Jelly View – A Technology for Arbitrarily Advanced Queries within RDBMS

Igor Wojnicki
Department of Mathematics and Computer Science, University of Missouri – St.Louis
One University Boulevard
St.Louis, MO 63121, USA
wojnicki@agh.edu.pl

Antoni Ligeza
Institute of Automatics, AGH – University of Science and Technology
Al. Mickiewicza 30
30-059 Krakow, Poland
ligeza@agh.edu.pl

ABSTRACT
Data processing capabilities of Relational Database Management Systems are limited. In particular, the following two categories of problems are hard to solve: Traversal of Structurally Complex Data Structures (such as graphs, trees, terms, lists etc.) and Search for Admissible Solutions Under Specified Constraints (finding specific subsets of a given set, generation of structural solutions satisfying specific constraints etc.) This paper covers a part of the Jelly View technology, which provides a new, practical methodology for knowledge decomposition, storage, and retrieval within RDBMS. It also briefly presents a prototype system implementing the Jelly View technology called ReDaReS. The technology tackles the above problems by introducing rule-based processing (intensional knowledge processing) to the database systems. To express intensional knowledge the Prolog language syntax is used in the form of clauses. The clauses of Prolog code are decomposed and stored in the RDBMS founding reusable components of Logic Program. The database becomes a complete source of knowledge, both extensional and intensional. Furthermore, to process intensional knowledge, an inference engine is coupled with the database systems. To process intensional knowledge the Prolog language syntax is used in the form of clauses. The clauses of Prolog code are decomposed and stored in the RDBMS founding reusable components of Logic Program. The database becomes a complete source of knowledge, both extensional and intensional. Furthermore, to process intensional knowledge, an inference engine is coupled with the RDBMS. The results of the inference process are visible as regular views, accessible through SQL. The state of the view is generated dynamically, on-demand by the inference engine. From the end-user point of view the processing capability becomes unlimited (arbitrarily complex queries can be constructed), while the most external queries are expressed with standard SQL. The relational database is extended with intensional knowledge, and coupled with an inference engine, which provides functionality analogous to that of the Deductive Databases.

Categories and Subject Descriptors
D.2.8 [Programming Techniques]: Logic Programming;
H.2.1 [Database Management]: Logical Design—data models; I.2.3 [Artificial Intelligence]: Deduction and Theorem Proving—Deduction

General Terms
Design, Languages

Keywords
RDBMS, Prolog, Intensional Knowledge, Deductive Database, Recursive Queries

1. MOTIVATION FOR THE JELLY VIEW
One of the most significant limitations of contemporary Relational Database Management Systems consists of well-known restrictions of the SQL, which is used as primary and native query language. Especially lack of recursive processing makes it hard to solve any more complex problems, such as traversal of structurally complex data structures (e.g. graphs, trees, terms, lists etc.), search for admissible solutions under specified constraints (finding specific subsets of a given set, generation of structural solutions satisfying specific constraints etc.). Moreover, lack of more sophisticated, i.e. rule-based data processing makes the database useless in many contemporary applications, such as decision making and decision support.

The limitations presented above follow from one generic drawback of relational databases which is the difficulty handling recursive queries. Such a difficulty is one of the eight principal weaknesses of Relational Database Management Systems [1].

The most important generic problem regarding this difficulty is the one mathematically defined as finding transitive closure of a relation. Apart from some small-scale examples this operation is computationally infeasible due to its complexity. However certain specific classes of problems can be solved even for practically useful systems. Particularly, two classes of such problems are of interest here:

C1 Traversal of structurally complex data structures (such as graphs, trees, terms, lists etc.).

The recursive SQL queries are described by the SQL99 standard, however this feature is not widely implemented; an exception is DB2 by IBM.
C2 Search for Admissible Solutions under specified constraints (finding specific subsets of a given set, generation of structural solutions satisfying specific constraints etc.).

