PRACE NAUKOWE AKADEMII EKONOMICZNEJ WE WROCŁAWIU
Nr 1064 2005
Pozyskiwanie wiedzy i zarządzanie wiedzą

Igor Wojnicki, Antoni Ligęza

University of Missouri – St.Louis, USA; AGH University of Science and Technology, Poland

HANDLING RECURSIVE QUERIES WITHIN RDBMS WITH JELLY VIEW TECHNOLOGY. SOME EXPERIMENTAL RESULTS WITH THE REDARES SYSTEM

1. Introduction

Some most significant limitations of information processing within contemporary Relational SQL. Two typical ·classes of unsolvable problems include the following examples: Database Management Systems follows from the lack of recursive queries at the level of

- **C1** Traversal of structurally complex data structures (such as graphs, trees, terms, lists etc.).
- **C2** Search for Admissible Solutions under specified constraints problems (finding specific subsets of a given set, generation of structural solutions satisfying specific constraints etc.).

These two classes cover many different problems such as: plan generation, searching for acceptable or optimal solutions, analysis of structures, decision sup-port systems, constraint programming problems. The limitations are inherited in SQL which is based on simple *relational algebra* concepts [1] while handling re-cursive queries requires more advanced approach (recursive queries are addressed by SQL99 standard, however this feature is implemented only in DB2 by IBM).

These problems are tackled by the ReDaReS system, which introduces rule-based processing to the database systems, providing virtually limitless processing capabilities. The system implements *Jelly View* technology [7,8,9]. The technology allows for encoding intensional knowledge using the **Prolog** language syntax within RDB tables (**Prolog**, being a superset of **Datalog**, has been proven to be a proper methodology to provide the Logical Data Model [3]). As the result the functionality of RDBMS is extended towards that of Deductive Databases [2,4,5].

The **Prolog** clauses are decomposed into data, and stored in the RDBMS founding so-called *Logic Program* (or just *Program*). The database becomes a complete source of knowledge, both extensional and intensional.In order to process intensional knowledge an inference engine is coupled with the RDBMS. The results of the inference process are visible as regular views, accessible by SQL.The state of the view is generated dynamically by the inference engine.

As the result problems of the **C1** or **C2** class can be smoothly approached keeping SQL as the outermost communication technology.The proposed methodology extends the system catalog towards storing, accessing, and processing intensional knowledge within RDBMS.

2. Intensional Knowledge Decomposition

The clauses, which provide intensional knowledge are decomposed and stored in a well defined manner in the database. The appropriate Entity Relationship Diagram is given in Fig.1. The entities `clause, argument logical operator` provide the *Program* - a set of decomposed **Prolog** clauses.

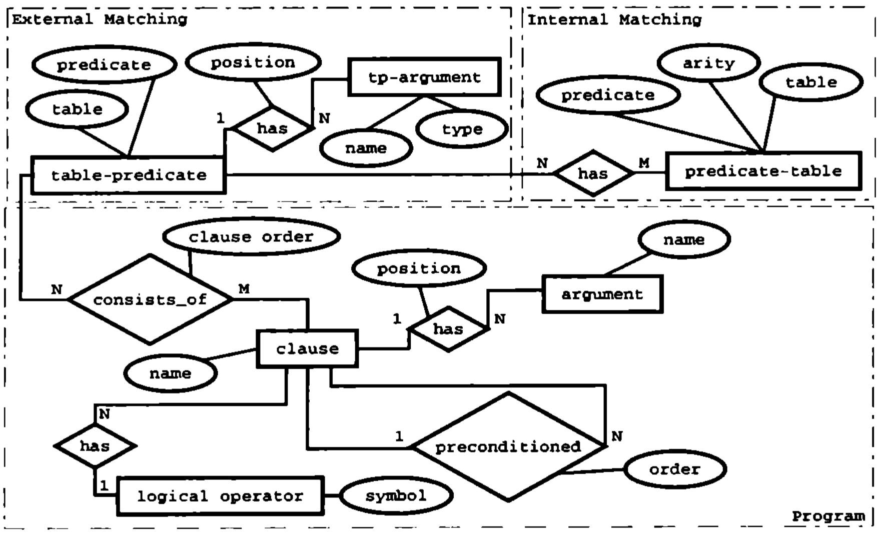

Figure 1. Storing Intensional Knowledge in RDBMS, Entity Relatio nship Diagram

The decomposition takes place at the predicate level, not the term level. Arguments of the predicates are not further decomposed (see entity argument). If the arguments were more complex expressions (terms, structures etc.) they would not be decomposed. They would be treated as atomic expressions instead. The lack of further decomposition is forced because of the performance reasons. If the arguments were decomposed, in order to recreate a clause, a recursive query would have to be issued. Since most of the RDBMS do not support such queries, there would be a performance loss (this issue is subject to the further research).

