Progettazione e implementazione di un DBMS relazionale

LAUREA TRIENNALE IN INFORMATICA

Candidato: Magliolo Leonardo

tesi presentata sotto la supervisione del relatore

Prof. Ruggero G. Pensa

Dipartimento di Informatica

Università di Torino

A.A. 2020/2021

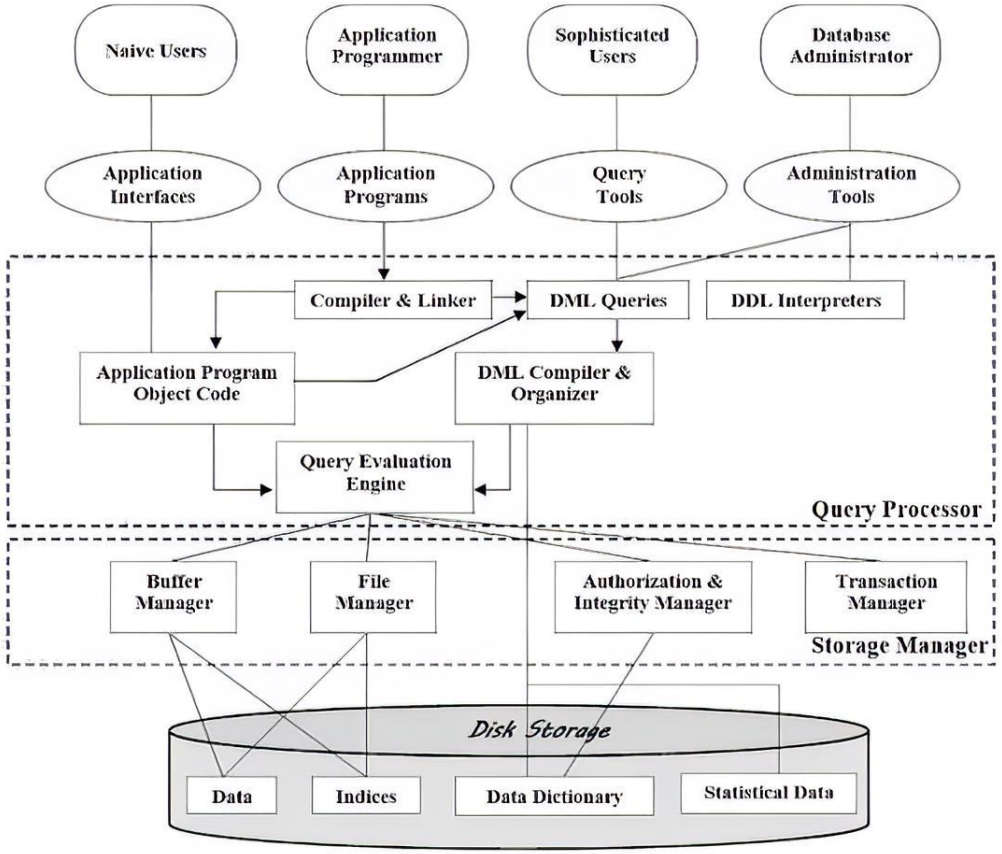
Introduzione

L'obiettivo della tesi consiste nella realizzazione di un Database Management System relazionale aventi le seguenti caratteristiche:

- Portabile
- Dalle ridotte dimensioni
- Progettazione in grado d'essere esaminata a fini didattici

La realizzazione del progetto prefigge, inoltre, lo scopo di consolidare ed ampliare le conoscenze apprese durante il corso di laurea di linguaggi formali, traduttori e basi di dati.

Cos'è un DBMS?



Un DBMS è un sistema software finalizzato alla gestione delle basi di dati.