These two classes cover many different problems such as: scheduling, plan generation, searching for acceptable or optimal solutions, analysis of structures, decision support systems, constraint programming problems. All of these problems can be solved basing on data encoded in RDBMS5, with some support from advanced information processing tools, external to the RDBMS.

This paper covers a part of the Jelly View technology which provides a new, practical methodology for knowledge decomposition, storage, and retrieval within RDBMS. It also briefly presents a prototype system, which implements the Jelly View technology, called ReDaReS. The technology tackles the above problems by introducing rule-based processing (intensional knowledge processing) to the database systems.

To express intensional knowledge the PROLOG language syntax is used in the form of clauses4. PROLOG, being a superset of Datalog, has been proven to be a proper methodology to provide the Logical Data Model [4], in other words, to extend the functionality of RDBMS towards that of Deductive Databases [2, 5, 6].

The PROLOG clauses are decomposed and stored in the RDBMS founding reusable components of Logic Program (or just Program). The database becomes a complete source of knowledge, both extensional and intensional. Furthermore, to process intensional knowledge, an inference engine is coupled with the RDBMS. The results of the inference process are presented as regular views, accessible through SQL. The state of the view is generated dynamically, on-demand by the inference engine. From the end-user point of view the processing capability becomes unlimited (arbitrarily complex queries can be constructed), while the most external queries are expressed with standard SQL.

Both, the methodology of storing and processing intensional knowledge within RDBMS, and the technology which interprets intensional knowledge, are named Jelly View [8, 9, 10]. The proposed technology enables solving problems of the C1 or C2 class within RDBMS. These problems can be smoothly approached keeping SQL as outermost communication technology. Moreover, the methodology extends the system catalog towards storing, accessing, and processing intensional knowledge within RDBMS.

2. IMPLEMENTATION OVERVIEW: THE REDARES SYSTEM

ReDaReS is a prototype system implementing Jelly View technology using PROLOG as an internal knowledge processing language. It is designed to be a loosely coupled mid-

2Relational Database
3Relational Database Management System
4PROLOG has been selected as a core, primary language for implementing the technology since it is both conceptually simple and powerful; simultaneously, as a language based syntactically on First-Order Predicate Calculus it covers relational data representation in natural way, and moreover it provides tools for recursive programming. However, the technology can be extended over practically any declarative and perhaps even procedural language of choice.

6The loose coupling has some important performance issues, though.
5The loose coupling has some important performance issues, though.
6The ODBC server functionality has not been implemented yet. It is replaced with text oriented query/response interface.

Figure 1: ReDaReS and RDBMS Data Flow

dlware between client applications and the RDBMS, using ODBC as a communication protocol. In this way, the system may be applied to any database supporting ODBC, regardless of its physical organization, vendor, or SQL dialect it uses. No modification of the RDBMS code is required.

The system is designed to be connected via ODBC to the RDBMS. Furthermore, it becomes an ODBC data source which mimics the original database. Instead of querying the database one should query ReDaReS. The system is transparent, which means that all the queries which were being accepted by the the database, are still accepted while querying ReDaReS, since they are forwarded to the database directly. Any response from the database is forwarded back to the client application.

If the query concerns a Jelly View, the system takes the processing over. It gathers intensional knowledge (from the database), 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 for further processing. Finally the database sends the reply to ReDaReS which forwards it back to the client application. The data flow explaining how the technology works is presented in Fig. 1.

The inference engine is based on SWI PROLOG [7]. It has direct access to extensional knowledge gathered in the RDBMS. The current implementation downloads all extensional knowledge to the inference engine before the inference process takes place.

3. INTENSIONAL KNOWLEDGE DECOMPOSITION

The PROLOG language syntax is chosen to express intensional knowledge. A PROLOG program consists of simple and complex clauses. Simple clauses (facts) provide extensional knowledge, while complex clauses (Horn clauses) provide intensional one.

