Proc. of First Intl. Workshop on Rules in Database Systems, Sep. 1993

Deductive and Active Databases: Two Paradigms or Ends of a Spectrum?

Jennifer Widom

IBM Almaden Research Center
650 Harry Road, K55/801
San Jose, CA 95120 USA

widom@almaden.ibm.com

Abstract

This position paper considers several existing relational database rule languages with a focus on exploring the fundamental differences between deductive and active databases. We find that deductive and active databases do not form two discernible classes, but rather they delineate two ends of a spectrum of database rule languages. We claim that this spectrum also corresponds to a notion of abstraction level, with deductive rule languages at a higher level and active rule languages at a lower level.

1 Introduction

Research on incorporating rule processing into database systems historically has been divided into two distinct areas: *deductive* databases and *active* databases. In deductive databases, logic programming style rules are used to provide a more powerful user interface than that provided by most database query languages [CGT90, Ull89]. In active databases, production style rules are used to provide automatic execution of database operations in response to certain events and/or conditions [DW92]. We claim that although some database rule languages are clearly classifiable as either deductive or active, there is no obvious delineation between the two types of languages.

We support our claim by considering six relational database rule languages prominent in the literature: *Datalog* [Ull89], *RDL* [KdMS90], *A-RDL* [SKdM92], *Ariel* [Han89, Han92], *Starburst* [WCL91, WF90], and *Postgres* [SJGP90]. We believe that these six rule languages form a spectrum along a "deductive to active axis", as shown in Figure 1. We also believe that the same spectrum describes the level of abstraction of these database rule languages, with deductive to active rule languages ranging accordingly from a higher level to a lower level.

In Section 2 we introduce a generic notation for database rules and a generic rule processing algorithm. In Sections 3–8 we expand these generic concepts to describe each of the six rule languages. We begin with Datalog, which we view as the "most deductive" language, and we end with Postgres, which we view as the "most active" language. It will be seen that the description of each language requires only small revisions to the description of the previous language, substantiating our claim that these languages form a spectrum rather than two distinct classes. Furthermore, as we move along the spectrum from deductive to active, we note that in general each rule language

DEDUCTIVE ... ... *ACTIVE*

higher level ... ... *lower level*

|———————————————————————————————|

Datalog ... RDL ... A-RDL ... Ariel ... Starburst ... Postgres

Figure 1: Spectrum of Database Rule Languages

includes more constructs than the previous language but also requires more behavior to be specified explicitly by the rule programmer. This leads to our claim that deductive rule languages provide a higher level of abstraction than active rule languages.[1] In Section 9 we draw conclusions and discuss implications of our claims.

It is important to note that in considering the six database rule languages we focus only on the "essence" of each language. That is, we consider features that have a direct bearing on whether the language should be considered as deductive or active, i.e. features such as basic language constructs and rule processing semantics, while we ignore features that have little or no bearing on the deductive versus active question, i.e. features such as syntax, minor semantic restrictions, assignment to variables, rule ordering, relationship to database transactions, etc. Note also that we sidestep the issue of whether these languages should be considered as *declarative* or *procedural.*[2]

2 Generic Framework

In all rule languages, each rule consists of at least some form of *antecedent* and some form of *consequent.* In addition, rules in the database context sometimes include an *activator.* (We specifically avoid terms such as *head*, *body*, *event*, *condition*, *action*, *trigger*, etc., since these often are associated with a particular database rule language paradigm.) Hence, we use the following generic notation for database rules, where square brackets indicate optionality:

$$[\ \textit{Activator} \leadsto\ ]\ \textit{Antecedent} \leadsto \textit{Consequent}$$

The generic interpretation of such a rule is: "[when the *Activator*] if the *Antecedent* then the *Consequent*", and a generic algorithm for processing a set of such rules is:

repeat until steps 1–2 can have no effect:

1. find a rule r [that is activated] whose antecedent is true
2. ensure r's consequent

[1] An analogy can be drawn with programming languages, where there are high-level languages with few constructs, such as Lisp, low-level languages with many constructs, such as assembly code, and "in-between" languages such as C.