Ad ogni richiesta di interazione verso il DBMS viene associata una transazione,
per ognuna di esse il DBMS deve garantire il soddisfacimento di proprietà che in gergo vengono chiamate ACID:

- Atomicità
- Coerenza
- Isolamento
- Durabilità

Strumenti utilizzati

Utilizzato per generare automaticamente le classi parser scritte in Java a partire dalla grammatica SQL

Utilizzato per rappresentare le classi progettuali e consentirne l'esecuzione

Perché il linguaggio Java?

Considerate funzionalità come:

- Gestione automatica della memoria
- Nessun riferimento a puntatori espliciti
- Gestione delle eccezioni
- Compilazione intermedia JIT

la scelta di Java permette di risolvere efficacemente il trade-off imposto tra prestazioni del software prodotto e tempistiche dedicate allo sviluppo e ai test.

Perchè ANTLR?

- Maggiore disaccoppiamento tra definizioni delle regole di produzione e relative azioni semantiche
- Tempistiche di sviluppo notevolmente ridotte
- Report automatico di errori dovuti ai processi di analisi lessicale e analisi sintattica
- Plugin di supporto IDE in grado di facilitare il debugging
- Ampiamente utilizzato dalla community di sviluppatori per realizzare traduttori basati su schemi SDT

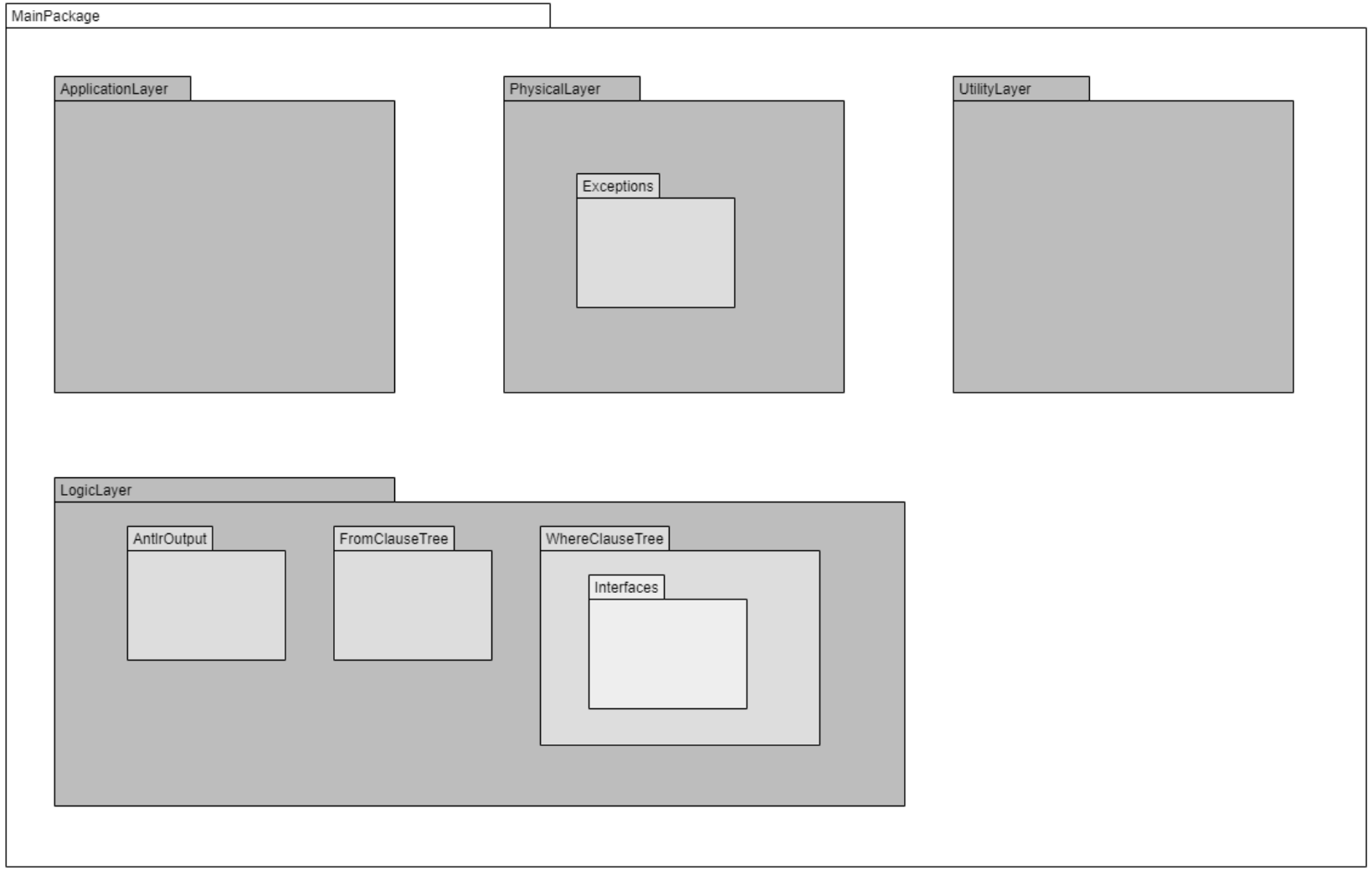
Esempio di definizione lexer con ANTLR

```
1  /*
2   * Lexer Rules
3   */
4  fragment A          : ('A'|'a') ;
5  fragment S          : ('S'|'s') ;
6  fragment Y          : ('Y'|'y') ;
7  fragment H          : ('H'|'h') ;
8  fragment O          : ('O'|'o') ;
9  fragment U          : ('U'|'u') ;
10 fragment T          : ('T'|'t') ;
11 fragment LOWERCASE  : [a-z] ;
12 fragment UPPERCASE  : [A-Z] ;
13 SAYS                : S A Y S ;
14 SHOUTS              : S H O U T S;
15 WORD                : (LOWERCASE | UPPERCASE | '_')+ ;
16 WHITESPACE          : (' ' | '\t') ;
17 NEWLINE             : ('\r'? '\n' | '\r')+ ;
18 TEXT                : ~[\])]+ ;
```

