Inter-Database Data Quality Management: A Relational-Model Based Approach

by

Eva Y. Tsai

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1996

Signature of Author.....
Department of Electrical Engineering and Computer Science
May 20, 1996

Certified by.....
Richard Y. Wang
Associate Professor of Information Technologies and Co-Director for Total Data Quality Management (TDQM) Research Program
Thesis Supervisor

Accepted by.....
Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Inter-Database Data Quality Management
A Relational-Model Based Approach

by

Eva Y. Tsai

Submitted to the Department of Electrical Engineering and Computer Science on
May 13, 1995, in partial fulfillment of the
requirements for the degree of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Poor data quality can have a detrimental impact on the overall effectiveness of an organization. However, there exists no complete tool today to manage the quality of organizational data. This thesis presents an inter-database software tool as part of the concerted research effort of Total Data Quality Management (TDQM) project to ensure data consistency. It focuses primarily on referential integrity and in particular, *business rules* pertaining to organizational needs.

A set of inter-database rules is proposed based on generalizations derived from the referential integrity and normalization literature. The rules are implemented in the form of a database using Microsoft Access and Structured Query Language (SQL), presented with an appropriate graphical user interface. The results of executions are delivered in comprehensible reports tailored for users' specific needs. Without the enforcement of such rules, inconsistent data can corrupt organizational databases and impede subsequent operations.

The software tool developed in this thesis enables organizations to execute inter-database business rules. With additions of intra-database tool and other techniques, TDQM will evolve into a complete tool to manage organizational data and subsequently improve overall effectiveness.

Thesis Supervisor: Richard Wang
Title: Associate Professor of Information Technologies, Sloan School of Management

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Professor Richard Wang, for his time, advice, and encouragement to the project. I would also like to thank my group members, Rith Peou and Ed Hwang, for their help and support.

Many thanks to Theodore Tonchev, for his timely assistance to any computer problems and patience to decipher the help manual and error messages.

Thanks to John Fan for his continual interest in my work and comments on my drafts of the thesis. Also thanks to my hall members for their attempts to fill my days with joy, and to my friends for the adoring photographs to remind me of the spring in the midst of my work.

Finally, thanks to my family for their unfailing understanding and support.

TABLE OF CONTENTS

List of Figures

Chapter 1

Introduction

1.1 Information Overload

Recent years have witnessed the explosive advancement in computer technologies and applications in such areas as client/server computing and the Internet. As computers become a necessity, information induced and enabled by the technologies also start to pour in everyday life. For example, before the wide acceptance of the Internet, a person already had to tackle the incredible amount of advertisement, news, movies, and a myriad of information sources. With the advent of the Internet era, however, considerably more information can be predicted. There then emerges a critical need to utilize technology to help people and corporations alike digest the overwhelming amount of information. In particular, to ensure consistency and report violations, data quality management forms an essential research foundation necessary for any further information control.

1.2 A Mini-case Study

A case study of a major bank corporation, X, readily justifies the importance of data quality management. To handle the influx of information, Bank X resorts to manual data entry as a primary means to achieve timely and effective input of the required data into the requisite systems. The amount of the manpower is determined by dividing the product of task time and volume by available staff hours. However, as Bank X has numerous sites around the world within a multi-system environment, manual input process spread over these sites has caused a relatively high error ratio. This error ratio has led to a general belief among senior management that the integrity of data is poor and subsequent reports are unreliable as measures of performance.

After high error ratio was detected, Bank X has attempted to remedy the situation. For example, the Head Office has audited static data and identified existing inconsistencies in the selected data sample. The result suggests that the lack of a central unit exclusively in charge of the data control process has contributed to the proliferation of systems and databases under the command of individual managers. This uncoordinated proliferation of databases, in turn, is responsible for inconsistent data models and methodology for measuring and maintaining the integrity of the underlying static data.

The above case study demonstrates the importance of data quality management. As corporations and individuals strive to eliminate or rectify inappropriate data with the help of the advanced computing power, elaborate software programs should be designed to address such purpose.

1.3 Overview of the Document

This thesis presents an inter-database software tool as part of the concerted research effort of Total Data Quality Management project (TDQM) at MIT Sloan School of Management to ensure data consistency. It focuses on referential integrity and in particular, business rules pertaining to organizational needs. It will also serve as a technical reference manual illustrated by detailed explanation of architecture design and examples.

Chapter 2 discusses the purpose of the research with the emphasis on TDQM project and referential integrity. Chapter 3 details the architecture design and functional requirement of the inter-database software tool. Chapter 4 demonstrates the implementation of the tool. Chapter 5 illustrates the usage with an elaborate example. Finally, chapter 6 presents conclusion of this thesis.

Chapter 2

The TDQM Cycle and Referential Integrity

2.1 Overview of TDQM

The thesis develops an inter-database software tool as part of the concerted research effort of Total Data Quality Management (TDQM) project [17, 22, 23]. TDQM is designed to bridge the chasm between theoretical data quality research and practical needs for DQ improvement in organizations. Building upon existing research, TDQM introduces and develops a TDQM cycle consisting of following four components:

- Methods for defining DQ rules
- Methods for measuring DQ
- Methods for analyzing the impact of poor data quality
- Methods for improving data

As mentioned in *A Software Tool for Total Data Quality Management* written by Richard Wang, the TDQM cycle devised at MIT Sloan School of Management is a synthesis of concepts from TQM, data quality (DQ), and database literature [21]. In the TQM literature, the widely-practiced Deming cycle for quality enhancement encompasses four components: Plan, Do, Check, and Act (PDCA) [13, 14]. The fitness-for-use by customer is further an accredited criterion for continuous measurement, analysis, and improvement of product quality [10, 15, 16]. In the DQ literature, DQ is defined as a multi-dimensional concept that embraces dimensions

such as accuracy, completeness, consistency, and timeliness [1, 2, 9, 10, 20]. Finally, in the database literature, emphasis is placed on domain, entity, and referential integrity constraints [6, 11, 24].

In addition to different literature, TDQM cycle further devises business rules that capture organizational quality operations. These rules are employed through all phases of the TDQM cycle for defining, measuring, analyzing, and improving DQ from data customers' perspective. The TDQM cycle adapted from TQM, DQ, and database literature is presented in Figure 1 [18].

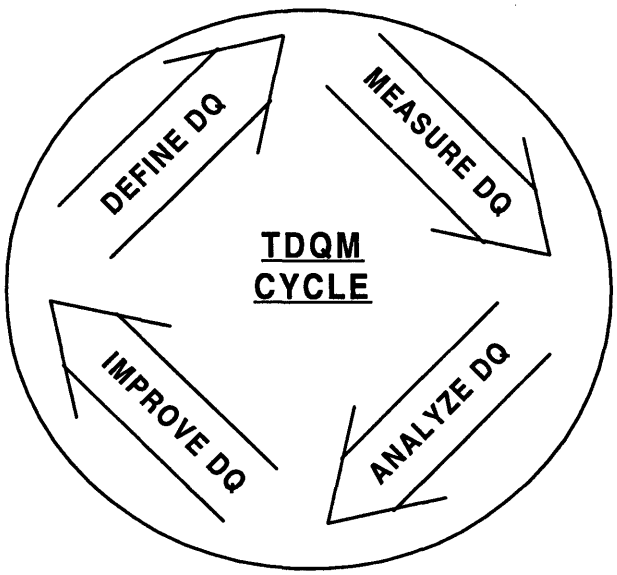


Figure 1: The TDQM Cycle

2. 2 Referential Integrity

The relational model presents two general integrity rules, domain integrity and referential integrity, as methods of DQ definition, measurement, and analysis. This thesis, as mentioned above, focuses on referential integrity [3, 4, 5] and in particular, *business rules* to ensure consistency in organizational databases. Without the enforcement of such rules, inconsistent data will easily corrupt databases and impede subsequent operations. To guarantee integrity in a pizza delivery business, for example, the following set of business rules need to be complied with [12]:

- The order data must always be greater than the date the business started and less than the current date.
- Customer zip codes must be within a certain range (the delivery area).
- The quantity ordered can never be fewer than 1 or greater than 50.
- The ordered pizzas must be constituents of the menu.
- New orders cannot be created for discontinued items.

2.3 Thesis

This thesis will first devise a set of business rules to fulfill the requirements of TDQM and referential integrity. Second, it will develop a software tool to enforce such rules.

Chapter 3

Functional Requirements

3.1 Introduction

Information stored in databases needs to comply with organization-specific business rules, before it can be used for further purposes. The gamut of business rules, for example, can range from checking if the DEPARTMENT of an EMPLOYEE is a constituent of all DEPARTMENTS to ensuring that the overall TOTAL equals the sum of all TRANSACTIONS.

Integrity constraints are such set of business rules designed to enforce consistency between databases. These rules impose comparison of related fields within a table or of different tables to report error percentage or individual violations as requested by users. Derived from related literature, the requisite functions of integrity constraints proposed in this thesis are detailed below.

3.2 Functional Specifications

The functional specifications are first stated mathematically, and sometimes illustrated by examples. The notation used in functional specifications is presented as follows:

X and *Y* represent different fields within a table or of different tables, C_i denotes a constant, and *FUN* encompasses available aggregate functions in SQL. Finally, *[A].[B]* symbolizes Field *B* in Table *A*.

- Field Comparison: *X {>, <, =, >=, <=, <>} Y* (This filter is called CompField in the inter-database software tool).

The default comparison is "=." An example of such rule is that if users want to ensure cost is less than revenue, they can select COST < REVENUE

- Functional Comparison: C_1 *{>, <, =, >=, <=, <>}* C_2** FUN (X) +* C_3 (This rule is called *CompField* in the software tool).

The default comparison, again, is "=", and values of C_2 and C_3 are 1 and 0, respectively. The available aggregate functions provided by SQL include AVG, COUNT, MIN, MAX, and SUM. An example of such rule is that if users need to check that employees do not earn more than twice the average salary in their department, they can specify SALARY <= 2 * AVG(SALARY) [7].

- Subset: *X IsIn Y* (This rule is called *IsIn* in the tool).

If users, for example, want to check that employees' departments are constituents of available departments listed in DEPT [8], they can select [EMPLOYEE].[DEPT_NAME] *IsIn* [DEPT].[DEPT.NAME].

- Negation: the use of *Not* (Negation can be invoked by the use of the *Not* button).

To facilitate expression of various business rules, negation of filters is implemented. For example, the above users can easily negate their rule of *IsIn*:

[EMPLOYEE].[DEPT_NAME] *Not IsIn* [DEPT].[DEPT.NAME]

- Connectors: the use of *And* and *Or* (Connectors can be invoked by the use of the *And* and *Or* buttons).

