An Evaluation of a Rule-Based Language for Classification Queries

Dennis P. Groth

Indiana University School of Informatics
Bloomington, IN 47408, USA.
Email: dgroth@indiana.edu

Abstract

This paper provides results from a usability experiment comparing two different database query languages. The research focuses on a specific type of query task, namely classification queries. Classification is the process of assigning input data to discrete classes according to application specific criteria. While SQL can be used to perform classification tasks, we seek to discover whether a different type of query language offers any advantages over SQL. We present a rule-based language, which organizes the queries in a logical way. The rule based language is specifically designed to support classification tasks. The usability experiment measures the effectiveness, efficiency and satisfaction of novice and expert users performing a variety of classification tasks. The results show that while both approaches are usable for classification tasks, the rule-based approach was preferred by expert users.

1 Introduction

Classification is the process of assigning input data to discrete classes according to application specific criteria. Simple examples abound for this type of task. For example, employees in an employee database may be classifed according to their salary into "High", "Medium" and "Low" classes.

In this research we consider classifications that are definable by the user in two ways. First, a classification is *definable* if the class values are explicitly assigned. The definition can be stated in the form of a series of *If .. Then* statements. For example, consider the following sentences that define the employee salary classification:

1. If an employee's salary is less than \$30,000 then assign the employee to the "low" salary class.
2. If an employee's salary is between \$30,000 and \$60,000 then assign the employee to the "medium" salary class.
3. If an employee's salary is more than \$60,000 then assign the employee to the "high" salary class.

A classification is *derivable* if the class values are assigned according to a calculation. For example, we can assign employees to classes using the statement $SalaryClass = floor(Salary/1000)$, which effectively "bins" employees into salary ranges.

This research considers schemes that may be either definable or derivable to the extent that a user can describe the classification in some declarable form. We do not consider algorithmic techniques encountered in data mining, such as automatic cluster detection or discretization, which are described in [5, 4].

In this research we present a declarative language that supports both definable and derivable classifications. We call the language a Mapping Query Language (MQL), since the user declares the mappings between the input data and the classifications. The language has a syntax that succinctly describes the classifications according to a set of rules. A BNF grammar is provided as an appendix to this paper. We compare our approach with an Structured Query Language (SQL) approach in a controlled usability experiment.

The usability experiment seeks to determine whether one approach is superior to another in a controlled fashion. Previous experiments have compared equivalent systems. That is, each system was fully equivalent in expressible power. differing in syntactic form, or in interactiion style. This research differs from previous approaches in that the language we describe is strictly weaker than SQL. Never the less, by focusing on a specific work task (classification) the systems are comparable.

This paper is structured as follows. Section 1.1 provides pointers to the relevant literature most closely related to this work. Section 2 provides a brief description of the rule-based language. In Section 3 we describe the setup for our experiment. Section 4 provides the results from the usability experiments. Lastly, in Section 5 we provide directions for future research activities and summarize our findings.

1.1 Related Work

In this section we describe the relevant work that is related to this research. First, from a query language perspective, our approach is based on Datalog. Descriptions of the theoretical aspects of Datalog can be found in database texts, such as [1]. A comprehensive description of logic-based query languages is described by Vianu in [10]. There are numerous references for SQL, including general texts [1, 7] as well as language standards [3]. From a data mining perspective, SQL has been used in support of various techniques. With regards to classification tasks, very little work has focused on relational query languages, with the exception being [6].

There have been numerous examples in the literature related to usability of query languages. The approaches tend to follow a fairly standard experimental design, in which two query languages are compared. Early work by Reisner [9] forms the basis for many other experiments. Surveys that provide a compilation of multiple experiments were provided by Reisner [8] and Welty [11]. A study by Yen and Scamell [12] compared SQL to Query By Example (QBE) serves as the basis of our experiment design. More recently, comparisons of diagramatic languages to SQL have been presented, including work by Catarci, et. al. [2].

2 The Rule-Based Language

Conceptually, we view the process of classifying data as shown in Figure 1. The class values that are assigned can be expressed with database queries. We will continue with the employee salary example for the purposes of describing our approach.

