WDS'11 Proceedings of Contributed Papers, Part I, 131–136, 2011. ISBN 978-80-7378-184-2 © MATFYZPRESS

Rules in Database Systems

J. Kozák

Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic.

Abstract. This paper deals with the principles of rule-based systems in databases and gives an overview of existing approaches. First, we introduce the concept of rules. The application of rule-based systems to databases is categorized. Deductive database systems are introduced as the first extension of classical database management systems. We briefly describe the concept of deductive databases and their language. The main part is dedicated to active database systems. Language and implementation of rules in active databases are discussed as well as rule processing and key issues connected with it. Also architecture of the active database systems is depicted. At the end, benchmark comparing different rule-based database technologies is presented.

Introduction

The very first thing coming into our minds might be a natural question: "What are the rules?" Generally, we can say that almost any commands, decision making or deductive reasoning can be understood as a set of rules. This brief description implies that we meet rules in real life all the time although usually we do not even realize it. The previous interpretation becomes obvious when we start to think about implementation of intelligence into a machine i.e., making machines think like people. If we want to do that, then we have to tell the machine what to do in various situations which means to teach it in the way of stating the rules.

These considerations lead to an idea of implementing such a rule system. In this article, we will continue in introduction of rule-based systems and rules in dabases specifically. Then we will describe the concepts of deductive databases and active databases. At the end, a benchmark comparing different rule-based technologies in databases will be presented.

Rule-based systems

The *rule-based systems* (RBSs) evolved from the work of researchers from artificial intelligence (AI). Already in 1985, [*Hayes-Roth*, 1985] stated that RBSs constitute the best currently available means for codifying the problem-solving know-how of human experts, because these experts often tend to express their problem-solving techniques in terms of a set of situation-action rules (i.e., in specific situation execute an action). According to [*Hayes-Roth*, 1985], RBSs can be used in very different areas such as quality assurance, process control, troubleshooting and so on. Situation-action behavior represents only one part of rules.

Referring to the informal description of rules at the beginning, we could separate another different type of rules which would represent deduction and inferring of something new from the known facts—deductive systems. The reason of separating these parts from each other will be seen in the following sections considering the extension of database technology.

Rules and databases

Databases were originally designed as repositories which should be able to store data for various purposes. Traditional database management systems (DBMSs) are passive in the sense that commands are executed by the database (e.g., query, update, delete) as and when requested by the user or application program [*Paton and Díaz*, 1999]. However, some situations do not have to match this pattern and need to be modeled in a different way. It should allow the DBMS to react automatically in such situations without the need of user access. This can be

solved by implementing a system called *active database system* which provides the ability of reactive behavior.

Large databases also offer extensive possibilities to deduce some new information from the existing facts. More or less, it can be done by stating a proper query in common query language (e.g., SQL). Unfortunately, not everything can be expressed using e.g., SQL. Therefore, new approaches were implemented under the name of *deductive database systems*.

The active and deductive database systems represent the extensions of database systems using rules. Next sections will deal with them in more detail and greater amount of space will be devoted to active databases.

Deductive database systems

Deductive database systems are DBMSs whose query language and (usually) storage structure are designed around a logical model of data. As relations are naturally thought of as the "value" of logical predicate, and relational languages such as SQL are syntactic sugarings of a limited form of logical expression, it is easy to see deductive database systems as an advanced form of relational systems [*Ramakrishnan and Ullman*, 1996]. The main goal of deductive systems is to provide the ability to express queries that form a superset of relational algebra. The main difference is represented in the possibility of formulation recursive queries.

Deductive database systems represent extensions of relational DBMSs. Their evolution was influenced by logical programming. Relational databases and logic programming have been found quite similar in their representation of data at the language level. They have also been found complementary in many aspects [*Liu*, 1999].

Deductive systems divide their data into two categories:

- *Data or facts* that are usually stored in relational DBMS and are represented by a predicate with constant arguments. For example, *parent(Frank,Anne)* represents that Frank is a parent of Anne. This category forms so called *extensional database*.