Users can combine numerous comparisons between fields with logical symbols of And and Or. The precedence of the logic follows the convention that *And* precedes *Or*. For instance, if an organization, desires to see if its departments under manufacturing division have been reporting profit, it can use the *And* connector to specify the following rules:

COST < REVENUE *And*

DEPARTMENT *IsIn* MANUFACTURING

- User-defined Precedence: the use of parenthesis (Precedence can be defined by the use of "(" and ")" buttons).

In addition to the precedence dictated by *And* and *Or*, users may sometimes find it necessary to define precedence themselves for certain complicated rules. When such needs arise, they can resort to parentheses. For instance, if a company needs to identify what departments in manufacturing division or reportedly profitable ones have earned more than 1/3 of the total revenue, it can choose the following combination of rules.

(COST < REVENUE *Or* DEPARTMENT *IsIn* MANUFACTURING) *And*

REVENUE > 1/3 * TOTAL(REVENUE)

Notice that without the insertion of parentheses, the *And* would precede *Or* and change the order:

COST < REVENUE *Or* (DEPARTMENT *IsIn* MANUFACTURING *And*

REVENUE > 1/3 * TOTAL(REVENUE))

- Rule Combination: Composition of a new rules based on defined ones (This rule is called *CompositeRule* in the tool).

Users may sometimes find it convenient to compose a new rule based on priorly defined ones, although they can always manually assemble all constituting filters with the above specifications. This function gains prominence after users have defined a myriad of complex rules and prefer to

build new rules upon the existing ones. For example, if users have created two separate rules, Rule1 and Rule2, and desire to combine them with *And*, they can easily use *CompositeRule* to accomplish this goal.

(Rule1) COST <= REVENUE

(Rule2) REVENUE > 1/3 * TOTAL(REVENUE)

(New Rules) Rule1 *And* Rule2

- Domain Integrity Rules: Uniqueness, Zero Value, Null Value, Format, and Format Restriction (These filters are named identically in the Domain Integrity software tool) [17].

As referential integrity extends its application from domain integrity, it should certainly use filters of domain integrity as basic building blocks in addition to its set of business rules. Domain integrity in TDQM encompasses five filters: Uniqueness, Zero Value, Null Value, Format, and Format Restriction. With these five supplementary filters, users possess more latitude in specifying rules. For example, if a mail order company desires to check that any order originating from Area X has a zip code between 02135 and 02139, it can select the following rules:

[ORDER].[CITY] = (AREA X) *And*

02135 <= [ORDER].[ZIPCODE] <= 02139

Notice that any rule specified in the Domain Integrity component of TDQM can be readily expressed in the Relation Integrity component. With the assistance of the *CompositeRule* function, the task of combining basic filters is further facilitated.

Chapter 4

Architectural Design

4.1 Overview

For the purpose of the TDQM research, Microsoft Access will be adopted as the computing platform for the following reasons. First, the computing environment in Sloan School of Management is oriented towards personal computers and Microsoft Windows applications. Second, Access offers friendly interface as the needed foundation for any TDQM software tools. Third, as the TDQM project aims to create a research protocol rather than any sophisticated software, it does not demand enormous computing power for best concurrency control nor the capacity to handle billions transactions at a time. In consideration of the above reasons, Microsoft Access emerges as a good platform for the TDQM project.

4.2 Overall Structure

The TDQM software tool employs functions embedded in Access as basic building blocks. These functions include table, form (used to display and edit data), query, report, and module (a collection of functions and subroutines). To comply with standards, we use the L/R Naming Standard for Access with the exception that system objects are prefixed with "DQ_".

4.2.1 Table and Query

There exist two permanent tables in this Inter-database software component: DQ_NewRules and DQ_NewRulesLog (see Appendix A.1). DQ_NewRules stores definitions of rules, while DQ_NewRulesLog stores results of rule execution. In addition to the permanent tables, there also exist temporary tables suffixed with "temp" used for rule execution and report creation which are deleted soon after the processing. Finally, the software tool also has tables prefixed with DQ_[rule name] to detail violations of the individual rules.

Analogous to the temporary tables, all queries in the software tool are built dynamically for report creation, and are deleted afterwards.

4.2.2 Form and Report

Forms are the key structures used to create the relation integrity menu, and rule definition menu (see Appendix A.2). Reports, on the other hand, are produced to delivery execution results to users. The design of the forms and reports will be discussed in more detail in section 4.3 and 4.4, respectively.

4.3 Design of the Rule Definition Menu

The design of the rule definition menu should be designed from users' perspective to meet their needs. With that objective in mind, the menu is designed to ask users appropriate questions and create a business rule based on the provided information. The menu is presented in Figure 2.

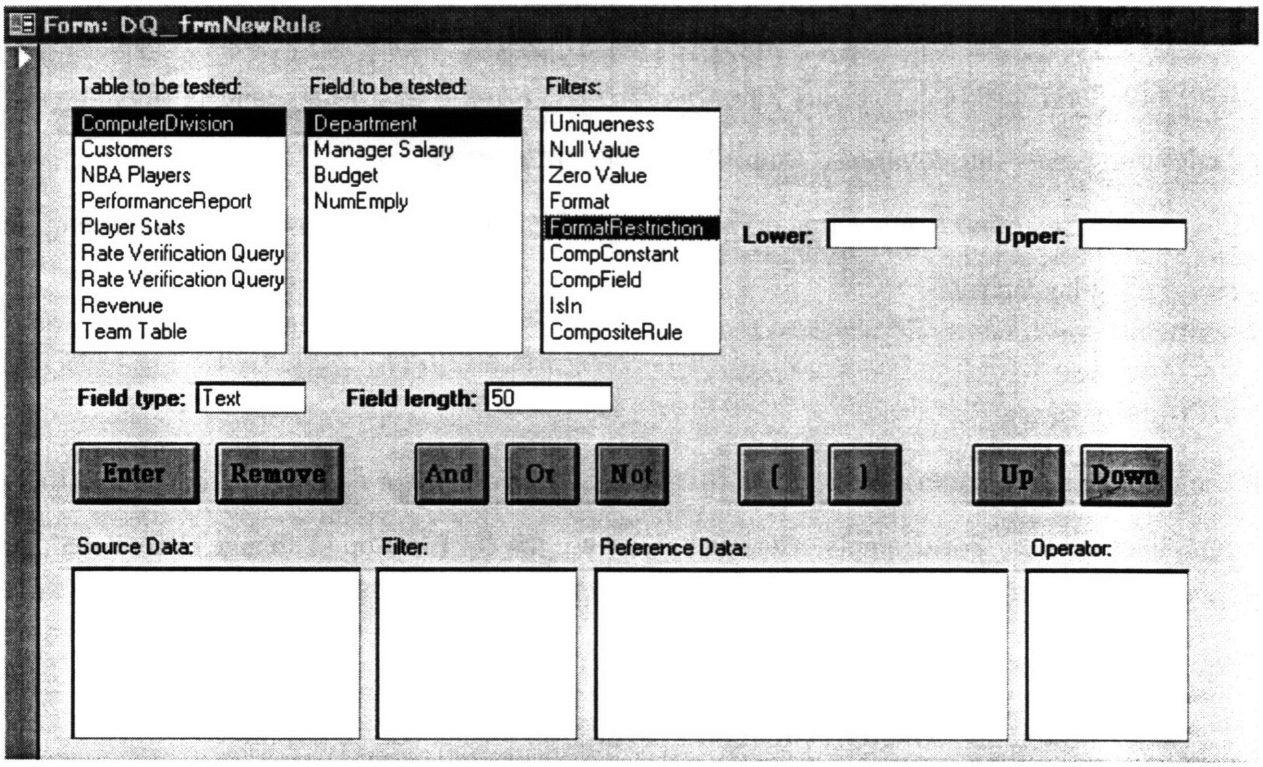


Figure 2: Rule Definition Menu

- **Filter-dependent input boxes**

To help users input requisite information in the menu, filter-specific input boxes are prompted only upon the selection of corresponding rules. Such design has the first advantage of concealing irrelevant input boxes to simplify the interface. Second, it instructs users as to what information should be provided after selection of certain rules. For example, in Figure 2, upon selection of *FormatRestriction* filter, *Lower* and *Upper* input boxes appear next to *FormatRestriction* prompting users to enter filter-dependent information.

- **Enter and Remove**

The menu is separated into two components by various buttons for the purpose of composing multiple filters. The upper half displays relevant information for producing a constituting filter onto a field, while the lower half stores the constituents of the rule. Individual filters can be added to the storage by *Enter*, and removed from by *Remove*. For example, *Enter* in Figure 2 add

[ComputerDivision].[Department] into *Source Data*, *Format* into *Filter*, and information from *Lower* and *Upper* into *Reference Data*. The default *operator* is *And*.

The *Enter* and *Remove* functions are especially important when users need to develop and revise complicated rules.

- **Up and Down**

The rule definition menu also provides *Up* and *Down* functions for users to rearrange the order of complicated rules. For example, users can easily use the *Up* function to change *(X or Y and Z)* to *(Y or X and Z)* without the need to remove both rules and start anew.

- **Other Functions: And, Or, Not, and Parenthesis**

The functions of *And, Or, Not*, and *Parenthesis* can be easily applied to highlighted constituting filters in the storage. The usage of *parenthesis* and *Not*, in particular, adheres to the intuition of mutual cancellation. In other words, a click of ")" cancels one "(" in the highlighted filter and vice versa, and negation of a negated filter is the filter itself.

4.4 Design of the Output

After users submit their business rules for execution, they await the delivery of the results in comprehensible reports for their perusal. Analogous to the rule definition menu, the report should also be designed from users' perspective. The following features are proposed for such purpose.

- **Summary and Report**

The reports arrive to users in two formats: summary and report. The summary format only produces the number of violations of individual filters, while the report format details overall violations and the violation ratio. The users can choose the format of the report for their specific purposes. The number of the violations in the report format is equal to or less than the sum of individual filter violations in the summary format, depending on the connector type. In the case of *And*, the union of violations of individual filters is taken to produce the final output. On the other hand, if the connector is *Or*, intersection of violations of individual filters is performed.

- **Exception fields**

In addition to the format selection, users at times may be interested in certain fields of the violation records. To eliminate irrelevant information, they can choose the fields of interest to them with the option of exception fields.

4.5 Module

Appendix A.3 lists all the functions and subroutines used for this inter-database software tool.

Chapter 5

Application Example

5.1 Introduction

This chapter illustrates how the inter-database software tool operates based a detailed example. The example uses two tables of company X: *PerformanceReport* in Figure 3 lists costs and revenues of various departments, and *ComputerDivision* in Figure 4 enumerates departments under the computer division.

