EXTENDING INGRES WITH A RULES SYSTEM

by

M. Stonebraker, R. Johnson and S. Rosenberg

Memorandum No. UCB/ERL M81/93

18 December 1981

EXTENDING INGRES WITH A RULES SYSTEM

by

Michael Stonebraker,

Rowland Johnson,

and Steven Rosenberg

Memorandum No. UCB/ERL M81/93

18 December 1981

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

This research was supported by the Navy Electronics Systems Command through contract N00039-81-C-0569, and by the Applied Mathemtical Research Program of the Dept. of Energy through contract W-7405-EN6-48.

EXTENDING INGRES WITH A RULES SYSTEM

by

Michael Stonebraker
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
UNIVERSITY OF CALIFORNIA
BERKELEY, CA.

Rowland Johnson
COMPUTATIONS DEPARTMENT
LAWRENCE LIVERMORE LABORATORIES
LIVERMORE, CA.

Steven Rosenberg
COMPUTER SCIENCE AND APPLIED MATH DEPARTMENT
LAWRENCE BERKELEY LABORATORIES
BERKELEY, CA.

ABSTRACT

This paper presents the specification and proposed implementation of a rules system for a relational data base manager. The motivation for this proposal is the fact that integrity constraints, protection, triggers, alerters, and views are ALL examples of special purpose rules systems. We suggest that all five services can be obtained in one unified way through a single rules system.

I INTRODUCTION

This paper presents the specification of a rule based facility for INGRES [STON76, STON80]. Its basic motivation is the observation that integrity constraints [STON75, HAMM76], alerters [BUNN79], triggers [ESWA76], protection [GRIF76, STON74], and query modification to support views [CHAM75, STON75] are all special purpose rule systems. This paper specifies a mechanism by which these facilities can be obtained in one unified way. Moreover, many rules not possible in any of the existing mechanisms can also be formulated.

Rule systems have been known for a long time in the artificial intelligence (AI) world. Knowledge based systems such as MYCIN [SHOR76] and PROSPECTOR [DUDA78] are basically rule driven systems. Moreover, AI programming languages such as KRL [BOBR77] and FRL [ROBE77] have facilities of the flavor we propose. Our contribution is to apply them in a data base environment in a simple way which is easy to use and efficient to implement.

We begin with a specification of our paradigm and the language constructs necessary for it. Then in Section III we indicate several examples of our rules system which indicates its power and generality. In Section IV we suggest an implementation of our construct in a relational system. . Lastly, Section V closes with some conclusions.

II RAISIN

The language by which a data base administrator or user specifies rules is called RAISIN (Rules from AI Specified for INgres). Its basic structure is a sequence of ON-THEN clauses. That is,

ON (condition) THEN (action)
ON (condition) THEN (action)
:

For each ON-THEN clause, the condition will specify constraints to be met by an incoming data manipulation command before the action can be applied. Moreover, the condition can depend on data in the data base system. The action will be a set of operations to be performed on the command as well as other possibly new operations on the data base.

In this section we specify the allowable conditions and actions in RAISIN. We begin with the keywords in the language, which are necessarily somewhat specific to INGRES and QUEL. Other relational systems such as System R [ASTR76, BLAS79] would require a slightly different syntax.

KEY WORDS

TV: This stands for the collection of tuple variables in the incoming QUEL command. For example, if

EMP (name, salary, manager, age, dept)

is a relation and we specify the command:

```
RANGE OF E IS EMP
REPLACE E(salary = 1.1 * E.salary)
    WHERE E.name = "Stonebraker"
```

then TV will have the value "E." Languages which do not have an explicit notion of a tuple variable, for example, SEQUEL [CHAM74], would replace TV by the relation names that the command affects.

LHS: This stands for the collection of column names in the target list of the QUEL command which appear to the left of the equals sign. In this example there is only one, salary.

RHS(name): This stands for the computation to the right of the equals sign in the target list. The desired one has a left hand side with value "name." In this case RHS(salary) has the value "1.1 * E.salary."

RV: The tuple variable which specifies the relation affected by delete and replace operations. In this case it is E.

TR: The target relation indicated for append commands and optionally for retrieve commands. If we perform the command:

RETRIEVE INTO TEMP (E.name) WHERE E.age < 35

then TR has the value "TEMP".