Esempio di definizione parser con ANTLR

```
1  cartesian returns [FromClauseNode nodeToReturn]:
2      n1=join  n2=cartesian_1[$n1.nodeToReturn] {$nodeToReturn = $n2.nodeToReturn;};
3
4  cartesian_1[FromClauseNode n] returns [FromClauseNode nodeToReturn]:
5      ',' n1=join n2=cartesian_1[new OperationTableNode(n, $n1.nodeToReturn, OperationTableNode.Op.CARTESIAN_PRODUCT , sessionID)] {$nodeToReturn
   = $n2.nodeToReturn;}|{$nodeToReturn = n;};
6
7  join returns [FromClauseNode nodeToReturn]:
8      n1=term n2=join_1[$n1.nodeToReturn]{$nodeToReturn = $n2.nodeToReturn;};
9
10 join_1[FromClauseNode n] returns [FromClauseNode nodeToReturn]:
11     JOIN n1=term {OperationTableNode joinNode = new OperationTableNode(n, $n1.nodeToReturn, OperationTableNode.Op.JOIN , sessionID);
12                   leaves = new ArrayList<>();} n2=join_1[joinNode] ON '(' constraint = booleanExpr ')'
13     {
14         WhereBooleanTree booleanTree = new WhereBooleanTree($constraint.node, leaves);
15         joinNode.setConstraint(booleanTree);
16     }
17     n2 = join_1[$n2.nodeToReturn]{$nodeToReturn = $n2.nodeToReturn;}
18     |{$nodeToReturn = n;};
```

Struttura dei package



Physical Layer: Le classi contenute rappresentano i concetti di tipo, tupla, pagina e relazione. Contiene anche il gestore delle pagine e delle tabelle.

Logic Layer: Le classi contenute permettono l'interpretazione della query vera e propria, al suo interno vengono rappresentati lexer e parser, comprese le classi generate da ANTLR.

