Abstract

We present open research Datalog-system. Datalog-system is capable to evaluate programs of Datalog, language of logic programming of databases. The fundamental feature of Datalog is capability to express recursion. One of the main features of our system is the functional approach to system realization. Datalog-system is written using programming language Scheme. Our system is loosely coupled with relational DBMS.

Introduction

One of lacks of traditional query languages to relational databases is their insufficient expressiveness. With usage of relational algebra is quite problematic to write to the database the arbitrary recursive query and to receive outcome. Datalog is the language of logic programming which has the greater expressiveness in matching with data definition languages and data manipulations of systems of databases.

At the same time, systems of logic programming do not provide technology of management large, reliable, permanent stored data sets with multiple access. Natural expansion of logic programming and management of databases consists in construction of the systems located on crossing of these two areas. Such systems are based on use of logic programming as language of queries and connect a formulation of queries and restrictions in style of logic programming to technology of databases for effective and reliable storing the data in a main memory.

Let's give informal conception how logic programming can be used as a query language, it is meant, that the reader is familiar with main concepts of logic programming. The foundation of a material of introduction is [1]. For a simplicity of perception in Figure 1 correspondences of concepts of logic programming and databases represented.

Concepts of databases	Concepts of logic programming
Relation	Predicate
Attribute	Predicate’s argument
Tuple	Ground clause (fact)
View	Rule
Query	Goal
Restriction	Goal (logic value)

Figure 1

Let's consider a relational database with relation PARENT (PARENT, CHILD). The tuple of relation PARENT contains pairs the subjects who are taking place in the relation the parent - the child. Let the database will consist of the facts: {parent (john, jeff), parent (jeff, margaret), parent (margaret, annie), parent (john, anthony)}.

The query « who children of John? » expresses with the help of the following Datalog’s goal:

? – parent (john, X).

Expected as a result of calculation of this query to a database the answer:

X = {jeff, anthony}.

Rules can be used for construction intensional databases (IDB) from extensional databases (EDB). EDB is simply relational database. In our example it includes relation PARENT. IDB is formed from EDB for of the rules determining its contents, instead of obvious storage of its tuples. For example - relation ANCESTOR includes as tuples all pairs an ancestor - the descendant, starting from parents:

ancestor(X, Y) :- parent(X, Y).

ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

Let's try to give the answer to a problem: «what does the Datalog program evaluate? ». We consider the program on Datalog the query operating above the extensional database and computing some result. Formally the result of the program of Datalog P applied to EDB E, will consist of all facts in issue which are logical consequence of a set of clauses $P \bigcup E$ and which predicate characters belong to in IPred $\Theta_P(E) = \{G \mid (P \cup E) \models G\}$.

Bypassing a set of formal assertions, we shall try to show, how the program of Datalog is evaluated. We shall consider a rule of Datalog R of sort L_0 :- L_1, …, L_n and the list of facts F_1, …., F_n. If there is a substitution θ such, that for all 1 <= i <=n $L_i\theta = F_i$, then from rule R and from facts F_1, …., F_n it is possible to infer the fact $L_0\theta$ for one step. The infered fact can be the new fact or already known. This rule is called as elementary products (EP).

For an evaluation of a set of all facts which can be infered for one step using EP, it is required to look all serially well-ordered sets of all facts in the database and to test, whether it is applicable by EP. If it is applicable, the new fact needs to be add to result. This procedure we name infer1.

Now with the help of injected concepts it is possible to show mechanically as the program on Datalog is evaluated.

```
ALGORITHM LFP(S)
INPUT: finite set S of clauses of Datalog
OUTPUT: result of the Datalog's program, i.e. a set of all ground facts being consequences S.
BEGIN
old := $\varnothing$;
new := FACTS(S);
WHILE new <> old DO
BEGIN
old := new;
new := old $\cup$ FACTS(S) $\cup$ infer1(RULES(S) $\cup$ old)
END;
RETURN result
END.
```

The written algorithm is the algorithm of calculation of the left fixed point of the Datalog's program.

Example. Consider set S of Datalog's clauses, including following rules R1: anc (X, Y):-anc (Z, Y), par (X, Z) and R2: anc (X, Y):-par (X, Y) and ground facts: {p (a, b)}, {p (b, c)}, {p (b, d)}, {p (c, e)}.