QUAL: This stands for the qualification of the QUEL command, i.e. everything after the "WHERE." In the example directly above, QUAL has value "E.age < 35."

COMMAND: This stands for the particular command being executed. In the above example, it has value "RETRIEVE."

USER: This stands for the login name of the user who executed the command.

TIME: This stands for the value of the current time of day clock.

DAY: This stands for the current day of the week.

IN: A keyword for set inclusion. For example "salary IN LHS" will return true for the first example command but false for the second.

TV->: The -> following a tuple variable indicates that the value desired is the relation which the tuple variable ranges over. In the above examples "E->" and "TV->" both have value "EMP."

We can now specify what conditions are allowable in RAISIN. The condition term

is a (possibly parenthesized) boolean combination of clauses. Each clause is of the form:

Left-side REL-OPERATOR Right-side

REL-OPERATOR indicates the standard collection of comparison operators {<, <=, =, >, >=, !=}. In addition IN and -> from the key word list above are added operators. Moreover, Left-side and Right-side are either constants or keywords from the above list.

The following indicate acceptable conditions in RAISIN:

ON (TV -> EMP AND COMMAND = REPLACE AND LHS = salary)

ON (TIME <800 OR TIME >1700)

ON (USER = INGRES)

ON (E.salary > 800 IN QUAL)

In addition, RAISIN can have conditions which involve data in the data base. Hence, clauses can additionally be:

1) a QUEL RANGE statement

2) a QUEL Qualification

The following are are additional acceptable conditions in RAISIN:

ON (COMMAND = APPEND AND RANGE OF T IS TR
AND COUNT (T.tid) >500)

ON (COMMAND = REPLACE AND RANGE OF M IS EMP
AND M.salary >100 AND M.name = "Smith")

The first condition will apply to an APPEND command for a relation with more than 500 tuples while the second will apply for any REPLACE command but only if an employee named Smith exists who earns more than 100. Notice that TR stands for the target relation of an APPEND command. Hence, it cannot be the name of a valid relation in the current data base.

In each of the RAISIN examples above, the specified condition is either true or false. If true, the associated action is performed. We now turn to the legal actions which can appear in a RAISIN statement

ON (condition) THEN action

Here, "action" is a collection of actions from the following list. They are executed in order and there is no branching allowed.

1) QUEL command

Any QUEL command using tuple variables which have been previously defined is a legal action. Moreover, any keyword can appear in such a QUEL statement. For example, we could declare an action of:

```
RANGE OF N IS NEW-EMP
APPEND TO JUNK (TV.all) WHERE QUAL
```

2) SUB string-2 FOR string-1

This action will process the incoming QUEL command and substitute string-2 for each occurence of string-1. For example, we could change a replace action into a delete action by:

```
SUB DELETE FOR REPLACE
```

Of course, we would require additional modifications to make the resulting

DELETE statement syntactically legal.

More generally, we allow any keyword to be embedded in string-1 or string-2. In this case, the keyword takes on a value before the substitution is attempted. For example, we could change tuple variables by:

SUB E.salary FOR TV.salary

Of course, the INGRES default "all" is allowed and indicates that all column names are to be substituted. For example we could change the scope of one of the above commands from EMP to NEW-EMP as follows:

SUB N.all FOR TV.all

3) ABORT

This action cancels the command.

4) EXEC

This action executes the command. Ordinarily the command is the last action to be accomplished and is done by default. If a user wants the command done earlier, he can force it by an EXEC. Two EXEC's in a row would cause the command to be run twice.

5) UNDO

This action undoes all the changes that have been made to the database since the beginning of the rule. With the inclusion of this action there is the implicit assumption that transactions are supported.

6) MESSAGE {TO user-id} "message text"

This command returns a message to the person or program which issued the

command unless the optional clause is included which directs the message to another user. The MESSAGE action is useful when a command must be aborted and an error message returned.