Application Layer: Le classi contenute permettono l'invio di comandi SQL mediante shell.

Sintassi SQL riconosciuta

- Possibilità di dichiarare chiavi primarie e chiavi esterne per mezzo della sintassi DDL
- Possibilità di eseguire query DML complete, omettendo eventualmente valori nulli
- Esecuzione di query DQL contenenti prodotti cartesiani e theta-join specificati per messo delle keyword ‘JOIN ON’.

  L’esecuzione dei join viene effettuata per mezzo una particolare

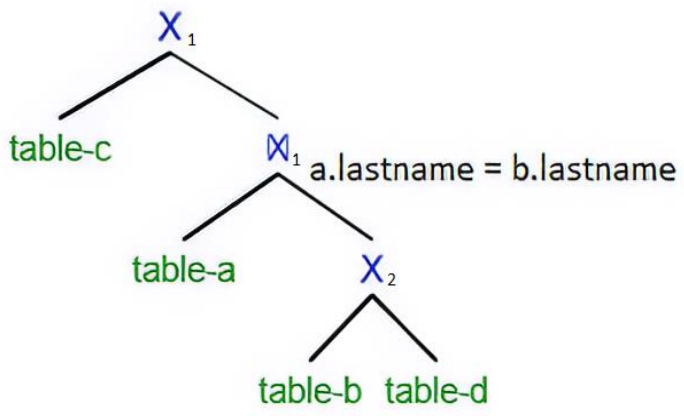
  implementazione dell'algoritmo block nested-loop join

Esempio di ottimizzazione logica

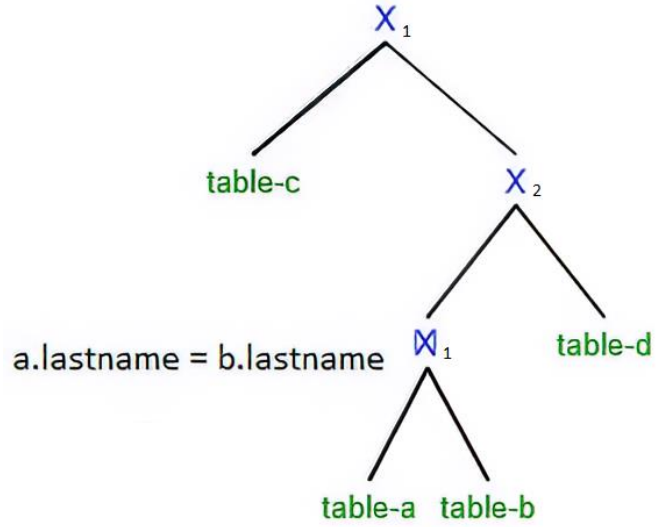
Prendendo in considerazione la porzione di query d'esempio:

persons c, persons a join persons b on (a.lastname = b.lastname), persons d

vengono riportati due possibili alberi sintattici differenti restituenti la medesima relazione:



Albero non ottimizzato



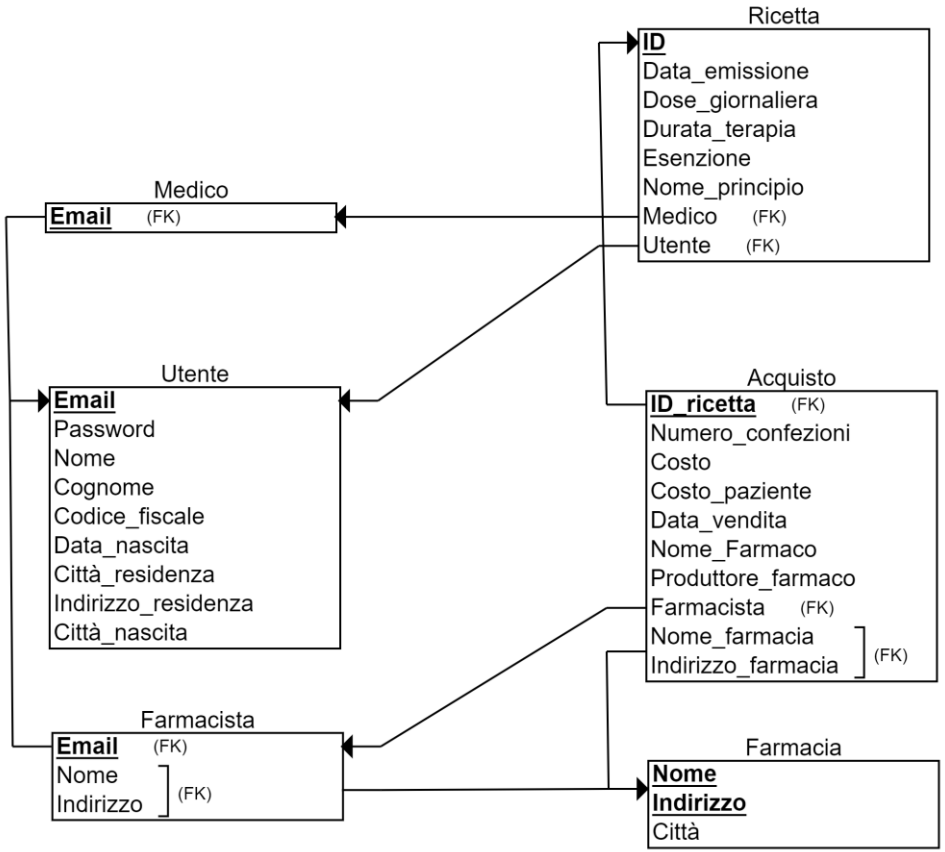
Albero ottimizzato

Analisi comparativa con SQLite

Specifiche Hardware macchina:

- CPU: Intel Core i5-3570
- RAM: 8 GB DDR3 1333MHz
- Memoria di massa: SSD Crucial BX500 1 TB

