  
PreparedStatement pst = conn.prepareStatement(query);
```

- La **query** viene **associata all'oggetto PreparedStatement**
- L'esecuzione sarà delegata ai metodi:
 - `int executeUpdate()` per query di tipo DDL e DML (es. istruzioni create/drop table, insert, update, delete,...)
 - `ResultSet executeQuery()` per query di selezione

Passaggio dei Parametri

Passaggio dei parametri

- Valori parametri sono assegnati dai metodi di `PreparedStatement`:
 - `void setXXX(int indice, XXX param)`, dove XXX e uno dei tipi di dato Java
Ex: `void setInt(int indice, int param)`, `void setString(int indice, String param)`
- ? in posizione indice viene sostituito dal valore della variabile param
- Il metodo viene scelto in base al tipo della variabile che contiene il parametro
- Una volta assegnati i valori ai parametri la query e pronta per essere eseguita

I metodi setXXX

setByte	setShort
setInt	setLong
setFloat	setDouble
setString	setBytes
setDate	setTime
setTimestamp	setAsciiStream
setBigDecimal	setBoolean
setNull	setUnicodeStream
setBinaryStream	setObject

Conversione di Tipi

- Metodi setXXX effettuano conversione tipi Java → tipi JDBC/SQL
- I tipi JDBC/SQL, definiti nella classe `java.sql.Types`, fanno da ponte verso i tipi SQL per garantire indipendenza dai tipi SQL definiti dai var DBMS
- Il driver JDBC converte i tipi JDBC/SQL nei corrispondenti tipi SQL del DBMS usato

Tipo o Classe Java	Tipo SQL
String	VARCHAR or CHAR
boolean	BIT
short	SMALLINT
int	INTEGER
float	REAL
double	DOUBLE

Conversione di Tipi

- Metodi setXXX effettuano conversione tipi Java → tipi JDBC/SQL
- I tipi JDBC/SQL, definiti nella classe `java.sql.Types`, fanno da ponte verso i tipi SQL per garantire indipendenza dai tipi SQL definiti dai var DBMS
- Il driver JDBC converte i tipi JDBC/SQL nei corrispondenti tipi SQL del DBMS usato

Tipo o Classe Java	Tipo SQL
<code>java.sql.Date</code>	DATE
<code>java.sql.Time</code>	TIME
<code>java.sql.Timestamp</code>	TIMESTAMP

Esempio

Example

```
Statement stmt; PreparedStatement pstmt; ResultSet rs;
stmt = con.createStatement();
//Selezione docenti nella DB
rs=stmt.executeQuery("SELECT DISTINCT id_studente FROM
frequenza,persona WHERE id_persona=id_studente");
//Creazione tabella S, per gli studenti nella BD
stmt.executeUpdate("create table S(id int primary key)");
//popolamento tabella degli studenti S mediante PreparedStatement
pstmt=con.prepareStatement("insert into S values(?)");
while(rs.next())
{
    int id=rs.getInt(1);
    pstmt.setInt(1,id);
    pstmt.executeUpdate();
}
```

Il Problema dell'SQL Injection (I)

SQL Injection

- Cosa succede se, data una query con parametri inseriti dall'utente (es. tramite interfaccia Web), questi ha la possibilità di agire direttamente sul valore dell'input di tipo stringa (oggetto String), aggiungendo, ad esempio, apici e altre istruzioni di controllo??
- Può inserire istruzioni arbitrarie che verranno eseguite dal DBMS!!!

Ex. <http://bobby-tables.com/>

Example

```
Statement = "SELECT * FROM users WHERE name = ' " +  
userName + " ';"
```

con la variabile userName assegnata al valore:

```
a';DROP TABLES users;
```

SQL Injection (II)

SQL Injection e PreparedStatement

- Questo tipo di vulnerabilit  viene detta SQL injection, in quanto l'utente pu iniettare statement SQL arbitrari con risultati catastrofici, come la divulgazione di dati sensibili o l'esecuzione di codice
- A prevenzione del problema, l'interfaccia PreparedStatement permette di gestire in modo corretto anche l'inserimenti di dati ostili

Bibliografia ed Approfondimenti

Bibliografia ed Approfondimenti

- R.A.Elmasri, S.B. Navathe. Sistemi di Basi di Dati – Fondamenti: Capitolo 9
- <http://java.sun.com/docs/tutorial/jdbc>