7) ANDQUAL (text-string)

This action will perform query modification on the current command. Specifically it will replace QUAL by QUAL AND text-string. Hence, text-string is "ANDed" to the qualification. For example, we can restrict somebody to the subset of employees under 30 by:

```
ANDQUAL (E.age < 30 )
```

8) ILLEGAL "message text"

This action inspects the current command to see if it is syntactically valid. If not, it will perform an "ABORT" and generate a message. Consequently, it has the following effect:

```
IF (syntax error) THEN
   BEGIN  ABORT; MESSAGE "message text"; END
```

We now turn to illustrating this facility with several examples of commonly desired features.

III EXAMPLES OF RAISIN

We indicate the use of RAISIN to accomplish integrity constraints, protection statements, triggers, alerters, and view support in turn. Then, we conclude the section with some other miscellaneous applications of our rules system which seem useful.

3.1 Integrity Constraints

If employees must make more than 1000, then the following integrity constraint in INGRES expresses this desire.

```
RANGE OF E IS EMP
INTEGRITY E.salary > 1000
```

In RAISIN this rule can be expressed as:

```
ON (TV -> EMP AND salary IN LHS AND
    (COMMAND = REPLACE)
    or COMMAND = APPEND)

THEN  ANDQUAL (RHS(salary) > 1000)
```

A more sophisticated example is the constraint that Smith must make more than 2000. This is expressed in INGRES by

```
RANGE OF E IS EMP
INTEGRITY E.salary >2000 or E.name != "Smith"
```

In RAISIN this rule can be expressed as follows:

```
ON ((TV-> EMP and salary IN LHS AND
COMMAND = REPLACE) or COMMAND = APPEND) AND
name = "Smith" IN QUAL)

THEN ANDQUAL (RHS(salary) >2000)
```

Finally, consider the case where the average salary must be less than 1800. The RAISIN rule for this is:

```
ON(TV-> EMP AND
```

```
((salary IN LHS AND COMMAND = REPLACE) OR
 COMMAND = APPEND OR
 COMMAND = DELETE)

THEN EXEC

ON (RANGE OF E IS EMP AND
   AVG(E.salary) < 1800)

THEN UNDO
    MESSAGE "command not done because it would
          raise average salary above 1800"
    ABORT
```

3.2 Protection

Suppose Jones is only allowed to update salaries of employees for whom he is the manager and only between 8 A.M. and 5 P.M. This can be expressed in INGRES as:

```
RANGE OF E IS EMP
PERMIT REPLACE of E(salary) to Jones
FROM 800 to 1700  WHERE E.manager = "Jones"
```

This can also be specified in RAISIN as:

```
ON (TV -> EMP AND salary IN LHS AND COMMAND = REPLACE AND
   USER = Jones AND TIME > 800 AND TIME < 1700)

THEN   ANDQUAL (TV.manager = "Jones")
```

3.3 Triggers

Whenever one appends a new tuple to the EMP relation, one might wish to trigger an auxiliary update to the NEWEMP relation. This could be accomplished as follows:

```
ON (TR = EMP and COMMAND = APPEND)

THEN EXEC
    SUB NEWEMP FOR TR
```

A second example would be to construct a trigger which would automatically keep a count of the number of employees in each department. In the case that the application designer knows that employees are added one at a time, the following rule will keep a correct total in the field COUNT in the DEPT relation.

```
ON (TR = EMP and COMMAND = APPEND)

THEN range of D is DEPT
    Replace D(COUNT = D.COUNT + 1) WHERE
        D.dname = RHS(dept)
```

3.4 Alerters

Suppose one wanted a message printed on a user's terminal if he performed a salary update for any employee. The required RAISIN code is the following:

```
ON (TV -> EMP and salary IN LHS)

THEN MESSAGE "alarm, you are updating salaries"
```

As a second example, suppose one wants a message printed out if the average salary rises above 2000. This alarm could be specified in RAISIN as:

```
ON (TV -> EMP AND
```

```
((salary IN LHS AND COMMAND = REPLACE) OR
 COMMAND = APPEND OR
 COMMAND = DELETE))

THEN EXEC

ON (RANGE OF E IS EMP AND
   AVG (E.salary) > 2000)

THEN  MESSAGE TO accounting "alarm, salaries too high"
```

3.5 Views

We will do three view examples in this section to illustrate the power of RAISIN. First we will explore a view which is a restriction of a single relation.