[2] Actually, it is our belief that here too the languages form a spectrum—with "more declarative" towards the left of Figure 1 and "more procedural" towards the right—but the notions of declarative and procedural are fuzzy enough in this context that we prefer to avoid the issue.

We expand these generic concepts for our six database rule languages, considering the languages in left-to-right order along the spectrum of Figure 1.

3 Datalog

In Datalog, rules do not include activators. The antecedent of a Datalog rule corresponds to a query over the database that produces a set of tuples. The consequent of a Datalog rule states that the tuples produced by the antecedent are in a specified relation in the database. Thus, a Datalog rule has the form:

Query producing t's $\rightsquigarrow$ t's are in relation R

The interpretation of a set of Datalog rules is based on a *fixpoint* semantics [Ull89]; it can be described by the following variation on our generic rule processing algorithm:

repeat until steps 1–2 can have no effect:

1. find a rule r whose antecedent produces tuples
2. ensure that the tuples are present in r's consequent relation

(In the repeat clause, "no effect" here means that no rule produces tuples in step 1 or no executions of step 2 add tuples to the consequent relation.) In an actual implementation there are many alternatives and optimizations to this rule processing approach, but they do not affect the semantics of the language. Note that the relations in the consequents of Datalog rules often are not actually stored in the database (i.e. they are *intensional* or *virtual*), but this too is not an inherent property of the rule language.

4 RDL

RDL rules are identical to Datalog rules except that the consequent of an RDL rule is a sequence, where each element of the sequence specifies either that the tuples produced by the antecedent are in a specified relation or that the tuples produced by the antecedent are not in a specified relation. Thus, an RDL rule has the form:

Query producing t's $\rightsquigarrow$ t's are in relation R_1
t's are not in relation R_2
...
t's are in relation R_n

The processing of RDL rules is described using the following slight variation on the rule processing algorithm given for Datalog:

repeat until steps 1–2 can have no effect:

1. find a rule r whose antecedent produces tuples
2. for each element in r's consequent:
   ensure that the tuples are present or absent in the specified relation

Hence, we see that the only significant difference between RDL and Datalog is that RDL allows rule consequents to specify both the presence and absence of tuples.[3]

5 A-RDL

A-RDL is described as "an extension of RDL with active capabilities" (corroborating our claim that we are moving along the spectrum from deductive to active rule languages). The A-RDL language is identical to RDL except that queries in A-RDL rule antecedents may refer to *delta relations* in addition to normal database relations. There are three delta relations—**inserted**, **deleted**, and **updated**—associated with each database relation. Thus, an A-RDL rule has the form:

$$\begin{array}{lll} \mathit{Query}_\Delta \text{ } \mathit{producing}\ t\text{'s} & \rightsquigarrow & t\text{'s are in relation } R_1 \\ & & t\text{'s are not in relation } R_2 \\ & & \ldots \\ & & t\text{'s are in relation } R_n \end{array}$$

where $Query_\Delta$ denotes that the *Query* may refer to delta relations. The rule processing algorithm for A-RDL is identical to that for RDL, except that rule processing occurs with respect to a previous database state S as well as the current state (intuitively, S corresponds to the database state preceding some set of modifications):

$S \leftarrow$ some previous database state
repeat until steps 1–2 can have no effect:

1. find a rule r whose antecedent produces tuples according to S
2. for each element in r's consequent:
   ensure that the tuples are present or absent in the specified relation

In step 1 rule antecedents may refer to delta relations; for each database relation, the three delta relations associated with it contain the tuples that have been inserted, deleted, and updated between previous state S and the current state.

Hence, we see that the only significant difference between A-RDL and RDL is that A-RDL rules are considered with respect to a database state transition along with the current database state and they may refer to this state transition in their antecedent.

[3]This is very similar to suggested extensions of Datalog as described in, e.g., [AS91].

6 Ariel

In Ariel there are two forms of rules: rules without activators, called *pattern-based* rules, and rules with activators, called *event-based* rules. For convenience, we refer to these two forms of rules as *Ariel-p* and *Ariel-e* respectively. Along our deductive-to-active axis, Ariel-p lies to the deductive (left) side of Ariel-e, so we consider Ariel-p first.