- *Rules* that are usually written in Prolog-style notation

$$p \text{ :- } q_1, q_2, \ldots, q_n$$

  This rule means: "If q_1 and q_2 and ... and q_n then p". Rules with terms being either constants or variables are often referred to as Datalog rules. Whole set of rules forms so called *intensional database*.

We can see that deductive languages are declarative and therefore allow the user to say what he or she wants but not how to do it. This is one of the greatest benefits of declarative languages. Standardization of syntax of the deductive languages was done in [RIF_1, 2010] and represents the recommendation of W3C.

The expressive power of deductive languages is generally greater than expressive power of relational algebra. But for nonrecursive range-restricted Datalog with negation can be proved that it is equivalent to relational algebra and domain-independent relational calculus (for details see e.g., [*Bry et al.*, 2007]).

Evaluation and optimization of rules in deductive databases usually use fixpoint techniques. These are extensively described in [*Bry et al.*, 2007] and go beyond the framework of this paper.

Many deductive languages and systems have been developed. Their historical overview and development, concrete implementations and their brief description of capabilities can be found in [*Ramakrishnan and Ullman*, 1996]. Some issues araising in deductive languages such as extensions of Datalog to support complex values and others are discussed in [*Liu*, 1999].

Management of larger sets of deductive rules becomes quite problematic but still it offers an interesting alternative to traditional ways of data analysis.

Active database systems

Active database systems are based on a different approach to rules in DBMSs than deductive database systems. They represent the active behavior originally implemented in AI and expert systems.

The rules in active databases are commonly made up from up to three parts: an event, a condition and an action. It perfectly fits the reactive behavior. When some event happens, the condition is evaluated and if it is true, the action is carried out. Such rules are known as *event-condition-action* or *ECA rules*. Thus, the semantics of active rules are procedural [*Ceri and Ramakrishnan*, 1996].

The active rules do not have to contain all three parts. The event or condition part can be omitted. Then we speak about *condition-action rules* (often referred as *production rules*) or *event-action rules*. Every type of active rule has its specific kind of use and production rules are quite similar to deductive rules.

In the article [*Paton and Díaz*, 1999] three categories of active database applications are distinguished:

- Database system extension—support for other parts of database such as integrity constraints and materialized views etc.
- Closed database application—rules directly support the semantics of the application e.g., repair actions in a modeling database
- Open database application—used in conjunction with monitoring devices, see e.g., [*Zoumboulakis et al.*, 2004].

All these types of applications have a general architecture of an active database system which comprises of two basic components—rule repository and rule engine. The rule engine can be further divided into more modules such as event detector, condition monitor, scheduler and query evaluator [*Paton and Díaz*, 1999].

Rule language and repository

A new language for rule formulation must be defined in order to allow users to operate the active database system. The idea is to keep it as close to the natural language as possible. This will make the system easy to use. On the other hand, there was no standard so each RBS developed its own language.

In the year 2005 Rule Interchange Format (RIF) Working Group of W3C was established. This group presented a document [RIF_2, 2010] which represents a recommendation of W3C for abstract syntax of production rules. They use "mathematical English" (a special form of English for communicating mathematical definitions, examples, etc.) as a presentation syntax and XML syntax as its concrete syntax. [RIF_2, 2010] describes the syntax which shares features with concrete production rule languages. Without any other comments, here is one example of production rule and its representation according to [RIF_2, 2010], which could be used for customer segmentation: A customer becomes a "Gold" customer when his cumulative purchases during the current year reach $5000.

```
Prefix(ex <http://example.com/2008/prd1#>)
(* ex:rule_1 *)
Forall ?customer ?purchasesYTD (
 If   And( ?customer#ex:Customer
           ?customer[ex:purchasesYTD->?purchasesYTD]
           External(pred:numeric-greater-than(?purchasesYTD 5000)) )
 Then Do( Modify(?customer[ex:status->"Gold"]) ) )
```

The language of rules is not the only issue when speaking about the rule repository. The rules in the repository can change quite frequently e.g., in order to meet the business needs and their management can become problematic. It follows that providing a powerful rule management environment should be important. Rule management is discussed e.g., in [*Viana et al.*, 2006] where a short overview about existing solutions is provided and graphical representation of rule anatomy (i.e., rule meta-model) is proposed.