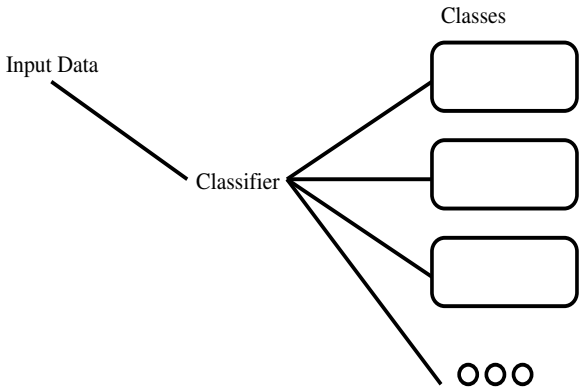

Fig. 1. The classification process

We assume a basic familiarity with relational database processing. Data is stored in tables that are described by their schema, which is a finite set of attribute names. Given a table $\mathbf{r}$ over attributes $R = \{A, B, C, \ldots\}$, let $t \in \mathbf{r}$ be a tuple. A query q is a mapping from input schema to an output schema.

Our rule-based language specifies queries in the form of a finite list of rules. Each rule is of the form $Head \leftarrow Body$, with the following semantics. The body of a rule is a boolean expression involving attribute names and constants, as well as a small number of supported functions. For example, basic mathematical functions such as addition and subtraction are supported. The head of a rule is an expression like the body, except that it is not restricted to a $\{0, 1\}$ result. Each rule head in a query must of the same data type. We refer to the set of attributes used in the rules as the schema of the query.

Let $P = \{p_1, \ldots, p_k\}$ be a rule query and $\mathbf{r}$ be a table. For each $t \in \mathbf{r}$, let $Head(t)$ be the value of the expression $Head$ when given t as input. $Body(t)$ is defined in the same manner. The output schema of P is $R \cup Class$, where R is the schema of the input table and $Class$ is the class value assigned to each tuple.[1] The following rule query defines the employee salary classification.

$'Low' \leftarrow Salary < 30000$
$'Medium' \leftarrow Salary >= 30000$ and $Salary <= 60000$
$'High' \leftarrow Salary > 60000$

Rules are evaluated in the order that they are defined. In addition, we support two different interpretation strategies. Queries may be interpreted *functionally* or *relationally*. Under a functional interpretation, rules are evaluated until a success is encountered. In contrast, a relational interpretation will evaluate every rule, which provides for a single input tuple to be mapped to multiple classes.

We support two special types of rules to provide greater control over the process. A rule of the form $Except \leftarrow Body$ excludes input tuples from further processing. A rule of the form $Head \leftarrow Else$ allows for a tuple to be mapped if all of the preceding rules failed. A BNF syntax description of the language is provided as an appendix to this paper.

By design, the rule-based query language is restricted in terms of the types of queries that can be expressed. While it is similar to Datalog in terms of its syntax, we do not support recursive queries or negation in the head of the rule. With these restrictions, the queries can be executed in linear time by processing one tuple at a time.

2.1 Equivalence to SQL

Each rule query is equivalent to an SQL query involving set operations. The proof of this equivalence is based on a transformation from the rule-based language to SQL. Rather than providing the proof we provide a sketch of the transformation process here. First, note that the body of a rule is essentially the same as the SQL where clause. The head of a rule is the same as the SQL select clause. The following SQL query is equivalent to the rule query that classifies employee salaries.

```
Select "Low", * from employee where salary < 30000
  Union
Select "Medium", * from employee where salary >= 30000 and salary <=60000
  Union
Select "High", * from employee where salary > 60000
```

Special care must be taken to support the *Except* and *Else* rules by creating more complicated where clauses in the SQL queries. From a processing perspective, it is likely that SQL queries used for classification tasks will be less efficient than the rule-based process. Note that the SQL queries may require multiple passes through the input data. In addition, using SQL in support of classification tasks may require complex queries to be written, which may be beyond the end-user's ability.