There are two more components which are vital to intensional knowledge processing. They define the way how the user accesses the *Jelly View*, what clauses a particular *Jelly View* is composed of, and how the inference engine accesses extensional knowledge gathered in the database. These are *External Matching* and *Internal Matching* respectively. The *External Matching* establishes the name of a *Jelly View* and its schema (entities: `table-predicate, tp-argument`), in terms of the Relational Model. It also indicates what is the goal for the inference engine (the attribute `predicate` of the `table-predicate` entity). The goal is used to generate the state of the *Jelly View*. It also denotes what clauses should be used by the inference engine for the particular *Jelly View* (the relationship `consists of` between the entities: `table-predicate` and `clause`). The arity of the goal matches the number of attributes of the *Jelly View*. Moreover, the goal has to be a valid predicate covered by the *Program* clauses.

It is worth pointing out, that some clauses may be shared among some number of *Jelly Views*. It enables modular programming and allows to reuse the code. In such a way libraries of problem-specific modules may be provided and stored for further use.

The *Internal Matching* enables the inference engine to access extensional knowledge gathered in the database. It indicates which extensional knowledge is available to the inference engine.It is provided by defining a correspondence (mapping) between a predicate and a relation (`the predicate-table` entity). In such a way simple clauses of the given predicates are provided by the tuples from the given relations. Moreover, the same mapping between the predicate and the relation might be used by more than a single *Jelly View*, since there is a many-to-many relationship (`has`) between the *External Matching* and *Internal Matching*.

All the relations used by the *External Matching*, *Internal Matching* and *Program* extend the *system catalog* of the RDBMS allowing it to store intensional knowledge. Since intensional knowledge is stored as (decomposed) **Prolog** clauses; a **Prolog** inference engine has to be coupled with the RDBMS to interpret it.

3. Implementation: the ReDaReS System

ReDaReS is a prototype system implementing the proposed *Jelly View* technology. It is designed to be a loosely coupled middleware between client applications and the RDBMS. It uses ODBC as a communication protocol. In this way the system may be applied to any standard relational database, which supports ODBC, regardless of its physical organization, vendor, or SQL dialect it uses (the loose coupling has some important performance issues, though).

The system is designed to be connected via ODBC to the RDBMS. Furthermore, it becomes an ODBC data source (being ODBC server has not been implemented yet, this functionality is currently replaced with text oriented query/response interface) which mimics the original database. Instead of querying the database one should query ReDaReS. The system is transparent. It means that all the queries, which were being accepted by the database, are still accepted while querying ReDaReS, since they are forwarded to the database directly. The database response is forwarded back to the client application.

If the query concerns a *Jelly View* the system takes the processing over. It gathers intensional knowledge, which is accompanied with particular *Jelly View*, and launches the inference engine. As the result, the inference engine provides the state of the *Jelly View*. The state is temporarily stored in the RDBMS (storing the state of *Jelly View* in the database is forced by the middleware architecture). Then, the original query is forwarded to the database. Finally the database sends the reply to ReDaReS and it forwards it back to the client application.The data flow is given in Fig.2. The inference engine is based on **SWI Prolog** [6].

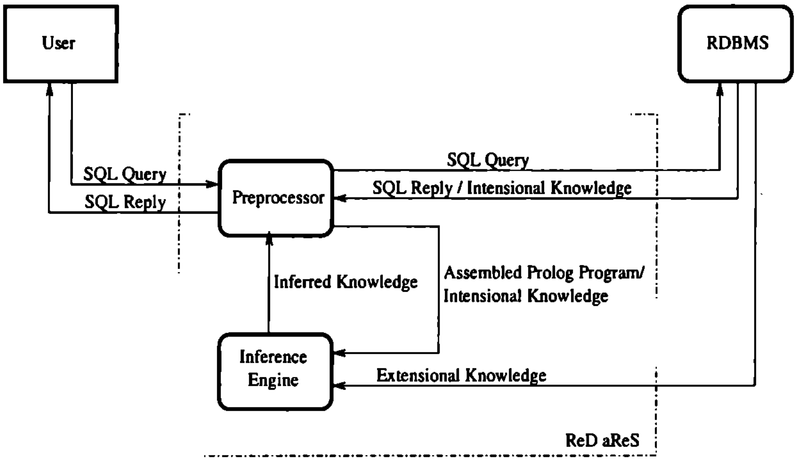

Figure 2. ReDaReS and RDBMS, Data Flow

The inference engine has a direct access to extensional knowledge gathered in the RDBMS. Current implementation downloads all extensional knowledge to the inference engine before the inference process takes place. It introduces a time-delay and this issue is subject to further research. The experiments showed that downloading extensional knowledge on-demand (during the inference process) causes even more slowdown.