6.1 Ariel-p

Syntactically, the antecedent of an Ariel-p rule is a query over the database, as in Datalog and RDL. However, the interpretation of an Ariel-p antecedent is that the query produces tuples by systematically replacing each normal database relation referenced in the query by a combination of delta relations **inserted** and **updated**.[4] Ariel-p includes a mechanism whereby this feature can be circumvented, i.e. entire relations are used. Therefore, the antecedents of Ariel-p rules are very similar to the antecedents of A-RDL rules, except Ariel-p rules cannot refer to delta relation **deleted**.

The consequent of an Ariel-p rule is a sequence of set-oriented relational database modification commands (**insert**, **delete**, and **update**). The tuples produced by the antecedent may be referenced in these commands along with normal database relations. Thus, an Ariel-p rule has the form:

$$Query_{\Delta} \textit{ producing } t\text{'s} \rightsquigarrow \textit{DB-Mod}_1(t\text{'s}) \ \cdots \ \textit{DB-Mod}_n(t\text{'s})$$

where *DB-Mod*(t's) denotes a database modification command that may reference the tuples produced by the antecedent.

The rule processing algorithm for Ariel-p is identical to that for A-RDL, except a different notion of previous state is used for evaluating delta relations, and more complex consequents are executed:

> $S \leftarrow$ some previous database state
> repeat until steps 1–2 can have no effect:
> 1. find a rule r whose antecedent produces tuples according to previous state S'
> 2. execute each command in r's consequent using the tuples from step 1

State S' in step 1 is S if it is the first time rule r is selected during rule processing; otherwise S' is the database state from the last time rule r was selected. (One additional slight difference in Ariel-p, as well as in the remaining languages discussed below, is that "no effect" in the repeat statement means only that no rule produces tuples in step 1, independent of step 2.)

[4]This interpretation ensures that the antecedent is satisfied if and only if it is satisfied by "new" data, similar to expert systems rule languages such as OPS5 [BFKM85].

Hence, we see that the most significant difference between Ariel-p and A-RDL is that the consequents in Ariel-p rules are more general. Less significant differences are that Ariel-p cannot reference **deleted**, and that the two languages use a different notion of previous state for delta relations.

6.2 Ariel-e

Rules in Ariel-e are identical to rules in Ariel-p except Ariel-e rules include activators. The activator in an Ariel-e rule is either $I(R)$, $D(R)$, or $U(R)$, corresponding to insertions, deletions, or updates on a relation R. Thus, an Ariel-e rule has the form:

$$I(R) \mid D(R) \mid U(R) \;\leadsto\; \textit{Query}_\Delta \textit{ producing } t\textit{'s} \;\leadsto\; \textit{DB-Mod}_1(t\text{'s}) \;\cdots\; \textit{DB-Mod}_n(t\text{'s})$$

As in Ariel-p, delta relations are referenced implicitly in rule antecedents. However, in Ariel-e only references to relation R (from the activator) implicitly reference delta relations, and they implicitly reference only the delta relation corresponding to the activator's operation. For example, in the antecedent and consequent of a rule with activator $D(R)$, references to R implicitly reference delta relation **deleted** for R.

The rule processing algorithm for Ariel-e is a slight variation on the rule processing algorithm for Ariel-p, taking activation into account:

> $S \leftarrow$ some previous database state
> repeat until steps 1–2 can have no effect:
> 1. find a rule r that is activated according to previous state S' and whose antecedent produces tuples according to S'
> 2. execute each command in r's consequent using the tuples from step 1

State S' is the same in Ariel-e as in Ariel-p. In step 1, a rule is activated if and only if the delta relation corresponding to its activator is non-empty.

Hence, we see that the most significant difference between Ariel-e and Ariel-p is that Ariel-e has the additional requirement of activation based on non-empty delta relations. A less significant difference is that Ariel-e can implicitly reference **deleted**, while Ariel-p cannot.