3 Experiment Design

In order to understand the usefulness of our approach we have performed a controlled usability experiment. This experiment seeks to quantify a user's ability to solve classification tasks with either SQL or the rule-based language. The focus is targeted only at classification tasks, and is not a broad comparison of fully equivalent systems. SQL was chosen as the most appropriate comparative tool due to its predominant use by both expert and novice users.

[1] The implementation provides the user with a mechanism for defining a unique name for $Class$.

3.1 Independent Variables

The independent variables used to control the experiment were:

1. User skill level (Low, Medium-Low, Medium-High, High)
2. Query language (SQL, Rule-Based Language)
3. Query complexity (Less complex, More complex)

Subjects for the experiment were recruited from a one-semester course in database systems as well as professional programmers. Table 1 describes how user skill levels were assigned as well as the number of subjects in each skill level.

User Description	Skill Level	Number
Undergraduate Student	Low	27
Graduate Student	Medium-Low	28
Professional experience < 3 years	Medium-High	6
Professional experience >= 3 years	High	4

Table 1. The base of experimental subjects

Half of the subjects performed classification tasks using either SQL or the rule language. The use of SQL and the rule language was balanced within the groups. The same problems were attempted by each user. Query complexity was based on the type of classification being performed. The complexity of the task was calculated based on a model answer for the problem as formulated in the rule-based language. A complexity score for a rule is defined according to the combined complexity of the head and the body, each of which is given a score of 1 (simple) or 2 (complex). An expression is simple if it involves only constants or simple boolean comparisons, otherwise it is complex. The total complexity score for a task is given by summing the score for each rule. The score for our running example would be 6 - each rule involves only simple expressions and there are 3 rules.

Tasks below the mean score for all tasks are considered less complex, while tasks that score above the mean are considered more complex. It is important to note that the queries required to solve most of the tasks in this experiment would be classified as complex in the previously reported experiments due to their use of set operations.

Environment and Evaluation While we have a fully functioning implementation of the rule-based language, we decided to administer the experiment by using a paper and pencil exam. This technique has been frequently employed in previous experiments, and benefits from being efficiently administered to multiple subjects simultaneously. Prior to participating in the exam, each subject filled out a short demographic questionaire, which identified their skill level.

The exam was comprised of twelve classification problems against three different datasets. The datasets were described in terms of the input schema of the table. The problems were worded in the form of english sentences describing the desired classification.

Each subject was provided a training manual for the tool they were to use to solve the classification tasks. Each user had some experience with writing SQL queries, so the SQL training material focused on the writing of queries invlolving set operations. The rule-based language training materials were slightly longer due to syntax differences. Both training manuals contained an identical set of example classification tasks as well as solutions. After reviewing the training materials the users were asked to solve the twelve classification problems.

The professional developers (expert users) were asked to complete both versions of the test. The order of exposure was controlled, ensuring that half of the expert subjects first used SQL and then used the rule-based language. The other half reversed this order of exposure. After completing both tests, the expert users completed a satisfaction survey comparing the two approaches.

3.2 Subject Group Comparison

Because of the design of the experiment, there are several groups whose characteristics need to be considered. Table 2 provides a summary of the information provided by the subjects that reported GPA's. A test for homogeneity of variances showed that the subject groups were comparable ($p < .10$).

Subject Description	Average GPA (SD)	Average Experience (SD)
Undergraduate Student	3.35 (0.47)	1.18 (0.62)
Graduate Student	3.64 (0.31)	1.00 (0.85)

Table 2. The average GPA and number of database courses for the student subjects.

The professional programmers were not asked to provide GPA's. Instead, they were asked to report their work experience (in years) with databases. As a group, the professional programmers averaged 4.2 years of experience.

3.3 Dependent Variables

The dependent variables we measured with this experiment were: 1. User Accuracy, and 2. User Satisfaction.

