A RULES SYSTEM FOR A RELATIONAL DATA BASE MANAGEMENT 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.

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

Rule systems have been extensively investigated in non data base contexts. For example, MYCIN [SHOR76] and PROSPECTOR [DUDA78] are basically rule driven systems. Programming languages such as KRL [BOBR77] and FRL [ROBE77] support rules in a fundamental way.

Many services of a data base management system (DBMS) can be interpreted as rules systems. For example, integrity constraints [STON75, HAMM76] specify conditions which must be guaranteed by a data manager. One such constraint for the relation

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

is that employee salaries be greater than 1000. It can be

Research Sponsored by Naval Electronics Systems Command Contract N00039-76-C-0022.

expressed in the current INGRES DBMS [STON76, STON80] as:

```
    range of E is EMP
    define integrity E.salary > 1000
```

This condition is automatically enforced by modifying each incoming salary update to one which is guaranteed not to violate the constraint. For example, the command

```
    range of E is EMP
    replace E(salary = .8 * E.salary) where E.name = "Smith"
```

is changed to

```
    range of E is EMP
    replace E(salary = .8 * E.salary) where
    E.name = "Smith" and .8 * E.salary > 1000.
```

The last clause ensures that Smith's updated salary cannot violate the constraint.

This modification procedure is triggered by an incoming command and performs a collection of actions which alter the command. Hence, it is of the form

```
    On condition
    Then action
```

As such, it is a special purpose rules system. In addition, alerters [BUNN79], triggers [ESWA76], protection services [GRIF76, STON74], and support systems for views [CHAM75, STON75] follow the same paradigm. Consequently, they are also rules systems.

Many DBMS implement such data base services individually. For example, INGRES implements integrity control, protection and views with three independent modules; each of which is a special purpose rules system. The purpose of this paper is to propose a single rules system which can provide all such data base services. In this way only one mechanism need be implemented, and an economy of data base code may result. Moreover, many rules not possible with existing DBMS services can also be formulated.

We begin with a specification of our rule paradigm in Section II. Then in Section III several examples of our rules system are presented which indicate its power and generality. Lastly, in Section IV we suggest an implementation of our constructs in a relational system.

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. The general form of a condition is the following:

```
ON          command(s)
TO          relations(s)
AFFECTING   field(s)
QUALIFYING  field(s)
BY          user-name(s)
DURING      time-range
FOR         day-range
WHERE       qualification
```

Hence a condition is a collection of terms, each of which is a keyword followed by a parameter. We give a few examples of conditions then explain the general syntax.

```
ON         replace
TO         EMP

ON         replace
TO         EMP
AFFECTING  salary
WHERE      EMP.name = "Smith"


ON         append, replace
TO         *
BY         Jones
DURING     8:00-17:00
FOR        mon-fri
```

The first condition applies to all replace operations to the EMP relation while the second applies to a salary update for an employee named Smith. Lastly, the third condition applies to all data base modifications made by Jones during normal working hours.

It should be noted that all terms in a condition except the first are optional and the wild card "*" is a valid parameter standing for "always". The TO clause specifies a list of relations in the current data base to which this rule applies while the AFFECTING term indicates what fields must be updated for the condition to apply. Moreover, the QUALIFYING clause indicates what fields must be present in the qualification of a user command for the condition to apply. For example, the command which gave a 20 percent salary decrease to Smith uses name in the qualification. A rule which included the term

```
QUALIFYING  name
```

would apply to this update.

The day-range, time-range and user-list constructs are self-explanatory. Lastly, the WHERE clause qualifies data to which the rule applies. Hence, it should be a valid qualification in a data manipulation language. In this exposition, we assume that qualification is a QUEL WHERE clause modified in one important way. In QUEL, all field names must have an attached range variable. Hence, E is declared to range over EMP in the above QUEL example and fields are designated by E.name and E.salary. In RAISIN qualifications we assume that a relation name is prepended to a field name instead of a range variable. Hence, EMP.name and EMP.salary would be valid field names.

For any incoming data manipulation command, the first condition of any rule is either true or false. If false, the rule does not apply. However, if true the action part of the rule is executed and the remainder of the ON-THEN statements (if any) are checked for applicability. We now turn to the legal actions which can appear in a RAISIN statement

The action portion of an ON-THEN statement is an ordered collection of commands from the following list.

1) EXECUTE

The user command is performed automatically as the last action of a rule. If a user wants the command done earlier, he must use an EXECUTE statement. Two EXECUTE statements in a row would cause the user command to be run twice.