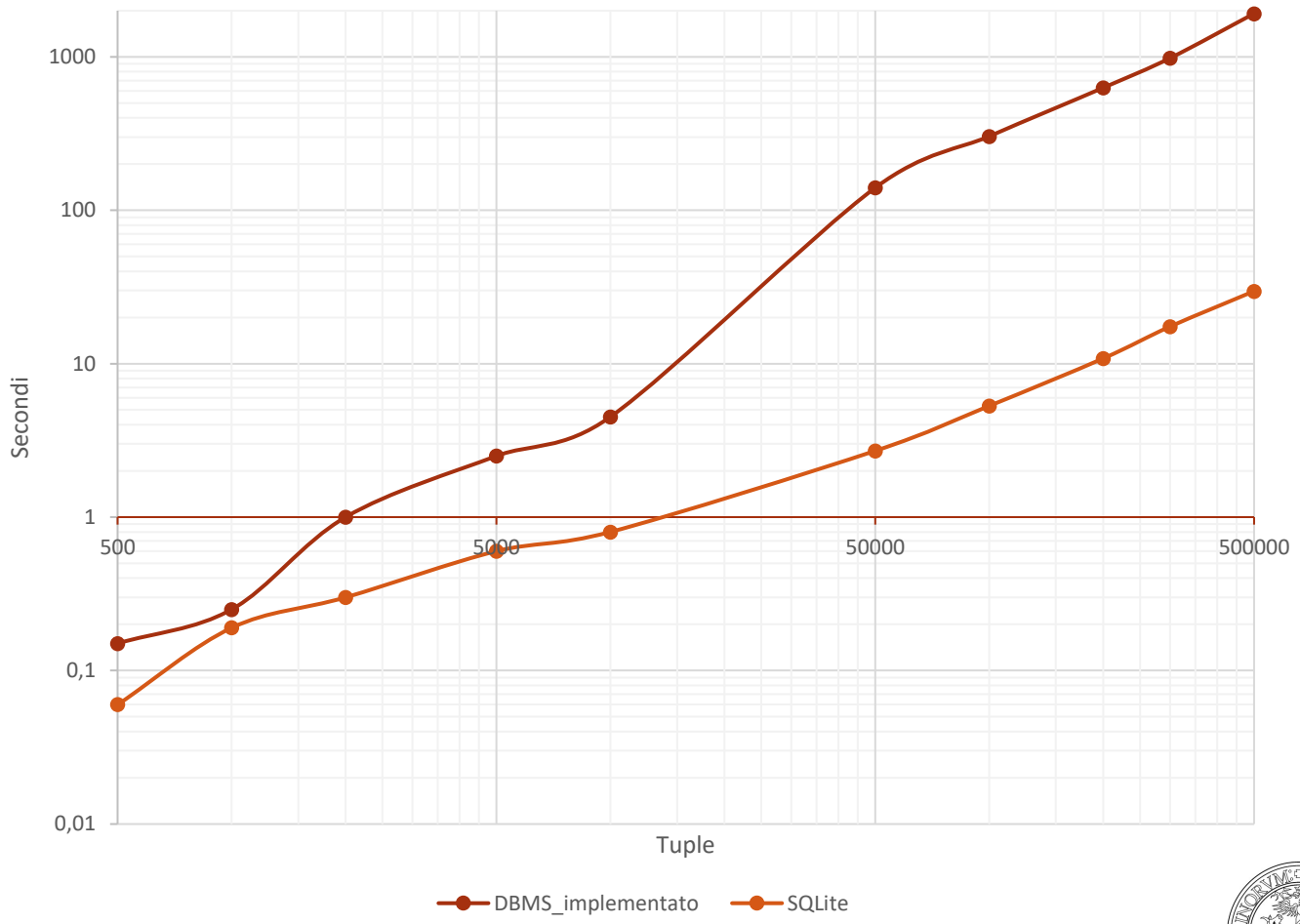
Schema relazionale d'esempio:



Esempio di query DML

Sono state valutate le tempistiche d'inserzione sulla relazione 'Farmacista':