Rule engine

Rule engine cooperates with the rule repository and the underlying database. This part of the whole system operates the main algorithms. The rule execution model can be divided into the sequence of tasks according to [*Paton and Díaz*, 1999]: signaling, triggering, evaluation, scheduling and execution. The schematic diagram can be seen in Figure 1.

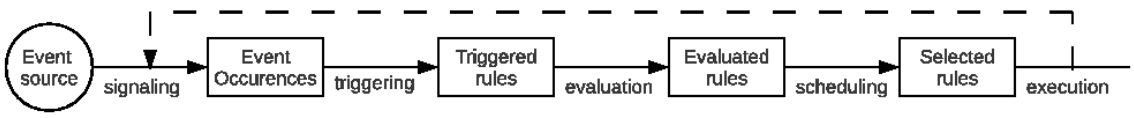

Figure 1. Rule execution diagram.

This schema implies possible parts of architecture of an active database system. First, an event must be detected. Then appropriate rules are selected and their conditions are evaluated. These processes select only applicable rules and a conflict set of rules is created. Conflicts are resolved and actions are executed.

Although the execution might look quite straightforward, there are many complicated issues hidden. At the very beginning the event monitor detecting complicated composite events can cause first problems (such as deciding for the right structure of composite event implementation) in an implementation of an active database system. But if we skip the event monitor part and concentrate rather on the production rules processing we get to the problem of condition evaluation at first.

Special algorithms for condition evaluation are often called *pattern matching algorithms*. They match facts against the rules. There is a number of pattern matching algorithms like TREAT, Rete etc [*Wang and Hanson*, 1992]. The Drools, production rule system developed by JBoss Community, uses an extended version of the Rete algorithm [*JBoss Community*, 2010].

Previous matching algorithm produces a conflict set of rules without specified order of execution. It means there has to be some kind of order of firing specified afterwards. This problem is often called a *conflict set resolution*. Some specific logic has to implemented to solve this problem dynamically or priority can be assigned to each rule in its definition. Dynamically prioritized conflict resolution is discussed in [*Dimitoglou and Rotenstreich*, 2007].

Benchmark

We presented two main types of rule systems which can be implemented in classical DBMS. But what kind of them performs better in real world situations? A unique benchmark was presented in [*Liang et al.*, 2009]. The authors compared several solutions from the area of rule based systems in databases. For our purposes the results of deductive and active databases are interesting. The authors chose DLV[1] and Ontobreaker[2] from the deductive databases and Drools[3] and Jess[4] from production and reactive rule systems.

[1] http://www.dbai.tuwien.ac.at/proj/dlv/

[2] http://www.ontoprise.de/en/products/ontobroker/

[3] http://www.jboss.org/drools

[4] http://www.jessrules.com

The benchmark was mainly designed to test the abilities which cannot be accomplished in tradional database languages such as relational algebra. The authors tested rule engines in large joins, datalog recursion and default negation. For illustration, we provide information about two tests performed in this benchmark.

One of the tests has a form of a non-recursive tree of binary joins which is expressed using the following inference rules written in Prolog syntax:

```
a(X,Y) :- b1(X,Z), b2(Z,Y)
b1(X,Y) :- c1(X,Z), c2(Z,Y)
b2(X,Y) :- c3(X,Z), c4(Z,Y)
c1(X,Y) :- d1(X,Z), d2(Z,Y)
```

The base relations, `c2`, `c3`, `c4`, `d1` and `d2` were randomly generated. Two data sets were used: one with 50000 facts and the other with 250000 facts.

Datalog recursion was tested e.g., on the same-generation problem i.e., finding all siblings in the same generation:

```
sg(X,Y) :- sib(X,Y)
sg(X,Y) :- par(X,X1), sg(X1,Y1), par(Y,Y1)
```

Tha base relations `par` and `sib` were randomly generated. Data were considered both cyclic and acyclic and for each type two data sizes were used: 6000 and 24000.

