

Realizzazione di un driver JDBC per l'accesso ad un database scritto in prolog

di Stefano Tamagnini
Matricola XYZXYZ

Corso : Linguaggi e Modelli Computazionali 2006/07

last update : 29 dicembre 2008

Indice

1	Introduzione	3
2	Il database	4
2.1	Database Prolog	4
2.2	Il metabase	4
3	Structured Query Language	6
3.1	Il parser	6
3.2	Comandi	6
3.3	Operatori	7
4	Il driver	8
4.1	Struttura	8
4.2	Operazion	8
4.3	Transazioni	8

1 Introduzione

Come progetto per l'esame di Linguaggi e Modelli Computazionali, si è richiesto la realizzazione di un driver JDBC che permettesse di poter eseguire query SQL su un database Prolog. Della sintassi SQL che è molto vasta sono richieste le funzionalità di base che permettano basilamente : Creation, Read, Update, Delete (CRUD), lasciando l'implementazione completa secondo lo standard SQL ad un secondo momento.

Questo driver, ho deciso di chiamarlo **PJDBC**, dove la lettera P sta appunto per Prolog ossia il linguaggio in cui è espresso il database da interrogare; le parti coinvolte in questo progetto sono :

1. il database
2. il linguaggio con cui interrogarlo (sql)
3. il driver fisico che permetta l'interazione del database con le richieste dell'utente

2 Il database

Una definizione di database può essere **il database rappresenta un archivio di dati strutturati**; la struttura più diffusa in questo momento è quella chiamata *relazionale*, che si compone di tabelle e di relazioni tra di esse.

2.1 Database Prolog

Il database che si andrà ad interrogare attraverso il driver sviluppato avrà una struttura relazionale e sarà descritto attraverso una teoria prolog; le relazioni saranno espresse mediante l'uso di opportuni predicati. La tabella all'interno di questo database sarà rappresentata da un insieme di termini *compound* prolog che rappresenteranno le righe della tabella; le righe avranno come sintassi:

```
nomeTabella(col1 , col2 , . . . . . , colN ) .
```

I termini per appartenere ad una tabella dovranno avere lo stesso numero di argomenti e avere come nome il nome della tabella. Esempio:

```
ta(col1 , col2 , col3 ) .  
ta(col11 , col21 , col31 ) .
```

questi due termini rappresentano due righe di una tabella;

```
ta(col1 , col2 , col3 ) .  
tb(col1 , col2 , col3 ) .
```

questi due termini rappresentano due righe di due tabelle rispettivamente ta e tb;

```
ta(col1 , col2 , col3 ) .  
ta(col1 , col2 ) .
```

queste due termini rappresentano due righe di due tabelle differenti questo in quanto non è predicibile quale argomento manchi per avere la stessa struttura del termine della prima riga, se fosse predicibile si potrebbe tranquillamente pensare che l'elemento mancante sia *NULL* o *any*.

2.2 Il metabase

Per avere la completezza di informazioni tipiche di un moderno database relazionale è necessario avere delle ulteriori informazioni su come è strutturato il database ed in particolare sulla struttura delle tabelle; queste informazioni solitamente compongono quello che è chiamato *metabase* o *metadatabase*. Un possibile insieme di informazioni utili possono essere :

1. il nome da associare ad una colonna
2. il tipo di dato che contiene una colonna
3. la descrizione del contenuto di una colonna

questi sono solo alcuni possibili elementi. Nel nostro caso il *metabase* può essere descritto attraverso l'uso di una tabella speciale dal nome **metabase**; la sintassi sarà :

```
metabase(  
    "nome tabella",  
    "posizione della colonna tra gli argomenti",  
    "nome della colonna",  
    "tipo di dato contenuto",  
    "descrizione"  
).
```

Questa particolare tabella non è però necessaria, il database sarà comunque consistente, ma la sua presenza facilita l'interazione dell'utente nello scrivere query sql. Un esempio :

```
metabase("ta",1,"colonna1","string","prima colonna").
```

3 Structured Query Language

Lo Structured Query Language o SQL in forma abbreviata, è un linguaggio standard utilizzato per interrogare i database relazionali; allo stato attuale i moderni database implementano solo la parte *Entry Level* di tale standard.

3.1 Il parser

Per poter interpretare le richieste sql dell'utente si è reso necessario la creazione di un parser; in particolare dovrà essere in grado di poter riconoscere un sottoinsieme dello standard SQL/92. La sua realizzazione è avvenuta attraverso lo strumento javacc che ha permesso di specificare solo la grammatica da riconoscere e di lasciare allo strumento la generazione del codice relativa all'effettiva analisi del testo. La grammatica utilizzata per generare il parser comprende :

- **Operatori del linguaggio**, ossia le parole significative, che nel nostro caso possono essere di due tipi :
 - Comandi
 - Operatori SQL
- **Tipi di dato**

3.2 Comandi

I Comandi possibili sono suddivisi in due categorie Data Definition Language o DDL e Data Manipulation Language o DML. Quelli che ho ritenuto necessari implementare sono:

- DDL :
 - Create Database
 - Drop Database
- DML :
 - Select
 - Update
 - Insert
 - Delete
 - Truncate Table

- Drop Table

Come si nota non ho inserito *Create Table* in quanto già la *Insert* permette di creare una tabella all'interno del database prolog; ovviamente in questo caso mancano i metadati ma possono essere inseriti manualmente con delle *Insert* direttamente con riferimento alla tabella metabase.

3.3 Operatori

Gli operatori, messi a disposizione dal SQL/92 si dividono in quattro categorie:

1. Operatori di confronto
2. Operatori aritmetici
3. Operatori condizionali
4. Operatori logici

Di questi sono stati presi in considerazione quelli aritmetici e quelli di confronto

4 Il driver

Il driver è stato sviluppato secondo le specifiche *JDBC 4.0* (JSR 221) del Novembre 2006; secondo queste specifiche il driver è di tipo 4, ossia interamente scritto in java. Per aderire a queste specifiche è necessario che vengano implementate le seguenti interfacce :

- java.sql.Driver
- java.sql.DatabaseMetaData
- java.sql.ParameterMetaData
- java.sql.ResultSetMetaData
- java.sql.Wrapper
- java.sql.DataSource

e supportare nei ResultSet la capacità di concorrenza in lettura.

4.1 Struttura

4.2 Operazion

4.3 Transazioni