Table: PerformanceReport

Month	Year	Cost	Revenue	DepartmentID	Department
5	96	755502	666600	A564	Product Line
5	96	470090	340054	A233	New Media
5	96	150021	5003	B022	Stamp
5	96	250032	340032	A320	Service
5	96	30022	240022	X406	Sales
5	96	1400022	10055	C677	Maintenance
5	96	500055	450033	B079	Food
5	96	80932	44021	D339	Applications

Figure 3: PerformanceReport Table

Table: ComputerDivision

Department	Manager Salary	Budget	NumEmply
Server	50000	100000	9
New Media	65000	345000	10
Applications	50000	254320	5
Product Line	48000	150044	5
Service	60000	205400	9

Figure 4: ComputerDivision Table

5.2 Basic Menu

Figure 5 presents the main menu of the TDQM project with the embedded four phases: definition, measurement, analysis, and improvement. The inter-database software component is part of the *definition* phase and can be invoked with a click on the *Relation Integrity Rules*.

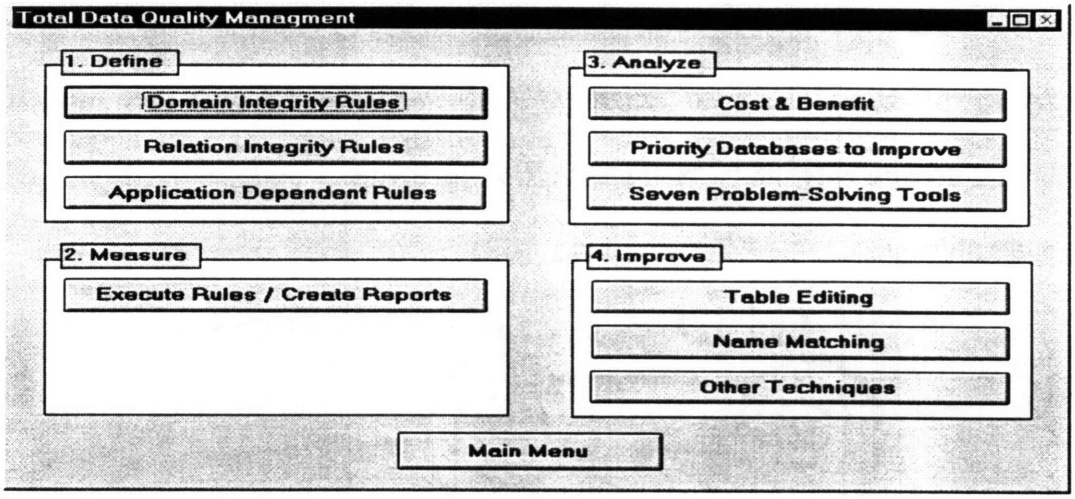


Figure 5: Main Menu

5.3 Rule Menu

Figure 6 demonstrates the rule menu of the inter-database software tool. In terms of rule manipulation, the menu encompasses functions of rule definition, edition, deletion, and execution. Upon selection of a rule from the rule repository, the *Rule Name, Table*, and *Rule Description* promptly display the corresponding information of the rule. In addition, the rule menu also supports exception table edition used solely for selecting fields of interest to the users in the final report[1].

[1] Exception table is introduced in section 4.4.

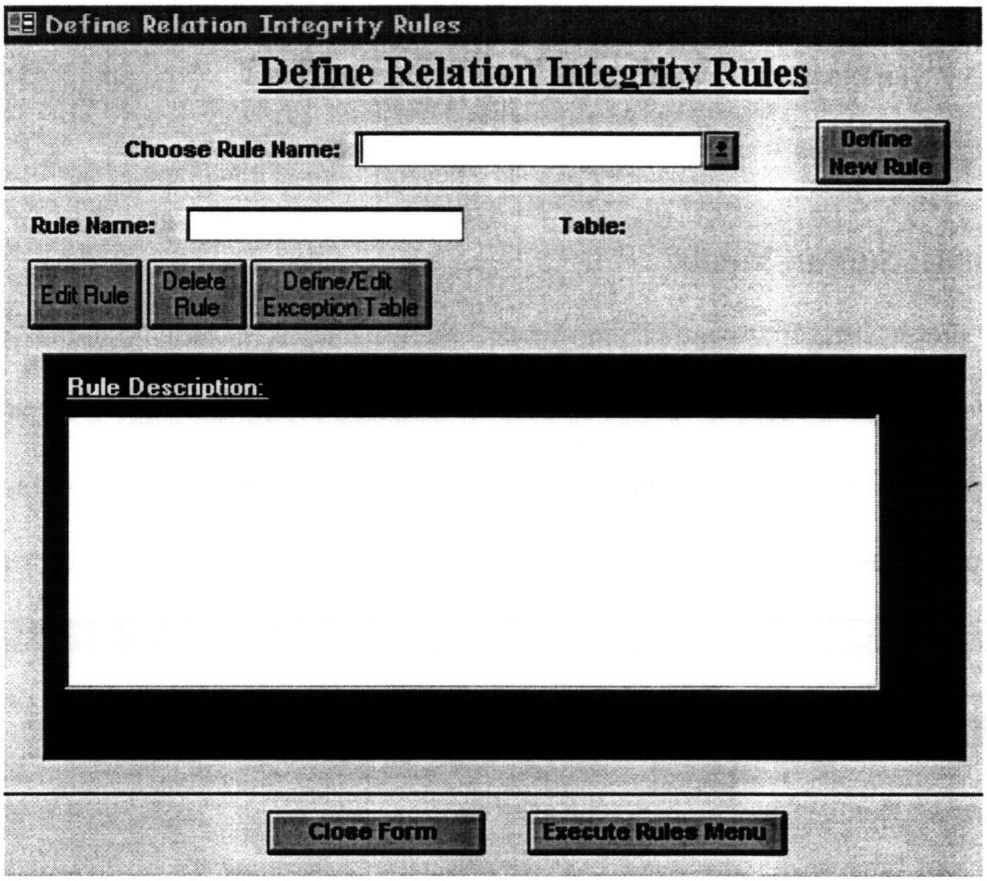


Figure 6: Relation Integrity Menu

In this application example, let's assume that company X desires to find non-profitable (COST >= REVENUE) computer departments from *PerformanceReport* table. In other words, a permissible department is either profitable (COST < REVENUE) or a non-computer department. To check this condition, company X names the rule *CompDeptLoss* and proceeds to the rule definition menu to specify the following two filters.

(Filter 1) [PerformanceReport].[Cost] < [PerformanceReport].[Revenue] *Or*

(Filter 2) [PerformanceReport].[Department] *Not IsIn* [ComputerDivision].[Department]

THE SOLVENT DEPENDENCE OF ENZYMATIC SELECTIVITY

by CHARLES R. WESCOTT

Submitted to the Department of Chemistry on May 9, 1996, in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Chemistry

ABSTRACT

The exacting selectivity of enzymatic catalysis is the most valuable characteristic of enzymes to the synthetic chemist. Ironically, this strict specificity also limits the generality of enzymatic synthesis, because enzymes that catalyze the reaction of interest with the desired selectivity are not always available. Nonaqueous enzymology, and the discovery that enzymatic selectivity can be changed by the reaction medium, thus greatly enhances the utility of enzyme-catalyzed syntheses. To further exploit this solvent effect, we seek herein to mechanistically explain the dependence of enzymatic selectivity on the solvent.

The substrate specificity of the serine protease subtilisin Carlsberg in the transesterification reaction of N-Ac-L-Ser-OEt and N-Ac-L-Phe-OEt with propanol has been examined in 20 anhydrous solvents. The serine substrate is strongly favored in some solvents, while the phenylalanine substrate is greatly preferred in others. An equation has been derived on the basis of the thermodynamics of substrate desolvation, which correctly predicts the substrate specificity as a function of the solvent-to-water partition coefficients of the substrates and the substrate specificity of the enzyme-catalyzed hydrolysis of the esters in water. This model is herein demonstrated to be independent of the enzyme and the substrate, so long as the latter is removed from the solvent in the transition state.

Experimentally measured solvent-to-water partition coefficients are nonideal for use in the prediction of the solvent dependence of enzymatic selectivity for several reasons. First, partition coefficients cannot be readily measured for water-miscible solvents. Second, the mutual solubility of aqueous and organic phases influences the measured partition coefficients. Third, the effects of additives to the reaction medium, such as a second substrate, cannot be included. These problems have been overcome by calculating the partition coefficients from the substrate activity coefficients using UNIFAC.

For the case of substrate specificity, the differential free energy of desolvation for two substrates is primarily driven by chemical differences in the substrates. In cases of stereoselectivity (e.g. enantioselectivity, prochiral selectivity, and regioselectivity), however, chemically identical substrates lead to the formation of multiple products. For such identical substrates, desolvation energy differences arise from differences in the solvation of the substrates in the transition states which lead to the various products. Our model has been expanded to account for partial transition state desolvation, which is assessed using molecular modeling based on the crystal structure of the enzyme. Using this methodology, we are able to quantitatively predict the solvent dependence of the enantioselectivity of cross-linked chymotrypsin crystals in the resolution of racemic methyl 2-hydroxy-3-phenylpropionate.

Thesis Supervisor: Dr. Alexander M. Klibanov, Professor of Chemistry

5.4.1 Creation of the First Filter

To create the first constituting filter, company X should highlight the table and the field to be tested. Upon selection of *PerformanceReport*, the *Field to be tested* will display corresponding fields in *PerformanceReport* so that *Cost* can be chosen. Next, as the filter is an instance of *field comparison*[3], company X should select the *CompField* filter and provide appropriate information as prompted by the *CompField*-dependent input boxes, namely the *comparison* and table and field to be compared. In this case, *PerformanceReport* and *Revenue* are the table and field to be compared. After company X has completed the above two steps, it can click on the *Enter* button and add this filter to the filter collection. The menu at the time should is shown in Figure 8.

