



LQL: Logic Query Language

Un nuovo modo di
interrogare i
Database!

Based on:



Progetto di: Burnacci Paolo



→ Outline

- Motivazioni e Obiettivi
- Analisi: che linguaggio vogliamo? Per quali problemi?
- Grammatica e Linguaggio
- Architettura e Scelte Implementative
- Limiti Sviluppi futuri
- DEMO di LQL executor



→ Motivazioni e Obiettivi

- Casus Belli: necessità di rintracciare i voti degli studenti e tutte le informazioni collegate
- Scarso *apprezzamento* per la sintassi SQL
- Prospettiva di un linguaggio quanto più relazionale possibile
- Un linguaggio per fare query Prolog Oriented
- Velocizzare e semplificare le query (concettualmente): sforzo mnemonico inferiore
- Introduzione di un minimo di estendibilità



→ Output desiderato (1/7):SELECT

Come si chiama lo studente con il numero di matricola 0000260024?

SQL

```
SELECT Nome , Cognome FROM Studenti  
WHERE Matricola = 0000260024
```

LQL

```
get nome(X,Y) from  
studenti(0000260024,X,Y,_).
```

Case Study: DB VOTI

TABELLE:

- Studenti
- Voto
- Registrazione
- Presenza
- CorsoDiLaurea
- Appelli



→ Output desiderato (2/7): JOIN

Che voto ha lo studente che si chiama Paolo Burnacci?

SQL

```
SELECT VotoFinale FROM Voto
JOIN Studenti
ON Voto.Studenti = Studenti.Matricola
WHERE Nome='Paolo' AND
Cognome='Burnacci'
```

LQL

```
get votoBurnacci(Y) from
voto(_,X,_,_),
studenti(X,Paolo,Burnacci,_).
```

Case Study: DB VOTI

TABELLE:

- Studenti
- Voto
- Registrazione
- Presenza
- CorsoDiLaurea
- Appelli



→ Output desiderato (3/7): UPDATE

*Cambiamo il numero di matricola a
chi si chiama Paolo Burnacci*

SQL

```
UPDATE Studenti SET 000022222  
WHERE Nome = 'Paolo'  
AND Cognome = 'Burnacci'
```

LQL

```
update  
studenti (_, Paolo, Burnacci, _)  
to  
studenti (000022222, Paolo, Burnacci, _).
```

Case Study: DB VOTI

TABELLE:

- Studenti
- Voto
- Registrazione
- Presenza
- CorsoDiLaurea
- Appelli



→ Output desiderato (4/7): INSERT

Inseriamo un nuovo studente

SQL

```
INSERT INTO  
Studenti(Matricola, Corso, Nome,  
Cognome, AnnoImmatricolazione)  
VALUE (0000158073, 22, Paolo,  
Burnacci, 2002/03)
```

LQL

```
insert  
studenti(0000158073, 22, Paolo,  
Burnacci, 2002/03).
```

Case Study: DB VOTI

TABELLE:



- Studenti
- Voto
- Registrazione
- Presenza
- CorsoDiLaurea
- Appelli



→ Output desiderato (5/7): DELETE

Cancelliamo uno studente dalla tabella

SQL

```
DELETE FROM Studenti
WHERE
Matricola= 0000158073 AND
AND Corso =22 AND Nome = 'Paolo' AND
Cognome = 'Burnacci' AND
AnnoImmatricolazione = '2002/03'
```

LQL

```
delete
studenti(0000158073,22, Paolo,
Burnacci, 2002/03).
```

Case Study: DB VOTI

TABELLE:

- Studenti 
- Voto
- Registrazione
- Presenza
- CorsoDiLaurea
- Appelli



→ Output desiderato (6/7): GROUP BY

Raggruppiamo gli studenti per ogni anno

SQL

```
SELECT Anno ,Matricola  
FROM Studenti  
GROUP BY Anno
```

LQL

```
get iscritti(A,List)  
from studenti(B,_,_,_,A)  
grouping B to List.
```

Case Study: DB VOTI

TABELLE:

- Studenti 
- Voto
- Registrazione
- Presenza
- CorsoDiLaurea
- Appelli



→ Output desiderato (7/7): Functions

Contiamo gli studenti per ogni anno

SQL

```
SELECT Anno ,COUNT(Matricola)
FROM Studenti
GROUP BY Anno
```

LQL

```
get iscritti(A,List)
from studenti(B,_,_,_,A)
grouping B to List,
get numeroiscritti(A,Count)
from iscritti(A,List)
apply count(List,Count).
```

Case Study: DB VOTI

TABELLE:

- Studenti 
- Voto
- Registrazione
- Presenza
- CorsoDiLaurea
- Appelli



→ Grammatica: le query fondamentali

Utilizzando una notazione EBNF:

- $S ::= E.$
- $E ::=$ insert **Fact**
| update **Predicate** to **Predicate**
| delete **Predicate**
| **Get**

Puro fatto
Prolog

Puro
predicato
Prolog

Nota: per comodità si sono indicati i VN con stringhe maiuscole invece che con <stringa>



→ Grammatica: fatti e predicati

Utilizzando una notazione EBNF:

- **Fact** ::= Name “ (“ **Attributes** ”) ”
- **Attributes** ::= (Name | Maiusc | Num)
 { , (Name | Maiusc | Num) }
- **Predicate** ::= Name “ (“ **Group** ”) ”
- **Group** ::= (Variable | Name | Num | Maiusc | Any)
 { , (Variable | Name | Num | Maiusc | Any) }

Veri
C
“
Stringhe
minuscole, con
apici e numeri

Valori e variabili
Prolog: stringhe
maiuscole e “ _ ”



→ Grammatica: la query GET

Utilizzando una notazione EBNF:

Get → get **Predicate** **Source**
[where **Condition**]
[grouping Variable **To**]
{, get **Predicate** **Source** [where **Condition**]
[grouping Variable **To** | apply **Predicate**]} ← - - -

Source → from **Predicate**{, **Predicate** }

To → to Variable

Condition → Expterm Compare Expterm

Expterm → T "(" Operator T ")"

T → Num | Variable | "(" Expterm ")"





→ Grammatica: i simboli terminali

Utilizzando una notazione EBNF:

Name: ["a"-"z"] {["a"-"z"]|["A"-"Z"]|["0"-"9"]}

Maiusc: "\" ["A"-"Z"] {["a"-"z"]|["A"-"Z"]|["0"-"9"]} "\"

| "\" Name "\"

| "\" Variable *\"

| "\" Name "\"

| "\" Variable "\"

| "\" Num "\"



Termini
con apici

Variable: ["A"-"Z"] {["a"-"z"]|["A"-"Z"]|["0"-"9"]}

Any: "_"

Num: ["0"-"9"] {["0"-"9"]}

Compare: ">"|"<"

Operator: "+"|"-"|"*"|"/"



→ Grammatica: considerazioni

- La grammatica vista secondo la classificazione di Chomsky è di **TIPO 2** in quanto in corrispondenza delle espressioni presenta **self-embedding**

→ $T \rightarrow \text{Num} \mid \text{Variable} \mid \boxed{\text{"(" Expterm ")"} }$

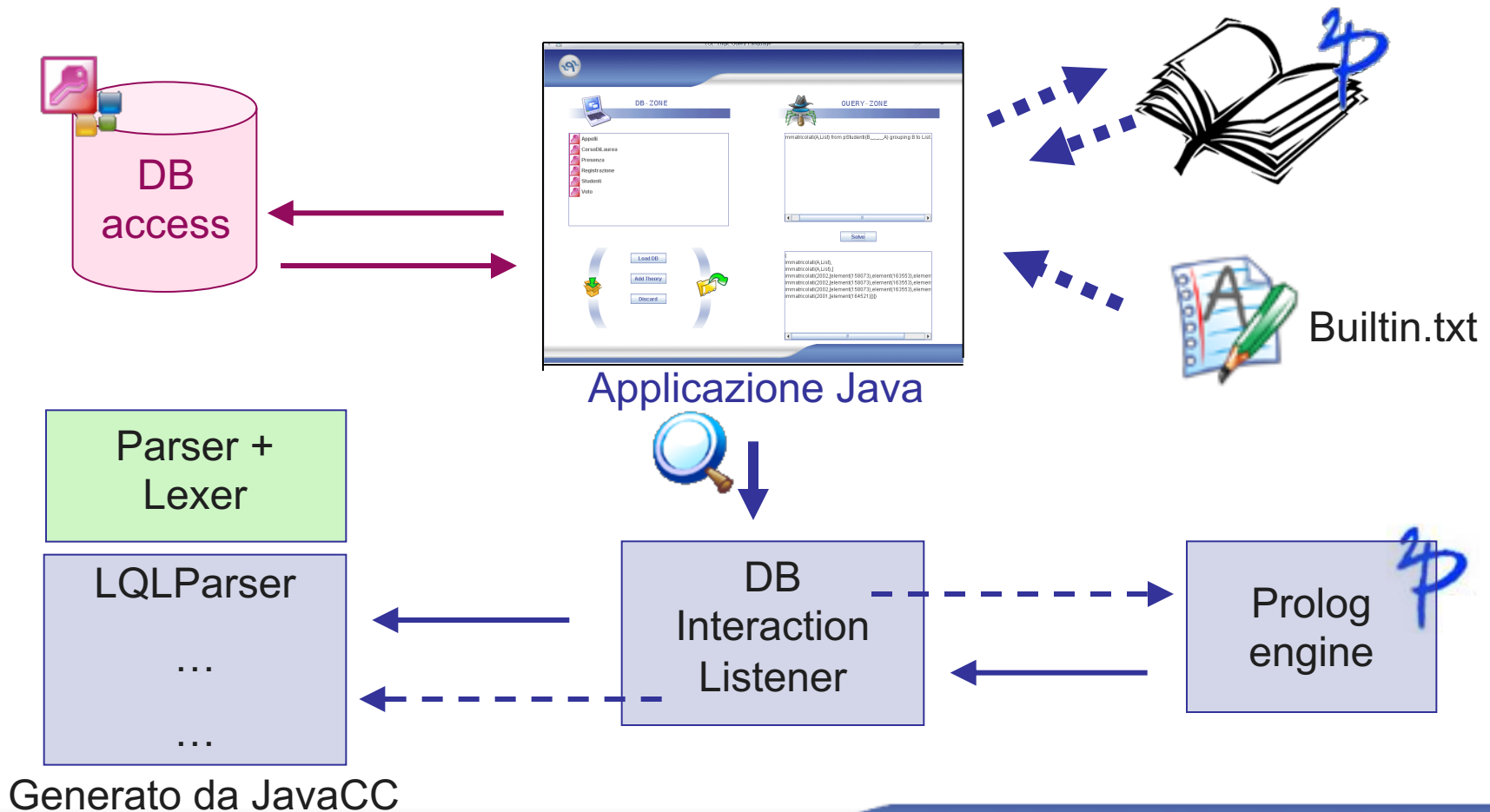
La grammatica vista è LL(1): basta sempre un simbolo terminale per individuare la produzione corretta (ci sono sempre parole chiave...)