For the sake of the implementation only complex clauses
are used, since the main focus of the research is to introduce intensional knowledge to extend the RDBMS knowledge representation and processing capabilities (simple clauses may be decomposed as well, though). Simple clauses are stored directly in the database relations (one fact is one record). Moreover, the database is populated with intensional knowledge. Upon a proper coupling of the inference engine with the database, the latter’s functionality is extended towards that of Deductive Databases [2, 5, 6].

There are three components of the design which deal with knowledge access and decomposition. These are: the Program, Internal Matching and External Matching. The Program covers the decomposition of intensional knowledge, while the External Matching provides information about how to access intensional knowledge from SQL. The Internal Matching states what extensional knowledge is accessible by the inference engine. The UML class diagram visualizing dependencies among relations, which provide the above components, is given in Fig. 2.

The relations supporting meta-data for Jelly Views (presented in Fig. 2) are called ra-relations for simplicity. Complex clauses are decomposed into two relations (raclause and raclarg) as it is shown in Fig. 3. The example presents a clause sibling/2. It states that two persons are siblings if they have a common parent (parent/2) and they are different persons.

Predicate names are placed in the raclause relation along with information about relationships among them. The raoperator denotes the operator which is placed after each predicate. If corder or precondition are NULL, it means that the predicate is the conclusion of the clause. Otherwise, precondition denotes id of the conclusion which the predicate preconditions; corder denotes the order of the preconditions.

The decomposition takes place at the predicate level, not the term level, so arguments of the predicates are not further decomposed, they are treated as atomic values. In the example (see Fig. 3) the arguments are variables, but if there were some more complex expressions (terms, structures etc.) they will be decomposed in the same manner. Lack of further decomposition is forced because of the performance reasons. If the arguments, being terms, structures, or lists, were further decomposed, a recursive query would have to be issued in order to recreate a clause. Since most of the RDBMS do not support such queries, there would be a performance loss.

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 particular Jelly View is composed of, and how the inference engine may access 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, in terms of the Relational Model. It also indicates what is the goal for the inference engine. 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 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.

The goal and the view name are stored in the rablepred relation as predicate and tab attributes respectively. Meta-data about attributes of the view is stored in the ratparg relation (attribute names and positions). There is also a many-to-many relationship between the External Matching and Program (established by the ratpclc relation) to indicate which clauses should be used to generate the particular Jelly View. It is worth pointing out, that some clauses may

7ru comes from the name of the prototype system: ReDaReS which may be spelled Red Ares as well, and it has nothing to do with the Egyptian God Ra.

8DB2 by IBM is an exception

9This issue is subject to the further research, especially if the inference engine is tightly integrated with the RDBMS.
be shared among some number of Jelly Views, which enables modular programming and allows to reuse the code (clauses).

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 by establishing a relationship between a predicate and a relation. In such a way, simple clauses of the given predicates are provided by the tuples from the given relations. The rapredtable relation indicates the predicate name (rapredicate.predicate), its arity (rapredicate.arity), and the name of the relation the simple clauses of the predicate will be generated from (rapredicate.tab). Since it is possible that the same mapping between the predicate and the relation is used by more than a single Jelly View, there is a many-to-many relationship (ratppt) between the External Matching and Internal Matching. It allows to reuse the same mappings for different Jelly Views.

All these relations used by 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 PROLOG clauses, a PROLOG inference engine is used to interpret it.

4. APPLICATIONS AND MODULAR PROGRAMMING

One of the common applications, which cannot be provided by SQL, regards finding relationships in a tree structure [3] (class C1, see Sec. 1). Such a structure might be represented as a single relation: subject(Parent_id, Item_id, Name). Where Item_id is a primary key, Parent_id is a foreign key denoting spatial relationship between the given node and its parent node.