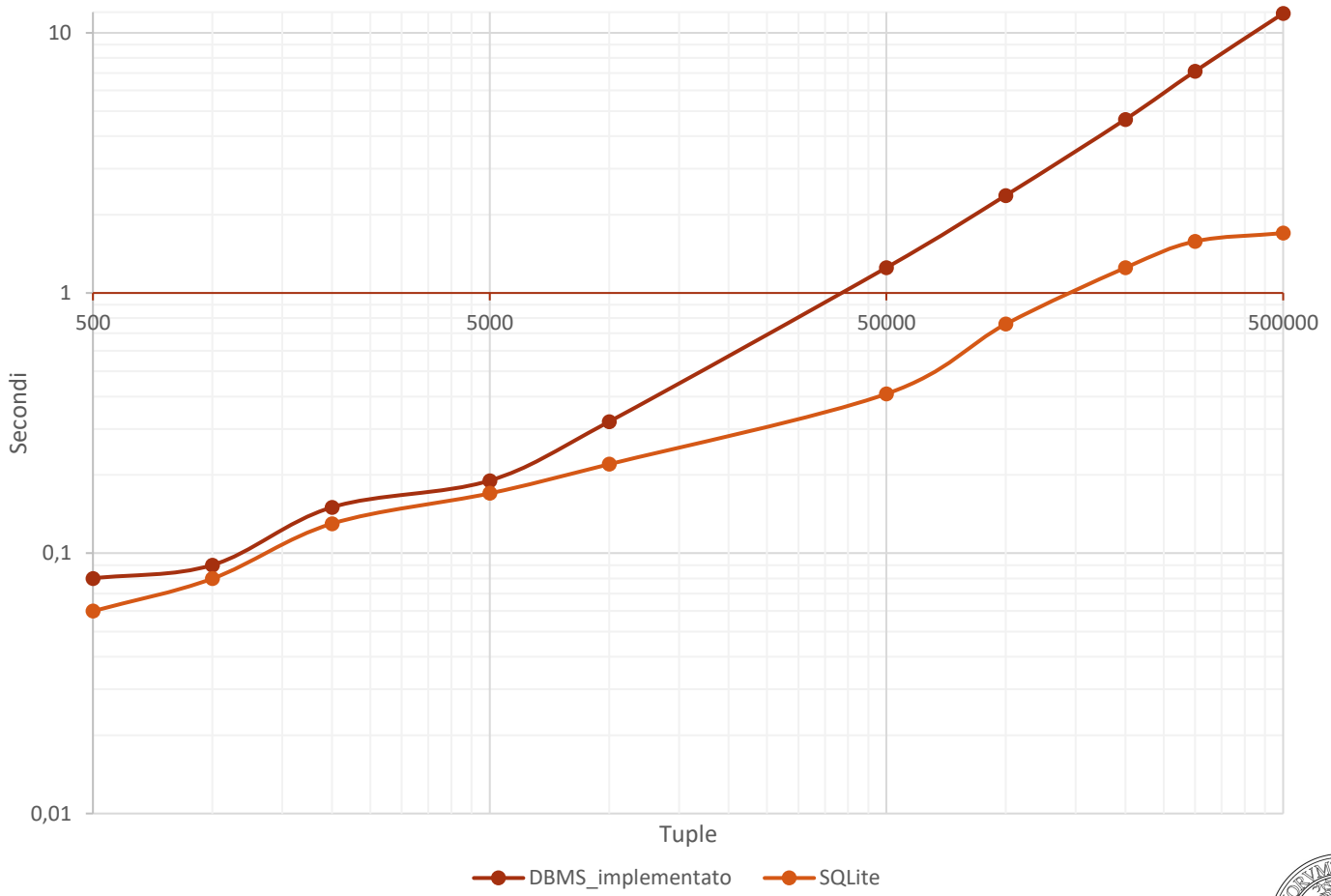
Numero di tuple inserite	Tempo di inserzione DBMS progettato	Tempo di inserzione SQLite
500	0.15 sec	0.06 sec
1000	0.25 sec	0.19 sec
2000	1.00 sec	0.30 sec
5000	2.50 sec	0.60 sec
10000	4.50 sec	0.8 sec
50000	2 min 20 sec	2.7 sec
100000	5 min 3 sec	5.30 sec
200000	10 min 28 sec	10.8 sec
300000	16 min 19 sec	17.4 sec
500000	31 min 42 sec	29.6 sec



Esempio di query DQL semplice

select * from farmacia where Citta = ‘nonEsistente’