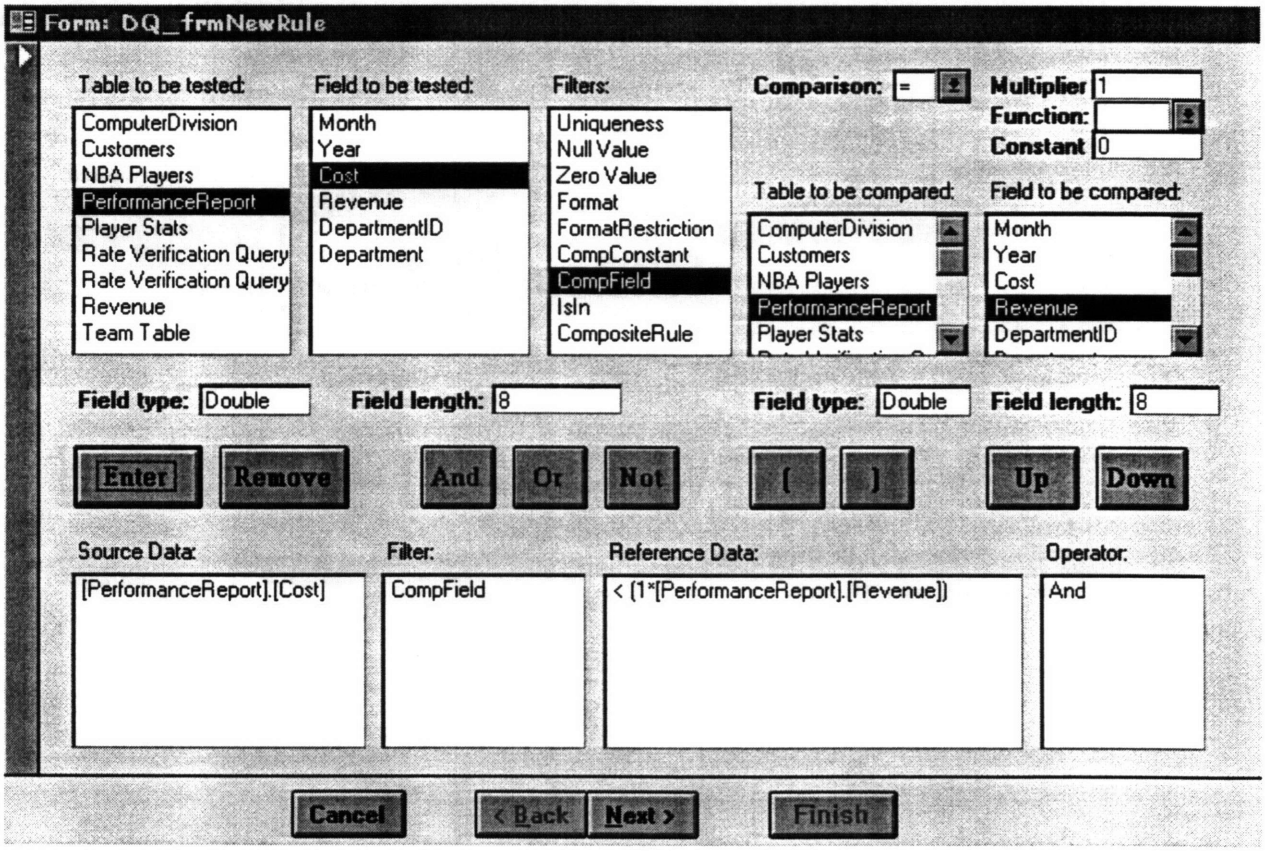


Figure 8: Creation of the First Filter

[3] Field Comparison is the first functional specification discussed in section 3.2.

5.4.2 Creation of the Second Filter and the Connector

To produce the second filter, company X should follow the procedure listed in section 5.4.1. First, it should highlight the table and the field to be tested. In this filter, the table is again *PerformanceReport* and the field is *Department*. Second, it should choose the *IsIn* filter and fill the *IsIn*-dependent input boxes -- table and field to be compared -- with *ComputerDivision* and *Department*, respectively. Third, after it adds this filter into the collection, it should apply the *Not* button to negate the filter[4]. Finally, it has to change the connector between the filters from the default value *And* to *Or*. Figure 9 demonstrates the resulting menu.

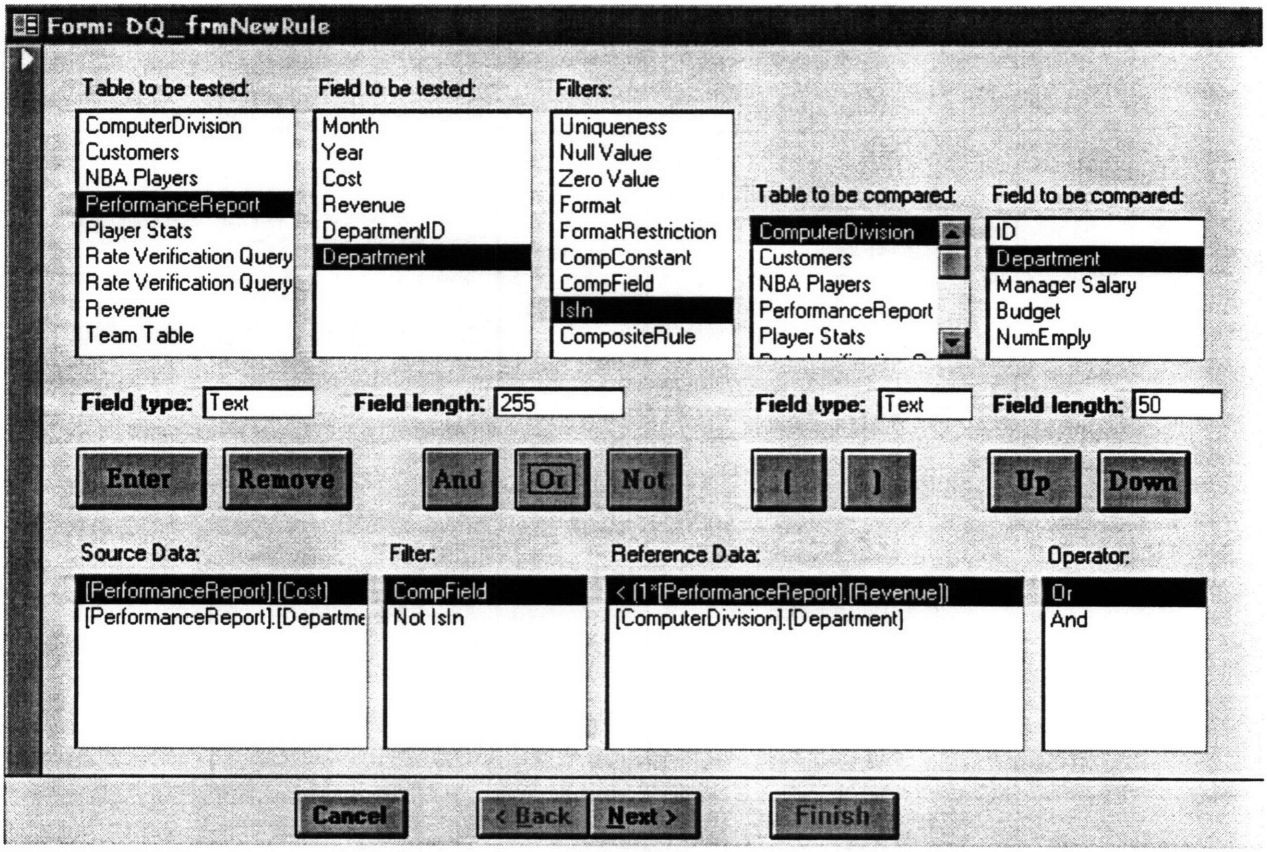


Figure 9: Creation of the Second Filter

[4] Refer to section 4.3 and 5.4.4.

5.4.3 Composite Rule

If company X desires to enforce modularity and create basic rules for later use, it can store Filter 1 and Filter 2 as two distinct rules and subsequently combine them to produce *CompDeptLoss*. Assume that company X in this example defines Filter 1 as *NegativeProfit*, and Filter 2 as *NonCompDept*. It can then utilize *CompositeRule* filter to compose *NegativeProfit* and *NonCompDept* into *CompDeptLoss*. Figure 10 details the specifications of such a composite rule.

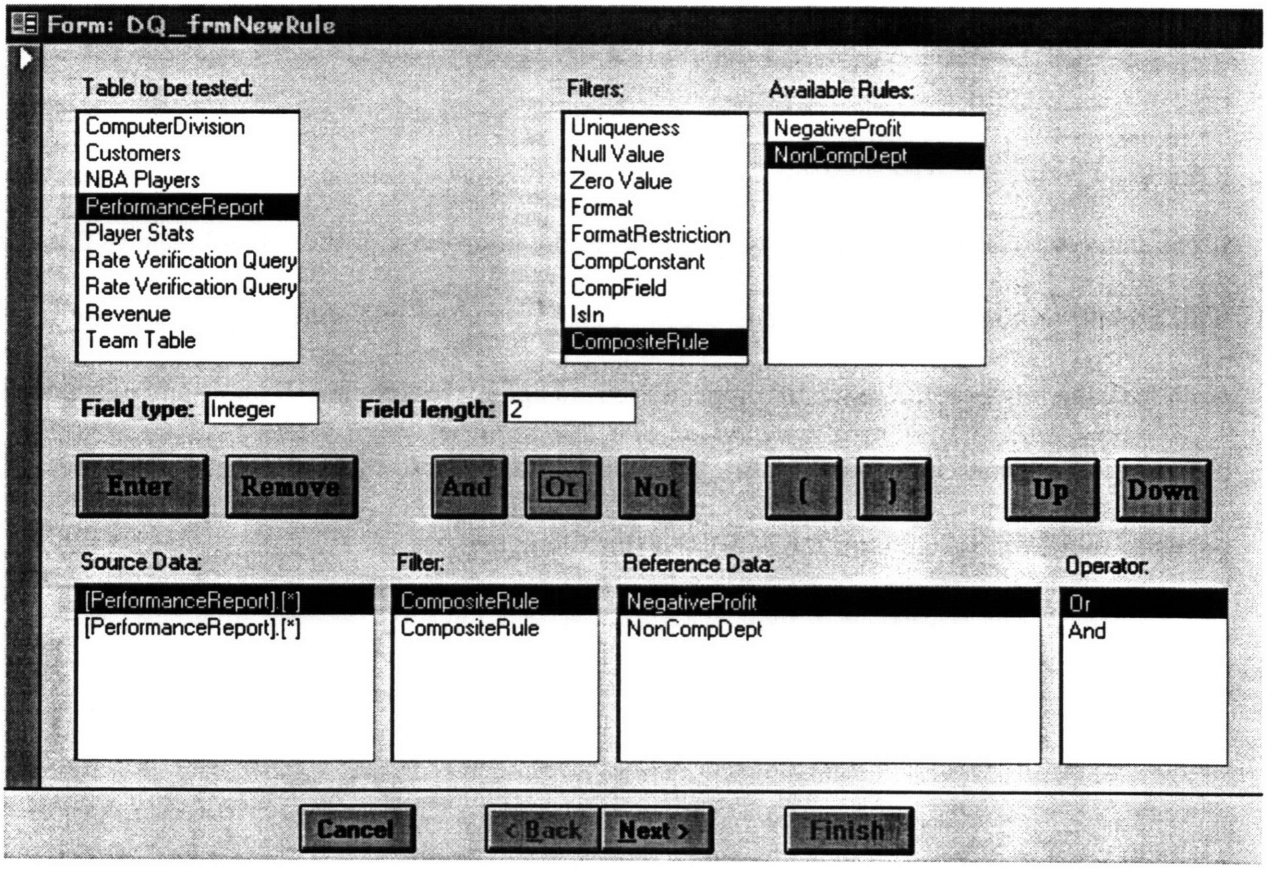


Figure 10: Composite Rule

5.4.4 Rule Edition

If company X makes any mistake during the rule creation, it can resort to the buttons to correct the mistake. For example, to remove a filter from the collection, company X only needs to highlight the filter and click on *Remove*. The functions of *Remove, And, Or, Not, Parenthesis, Up,* and *Down* are all designed to be applied to highlighted constituting filter in the storage. The usage of *Parenthesis* and *Not*, in particular, adheres to the intuition of mutual cancellation so that a click of ")" cancels one "(" in the highlighted filter and vice versa, and negation on the negated filter is the filter itself.