We can see results of these two chosen tests in the following tables which show the execution times. In these tables, size means the total size of all base relations; *error* means that the system gave an error during the evaluation; *timeout* indicates that evaluation did not finish within a set time limit of 30 minutes. All times are given in seconds and exclude the loading time.

Table 1. Large join test.

query	a(X,Y)		b1(X,Y)		b2(X,Y)	
size	50000	250000	50000	250000	50000	250000
ontobroker	4.089	28.385	0.213	4.806	0.019	0.168
dlv	85.459	838.781	7.177	60.239	0.820	9.392
drools	error	error	27.414	error	2.111	94.474
jess	310.000	timeout	12.000	317.000	1.000	26.000

Table 2. Same generation test.

size	6000	6000	24000	24000
cyclic data	no	yes	no	yes
ontobroker	1.402	1.926	5.424	5.309
dlv	20.274	31.346	365.136	438.008
drools	104.884	error	error	error
jess	64.000	error	1517.000	error

The results of two presented tests indicate that Ontobroker was the best performing system. Although Ontobroker was not the best system in all other tests, in the conclusion of the paper Ontobreaker (and partly DLV) were assigned to be the best performing systems. Results of other above enumerated systems were significantly worse.

Conclusion

We described the idea of rule-based systems and their implementation in databases. Two main branches, deductive database systems and active database systems, were introduced. Their main features and connected issues were discussed and results of a benchmark were presented. Of course, the description of RBSs was not exhaustive. The rules in databases is quite a vast area to be explored and there are many more topics to be discovered and problems to be solved.

For example, this paper did not deal with the static analysis of rules (termination problem etc.) at all.

Although the presented theory is interesting and forms the necessary background for all RBSs, the further research will rather concentrate on more recent problems of knowledge representation and reasoning such as ontological databases and reasoning on the Web which have become quite prominent now.

References

Bry F., Eisinger N., Eiter T., Furche T., Gottlob G., Ley C., Linse B, Pichler R., Wei F., Foundations of Rule-Based Query Answering, *Reasoning Web*, 1–153, 2007.

Ceri S., Ramakrishnan R., Rules in Database Systems, *ACM Computing Surveys*, 28, 109–111, 1996.

Dimitoglou G., Rotenstreich S., Architecture and Algorithms for Distributed Rule Management and Processing, *International Journal of Computer Science and Network Security*, 7, 397–404, 2007.

Hayes-Roth F., Rule-based Systems, *Communications of the ACM*, 28, 921–932, 1985.

JBoss Community, Drools Expert 5.1 Documentation, Available on *http://www.jboss.org/drools/documentation*, 2010.

Liang S., Fodor P., Wan H., Kifer M., OpenRuleBench: An Analysis of the Performance of Rule Engines, *Proceedings of the 18th international conference on World wide web*, 601–610, 2009.

Liu M., Deductive Database Languages: Problems and Solutions, *ACM Computing Surveys*, 31, 27–62, 1999.

Paton N. W., Díaz O., Active Database Systems, *ACM Computing Surveys*, 31, 63–103, 1999.

Ramakrishnan R., Ullman J. D., A Survey of Deductive Database Systems, *The Journal of Logic Programming*,23 ,125–149, 1995.

RIF_1 Working Group, RIF Basic Logic Dialect (BLD), Available on *http://www.w3.org/TR/rif-bld*, 2010.

RIF_2 Working Group, RIF Production Rule Dialect (PRD), Available on *http://www.w3.org/TR/rif-prd*, 2010.

Viana S., Pavón J., Rady de Almeida Junior J., Rule Management in Active Database Systems, *Proceedings of the 15th International Conference on Computing (CIC'06)*, 315–322, 2006.

Wang Y., Hanson, E. N., A performance comparison of the Rete and TREAT algorithms for testing database rule conditions, *Proceedings of the Eighth International Conference on Data Engineering*, 88–97, 1992.

Zoumboulakis M., Roussos G., Poulovassilis, Active Rules for Sensor Databases, *Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004)*, 98–103, 2004.