The PROLOG code for finding relationships among tree nodes is given below:

```
-- RedaReS data
-- Ext.matching
INSERT INTO ratablepred VALUES (1, 'find', 'find');
INSERT INTO ratparg VALUES(1, 'parent_id', 'integer', 0, 1);
INSERT INTO ratparg VALUES(2, 'child_id', 'integer', 1, 1);
-- Int.matching
INSERT INTO rapredtable VALUES (1, 'tree', 'subject', 3);
-- Logic Program
-- 1st complex clause
INSERT INTO raclause VALUES (1, 'find', NULL, 1, NULL);
INSERT INTO raclarg VALUES (1, 'Parent', 1, 1);
INSERT INTO raclarg VALUES (2, 'Child', 2, 1);
-- Internal Matching
INSERT INTO raclarg VALUES(3, 'Parent', 1, 2);
INSERT INTO raclarg VALUES(4, 'Child', 2, 2);
INSERT INTO raclarg VALUES(5, '-', 3, 2);
-- 2nd complex clause
INSERT INTO raclarg VALUES(6, 'Parent', 1, 3);
INSERT INTO raclarg VALUES(7, 'Child', 2, 3);
INSERT INTO raclarg VALUES(8, 'Parent', 1, 4);
INSERT INTO raclarg VALUES(9, 'Child', 2, 4);
INSERT INTO raclarg VALUES(10, '-', 3, 4);
-- Supporting
INSERT INTO ratpt VALUES (1, 1);
INSERT INTO ratpcl VALUES (1, 1, 10);
INSERT INTO ratpcl VALUES (1, 1, 20);
```

Figure 4: Browsing a tree structure, decomposed clauses into the ra-relations.

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

To make a Jelly View, which is capable of finding the relationships, the above program is decomposed, and appropriate values are stored in the relations raclause and raoperator (see Sec. 3). 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 of the tree/3 predicate are taken from the subject relation. The detailed SQL code, which updates the state of the ra-relations, is given in Fig 4.

This particular find(parent_id, child_id) Jelly View is capable of searching for relationships in the relation subject. However, another Jelly View may be easily created for browsing a tree stored in a different relation, which uses the same Program (decomposed clauses). The difference will be with the Internal Matching (ratpt and rapredtable relations) which will state what relation should correspond to tree/3 predicate, and with the External Matching defining a new Jelly View. This introduces modular programming. Already decomposed, modularized clauses may be reused for creating other Jelly Views. In this manner, even complicated PROLOG programs may be provided as modules, and used even
by programmers not familiar with Prolog or the Logical Data Model. Specific libraries of modules can be defined and provided for typical classes of problems (the tree traversal, graph traversal, rule-based processing, forward chaining).

5. SUMMARY AND FURTHER RESEARCH

The Jelly View methodology extends the Relational Database concept, in such a way, that even more complex problems, than these specified in Section 1, can be smoothly approached keeping SQL as the outermost communication technology. Some real-life examples include: flight or trip planning (in general TSP), budget planning, finding alternative routes (admissible solutions), phone billing systems, tax calculations (rule-based processing).

Functionality of Relational Databases is significantly extended towards solving recursive queries and searching for admissible solutions by integrating the proposed technology into a database. The technology is based on coupling of the Prolog inference engine with the database.

The methodology has the following advantages. Both data (extensional knowledge) and knowledge (intensional knowledge) are stored within the Relational Database; no additional knowledge base is necessary. The intensional knowledge is represented as modules specified in Prolog for extending data processing capabilities – these modules are stored as data in the database. Such a decomposition provides easy alteration of the modules and enables their reuse.

The prototype system has been implemented and tested [8]; it possesses the following key properties. The original functionality of the database is preserved. The database is coupled with the standard Prolog inference engine. 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.

The proposed solution is more flexible than PSMs11. It allows modular programming and may be supported with a CAD system easily. Moreover, it is designed to work with any database system regardless of its PSM capabilities. There is also an ongoing research in this domain, which is focused on further decomposition of intensional knowledge.

6. REFERENCES


11PSM – Permanent Stored Module, also known as Stored SQL Function/Procedure