7 Starburst

Starburst rules are very similar to Ariel-e, although there are several minor differences:[5] A Starburst rule may specify the disjunction of multiple activators. A Starburst rule references delta relations corresponding to the rule's activator(s) explicitly rather than implicitly, and its consequent may not reference antecedent tuples but may reference delta relations. A Starburst rule consequent

[5] Some of these differences are particularly good examples of one language (Starburst) providing more constructs than the other (Ariel), but also requiring more behavior to be specified explicitly by the rule programmer.

may include database retrieval commands as well as database modification commands. Thus, a Starburst rule has the form:

$$[\,I(R)\,]\ \ [\,D(R)\,]\ \ [\,U(R)\,] \ \leadsto\ \mathit{Query}_\Delta\ \leadsto\ \mathit{DB\text{-}Command}^1_\Delta\ \ \cdots\ \ \mathit{DB\text{-}Command}^n_\Delta$$

where at least one activator is specified and $\mathit{DB\text{-}Command}_\Delta$ denotes a database retrieval or modification command that may reference delta relations. The rule processing algorithm for Starburst is a very slight variation on the rule processing algorithm for Ariel-e:

> $S \leftarrow$ some previous database state
> repeat until steps 1–2 can have no effect:
> 1. find a rule r that is activated according to previous state S' and whose antecedent produces tuples according to S'
> 2. execute each command in r's consequent according to previous state S'

State S' is the same in Starburst as in Ariel-e.

Hence, we see that there are no significant differences between Starburst and Ariel-e.

8 Postgres

Like Starburst and Ariel-e, Postgres rules have activators. The activator of a Postgres rule includes the same alternatives as Starburst and Ariel-e with the addition of $S(R)$, corresponding to data retrieval (or "Selection") on a relation R. Postgres rules do not include disjunction of activators as in Starburst. A Postgres antecedent is a query that may explicitly reference the delta relation corresponding to the rule's activator, and a Postgres consequent is a sequence of data modification and retrieval commands that also may explicitly reference this delta relation. A Postgres consequent may include "instead", indicating that the consequent commands should be executed from a previous database state rather than from the current state (see below). Thus, a Postgres rule has the form:

$$I(R)\mid D(R)\mid U(R)\mid S(R)\ \leadsto\ \mathit{Query}_\Delta\ \leadsto\ [\ \text{instead}\ ]\ \mathit{DB\text{-}Command}^1_\Delta\ \ \cdots\ \ \mathit{DB\text{-}Command}^n_\Delta$$

The rule processing algorithm for Postgres is a slight variation on the rule processing algorithm for Starburst:

> $S \leftarrow$ immediate previous database state
> repeat until steps 1–2 can have no effect:
> 1. find a rule r that is activated according to previous state S and whose antecedent produces tuples according to S
> 2. execute each command in r's consequent according to previous state S

Note that here S is defined specifically to be the "immediate previous state"—the state preceding the most recent (tuple-level) operation. One implication of this is that when commands are executed in step 2 the rule processing algorithm is called recursively rather than iteratively. We feel that the recursive versus iterative issue, although important, is orthogonal to the deductive versus active issue. Also note that in step 2, when a rule consequent includes "instead", the notion of executing commands according to previous state S also indicates executing the commands from state S.

Hence, we see that the only significant differences between Postgres and Starburst are that Postgres includes a data retrieval activator and that Postgres consequents may specify execution from a previous state.

9 Conclusions and Implications

In summary, using a generic framework we have described the essential features of six important database rule languages. Our descriptions began with Datalog, which is considered to be a typical deductive database rule language; by making only incremental changes in each step we proceeded to describe RDL, A-RDL, Ariel, Starburst, and finally Postgres, which is considered to be a typical active database rule language. The fact that we were able to traverse smoothly from Datalog to Postgres indicates that database rule languages form a spectrum rather than two distinct classes.