Let's show the work of LFP, listing of values which are accepted with a variable new at each iteration. Let new_i means value new after i repetitions of cycle.

new_0 = FACTS(S) = {{p(a, b)}, {p(b, c)}, {p(b, d)}, {p(c, e)}};

new_1 = new_0 $\cup$ FACTS(S) $\cup$ infer1(RULES(S)) = new_0 $\cup$ {{a(a, b)}, {a(b, c)}, {p(b, d)}, {p(c, e)}};

new_2 = new_1 $\cup$ FACTS(S) $\cup$ infer1 (RULES(S) $\cup$ new_1) = new_1 $\cup$ {{a(a, c)}, {p(a, d)}, {a(b, e)}};

new_3 = new_2 $\cup$ FACTS(S) $\cup$ infer1 (RULES(S) $\cup$ new_2) = new_2 $\cup$ {{a(a, e)}};

new_4 = new_3.

The evaluation is realized by algorithm LFP, is named as a bottom-up evaluation since the algorithm begins with reviewing ground facts of S and realizes driving "up", producing at the beginning all facts which can be inferred for one step from the lowermost clauses. Then all facts which can be inferred for one step from the lowermost clauses or from the facts which have been infered for one step, etc. One step of EP also is named as a direct conclusion because if rules of Datalog are represented as implication, they are handled in a "direct" direction in sense of the implication's sign.

There is also completely other alternative bottom-up (to a direct inference) to an evaluation of Datalog's programs. It is shown, as rules of Datalog can be handled in the opposite direction (since goal clause) by construction a proof tree by top-bottom evaluation. This method is named also as a backward inference. It is shown also as the known method of the resolution can be used for the answer to queries of Datalog.

Nonrecursive Datalog in accuracy is equivalent RA^+. It is possible to translate each rule of Datalog in equation RA^+ and the back. RA^+ (the positive relational algebra) is a relational algebra without operation a difference. In RA^+ we consider only base operations - sample, a projection, the cartesian product, association and a difference. It means, that Datalog is at least so expressive, as well as RA^+. Thus, full Datalog is strictly more expressive, than RA^+ as using Datalog it is possible to express inexpressible in RA^+ recursive queries. For example, relation ANC can be expressed only with the help of recursive rules or recursive equations RA^+. It is fair for the most of recursive programs of Datalog.

On the other hand, there are expressions of full relational algebra which cannot be expressed with the help of Datalog's programs. These are the queries using the operator difference. Let there are sets, for example, two binary relations R and S. There is no Datalog - rule which expresses relation R-S. Nevertheless, such expressions can be written down with the help of logic denying. Also there are expansions of Datalog calculating queries with negation. However we consider "pure" Datalog.

Paths of implementation

The overall aim of creation of experimental Datalog-system consists in representation educational and to a scientific organization of the completed system of the logic programming basing in language of Datalog which could be used in various directions, including in practical works and scientific researches of students and experts.

From experimental of the system follows, that there is no need to involving all set of methods and the algorithms used to Datalog. It was necessary to present any subset of all methods to understand, what users and on what it is possible to concentrate the further developments really require. Outgoing from this and limitations of possibilities of creators main reqiurements to the system have been formulated.

One of key features of the system consists in the functional approach to its implementation. Functional programming and logic programming - are main paradigms of declarative programming, therefore their likeness speaks about much. A set of methods of an evaluation of programs on Datalog can be easily and naturally written with usage of recursion which is an integral part of functional languages. On the other hand, efficiency of implementations of languages of functional programming increases, that allows to speak about practical applications.

The language of implementation became Scheme [2] - a modern dialect of Lisp. Scheme is the functional programming language, it is simple and has the rich built - in possibilities of manipulation by complex data structures. Typically that is primary Scheme formed as the algorithms notation language.

There is a set of implementations Scheme, but one of the most effective compilers of Scheme language - Bigloo [3] which has several doubtless advantages has been selected. First, as against many other implementations of Lisp and Scheme, Bigloo is the compiler. Second, it allows to create executable files for various platforms, including a Windows 2000, XP (without usage Cygwin) and .NET. Thus, the Datalog - system is platform-independent.

It is impossible to tell unequivocally, what coupling - loose or strong is better. It depends on many factors. In this realization the Datalog - calculator has been coupled with DBMS by a principle of ' loose coupling '. The choice of concrete DBMS isn't very important, the system can be couple with MySQL, PostgreSQL, Oracle. Hovewer Datalog - system has been coupled with SQLite DBMS [4].