Accuracy is a quantitave measurement of the user's ability to solve classification problems with a specific tool. Satisfaction is a qualitative measurement of the user's feelings towards using a specific tool to solve classification problems. We did not measure the time required to solve each problem for two reasons. First, measuring time would have required the use of either an SQL interface as well as the implemented rule-based system. The queries are more efficiently processed using the rule-based system, which we believe would unduly influence the satisfaction measurement. Second, interacting with either system introduces possible side-effects to the accuracy measurements. For example, subjects may fail to get the syntax of the query exactly correct and become frustrated with either system leading to fewer problems attempted.

We determined the accuracy of a user's solution using a technique similar to [12]. Each problem was assigned the lowest of the possible scores shown in Table 3.

Score	Description
3	Correct solution
2	Essentially correct solution (typographical errors)
1	Partially correct solution (missing conditions)
0	Incorrect solution, or unsolved

Table 3. Possible scores assigned to each problem solution

The scoring method is intentionally coarse. Each subject's total score was computed by totaling their score for each problem and dividing by the maximum number of possible points.

User satisfaction was determined by using a qualitative assessment survey. Two surveys were completed by each student. The first survey was completed after reviewing the training material. The second survey was completed after completing the exam problems.

3.4 Hypotheses

Figure 2 provides the hypotheses we seek to test with this experiment. By convention, each hypothesis is stated in its negative form.

The expert users were the only subjects that evaluated both approaches. As a result, the between-groups comparisons are limited to the first approach evaluated by the expert users. Within-groups comparisons are based on the order of exposure for expert users.

H1: There will be no difference in accuracy based on tool selection.
H2: There will be no difference in accuracy based on user skill level.
H3: There will be no difference in accuracy based on task complexity level.
H4: There will be no interaction between user skill level and task complexity on accuracy.
H5: There will be no interaction between user skill level and tool selection on accuracy.
H6: There will be no interaction between task complexity level and tool selection on accuracy.
H7: There will be no difference in satisfaction based on tool selection.
H8: There will be no difference in satisfaction based on user skill level.
H9: There will be no difference in satisfaction based on task complexity level.
H10: There will be no interaction between user skill level and tool selection on satisfaction.
H11: There will be no difference on user satisfaction for expert users based on the order of exposure.

Fig. 2. Experimental hypotheses

4 Results

In this section we report the results of the experiment. In the following subsections we report: 1. Efficiency, 2. Accuracy, 3. Satisfaction, 4. Professional programmers.

The statistical test employed for this analysis was the standard T-Test. The significance level employed for all tests was 0.10. Note that the risk associated with making a type I error with either approach is small.

4.1 Efficiency

Often, usability experiments of this type will measure the efficiency with which users can solve problems. Efficiency may be measured in terms of time; however, for this experiment time was a constraint placed upon the user. Consequently, the time involved in solving problems would have a more significant effect on effectiveness measures, since the lack of time to solve all of the problems would preclude users from fully completing the problems. This was certainly the case with the student subjects, with only two subjects providing solutions for every problem.

A different measure of effectiveness which can be reported for this experiment is the number of steps involved in solving problems with each system. For example, a smaller number of steps with one system may indicate an improvement efficiency . The problems given to the users in this experiment involved similar steps with either system:

1. Identify the classes to be generated.
2. Define a condition for each class.
3. Define the output value to be generated for each condition.
4. Construct a simple query in SQL or the Rule language.
5. Put the queries or rules in the correct order.

Syntactically, the solutions had the same number of rules or queries. For this experiment, then, the systems are not separable. In the future, more elaborate experiments could be constructed to more closely measure the cognitive impact of these languages.