2) CANCEL

This action cancels the execution of the user's command.

3) UNDO

This action undoes all changes to the data base since the beginning of the rule. With the inclusion of this action there is the implicit assumption that transactions are supported.

4) CHANGE relation-1 TO relation-2,..,relation-N

This action will change the scope of the user command from relation-1 to relation-2,.., relation-N. More precisely, whenever one has

```
    range of var-1 is relation-1
```

this is changed to

```
    range of var-2 is relation-2
      .
      .
      .
    range of var-N is relation-N
```

Var-2, .., var-N are internally assigned by a RAISIN implementation. Moreover, for any given field name, F, in relation-1, it is assumed that only one relation, say relation-j, has a field of the same name. Hence,

```
    var-1.F
```

is changed to

```
    var-j.F
```

For example, one can deflect all operations on the EMP relation to the NEW-EMP relation by the following rule.

```
     ON      *
     TO      EMP
     THEN
     CHANGE  EMP to NEW-EMP
```

5) RENAME field-1 TO field-2

This action causes all references to field-1 to be changed to field-2. If, for example, NEW-EMP has a salary field named dollars, the action statements of the above rule should be extended to the following:

```
    RENAME salary   TO dollars
    CHANGE EMP TO NEW-EMP
```

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

A message is returned to the person who issued the command that activated the rule. If the optional clause TO user-name is included, the message is directed to another user. The MESSAGE action is useful when a command must be aborted and an error message returned.

7) ILLEGAL "message text"

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

```
     ON       syntax error
     THEN
     CANCEL
     MESSAGE "message text"
```

8) QUEL command

Any QUEL command is a legal action. For example, suppose RULES is a relation with two fields, a rule number and a count field indicating how many times any given rule has been executed. The action statement needed to correctly update this relation for rule number 16 follows.

```
    range of R is RULES
    replace R (count = R.count + 1) where R.number = 16
```

Unfortunately, this action statement must be repeated for each rule currently being enforced.

One extension is needed to QUEL commands in a RAISIN context. Portions of the user command which activated the rule can be substituted into a QUEL statement which is applied as an action. The following keywords indicate the needed portions.

qualification	-	a keyword for the qualification in the users command
command	-	a keyword for the whole user command
new.field-name	-	a keyword for the value being assigned to field-name by the user command.

These can appear where they are semantically valid in a QUEL command. For example, in the command which gave a 20 percent pay decrease to Smith, qualification has the value

```
    E.name = "Smith"
```

while new.salary has the value

```
     .8 * E.salary
```

9) ADDQUAL qualification

This action will perform query modification [STON75] on the current command. Specifically it will add the indicated qualification to the one specified by the user. This extra qualification follows the syntax of QUEL WHERE clauses except each field name has a relation name prepended instead of a range variable. Since the user's command will have a range variable in front of each field name, the qualification must be preprocessed to find each field name, remove the prepended relation name and substitute the user's range variable.

For example, we can restrict Jones to the subset of employees under 30 by the following rule:

```
    ON        *
    TO        EMP
    BY        Jones
    THEN
    ADDQUAL   EMP.age < 30
```

If Jones issues a query such as

```
    range of E is EMP
    retrieve (E.salary) where E.name = "Smith"
```

then it will be modified to

```
    retrieve (E.salary) where E.name = "Smith"
                                   and
                              E.age < 30
```

Notice that EMP.age is preprocessed to E.age before being added to the command. One other processing step must take place. The keywords noted in command (8) are also valid here, and the appropriate substitutions must take place.

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
define integrity E.salary > 1000
```

In RAISIN this rule can be expressed as:

```
ON      replace, append
TO      EMP
THEN
ADDQUAL new.salary > 1000
```

Note that new.salary refers to the value assigned to salary by the user command. 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
define integrity E.salary > 2000 or E.name != "Smith"
```

In RAISIN this rule can be expressed as follows:

```
ON         replace, append
TO         EMP
AFFECTING  salary
WHERE      EMP.name = "Smith"
THEN
ADDQUAL    new.salary > 2000
```

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

```
ON        replace, append, delete
TO        EMP
THEN

ON        *
WHERE     AVG(EMP.salary < 1800)
THEN
UNDO
MESSAGE "command not done because it would
         raise average salary above 1800"
CANCEL
```

A last example is to specify that employee salaries cannot decrease. This is not expressible by the integrity constraints of [STON75]. However, in RAISIN, the desired rule is

```
ON        replace
TO        EMP
AFFECTING salary
THEN
ADDQUAL   new.salary > EMP.salary
```

3.2 Protection

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

```
range of E is EMP
define permit replace of E(salary) to Jones
FROM 800 to 1700  WHERE E.manager = "Jones"
```

This can also be specified in RAISIN as:

```
ON          replace
TO          EMP
AFFECTING   salary
BY          Jones
DURING      8:00-17:00
THEN
ADDQUAL     EMP.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 NEW-EMP relation. This could be accomplished as follows:

```
ON          append
TO          EMP
THEN
EXECUTE
CHANGE      EMP TO NEW-EMP
```

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 count in the DEPT-COUNT relation.