5.4.5 Rule Name and Description

If company X is satisfied with the composed rule, it can proceed to the second page of the menu with a click on the *Next* button. The second page, presented in Figure 11, displays the rule in plain text and requests the requisite input of the rule name. If company X spots any mistakes and needs to edit the rule, it can click on the *Back* button to return to the first page. Otherwise, it can continue to the third page after it has entered the rule name.

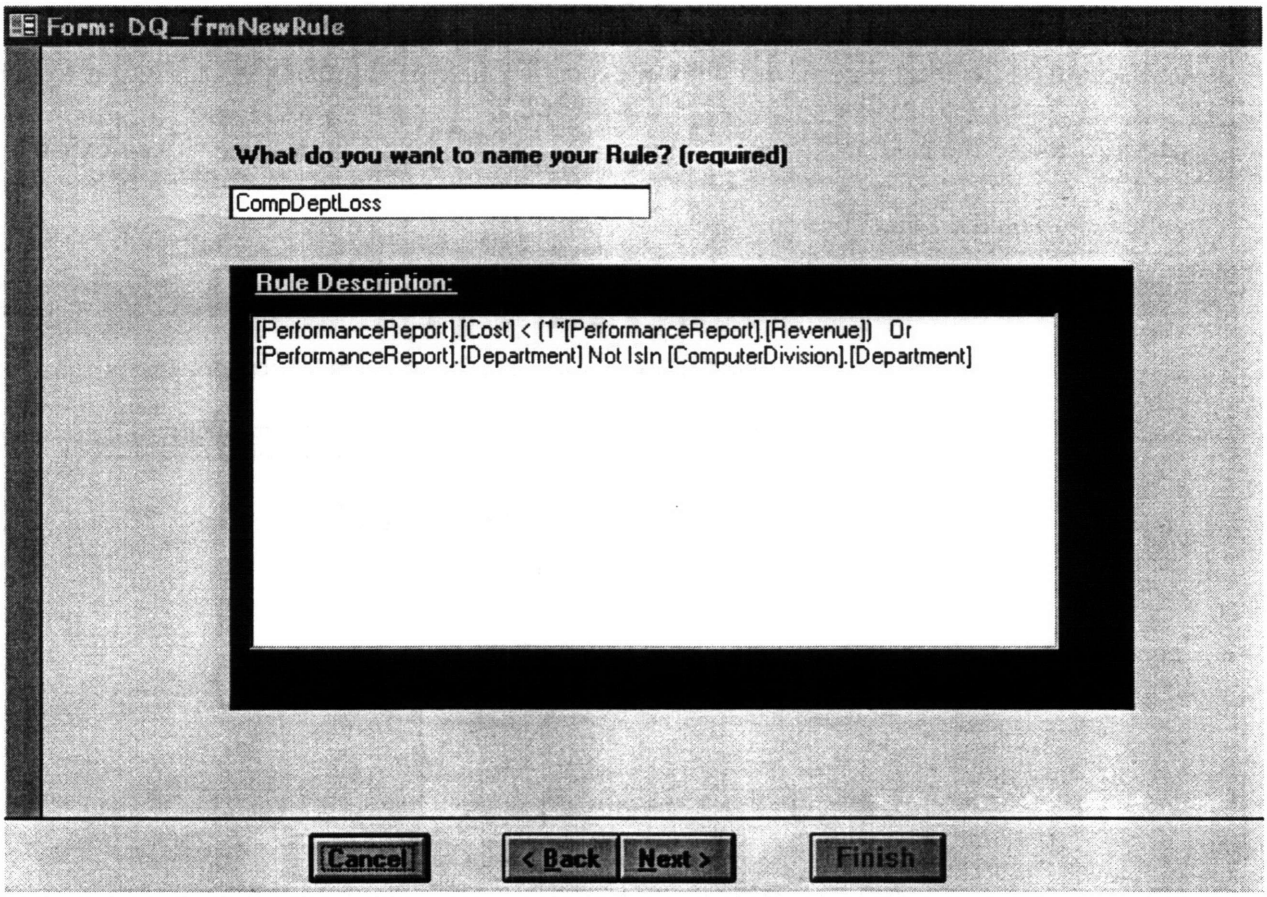


Figure 11: Rule Name and Description

5.4.6 Exception Table and Execution Options

The third page allows company X to choose exception fields to be displayed on the final report. If company X, for example, desires only to see *Cost*, *Revenue, DepartmentID,* and *Department* on the report, it can create this output by selecting these four fields from *Available Fields* to *Exception Fields* as shown in Figure 12. Exception table is especially useful when users intend to view a few relevant fields out of a multitude in a table.

This page also offers three execution options. The first option simply saves the rule. The second option executes the rule and produces the violation ratio. The final option runs the rule

and outputs a detailed report with chosen exception fields. Company X can return to previous pages to revise the rule at anytime. After it finishes the rule composition, it can exit the menu with a click on the *Finish* button.

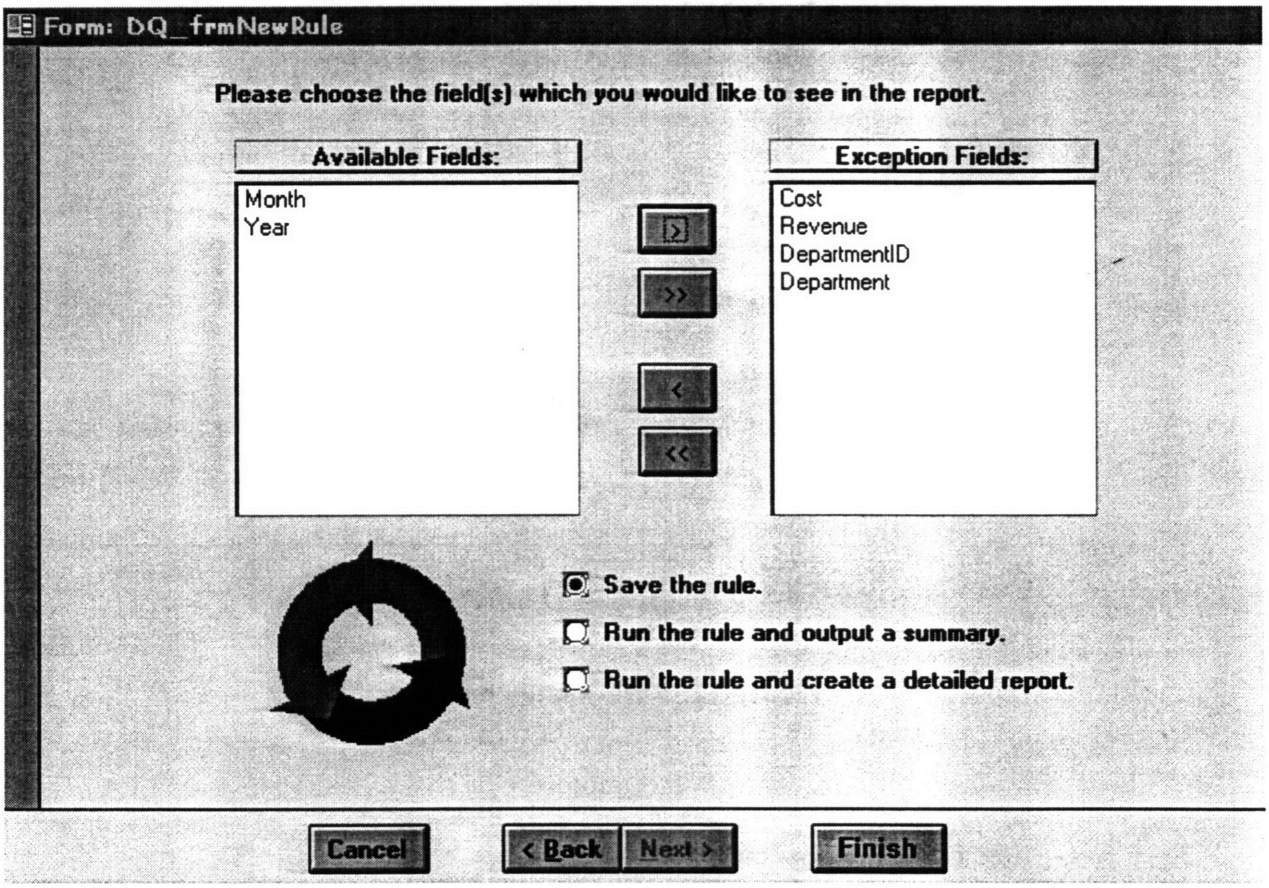


Figure 12: Exception Table and Execution Options

5.5 Rule execution

Upon the creation of *NonCompDept*, the inter-database rule menu in Figure 6 is updated to reflect the information as shown in Figure 13.