It is clear that the rule language is less verbose than SQL, which allows for a comparison on this basis. For example, the following rule program classifies patients according to their age:

$'Newborns' \leftarrow Age < 3$
$'Children' \leftarrow Age \geq 3$ AND $Age \leq 12$
$'Adolescents' \leftarrow Age \geq 13$ AND $Age \leq 18$
$'Adults' \leftarrow Age \geq 19$ AND $Age \leq 65$
$'Elderly' \leftarrow Age > 65$

The equivalent SQL query is much more verbose:

```
Select "Newborns", * from Patient where Age < 3
  Union
Select "Children", * from Patient where Age >= 3 and Age <= 12
  Union
Select "Adolescents", * from Patient where Age >= 13 and Age <= 18
  Union
Select "Adults", * from Patient where Age >= 19 and Age <= 65
  Union
Select "Elderly", * from Patient where Age > 65
```

A word count (each term) of the rule language yields 37 words, while the SQL query has 61 words. Using this measure, the SQL queries were all longer than the rule programs. The average word count for the rule programs was 29.2 words, while the average for the SQL queries was 44.8 words.

Even though the rule language has the propensity to be more efficient using this measure, it is difficult to draw any direct conclusions. It is more likely that the experiment has revealed this in an indirect way. For instance, it is plausible that the lower satisfaction scores for SQL are related to the length of the solutions.

4.2 Accuracy

Table 4 shows the accuracy scores for each subject group.

Subject	SQL	Mapping Language
Undergraduate Student	27 (17)	37 (10)
Graduate Student	26 (19)	24 (21)
Professional	97 (0.1)	96 (0.2)

Table 4. Mean accuracy scores for each group, as a percent of total. (standard deviation)

The low scores for the students is related to the limited time that was provided for the exam. The difference in performance between the undergraduate and graduate student subjects is interesting. Note that the performance of graduate and undergraduate students is indistinguishable when using SQL. However, when using the mapping language, the undergraduate students performed better.

The lack of a similar relationship between accuracy scores for graduate students is a result of 4 students that did not get any problem correct. When omitting these students the average accuracy score for graduate students increases to 34% (SD 17), which tracks more accurately with the undergraduate result. The best explanation we have is based on the mix of students in the graduate class, which has a higher number of international students. It is possible that a language barrier was the primary influencer of the lower scores for these students. Since we did not control for this variable, we retained these students scores, rather than removing them from our result.

Table 5 reports the statistical tests for Hypothesis 1 - 3. Each result was tested at a 0.10 significance level. When the hypothesis is rejected, we report the lowest significance level, even though our a priori test was at 0.10. For the professional programmers we report both parts of the experiment: Prof-1 refers to the first exam, Prof-2 refers to the second exam. For Hypothesis 3, we tested both SQL and the mapping language.

The undergraduate students accuracy scores were higher for the mapping language. However, the trend on accuracy as experience increases indicates that either tool can be used to solve such classification problems. This is a positive result for the mapping language, since the subjects had no prior knowledge of the language. Future work certainly needs to consider the effect of experience with the mapping language.

The results for Hypothesis 3 show that complexity of the solution does impact the accuracy. However, the actual result is counterintuitive - subjects were more accurate with complex solutions. This was especially true with SQL. In retrospect, controlling for complexity in the way we did is probably faulty. Note that the more complex solutions were shorter, which most likely skewed the results.

Hypothesis	t (critical value)	Result (p)
H1	-0.56 ($\pm$1.665)	Not Rejected ($p > .10$)
H2: Undergrad	-1.86 ($\pm$1.708)	Rejected ($p < .10$)
H2: Grad	0.29 ($\pm$1.706)	Not Rejected ($p > .10$)
H2: Prof - 1	0.44 ($\pm$1.86)	Not Rejected ($p > .10$)
H2: Prof - 2	0.26 ($\pm$1.86)	Not Rejected ($p > .10$)
H3: SQL	2.4 ($\pm$1.665)	Rejected ($p < .10$)
H3: MQL	1.82 ($\pm$1.665)	Rejected ($p < .10$)

Table 5. Statistical tests of the accuracy results (Hypothesis 1 - 3).

4.3 Satisfaction

Table 6 shows the results of the satisfaction surveys for the SQL group. Table 7 shows the results of the satisfaction surveys for the Mapping Language group.