Through our rule language descriptions it can be observed that, in general, each language provides more features than the previous language. In particular, along the spectrum from deductive to active languages we see the addition of:

- A notion of database state transition, including the ability to reference state transitions in more and more complicated ways
- More and more powerful operations in rule consequents
- A notion of rule activator, with more and more complicated activators allowed

One feature that appears to be lost when moving from deductive to active rule languages is a notion of passing data from the antecedent to the consequent of rules. However, it can be seen that in the more active rule languages, although data from a rule's antecedent is not implicitly operated on by the rule's consequent, the same effect can be achieved by using the powerful rule consequents and delta relations provided by these languages. This is one (important) instance of our general observation that in the more deductive rule languages certain behavior is implicit while the same behavior must be specified explicitly in the more active rule languages, i.e. deductive rule languages provide a higher level of abstraction than active rule languages.

Given our claim that the more deductive rule languages (i.e. languages towards the left of the spectrum) are at a higher level than the more active rule languages (i.e. languages towards the right of the spectrum), it should be possible to translate rules in a language L into rules in a language L', where L' lies to the right of L on the spectrum. That is, it should be possible to use a lower-level rule language as the target language of a compiler for a higher-level rule language. By examining the descriptions in Sections 3–8 it is evident that, with the exception of a few fine points, such compilation is indeed possible. In fact, as a complete example of this, a compiler from Datalog to Starburst is described in [CW92].[6]

To conclude, in designing a database rule system we do not believe that a choice must be made between deductive and active paradigms, but rather a point must be chosen along a spectrum of database rule languages. The appropriate point depends on the desired level of abstraction and the desired suite of features provided by the rule language. We close by noting that we have considered relational database rule languages only; in an object-oriented setting the distinction between deductive and active database rule languages may be even fuzzier and more complex.

Acknowledgements

Thanks to Elena Baralis, Stefano Ceri, Bobbie Cochrane, and Eric Simon for helpful and varied reactions to an initial draft.

References

[AS91] S. Abiteboul and E. Simon. Fundamental properties of deterministic and nondeterministic extensions of datalog. *Theoretical Computer Science*, 78:137–158, 1991.

[BFKM85] L. Brownston, R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming.* Addison-Wesley, Reading, Massachusetts, 1985.

[CGT90] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer-Verlag, Berlin, 1990.

[CW92] S. Ceri and J. Widom. Deriving incremental production rules for deductive data. IBM Research Report RJ 9071, IBM Almaden Research Center, November 1992.

[DW92] U. Dayal and J. Widom. Active database systems. In *ACM SIGMOD International Conference on Management of Data* (tutorial), San Diego, California, June 1992.

[Han89] E.N. Hanson. An initial report on the design of Ariel: A DBMS with an integrated production rule system. *SIGMOD Record, Special Issue on Rule Management and Processing in Expert Database Systems*, 18(3):12–19, September 1989.

[Han92] E.N. Hanson. Rule condition testing and action execution in Ariel. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 49–58, San Diego, California, June 1992.

[KdMS90] J. Kiernan, C. de Maindreville, and E. Simon. Making deductive databases a practical technology: A step forward. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 237–246, Atlantic City, New Jersey, May 1990.

[SJGP90] M. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos. On rules, procedures, caching and views in data base systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 281–290, Atlantic City, New Jersey, May 1990.

[6] It is interesting to note that a "reverse" compilation (from an active to a deductive rule language) is suggested in [Tan91]; however, it is obvious that this translation is far less natural and requires significant capabilities in the deductive rule language.

[SKdM92] E. Simon, J. Kiernan, and C. de Maindreville. Implementing high level active rules on top of a relational DBMS. In *Proceedings of the Eighteenth International Conference on Very Large Data Bases*, pages 315–326, Vancouver, British Columbia, August 1992.

[Tan91] L. Tanca. (Re-)Action in deductive databases. In *Proceedings of the Second International Workshop on Intelligent and Cooperative Information Systems*, pages 55–61, Como, Italy, October 1991.

[Ull89] J.D. Ullman. *Principles of Database and Knowledge-Base Systems, Volumes I and II.* Computer Science Press, Rockville, Maryland, 1989.

[WCL91] J. Widom, R.J. Cochrane, and B.G. Lindsay. Implementing set-oriented production rules as an extension to Starburst. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, pages 275–285, Barcelona, Spain, September 1991.

[WF90] J. Widom and S.J. Finkelstein. Set-oriented production rules in relational database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 259–270, Atlantic City, New Jersey, May 1990.