Si usa
JavaCC
facilmente

- La notazione EBNF elimina ogni ricorsione, anche se sarebbero presenti solo ricorsioni destre
(**Attributes**, **Group**, **Source**)



→ Architettura dell'applicazione





→ Scelte implementative (1/2): JavaCC

→ Creazione del Parser con JavaCC.

File LQL.jj senza generazione di AST



Valutazione embedded: le azioni semantiche incorporano la valutazione

Ad ogni produzione è associata la creazione di una stringa (talvolta l'aggiunta di un predicato) che non richiede una visita ulteriore



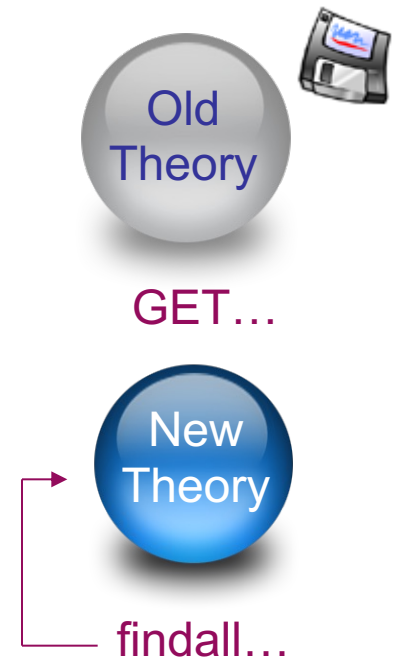
→ Scelte implementative (2/2): query

→ **GET**: introduzione di un predicato, findall, e rimozione del predicato (tramite copia di backup della teoria)

→ **INSERT**: assert del fatto

→ **DELETE**: retract nel caso semplice e findall seguita da retractList (built-in) nel caso articolato

→ **UPDATE**: 2 findall in successione dei nuovi e vecchi fatti, assertList sui nuovi e retractList sui vecchi





→ Limiti.

- Impossibilità di eseguire un JOIN e una GROUP BY contemporaneamente:

```
get iscritti(A,List) from  
appelli(B,A,_,_,_) grouping B to List.
```



1. Si inserisce nella teoria il predicato:
iscritti(A,List):-appelli(_,A,_,_,_), findall(element(B),appelli(B,A,_,_,_),List).
2. ?- findall(iscritti(A,List),iscritti(A,List),OUTPUTLIST).



- Manca l'ORDER BY: implementazione snervante (esula dagli scopi) → per le stringhe atom_chars, char_code e ordine tra i code

Attenzione, Sì,
limite, ma...



→ Sviluppi Futuri

- Implementazione della ORDER BY come apply grazie alla facile estendibilità (file builtin.txt)



Estenderlo con tutte le funzionalità volute

- Generalizzazione di insert:
Es: “insert get...” → collettiva
- Possibilità con l'applicazione di incorporare anche database in forma di file xml (generati da Uniwex)
- Possibilità di scrittura diretta su un DB (Access)





Fine