The specifications of YOUNGEMP in QUEL are:

```
RANGE OF E IS EMP
DEFINE VIEW YOUNGEMP (E.all) WHERE E.age < 32
```

This same view can be indicated in RAISIN as:

```
ON (TV -> YOUNGEMP)

THEN  RANGE OF E IS EMP
      SUB E.all FOR TV.all
      ANDQUAL (E.age < 32)
      ILLEGAL "bad view mapping for YOUNGEMP"
```

This collection of action statements specifies the normal INGRES query modification procedure. Now, if we have a second relation:

DEPT (dname, floor)

then we can define a second view as follows:

```
RANGE OF D IS DEPT
DEFINE VIEW EMP-FLOOR (E.all, D.floor)
        WHERE E.dept = D.dname
```

This view is the natural join of EMP and DEPT. The normal query modification [STON75] facility to support this view is expressed in RAISIN as follows:

```
ON ( TV -> DEPT-FLOOR)

THEN  RANGE OF E IS EMP
    RANGE OF D IS DEPT
    SUB E.all FOR TV.all
    SUB D.floor FOR E.floor
    ANDQUAL (E.dept = D.dname)
    ILLEGAL "Your command on EMP-FLOOR
            could not be mapped"
```

There are several classes of updates that cannot be translated unambiguously to the underlying relations. INGRES currently gives up on these and issues an error message. One such command is the following:

```
RANGE OF F IS EMP-FLOOR
REPLACE F (floor = 6, dept = "toy") WHERE F.name = "Mike"
```

This command cannot be mapped to EMP and DEPT unambiguously unless Mike is the only employee in the toy department. The problem is that we will have to move the toy dept and will, as a result, move all other employees in the toy department. This was not asked for by the user. Let us suppose that we wish

Mike moved to the toy department and in addition the toy department moved to the 6th floor.as a result of the above command. This can be expressed in RAISIN as:

```
ON ( floor IN LHS AND dept IN LHS AND
  COMMAND = REPLACE AND TV -> EMP-FLOOR)

THEN  RANGE OF E IS EMP
     RANGE OF D IS DEPT
     SUB E.all FOR TV.all
     REPLACE D (floor = RHS(floor) WHERE QUAL AND
      E.dept = D.dname AND D.floor = RHS(floor)
     REPLACE E (dept = RHS(dept)) WHERE QUAL
        AND E.dept = D.dname
     ABORT
```

Notice that fairly general semantics can be specified by a data base administrator for ambiguous views.

3.6 Other Applications

It is easy to log the text for each command which is submitted to the data manager by the following rule:

```
ON ( )

THEN RANGE OF L IS LOG
   APPEND TO LOG (text =
       concat(COMMAND,RV, LHS, RHS, QUAL))
```

This will come very close to putting the command in proper syntactic order into

the LOG relation.

It is also possible to accumulate statistics about data base activity by application of a rule. For example:

```
ON (COMMAND = replace AND salary IN RHS AND TV->EMP)

THEN RANGE OF S IS STATISTICS
    REPLACE S(salcount = S.salcount + 1)
        WHERE S.name = "EMP"
```

IV IMPLEMENTATION CONSIDERATIONS

The action statements can be stored in parsed form in a system relation since they never need to be used for searching. This situation is analogous to current INGRES specifications for views, protection and integrity constraints which are stored in this fashion. If the parse tree exceeds the length of the longest character string (currently 255 bytes), then INGRES must cut the tree into 255 bytes pieces and store each with a sequence number. If INGRES had a mechanism for storing arbitrary length character string fields, it would make storing such parse trees much easier than currently. This appears to be a suitable use of data base experts [STON80a].

The ON condition will need to be stored in encoded form for efficient access. The structure we expect to use is:

```
CREATE RULES-REL(
  relation = C11,   /*relation pointed to*/
  column qual = i2, /*bit vector for which columns
              appear in the ON condition*/
  command = i1,     /*bit vector for which commands
              appear in the ON condition
  time = i1,        /*flag indicating whether time
              appears in the ON condition*/
  user = i1,        /*flag indicating whether user
              is restricted by the ON condition*/
```

```
  rule-id = i2,     /*id for the rule*/
    )

CREATE RULE-TEXT(
  rule-id = i2,     /*join field to RULES-REL*/
  ON = C255,        /*parsed form of ON condition*/
  ACTION = C255,    /*parsed form of ACTION condition*/
  sequence = i1,    /*sequence number in case parsed form
                    exceeds 255 bytes*/
    )
```

It appears that this structure will be at least as efficient as the existing INGRES structure which stores views, integrity constraints and protection statements in three different relations. Here, we need only access one relation to find all rules which apply to any given command.