4. Performance Issues

A series of experiments have been carried out to investigate performance of the proposed system. The main test subject is the tree traversal problem. It is finding predecessor or ancestor nodes in a tree structure stored in a relation. The test tree is composed of 12 levels, having 3 children at each node. The number of nodes is given as a Sum of Geometric Series:

$$S=a+ar+ar^2+...+ar^{(n-1)}=a\frac{(1-r^n)}{(1-r)},$$

and with a=1, n=12, r=3, there are: S=265720 nodes. Such a tree structure is represented as a single relation: `subject(Parent_id, Item_id, Name)`. The **Prolog** code for finding relationships among nodes is given below:

```
find(Parent,Child):- tree(Parent,Child,_)
find(Parent,Child):- tree(Parent,C1,_),
                     find(C1,Child)
```

In order to generate a *Jelly View*, which is capable of finding the relationships, the above program is decomposed. The *External Matching* is set to define the *Jelly View*, which has the following schema: `find(parent_id, child_id)`, and uses `find/2` as the goal. Furthermore, the *Jelly View* corresponds to the above clauses. The *Internal Matching* defines, that simple clauses (facts) of the `tree/3` predicate are taken from the `subject` relation.

There has been a series of experiments carried out in an isolate environment (with the PostgreSQL RDBMS, and ReDaReS running only). In general, they focused on finding all child nodes of the node at different levels of the tree structure. In particular these levels are: 11, 9, 7, 5, 3, 1, where the level number 1 is the root. The query, used in the experiments, finds all child nodes of the given parent one. The parent node is selected by passing the first argument to the *Jelly View* and setting the second one unbounded. The precise value passed as the `parent_id` is a result of a sub-query given as the argument.The query is given below.

SELECT * FROM find('SELECT XXX',);

where `XXX` is the following `parent_id` queried in turns: 29523, 3279, 363, 39, 3, 0, which correspond to the levels of the tree structure. The chart in Fig.3 shows the results.The X-axis is the number of nodes obtained from the query (its different for different levels of the tree). The Y-axis is the elapsed processing time which includes the RDBMS processing time, the ReDaReS processing time and the communication overhead.

The performance of ReDaReS is compared with the performance of a PSM (Permanent Stored Module, also known as Stored SQL Function/Procedure) with the same functionality. The PSM is written in PL/pgSQL, which is a native PostgreSQL procedural language. The experiments take into account the database indexing as well. The graphs labeled `elapsed` and `elapsed idx` represent the timings of ReDaReS on the relation `subject`, without and with the indexing turned on respectively. The graphs `elapsed pl` and `elapsed pl idx` represent the timings provided by the PSM without and with the indexing. As it is showed, the ReDaReS system outperforms the PSM if there is no indexing involved. Turning the indexing on (on the attributes of the `subject` relation) has a tremendous impact on PSM based solution, which becomes faster than ReDaReS.

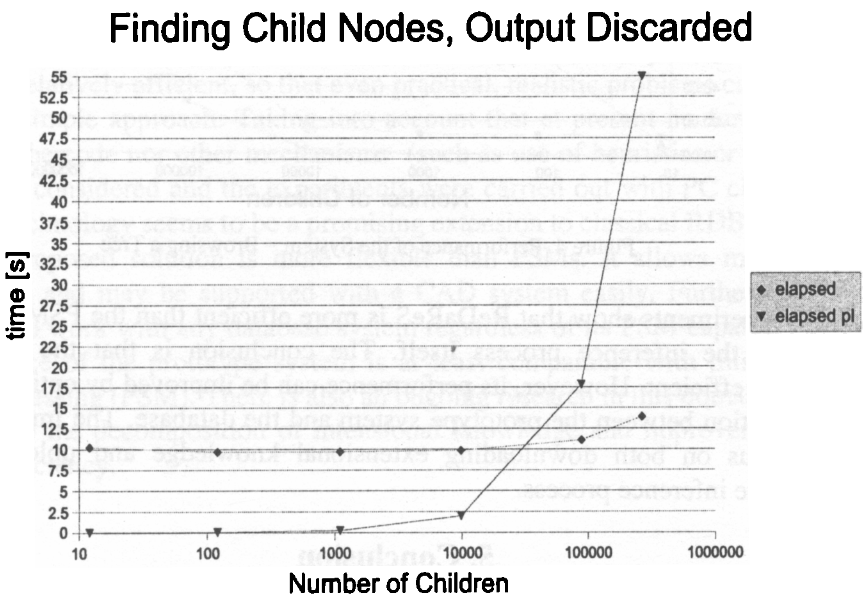

Figure 3. Performance of the System – Browsing a Tree (Output Discarded)