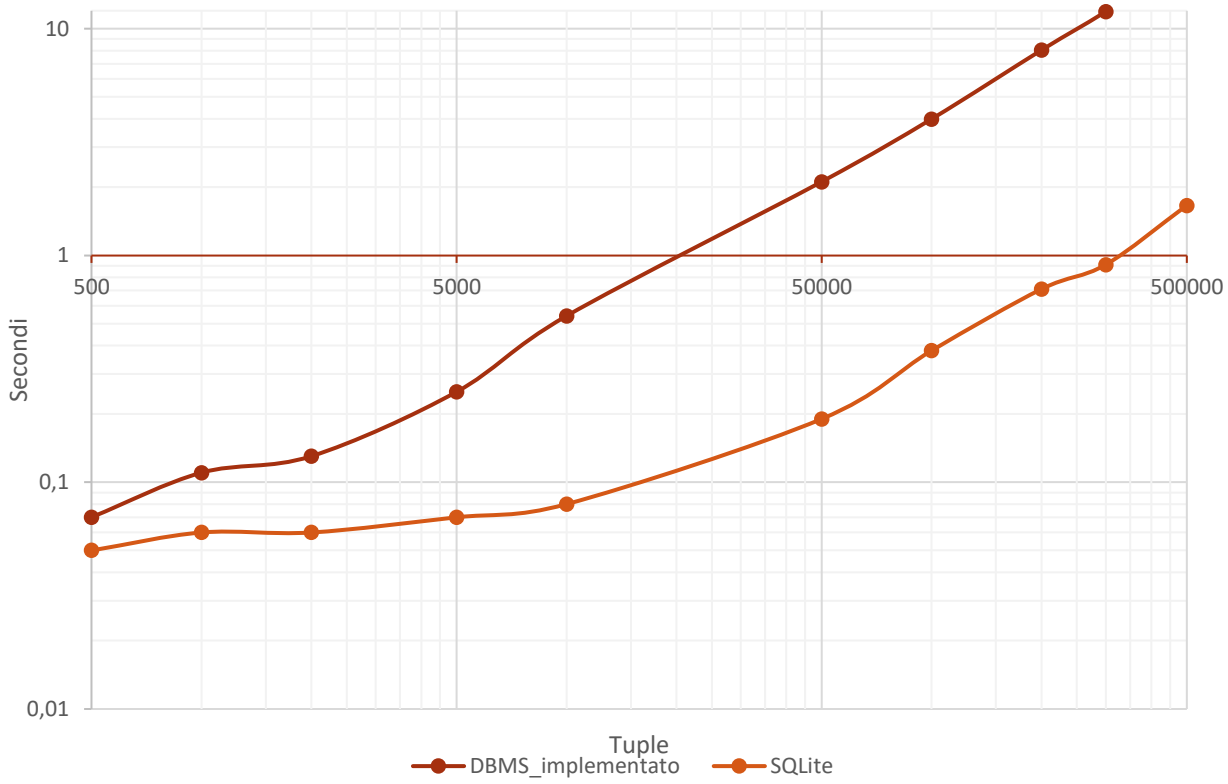
Cardinalità della relazione ‘farmacia’	Tempo di esecuzione DBMS progettato	Tempo di esecuzione SQLite
500	0.08 sec	0.06 sec
1000	0.09 sec	0.08 sec
2000	0.15 sec	0.13 sec
5000	0.19 sec	0.17 sec
10000	0.32 sec	0.22 sec
50000	1.25 sec	0.41 sec
100000	2.37 sec	0.76 sec
200000	4.64 sec	1.25 sec
300000	7.11 sec	1.58 sec
500000	11.85 sec	1.70 sec



Esempio di query DQL con sequenze di join

```
select a.ID_ricetta, a.Numero_confezioni, a.Costo, a.Costo_paziente, a.Data_vendita,a.Nome_Farmaco, a.Produttore_farmaco,
u.Nome, u.Cognome, u.Email,a.Nome_farmacia, a.Indirizzo_farmacia
from acquisto a join utente u on (a.Farmacista = u.Email) join ricetta r on (a.ID_ricetta = r.ID)
where u.Email = 'valoreUnico';
```

Cardinalità delle relazioni	Tempo di esecuzione DBMS progettato	Tempo di esecuzione SQLite
500	0.07 sec	0.05 sec
1000	0.11 sec	0.06 sec
2000	0.13 sec	0.06 sec
5000	0.25 sec	0.07 sec
10000	0.54 sec	0.08 sec
50000	2.11 sec	0.19 sec
100000	3.99 sec	0.38 sec
200000	8.03 sec	0.71 sec
300000	11.87 sec	0.91 sec
500000	20.18 sec	1.66 sec



Futuri sviluppi

- Sviluppo di un driver dedicato alla comunicazione tra applicativi esterni e DBMS
- Implementazione di più tipi accettati dalla sintassi DDL
- Estensione della sintassi DQL, applicabile all'interpretazione delle funzioni e delle sub-queries
- Implementazione di indici mediante B+Tree
- Ottimizzazioni logiche ulteriori