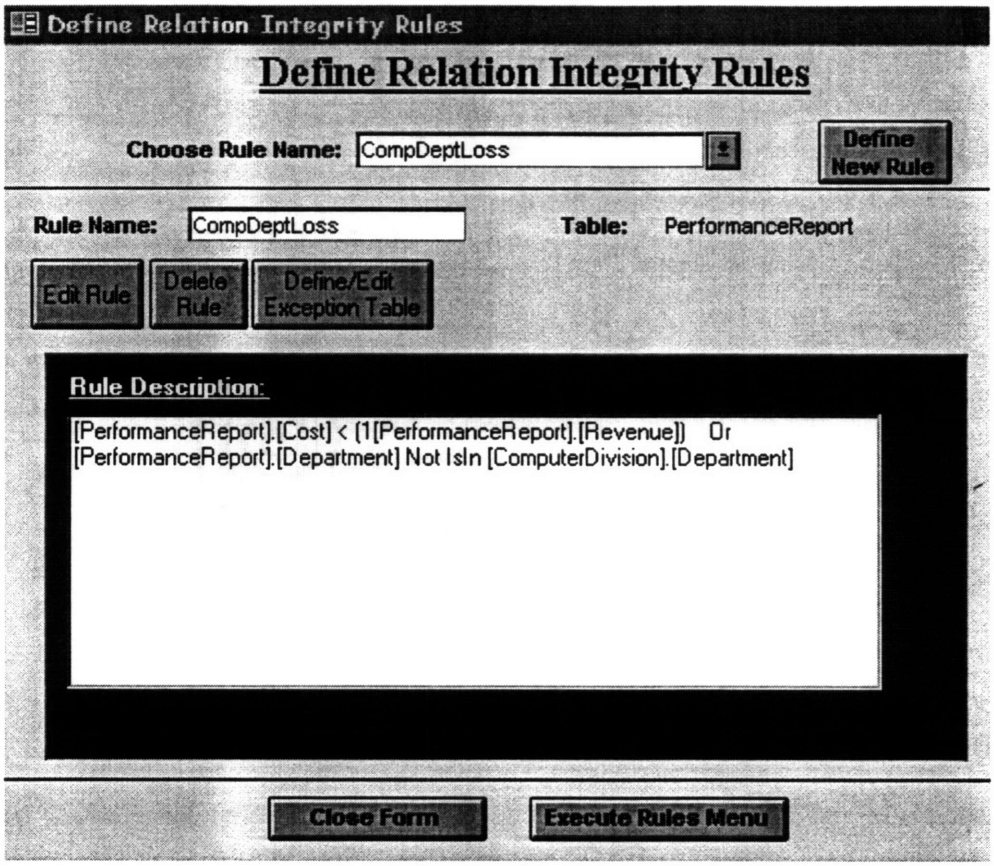


Figure 13: Updated Relation Integrity Menu

Rules can be executed from either the definition menu or the *Execute Rule Menu* button from Figure 13. The Rule Execution Menu in Figure 14 is launched with a click on the button. The menu offers options to execute rule, create report, and return to Figure 13.

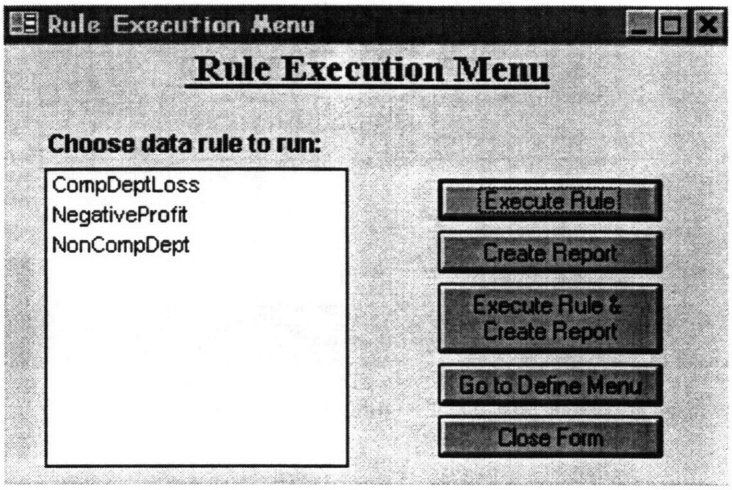


Figure 14: Rule Execution Menu

5.5.1 Summary

To execute a rule and output a summary, choose the rule and click on *Execute Rule* button in Figure 14. The summary of *CompDeptLoss* created by company X is demonstrated in Figure 15.

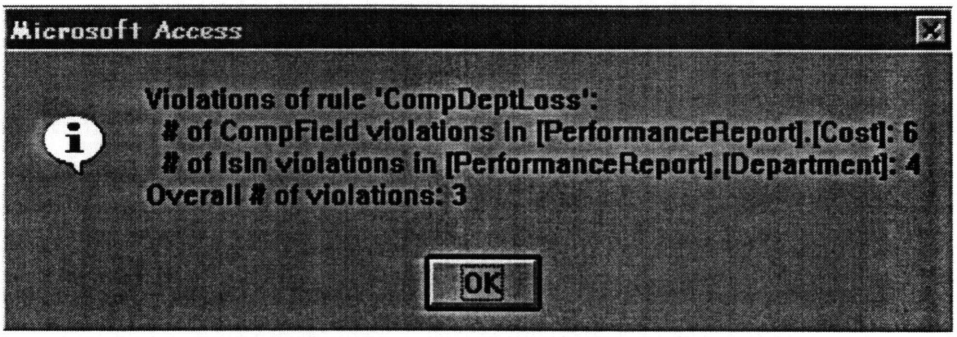


Figure 15: Summary of CompDeptLoss

5.5.2 Report

If company X also hopes to peruse the detailed report of *CompDeptLoss*, it can click on either the *Create Report* or the *Execute Rule & Create Report* button. Figure 16 displays all execution

records of *CompDeptLoss* and prompts company X to select one for the purpose of report creation.

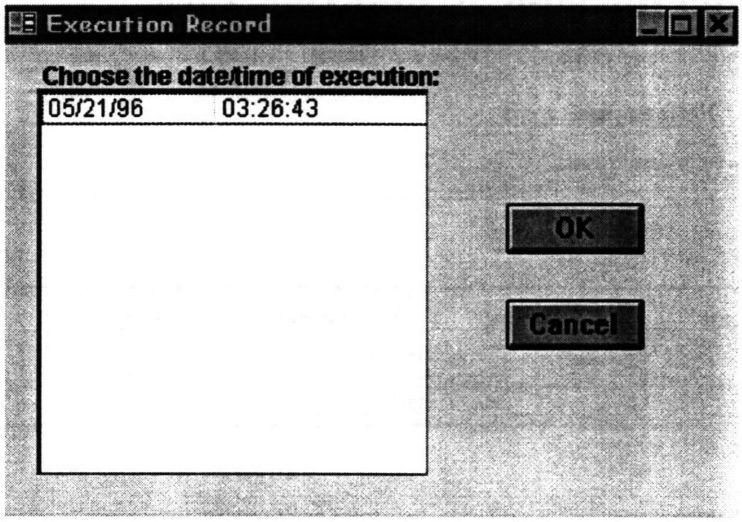


Figure 16: Execution Record

Figure 17 presents the report enumerating violations produced after company X chooses the data/time of the execution. As company X only desires to see *Cost, Revenue, DepartmentID*, and *Department* on the report, the report solely displays these four fields from *PerformanceReport* table. The report contains number of violation, and the violation ratio calculated from dividing the number of violation by the number of records in the table. In this example, as number of violation in *CompDeptLoss* is 3, and the number of records is 8, the violation ratio is accordingly 3/8 (38%). Furthermore, the number of the violations in the report is less than the sum of individual filter violations in the summary, as the connector in *CompDeptLoss* is *Or* and intersection is performed[5].

[5] Refer to section 4.4.

Report for rule "CompDeptLoss"

This rule was executed at 08:33:12 on 05/24/96

Table: PerformanceReport

The "PerformanceReport" table contains 8 records.

Number of violations: 3 Violation ratio: 38%

Cost	Revenue	DepartmentID	Department
755502	666600	A564	Product Line
470090	340054	A233	New Media
80932	44021	D339	Applications

Figure 17: Report of CompDeptLoss

5.6 Rule Edition

If company X needs to edit or delete *CompDeptLoss* after the execution, it can return to Figure 13 and click on appropriate buttons. In addition, it can also modify the exception fields and produce a new report with updated fields.

5.7 Summary

This chapter illustrates how the inter-database software tool operates based a detailed example of *CompDeptLoss*. More details of the tool and actual code are included in the Appendix.

Chapter 6
Conclusion

This thesis presents the inter-database software tool as part of concerted research effort of the Total Data Quality Management (TDQM) project. It focuses primarily on referential integrity and in particular, business rules pertaining to organizational needs. The set of business rules proposed and implemented in this thesis are based on generalizations derived from integrity constraints and normalization literature.

As mentioned in section 3.2, this inter-database software tool extends its application from domain integrity concerned with intra-database operations. Thus, it employs the intra-database filters as basic building blocks in addition to its own set of rules, and possesses the capacity to execute intra-database rules.

From the perspective of the TDQM cycle, this inter-database software tool addresses referential integrity and follows the first three phases of the cycle: definition, measurement, and analysis. With future additions, the TDQM project will evolve into a complete tool to manage the quality of organizational data.

References

[1] Ballou, D. P. and H. L. Pazer, Designing Information Systems to Optimize the Accuracy-Timeliness Tradeoff. *Information Systems Research (ISR),* 6(1) 1995, pp. 51-72.

[2] Ballou, D. P. and H. L. Pazer, Modeling Data and Process Quality in Multi-input, Multi-output Information Systems. *Management Science*, 31(2) 1985, pp. 150-162.

[3] Codd. *The Relational Model for Database Management.* Version 2. Addison-Wesley, 1990.

[4] Date, C. J., *An Introduction to Database Systems.* Volume 2. Addison-Wesley, 1981.

[5] Date, C. J., *An Introduction to Database Systems.* 4th ed. Addison-Wesley, 1986.

[6] Date, C. J., *An Introduction to Database Systems.* 5th ed. Addison-Wesley, Reading, 1990.

[7] Delobel C. and M. Adiba, *Relational Database Systems.* North-Holland, 1985, p. 286.

[8] Delobel C. and M. Adiba, *Relational Database Systems.* North-Holland, 1985, p. 287.

[9] Delone, W. H. and E. R. McLean, Information Systems Success: The Quest for the Dependent Variable. *Information Systems Research*, 3(1) 1002, pp. 60-95.

[10] Deming, E.W., *Out of the Crisis.* Center for Advanced Engineering Study, MIT, Cambridge, 1986.

[11] Elmasri, R. and S. Navathe, *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Co., Inc., Reading, MA., 1994.

[12] Getz, K., P. Litwin and G. Reddick, *Microsoft Access 2 Developer's Handbook*, 1992, pp. 43-44.

[13] Huge, E. C., *Total Quality: An Executive's Guide for the 1990s.* Richard D. Irwin, Inc., Homeward, Illinois, 1990.

[14] Ishikawa, K., *What is Total Quality Control?-the Japanese Way.* Prentice-Hall, Englewood Cliffs, 1985.

[15] Juran, J. M., *Juran on Quality by Design: The New Steps for Planning Quality Into Goods and Services.* Free Press, New York, 1992.

[16] Juran, J. M. & G., F. M., *Quality Control Handbook.* 4 ed. McGraw-Hill Book Co., New York, 1988.

[17] Madnick, S. & R. Y. Wang (1992). *Introduction to the TDQM Research Program, TDQM Working Paper.* (No. TDQM-92-01). Total Data Quality Management (TDQM) Research Program, MIT Sloan School of Management.