The choice of DBMS for coupling has been made for the benefit of SQLite as it, firstly, is the most simple and simultaneously possesses rich opportunities in sense of the interface. Secondy, it is least confusing, it works on small databases (up to 14 Mb) faster others, for example PostgreSQL or MySQL [<http: // www.sqlite.org/speed.html>].

System Architecture

The architecture of system is submitted in Figure 2. The Datalog - program comes in as an input. The Datalog - system makes the syntactic analysis of the program, receives internal representation. Depending on the chosen method of calculation Datalog - rules can be translated

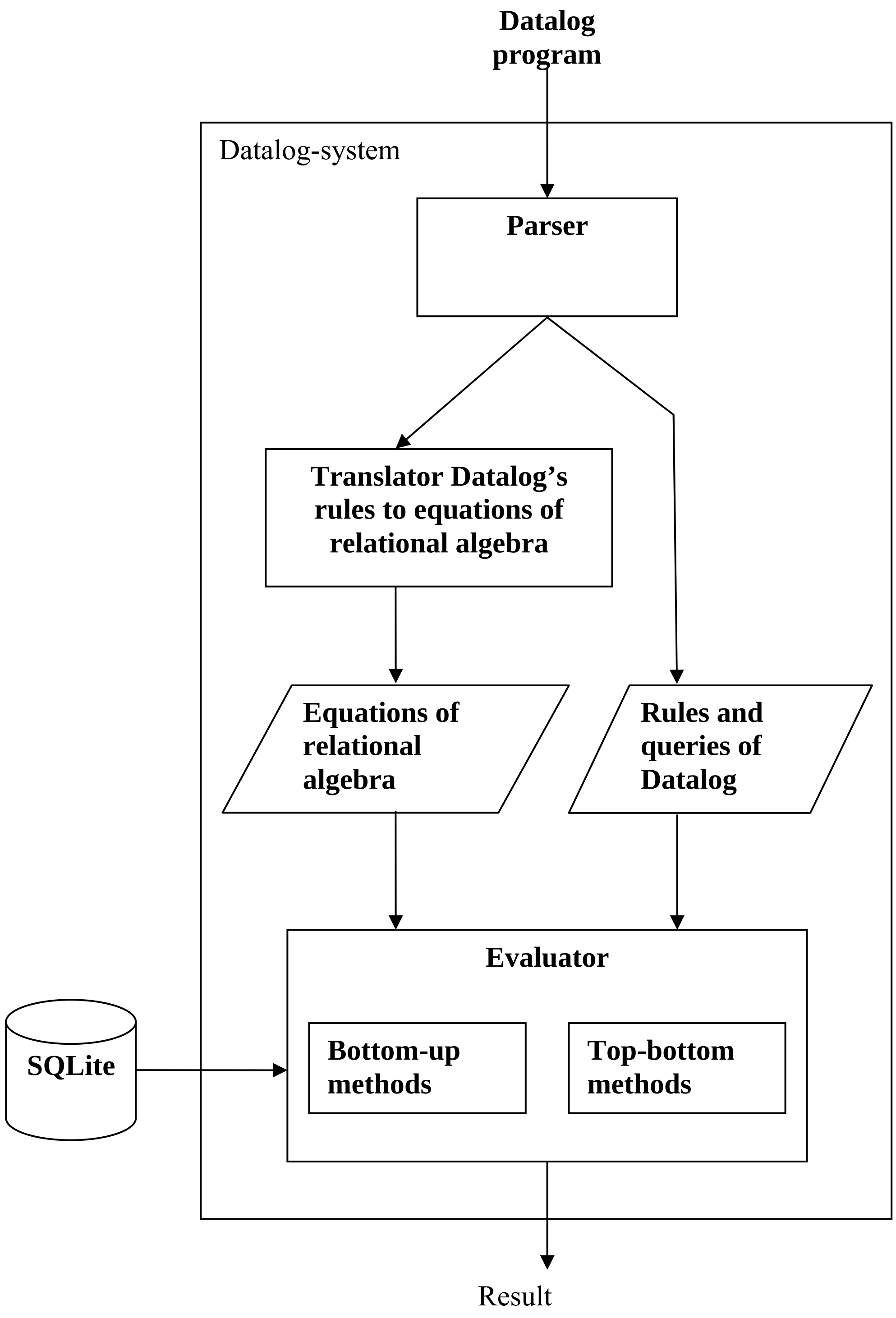
Datalog program
Datalog-system
Parser
Translator Datalog's rules to equations of relational algebra
Equations of relational algebra
Rules and queries of Datalog
Evaluator
Bottom-up methods
Top-bottom methods
SQLite
Result

Figure 2 System architecture

to the equations of relational algebra and come in the evaluator.