Subject	Pre-Exam	Post-Exam
Undergraduate Student	1.82 (0.65)	2.45 (0.67)
Graduate Student	1.92 (0.80)	2.48 (0.81)
Professional	1.90 (0.78)	2.62 (0.56)

Table 6. Mean satisfaction scores (1=Best, ..., 5=Worst) for the SQL group. (standard deviation)

Subject	Pre-Exam	Post-Exam
Undergraduate Student	2.09 (0.78)	2.01 (0.92)
Graduate Student	1.83 (0.57)	2.42 (0.58)
Professional	2.10 (0.60)	2.03 (0.68)

Table 7. Mean satisfaction scores (1=Best, ..., 5=Worst) for the Mapping Language group. (standard deviation)

What is interesting about the satisfaction scores is the relationship between the pre-exam and post-exam scores. For the SQL group, the post-exam score is higher. Again, the undergraduate students are interesting, in that their post-exam score is actually lower than the pre-exam score. Omitting the same 4 students as we previously omitted still resulted in a higher post-exam satisfaction score for the graduate students, although not as high as the SQL, graduate student group.

We compared the satisfaction scores to a target satisfaction score of 2.0, which would indicate that the user subjectively believes that the tool is a "good" tool to use. The statistical test employed was a standard T-Test with degrees of freedom set to the size of the sample minus 1. As a group, the SQL group's post-exam satisfaction scores indicated that SQL was not as good as they initially believed. For the Mapping Language group, only the Graduate students exhibited the same behavior.

Table 8 reports the statistical results for Hypothesis 4 and 5. Again, we distinguish between the professional programmer's first and second exams.

Hypothesis 4 was rejected. Note that this result compares the mean satisfaction score to 2.0 for the post-exam satisfaction survey. This result is influenced by the decreased satisfaction in SQL. For Hypothesis 5 the results do not consider the tool. Rather, they simply indicate that either tool is subjectively considered to be a good tool to solve classification problems.

Hypothesis	t (critical value)	Result (p)
H4	1.69 ($\pm$1.665)	Rejected ($p < .10$)
H5: Undergrad	-1.45 ($\pm$1.708)	Not Rejected ($p > .10$)
H5: Grad	0.22 ($\pm$1.706)	Not Rejected ($p > .10$)
H5: Prof - 1	1.53 ($\pm$1.86)	Not Rejected ($p > .10$)
H5: Prof - 2	-0.86 ($\pm$1.86)	Not Rejected ($p > .10$)

Table 8. Statistical tests of the satisfaction results (Hypothesis 4 and 5).

4.4 Details of the Professional Programmer Experiment

In this subsection we report the results of statistical testing of Hypothesis 6 - 8. These hypotheses consider whether there is any difference in accuracy and satisfaction for the professional programmers when given both tools to solve problems. Table 9 reports the statistical results for these hypotheses.

Hypothesis	t (critical value)	Result (p)
H6	-0.01 ($\pm$1.86)	Not Rejected ($p > .10$)
H7	-1.68 ($\pm$1.86)	Not Rejected ($p > .10$)
H8	6.11 ($\pm$1.833)	Rejected ($p < .10$)

Table 9. Statistical tests of the satisfaction results (Hypothesis 4 and 5).

Professional programmers had no problem with solving the problems with either tool, so there is no surprise about the accuracy hypothesis (H6). The satisfaction hypothesis (H7) is close to rejection. Again, the trend indicates that the subjects were less satisfied with SQL.

The significant result from Hypothesis 8 is based on a preference survey administered after using both tools. The raw data was scored on a 1 to 5 scale:

1: Strong preference for SQL.
2: Preference for SQL.
3: No preference.
4: Preference for MQL.
5: Strong preference for MQL.

The professional programmers preferred the mapping language, with a mean preference score of 3.7 (sd 0.35). The mean score indicates a somewhat weak preference for the mapping language. However, for the type of problem (classifications) no subject had a preference for SQL. This is especially positive for MQL, since the subjects had no prior knowledge of the language.

5 Conclusion and Future Work

The usability experiment showed that the mapping language could be used by both experienced and lesser experienced users with about as much accuracy as SQL. This in itself is encouraging, since the mapping language was a new concept for the subjects.