[18] Peou (1996), R., *Design & Implementation of System DQ: A Data Quality Management System.* Bachelor Thesis, Electrical Engineering and Computer Science (EECS) Department, Massachusetts Institute of Technology.

[19] Wand, Y. and R.Y. Wang, Anchoring Data Quality Dimensions in Ontological Foundations. *Communications of the ACM*, 1995.

[20] Wang, R.Y., V. C. Storey and C. P. Firth, A Framework for Analysis of Data Quality Research. *IEEE Transactions on knowledge and Data Engineering*, 7(4) 1995, pp. 623-640.

[21] Wang, R. Y (1996), *A Software Tool for Total Data Quality Management, TDQM Working Paper.* (No. TDQM-96-01). Total Data Quality Management (TDQM) Research Program, MIT Sloan School of Management.

[22] Wang, R. Y. and D. M. Strong. Beyond Accuracy: What Data Quality Means to Data Consumers. (No. TDQM-96). *Journal Of Management Information Systems (JMIS),* Volume 12, Issue 4 1996, pp. 5-34.

[23] Wang, R. Y. and H. B. Kon, *Towards Total Data Quality Management (TDQM)*, in *Information Technology in Action: Trends and Perspectives*, R. Y. Wang, Editor. 1993, Prentice Hall, Englewood Cliffs, NJ., 1993.

[24] Wiederhold, G., Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 1991.

Appendix

A.1 Table Definition

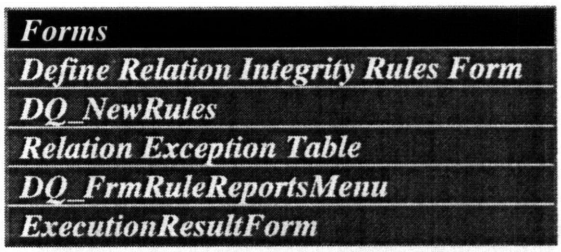
DQ_NewRules:

Field Name	Data Type	Description
RuleName	Text	
RuleNumber	Text	
NumRule	Number	
Date	Text	
Time	Text	Date/Time when rule was first created.
SourceData	Text	
ExceptList	Text	Exception fields
Filter	Text	
FormatComp	Text	Format variable
FormatSize	Number	Format variable
LowerRange	Number	FormatRestriction variable
UpperRange	Number	FormatRestriction variable
Compare1	Text	CompConstant variable
Constant	Text	CompConstant variable
Compare2	Text	CompField variable
Multiplier	Number	CompField variable
Function	Text	CompField variable
CompSource	Text	
FieldConstant	Number	CompField variable
CompositeRule	Text	
Operator	Text	
Paren	Number	

DQ_NewRulesLog:

Field Name	Data Type	Description
RuleName	Text	
Date	Text	
Time	Text	
SourceData	Text	
ExceptList	Text	Exception fields
NumViolation	Number	Number of violations
TableSize	Number	Table size