V CONCLUSIONS

It should be noted that RAISIN is an unappealing language as currently structured. Obviously, the current INGRES specification for views integrity controls and protection is more user friendly than the corresponding RAISIN statement. As such RAISIN is only appropriate for a data base administrator or sophisticated programmer. However, we expect to write front end "sanitizers" for the rules interface. For example, it it relatively straight forward to preserve the existing user interface for the specification of views protection and integrity control. A simple program could generate the necessary RAISIN statements. More complex rules could always be expressed in RAISIN providing increased functionality to the sophisticated data base administrator if needed.

Also, in the current INGRES implementation there are three separate modules to handle integrity constraints, views and protection. These rule systems are different enough that the modules share virtually no code. Under a RAISIN implementation there would be one module for rules. It is likely that a general RAISIN implementation would be no more complex than the current INGRES query modification procedures.

As a result we expect to provide increased functionality in the form of a more powerful rules system without the necessity of any additional software. In fact, RAISIN might take less code than the current implementation of three special rules systems. Lastly, such an implementation might well be more efficient than currently.

REFERENCES

[ASTR76] Astrahan, M. M. et. al., "System R: A Relational Approach to Database Management," TODS 2, 2, June 1976.

[BLAS79] Blasgen, M., et. al., "System R: An Architectural Update," IBM Research, San Jose, Ca., RJ 3091, September 1979.

[BOBR77] Bobrow, D. and Winograd, T., "An Overview of KRL, a Knowledge Representation Language," Cognitive Science, 1,1 1977

[BUNE79] Bunemann, O. and Clemons, E., "Efficiently Monitoring Relational Databases," TODS, Sept. 1979.

[CHAM74] Chamberlin, D. and Boyce, R., "SEQUEL: A Structured English Query Language," Proc. 1974 ACM-SIGMOD Conference on Management of Data, Ann Arbor, Mich., May 1974.

[CHAM75] Chamberlin, D., et. al., "Views, Authorization and Locking in a Relational Data Base System," Proc. 1975 National Computer Conference, Anaheim, Ca., May 1975.

[DUDA78] Duda, R. et. al., "Development of the Prospector Consultation System for Mineral Exploration," SRI International, October 1978.

[ESWA76] Eswaren, K., "Specifications, Implementations and Interactions of a Trigger Subsystem in an Integrated Database System," IBM Research, RJ 1820, San Jose, Ca., August 1976.

[GRIF76] Griffiths, P. and Wade, B., "An Authorization Mechanism for a Relational Data Base System," TODS, 2, 3, September 1976.

[HAMM76] Hammer, M. and McLeod, D., "A Framework for Data Base Semantic Integrity," Proc. 2nd. International Conference on Software Engineering, San Francisco, Ca., October 1976.

[ROBE77] Roberts, R. and Goldstein, I., "The FRL Manual," MIT, AI Laboratory, Memo No. 409, Sept 1977.

[SHOR76] Shortliffe, E., "Computer Based Medical Consultations: MYCIN," Elsevier, New York, 1976.

[STON74] Stonebraker, M. and Wong, E., "Access Control in a Relational Data Base System by Query Modification," Proc. 1974 ACM Annual Conference, San Diego, Ca., November 1974.

[STON75] Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification," Proc. 1975 ACM-SIGMOD Conference on Management of Data, San Jose, Ca., June 1975.

[STON76] Stonebraker, M. et. al., "The Design and Implementation of INGRES," TODS 2, 3, September 1976.

[STON80] Stonebraker, M., Retrospection on a Data Base System," TODS, September, 1980.

[STON80a] Stonebraker, M. and Keller, K., "Embedding Experts and Hypothetical Data Bases in A Relational data Base System," Proc. 1980 ACM-SIGMOD Conference on management of Data, Santa Monica, Ca., May 1980.