Some of the ReDaReS slowdowns regard the fact, that the system has to feed the database with the results from the inference process. The second set of experiments takes it into account (see Fig.4). This time the output tuples are discarded. They are not generated by ReDaReS nor PSM. Such an approach investigates timings of the inference process alone, without the communication overhead (which is pretty significant considering 265720 tuples returned while querying at level 1).

The *entry time-delay* of the ReDaReS system is caused by the necessity of downloading extensional knowledge into the inference engine, plus the time needed to start the engine up. It takes about 10 seconds for this particular experiment, which is downloading 265720 tuples into the inference engine.

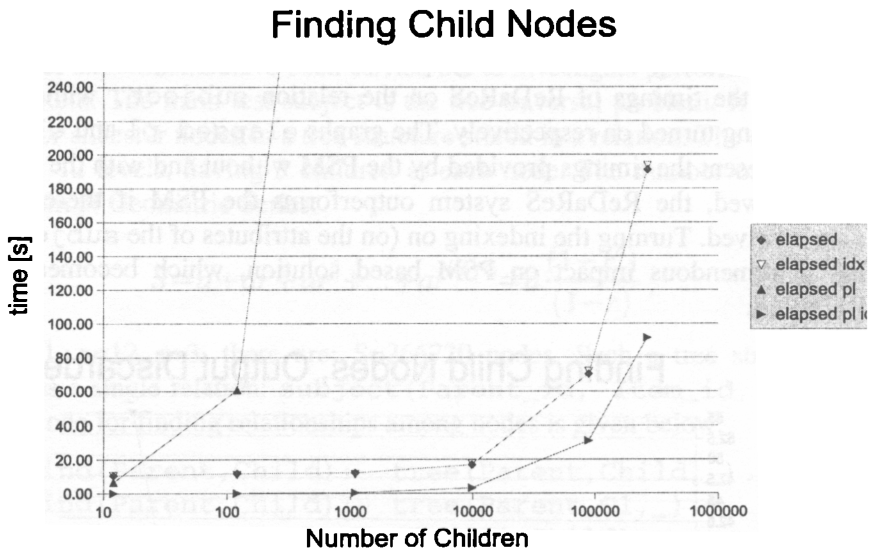

Figure 4. Performance of the System – Browsing a Tree

The experiments show that ReDaReS is more efficient than the PSM approach concerning the inference process itself. The conclusion is that the system is sufficiently efficient. However, its performance can be improved by optimizing the communication between the prototype system and the database. The improvement should focus on both downloading extensional knowledge and uploading the results of the inference process.

5. Conclusion

The *Jelly View* technology extends the Relational Database systems, in such a way, that even more complex problems than these specified in Section 1 can be smoothly approached keeping SQL as outermost communication technology. These problems are tackled by introducing rule-based processing to the database systems. The proposed approach allows to solve problems of **C1** and **C2** classes (see Section 1), which cover: plan generation, searching for acceptable or optimal solutions, analysis of structures, decision support systems, constraint programming problems.

The functionality of Relational Databases is significantly extended towards that of Deductive Databases [2,4,5], by integrating the proposed technology into a database. The technology is based on coupling the existing **Prolog** inference engine with the database.

The prototype ReDaReS system, providing the proposed technology, has been implemented and tested on a number of problems. It has shown out the following key properties. The original functionality of the database is preserved. The database is coupled with the **Prolog** inference engine.

Both data and knowledge are stored within the Relational Database; no additional knowledge base is necessary.

There are modules specified in **Prolog** for extending data processing capabilities, which are decomposed and stored as data in the database.

Results of the inference process, which is inferred knowledge, are accessible as dynamically generated SQL views; the necessary code is generated on request from components stored in the database. The communication method between the user and the database remains SQL.

Obviously, the problems examined (see Fig. 3, Fig. 4) are of exponential computational complexity. However, the technology and its implementation turned out to be relatively efficient, so that even practical, realistic problems can be solved with this simple approach. Taking into account that at present no further optimization of the code nor other mechanisms (such as use of heuristics or constraints) have been considered and the experiments were carried out with PC class computers, the technology seems to be a promising extension to classical RDBMS.