```
ON          append
TO          EMP
THEN
range of D is DEPT-COUNT
replace D( count = count + 1) where
            D.name = new.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         replace
TO         EMP
AFFECTING salary
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         append, delete, replace
TO         EMP
AFFECTING salary
THEN

ON         *
```

```
    WHERE      AVG(EMP.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 specification of YOUNGEMP in QUEL is:

```
    range of E is EMP
    define view YOUNGEMP (E.all) WHERE E.age < 32
```

This same view can be indicated in RAISIN as:

```
    ON         *
    TO         YOUNGEMP
    THEN
    CHANGE     YOUNGEMP TO EMP
    ADDQUAL    EMP.age < 32
```

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         *
    TO         EMP-FLOOR
    THEN
    range of E is EMP
    range of D is DEPT
    CHANGE EMP-FLOOR TO EMP, DEPT
    ADDQUAL    EMP.dept = DEPT.dname
    ILLEGAL     "Your command on EMP-FLOOR
                 could not be mapped"
```

There are several classes of updates that cannot be translated unambiguously to underlying relations. INGRES currently issues an error message for such commands. For example,

```
    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. If we wish Mike moved to the toy department and in addition the toy department moved to the 6th floor, we can express this desire in RAISIN as follows:

```
    ON         replace
    TO         EMP-FLOOR
    AFFECTING  dept, floor
    THEN
    range of E is EMP
    range of D is DEPT
    CHANGE EMP-FLOOR TO EMP, DEPT
    REPLACE D (floor = new.floor) WHERE qualification AND
            E.dept = D.dname AND D.floor = new.floor
    REPLACE E (dept = new.dept) WHERE qualification
               AND E.dept = D.dname
    CANCEL
```

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 each command which is submitted to the data manager by the following rule:

```
    ON       *
    THEN
    range of L is LOG
    append to LOG (text = command)
```

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

```
    ON         replace
    TO         EMP
    AFFECTING salary
    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. Moreover, any subsequent ON-THEN clauses in the current rule can be added to the end of the action statement for the first clause. This situation is analogous to current INGRES specifications for views, protection and integrity constraints which are stored in this fashion. If the parsed representation exceeds the length of the longest character string (currently 255 bytes), then INGRES must cut it into 255 bytes pieces and store each with a sequence number. A mechanism for storing arbitrary length character string fields would facilitate storing such data. 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*/
    command = i1,      /*bit vector for which commands
                        appear in the ON condition
    affecting = i2,    /*bit vector for which columns
                        appear in the AFFECTING condition*/
    qualifying = i2,  /*bit vector for which columns appear
                       in the QUALIFYING condition*/
    during = i1,      /*flag indicating whether there
                       is a DURING clause */
    by = i1,          /*flag indicating whether there
                       is a BY clause */
    for = i1,         /*flag indicating whether there
                       is an ON clause*/
    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*/
        )
```

The WHERE term in a condition could be used as input to a theorem prover which could check if the intersection of the term with the users qualification was empty. If so, the rule does not apply. Otherwise, the WHERE term must be translated to an action which OR's the user command with:

NOT qualification

It appears that this proposed structure will be at least as efficient as the existing INGRES implementation 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 not an appealing language. Obviously, the current INGRES specification for views integrity controls and protection seems generally more user friendly than the corresponding RAISIN statements. It is straightforward to build a small language processor which accepts current INGRES specifications and translates them into RAISIN internal form. In addition, special translators might be useful for alerters and triggers. Only a user who wished to perform complex view resolution would use RAISIN directly. Moreover, it is hoped that a more user friendly specification of RAISIN can be designed in the future.

In the current INGRES implementation there are three separate modules to handle integrity constraints, views and protection. These rule systems 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 that RAISIN could provide increased functionality in the form of a more powerful rules system with a comparable amount of software. Lastly, such an implementation might well be more efficient than the current one.

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.