Before the beginning of calculations the system loads tuples of necessary relations from a database, using the program interface of DBMS. The received program and queries are calculated with the help of bottom-up methods (the Jacoby, Gauss-Zeidel, semi-naive) and top-bottom QSQ (the query - subquery). Then the result is given out.

The parser works, using the following grammar:

<Program>	::=	<Statement List>
<Statement List>	::=	<Statement> | <Statement List> <Statement>
<Statement>	::=	<Rule> | <Query> | <Processing Instruction>
<Processing Instruction>	::=	! <Instruction>
<Instruction>	::=	Edb filename
<Rule>	::=	<Predicate> :- <Predicate List> .
<Predicate>	::=	constant (<Term List>)
<Term List>	::=	<Term> | <Term List>, <Term>
<Term>	::=	constant | <variable>
<variable>	::=	identifier | _
<Predicate List>	::=	<Predicate> | <Predicate List>, <Predicate>
<Query>	::=	? <Predicate List> .

The line starting with ';' is the comment.

Benchmarks and analysis of the results

At present in Datalog - system there is no strategy of application of the certain computing method at different programs and queries. Therefore comparative testing computing methods has been carried out.

Benchmarking of algorithms was carried out on three programs. Results are represented as tables and approximating schedules. In the first column of tables there is number of experience, in the second and the third the initial and inferred amount of the facts is showed accordingly. For EDB with several relations (an example 2 and 3) the total amount of the facts is resulted. In the staying columns - spent time for different algorithms. Calculations were made under Linux RedHat 8.0 on processor Intel Pentium IV 1.5Ghz, 256 Mb RAM.

Program L1:

```
anc(X, Y) :- anc(X, Z), par(Z, Y).
anc(X, Y) :- par(X, Y).
? anc(X, Y).
```

25 iterations of experiences, in each of which have been carried out were random formed EDB. In the first EDB there were 10 facts, in 2-nd - 20, etc. In 25 EDB there were 250 initial facts. The facts represent pairs integers, each of numbers in this case changed from 1 up to 80. Thus, the maximal number of the facts in given EDB equally 6400. Results of five experiences are resulted in Table 1. Method QSQ was applied here to the query? anc (X, Y). Here it returns all predicate anc.

№	Total amount of facts		Measured time (seconds)			
	Initial	Inferred	Jacobi	Gauss-Z		QSQ
1	10	5	0	0	0	0
5	50	49	0.02	0.02	0.01	0.02
10	100	390	0.61	0.6	0.28	0.57
15	150	3753	57.01	57.28	20.89	60.00
20	200	4989	92.51	91.88	29.24	95.16
25	250	5602	115.49	110	34.9	115.44

Table 1 Evaluation of program L1

On the average operating times of methods Гаусса-Зейделя, semi-naive and QSQ make in percentage terms relatively the Jacoby accordingly 97.9 %, 32.62 % and 103.54 %. Clearly, that the semi-naive method outstrips methods of the Jacoby and Гаусса-Зейделя. As program L1 contains only one intensional predicate, the Gauss-Zeidel approximately also is effective, as well as the Jacoby. QSQ it is created for calculation of the marked queries, therefore it lags behind all others on the query containing only variables.

Table 2 is called for the analysis of efficiency QSQ on the marked queries to L1. The same are used EDB.

№	Total amount of facts			Measured time (seconds)			
	Initial	Inferred	Relevant	Jacobi	Gauss-Z	s/n	QSQ
1	10	5	0	0	0	0	0
8	50	49	22	1.21	1.21	0.56	0.74
10	100	390	13	0.61	0.6	0.28	0.21
15	150	3753	61	57.04	57.31	20.92	25.91
20	200	4989	72	92.54	91.91	29.27	41.15
25	250	5602	75	115.53	110.04	34.94	26.69

Table 2 Evaluation of program L1, ? anc(X, “11”).