The proposed solution is more flexible than PSMs. It allows modular programming, and may be supported with a CAD system easily. Furthermore, it is designed to work with any database system regardless of its PSM capabilities. The performance of the prototype system is at least comparable with this of server-based processing (PSM). There is also an ongoing research in this domain, which is focused on the decomposition of intensional knowledge and improvement of the system efficiency.

References

[1] Connolly T., Begg C., Strachan A., *Database Systems, A Practical Approach to Design, Implementation, and Management*, Addison-Wesley, 2nd edition, 1999.

[2] Derr M.A., Morishita S., Phipps G., *The Glue-nail Deductive Database System: Design, Implementation and Evaluation*, VLDB Journal, 3(2):123 160, 1994.

[3] Nilsson U.,Mabuszynski J., *Logic, Programming and Prolog*, John Wiley & Sons, 1990.

[4] Ramakrishnan R., Srivastava D., Sudarshan S., Seshadri P., *The CORAL Deductive System*, VLDB Journal: Very Large Data Bases, 3(2):161 210, 1994.

[5] Vaghani J., Ramamohanarao K., Kemp D.B., Somogyi Z., Stuckey P.J., Leask T.S., Harland J., *The Aditi Deductive Database System*, Technical report, University of Melbourne, 1994.

[6] Wielemaker J., *SWI-Prolog Reference Manual*, Dept. of Social Science Informatics, University of Amsterdam, 2002.

[7] Wojnicki I., *A Rule-based Inference Engine Extending Knowledge Processing Capabilities of Relational Database Management Systems*. PhD thesis, AGH University of Science and Technology, 2004. A copy is availbale upon request from the author: wojnickiagh.edu.pl.

[8] Wojnicki I., Janikow C.Z., *Extending Data Processing Capabilities of Relational Database Management Systems*, [in:] H.R. Arabnia, Rose Joshua, and Youngsong Mun, editors,

International Conference on Artificial Intelligence, volume I, pages 388 393, Las Vegas, Nevada, USA, 2003. CSREA Press.

[9] Wojnicki I., Ligeza A., *An Inference Engine for Rdbms*, In 6th International Conference on Soft Computing and Distributed Processing, Rzeszów, Poland, 2002.

OBSŁUGA REKURENCYJNYCH ZAPYTAŃ W SZRBD Z TECHNOLOGIĄ *JELLY VIEW*. REZULTATY BADAWCZE Z SYSTEMEM ReDaReS

Streszczenie

Możliwości przetwarzania relacyjnych baz danych są ograniczone. Szczególnie dwie kategorie problemów są trudne do rozwiązania: przechodzenie po złożonych strukturach danych (takich jak wykresy, schematy, zestawienia itd.) oraz poszukiwanie dopuszczalnych rozwiązań zgodnie z określonymi ograniczeniami (znajdowanie specyficznych podzbiorów danego zbioru, generowanie rozwiązań spełniających określone ograniczenia itd.).

W artykule przedstawiono problemy wydajnościowe dotyczące systemu ReDaReS. Do systemu wprowadzono technologię *Jelly View*, która stanowi nowy, praktyczny sposób dekompozycji, przechowywania i wyszukiwania wiedzy w obrębie SZRBD. Technologia stawia czoło powyższym problemom przez wprowadzenie przetwarzania opartego na regułach (zamierzone przetwarzanie wiedzy) do systemów bazy danych. Wiedza intencjonalna jest przedstawiana w formie klauzul języka **Prolog**. Są one dekomponowane na dane i przechowywane w bazie danych, rozszerzając katalog systemu w kierunku przechowywania, wejścia i przetwarzania zamierzonej wiedzy w ramach SZRBD. W celu przetworzenia wiedzy intencjonalnej mechanizm wnioskowania jest połączony z SZBRD. Rezultaty procesu wnioskowania są widoczne jako zwykłe perspektywy dostępne za pośrednictwem języka SQL. Perspektywy są generowane w sposób dynamiczny, na żądanie, przez mechanizm wnioskowania. Z punktu widzenia użytkownika możliwości przetwarzania stają się nieograniczone (mogą być tworzone bardzo złożone kwerendy) w środowisku języka SQL. W konsekwencji funkcjonalność SZRBD jest poszerzona o własności dedukcyjnych baz danych. Baza danych staje się kompletnym źródłem wiedzy (zarówno intencjonalnym, jak i ekstensjonalnym). Artykuł skupia się na przedstawieniu systemu ReDaReS – pierwowzoru wprowadzania proponowanej technologii *Jelly View* – i prezentuje wnioski z cyklu eksperymentów.