The subjects were satisfied with both SQL and the mapping language. In general, the subjects were more satisfied with the mapping language.

We took advantage of the time the professional programmers made available for the study. Ideally, we would have liked many more professionals to participate in the study. The professionals had a strong preference for the mapping language for all types of classification tasks. At the same time, they tended to like SQL, since they had much more experience with it. The professionals were able to envision the benefits of the mapping language, which translated into the higher preference scores.

References

1. Abiteboul, S., Hull, R., and Vianu, V. *Foundations of Databases.* Addison-Wesley, 1995.
2. Catarci, T. What happened when database researchers met usability. *Information Systems 25*, 3 (2000), 177–212.
3. Date, C. J., and Darwen, H. *A Guide to the SQL Standard.* Addison-Wesley, Reading, Mass., 1993.
4. Han, J., and Kamber, M. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2001.
5. Hand, D., Mannila, H., and Smyth, P. *Principles of Data Mining.* The MIT Press, 2001.
6. Meier, A., Savary, C., Schindler, G., and Veryha, Y. Database schema with fuzzy classification and classification query language. In *Computational Intelligence: Methods and Applications* (2001).
7. Ramakrishnan, R. *Database Management Systems.* McGraw-Hill, 1998.
8. Reisner, P. Human factors studies of database query languages: A survey and assessment. *ACM Computing Surveys 13*, 1 (1981), 13–31.
9. Reisner, P., Boyce, R., and Chamberlin, D. Human factors evaluation of two database query languages - square and sequel. In *Proceedings of the National Computer Conference* (1975), pp. 447–452.
10. Vianu, V. Rule-based languages. *Annals of Mathematics and Artificial Intelligence 19* (1997), 215–259.
11. Welty, C., and Stemple, D. Human factors comparison of a procedural and a nonprocedural query language. *ACM Transactions on Database Systems 6*, 4 (1981), 626–649.
12. Yen, M., and Scamell, R. A human factors experimental comparison of SQL and QBE. *IEEE Transactions on Software Engineering 19*, 4 (1993), 390–402.

Appendix

<MAP_PROGRAM> ::= <RULE> [<Line Feed> <RULE>]*
<RULE> ::= <HEAD> "<-" <BODY>
<HEAD> ::= <EXPRESSION>
| "Except"
<BODY> ::= <BOOLEAN_EXPR>
| "Else"
| Null
<BOOLEAN_EXPR> ::= <CONDITIONAL_EXPR> [<BOOLEAN_OPERATOR> <CONDITIONAL_EXPR>]*
| ["NOT"] "(" <BOOLEAN_EXPR> [<BOOLEAN_OPERATOR> <BOOLEAN_EXPR>]* ")"
<BOOLEAN_OPERATOR> ::= "AND" | "OR"
<CONDITIONAL_EXPR> ::= <STATEMENT> [<RELATION> <STATEMENT>]
<STATEMENT> ::= ["NOT"] <EXPRESSION>
<RELATION> ::= "<" | "<=" | ">" | ">=" | "=" | "!="
<EXPRESSION> ::= <EXPRESSION> [<OPERATOR> <EXPRESSION>]*
| "(" <EXPRESSION> ")"
| <FUNCTION>
| <ELEMENT>
<OPERATOR> ::= "+" | "-" | "*" | "/" | "^" | "&" | "#" | "|"
<FUNCTION> ::= "ABS(" <EXPRESSION> ")"
| "FLOOR(" <EXPRESSION> ")"
| "CEIL(" <EXPRESSION> ")"
| "SQRT(" <EXPRESSION> ")"
| "GRAYCODE(" <EXPRESSION> ["," <EXPRESSION>]* ")"
| "LENGTH(" <EXPRESSION> ")"
| "SUBSTRING(" <EXPRESSION> "," <EXPRESSION> "," <EXPRESSION> ")"
<ELEMENT> ::= <VARIABLE>
| <CONSTANT>
| "(" <EXPRESSION> ")"
<VARIABLE> ::= $A_i \in R$
<CONSTANT> ::= number
| "string"