Testing has shown, what at the query? anc ("11", "11") to program L1 operating time QSQ almost 0 whereas other methods remains approximately same, as well as in Table 1. The result is easily explained to that QSQ practically at once chooses the unique necessary fact, and other methods at any query all over again calculate the program, and then choose the necessary facts. Also it has been noticed, that time of performance of any simple query (from one subquery) remains to constants (to the same IDB).

On Figure 3 dependences of performance time of the query ? anc (X, "11") from the ratio of number of the relevant facts to total number of the IDB facts are shown. Approximation was carried out by all experiences by linear functions. Benchmarking was carried out only semi-naive and QSQ methods as from Table 2 it is visible, that the Jacoby and Гаусса-Зейделя essentially lag behind.

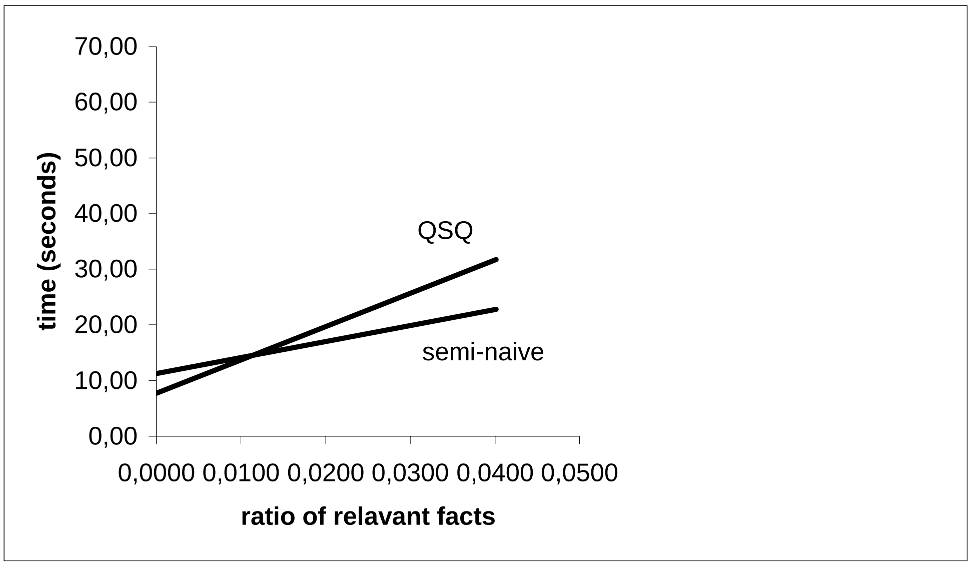

Figure 3 Evaluation of program L1, ? anc(X, “11”).

Interpretation is simple: the more the ratio of the relevant facts, the worse QSQ works and the better semi-naive works. Top-bottom method QSQ wins for semi-naive in a case, when a ratio of the relevant facts from a total number infered less than the defined level. Thus, having any a priori knowledge of allocation of values in IDB and knowledge (empirical) this threshold value it is possible to draw conclusions about the strategy of application of this or that method to various programs.

Conclusion

We have presented the loosely coupled Datalog - system, storing the facts in the lightweight SQLite DBMS, parsing program on Datalog and calculating this program using four methods: the Jacoby, Gauss-Zeidel, semi-naive and QSQ.

Comparative testing computing methods during which the coordination of practical results with the theory has been checked up were made.

Possible perspective directions of work are construction of the system realizing Datalog with negations, parallel realization of computing methods with the purpose of use on clusters. Integration of Datalog - system with more respectable DBMS, for example, DB2 or Oracle DBMS is possible too.

Literature:

[1] S. Ceri, G. Gottlob, L. *Tanca: Logic Programming and Databases*, Springer, 1990.

[2] R. Kelsey, W. Clinger, J. Rees (eds.), Revised^5 Report on the Algorithmic Language Scheme, *Higher-Order and Symbolic Computation*, Vol. 11, No. 1, September, 1998. http://www.schemers.org/Documents/Standards/R5RS

[3] Bigloo Homepage http://www-sop.inria.fr/mimosa/fp/Bigloo.

[4] SQLite Homepage http://www.sqlite.org/

[5] R. Ramakrishnah, J. Ullman. A Survey of Research on Deductive Database Systems. 1993

[6] Rajs. Sunderraman, Rajar. Sunderraman A Deductive Rules Processor for SQL Databases. ACM, 1998.