A.2 Forms



A.3 Module

```
Function Anyposition (start%, ss$, i%, s$) As Integer
' Purpose: Return the position of the first character after ith "s" in string ss
'                starting from start

  Dim j%, k%

  k = start
  For j = 1 To i
    k = InStr(k, ss, s) + 1
  Next j
  Anyposition = k

End Function

Function ChangeStringField (list$, Fnum%, fld$) As String
' Purpose: Replace a string field specified by Fnum in a string list
'          with new field fld

    Dim x%, y%
    x = Position(1, list, Fnum - 1)
    y = InStr(x, list, ";")
    ChangeStringField = Mid$(list, 1, x - 1) & fld & Mid$(list, y)

End Function

Function Createfrm1 (table_name$)
' Purpose: Convert the violation table, table_name, to a form with all fields
    On Error GoTo Err_CreateFrm1

    Dim frmcreate As Form, MyRecSet As Recordset, mycontrol As Control
    Dim mydb As Database, N%, m%, result$, fname$, x%, y%

    Set mydb = DBEngine.Workspaces(0).Databases(0)
    Set MyRecSet = mydb.OpenRecordset(table_name, DB_OPEN_TABLE)
    x = 0
    y = 0

    Set frmcreate = CreateForm()
    frmcreate.viewsallowed = 2
    frmcreate.defaultview = 2
    frmcreate.recordsource = table_name    'specify the default table
    m = MyRecSet.Fields.Count
    N = 0
    frmcreate.section(0).height = m * 500
    Do
        fname = MyRecSet.Fields(N).name
```

```
        Set mycontrol = CreateControl(frmcreate.name, 109, 0, "", fname, x, y, 1000,
200)
        mycontrol.controlsource = fname
        mycontrol.name = fname
        y = y + 300
        N = N + 1
    Loop Until N >= m

    frmcreate.SetFocus

Exit_CreateFrm1:
    DoCmd Echo True
    Exit Function

Err_CreateFrm1:
    MsgBox Error$, 0 Or 48, "Createfrm1"
    Resume Exit_CreateFrm1

End Function

Function Createfrm2 (table_name$, field_list() As String)
' Purpose: Convert the violation table, table_name, to a form with fields
'          specificed by field_list

    On Error GoTo Err_Createfrm2

    Dim frmcreate As Form, mycontrol As Control, mydb As Database
    Dim N%, m%, result$, fname$, x%, y%


    Set mydb = DBEngine.Workspaces(0).Databases(0)
    x = 0
    y = 0

    Set frmcreate = CreateForm()
    frmcreate.viewsallowed = 2
    frmcreate.defaultview = 2
    frmcreate.recordsource = table_name
    m = UBound(field_list)
    N = LBound(field_list)
    frmcreate.section(0).height = (m - N + 1) * 300
    For N = LBound(field_list) To m
        fname = field_list(N)
        Set mycontrol = CreateControl(frmcreate.name, 109, 0, "", fname, x, y, 1000,
200)
        mycontrol.controlsource = fname
        mycontrol.name = fname
        y = y + 300
    Next N

    frmcreate.SetFocus

Exit_Createfrm2:
    DoCmd Echo True
    Exit Function

Err_Createfrm2:
    MsgBox Error$, 0 Or 48, "Createfrm2"
    Resume Exit_Createfrm2

End Function

Sub DeleteTable (tbl$)
' Purpose: Delete table tbl

    DoCmd SetWarnings False
```

```
    Dim sel$

    sel = "DROP TABLE [" & tbl & "];"
    DoCmd RunSQL sel

End Sub

Function DelStringField (list$, Fnum%) As String
' Purpose: Delete a field specified by Fnum in a string list

    Dim x%, y%

    x = Position(1, list, Fnum - 1)
    y = InStr(x, list, ";")
    DelStringField = Mid$(list, 1, x - 1) & Mid$(list, y + 1)

End Function

Function EvaluateNewRule (RuleName$, prefix$) As String
' Purpose: Evaluate a referential rule


    On Error GoTo Err_EvaluateNewRule

    Dim mydb As Database, MyQuery As QueryDef, myset As Recordset
    Dim tset As Recordset, myset_tbl1 As Recordset, myset_tbl2 As Recordset
    Dim cnt%, level%, tblist$, dummy As Variant, tbl1$, tbl2$
    Dim TableSize%, x%, message$, i%, j%, sel$, lst$, rname$
    Dim timestamp$, ddate$, ttime$, ExceptList$, num%, MyRecSet As Recordset

    Set mydb = DBEngine.Workspaces(0).Databases(0)

    Set myset = mydb.OpenRecordset("DQ_NewRules", DB_OPEN_TABLE)      ' Open table.
    myset.Index = "PrimaryKey"    ' Select index.

    rname = RuleName
    lst = prefix$ & rname & "_Violations"
    cnt = 0
    level = 0

    rname = RuleName & "." & cnt
    myset.Seek "=", rname
    If myset.NoMatch Then
        EvaluateNewRule = ""
        GoTo Exit_EvaluateNewRule
    End If
    num = myset!NumRule
    If Not IsNull(myset!ExceptList) Then
        ExceptList = myset!ExceptList
    Else
        ExceptList = ";"
    End If

    ReDim tables(num) As String, op(num) As String, paren(num) As Integer

    Do Until myset.NoMatch
      cnt = cnt + 1
      level = level + 1
      rname = RuleName & "." & cnt
      myset.Seek "=", rname                          ' Seek record.
      If Not myset.NoMatch Then                      ' If record is found.
        tables(level) = ";" & Operate(myset, prefix$, level) & ";"
        If (cnt = num) Then
           op(level) = "Or"     'last operator needs to be Or
        Else
           op(level) = myset!Operator
        End If
```

```
        paren(level) = myset!paren

        While (level > 1 And (paren(level) < 0 Or paren(level) = 0 And Not (op(level)
= "And" And op(level - 1) = "Or")))
              level = level - 1
              tables(level) = ExecuteOp(op(level), tables(level), tables(level + 1),
prefix$, level)
              op(level) = op(level + 1)
              paren(level) = paren(level) + paren(level + 1)
        Wend
      End If
    Loop

    myset.Close

    'tables(1) is a list of resulting violation tables
    x = Anyposition(1, tables(1), 2, ";")
    If x <> 1 Then
        tbl2 = Mid$(tables(1), 2, x - 2 - 1)
    End If

    EvaluateNewRule = tbl2

Exit_EvaluateNewRule:
    Exit Function

Err_EvaluateNewRule:
    MsgBox Error$, , "EvaluateNewRule"
    Resume Exit_EvaluateNewRule

End Function

Sub ExecuteIntersec2 (table1$, table2$)
' Purpose: Interset table1 and table2 into table1
' Warning: Table1 and table2 must have the same number and type of fields

    DoCmd SetWarnings False

    Dim mydb As Database, MyRecSet As Recordset, sel As String, fname As String
    Dim i As Integer, MyQuery As QueryDef, temptable As String


    temptable = "DQ_tmp_" & table1
    sel = "SELECT [" & table1 & "].* INTO [" & temptable & "] FROM [" & table1
    sel = sel & "] WHERE (((0)=1));"
    DoCmd RunSQL sel
    Set mydb = DBEngine.Workspaces(0).Databases(0)
    Set MyRecSet = mydb.OpenRecordset(table1, DB_OPEN_TABLE)


    For i = 0 To MyRecSet.Fields.Count - 1
        fname = MyRecSet.Fields(i).name
        If i = 0 Then
            sel = "INSERT INTO [" & temptable & "] SELECT [" & table1 & "].* FROM ["
             sel = sel & table1 & "] WHERE EXISTS (SELECT * FROM ["
            sel = sel & table2 & "] WHERE ([" & table1 & "].[" & fname & "] = ["
             sel = sel & table2 & "].[" & fname & "]"
        Else
            sel = sel & " AND [" & table1 & "].[" & fname & "] = [" & table2 & "].["
             sel = sel & fname & "]"
        End If
    Next i

    MyRecSet.Close

    sel = sel & "));"
```

```
    DoCmd RunSQL sel

    Call DeleteTable(table1)
    Call RenameTable(temptable, table1)

End Sub

Function ExecutenewRule (RuleName$, prefix$) As String
' Purpose: Execute referential rules
' Subroutines: Operate, ExecuteOp, EvaluateNewRule
' Warning: conditions are related by And if they are based on the same table


    On Error GoTo Err_ExecuteNewRule

    DoCmd Hourglass False
    DoCmd SetWarnings False


    Dim mydb As Database, MyQuery As QueryDef, myset As Recordset, MyRecSet As
Recordset
    Dim tset As Recordset, myset_tbl1 As Recordset, myset_tbl2 As Recordset
    Dim cnt%, level%, tblist$, dummy As Variant, tbl1$, tbl2$
    Dim TableSize%, x%, message$, i%, j%, sel$, lst$, rname$
    Dim timestamp$, ddate$, ttime$, ExceptList$, num%

    Set mydb = DBEngine.Workspaces(0).Databases(0)
    Set myset = mydb.OpenRecordset("DQ_NewRules", DB_OPEN_TABLE)      ' Open table.
    myset.Index = "PrimaryKey"    ' Select index.

    rname = RuleName & "." & cnt
    myset.Seek "=", rname
    num = myset!NumRule
    If Not IsNull(myset!ExceptList) Then
        ExceptList = myset!ExceptList
    Else
        ExceptList = ";"
    End If

    tbl1 = myset.SourceData
    Set myset_tbl1 = mydb.OpenRecordset(tbl1, DB_OPEN_TABLE)
    TableSize = myset_tbl1.RecordCount
    myset_tbl1.Close

    myset.Close

    vio_summary = ""

    tbl2 = EvaluateNewRule(RuleName, prefix)

    'Add record to DQ_NewRulesLog
    Set MyRecSet = mydb.OpenRecordset("DQ_NewRulesLog", DB_OPEN_TABLE)
    MyRecSet.AddNew
    MyRecSet.RuleName = RuleName
    i = InStr(1, prefix, "(")
    j = InStr(1, prefix, ")")
    x = InStr(1, prefix, " ")
    ddate = Mid$(prefix, x + 1, j - x - 1)
    ttime = Mid$(prefix, i + 1, x - i - 1)
    MyRecSet.Date = ddate
    MyRecSet.Time = ttime
    MyRecSet.SourceData = tbl1
    MyRecSet.ExceptList = ExceptList
    MyRecSet.TableSize = TableSize

    'tables(1) is a list of resulting violation tables
    Set myset_tbl2 = mydb.OpenRecordset(tbl2, DB_OPEN_TABLE)
```

```
    MyRecSet.NumViolation = myset_tbl2.RecordCount
    MyRecSet.Update


    ExecutenewRule = tbl2

    vio_summary = "Violations of rule '" & RuleName & "':" & Chr(10) & Chr(13)&
vio_summary
    vio_summary = vio_summary & "Overall # of violations: " & myset_tbl2.RecordCount

    If vio_summary <> "" Then
        MsgBox vio_summary, 64
    Else
        MsgBox "This rule contains no violation checks."
    End If


Exit_ExecuteNewRule:
    DoCmd Echo True
    DoCmd SetWarnings True
    DoCmd Hourglass False
    Exit Function

Err_ExecuteNewRule:
    MsgBox Error$, , "ExecuteRule"
    Resume Exit_ExecuteNewRule


End Function


Function ExecuteOp (op$, tables1$, tables2$, prefix$, level%) As String
' Purpose: Execute the operator op on tables1 and tables2 and return the
'          result in tables1. Op can be "And" or "Or"

    Dim result$, pos1%, pos2%, pos3%, pos4%
    Dim tbl1$, tbl2$

    result$ = ";"

    While tables1$ <> ";"
        pos1 = InStr(2, tables1$, ";")              'end of first table name in
tables1$
        tbl1 = Mid$(tables1$, 2, pos1 - 2)         'tbl1$ = first table name
        pos2 = Len(prefix$) + 1                     'start of 'level' in tbl1$
        pos3 = InStr(pos2, tbl1$, "_")              'end of 'level' in tbl1$

        'replace 'level' with 'level+1' to get the name of the table in tables2$
        tbl2 = prefix$ & (level + 1) & Mid$(tbl1$, pos3)
        pos4 = InStr(tables2$, ";" & tbl2$ & ";")    'start of tbl2$ in tables2$

        If pos4 = 0 Then                            'tbl2$ does not exist
            If op$ = "And" Then
                result$ = result$ & tbl1$ & ";"     'union = tbl1$
            Else
                DeleteTable (tbl1$)                 'intersection = nothing
            End If
        Else                                        'tbl2$ exists
            If op$ = "And" Then                     'perform the operation
                Call ExecuteUnion2(tbl1$, tbl2$)
            Else
                Call ExecuteIntersec2(tbl1$, tbl2$)
            End If
            result$ = result$ & tbl1$ & ";"         'add the modified tbl1$ to results

            'erase tbl2$ from tables2$
            tables2$ = Left$(tables2$, pos4) & Mid$(tables2$, pos4 + Len(tbl2$) + 2)
```

```
            DeleteTable (tbl2$)                         'Delete tbl2$
        End If
        tables1$ = Mid$(tables1$, pos1)                 'erase tbl1$ from tables1$
    Wend

    'now deal with the remaining tables in tables2$
    While tables2$ <> ";"
        pos1 = InStr(2, tables2$, ";")                  'end of first table name in
tables2$
        tbl2$ = Mid$(tables2$, 2, pos1 - 2)             'tbl2$ = first table name

        If op$ = "And" Then
            'union
            pos2 = Len(prefix$) + 1                     'start of 'level+1' in tbl2$
            pos3 = InStr(pos2, tbl2$, "_")              'end of 'level+1' in tbl2$
            'replace 'level+1' with 'level' to get the name of the table in level
            tbl1$ = prefix$ & level & Mid$(tbl2$, pos3)
            Call RenameTable(tbl2$, tbl1$)                  'switch to level from level+1
        Else
            'intersection
            Call DeleteTable(tbl2$)
        End If
        tables2$ = Mid$(tables2$, pos1)
    Wend

    ExecuteOp = result$

End Function

Sub ExecuteUnion2 (table1$, table2$)
' Purpose: Union table1 and table2 into table1
' Warning: table1 and table2 must have the same number and type of fields

    Dim mydb As Database, sel$, myset As Recordset, MyQuery As QueryDef
    Dim temptable$


    temptable = "DQ_tmp_" & table1$
    Set mydb = DBEngine.Workspaces(0).Databases(0)
    sel = "SELECT * FROM [" & table1$ & "] UNION SELECT * FROM [" & table2$ & "];"
    Set MyQuery = mydb.CreateQueryDef("QD_tempquery", sel)

    sel = "SELECT DISTINCTROW QD_tempquery.* INTO [" & temptable$ & "] FROM
QD_tempquery;"
    DoCmd RunSQL sel
    mydb.DeleteQueryDef "QD_tempquery"
    Call RenameTable(temptable$, table1$)

End Sub


Function get_field_list (table_name)
' Purpose: Obtain the fields of the table table_name

    On Error GoTo Err_Get_Field_List

    Dim mydb As Database, N%, m%, result$, fname$

    Set mydb = DBEngine.Workspaces(0).Databases(0)
    m = mydb.TableDefs(table_name).Fields.Count

    Do
        fname = mydb.TableDefs(table_name).Fields(N).name
        result = result & fname & ";"
        N = N + 1
    Loop Until N >= m
```

```
    get_field_list = result

Exit_Get_Field_List:
    DoCmd Echo True
    Exit Function

Err_Get_Field_List:
    MsgBox Err & " : " & Error, , "Get_Field_List"
    Resume Exit_Get_Field_List

End Function


Function get_rule_list (table_name) As String
' Purpose: Obtain the list of rules associated with this table


    Dim mydb As Database, myset As Recordset, criteria$, lst$

    Set mydb = DBEngine.Workspaces(0).Databases(0)
    Set myset = mydb.OpenRecordset("DQ_NewRules", DB_OPEN_DYNASET)

    criteria = "SourceData = '" & table_name & "'"
    myset.FindFirst criteria

    lst = ""

    Do Until myset.NoMatch
        lst = lst & myset!RuleName & ";"
        myset.FindNext criteria
    Loop
    myset.Close

    get_rule_list = lst
End Function


Function get_table_list ()
' Purpose: Return a list of tables in this database that does not begin
'          with "MSy" or "DQ_"; each table name is separated by a semi-colon.

    On Error GoTo Err_Get_Table_List

    Dim mydb As Database, MyTD As TableDef, N As Integer, result As String
    Dim tbname As String, m As Integer

    Set mydb = DBEngine.Workspaces(0).Databases(0)
    m = mydb.TableDefs.Count    'count of number of tables in db
    N = 0                       'intialize n
    Do
        tbname = mydb.TableDefs(N).name
        If Mid(tbname, 1, 3) <> "MSy" And Mid(tbname, 1, 3) <> "DQ_" And Mid(tbname,
1, 3) <> "~TM" Then
            result = result & tbname & ";"
        End If
        N = N + 1
    Loop Until N >= m

    get_table_list = result

Exit_Get_Table_List:
    Exit Function

Err_Get_Table_List:
    MsgBox Err & " : " & Error, , "Get_Table_List"
    Resume Exit_Get_Table_List
```

```
End Function

Function GetStringField (list$, Fnum%) As String
' Purpose: Return the field specified by Fnum in string list

    Dim x%, y%

    x = Position(1, list, Fnum - 1)
    y = InStr(x, list, ";")
    GetStringField = Mid$(list, x, y - x)

End Function


Function MatchString (list$, elt$) As Variant
' Purpose: To determine whether field elt is in string list.
'          Elements in list are separated by semicolons.

    Dim i As Integer, j As Integer, found As Variant, counter As Integer

    i = 1
    j = Len(list)
    found = False
    counter = 1

    Do Until (i > j)
       If GetStringField(list, counter) = elt Then
          found = True
          Exit Do
       Else
          counter = counter + 1
          i = Position(i, list, 1)
       End If
    Loop

    If found = True Then
       MatchString = True
    Else
       MatchString = False
    End If

End Function


Function Operate (myset As Recordset, prefix$, level%) As String
' Purpose: Operate the input filter on the sourcedata and input the
'          result into a table


    DoCmd Hourglass False
    DoCmd SetWarnings False

    Dim mydb As Database, MyQuery As QueryDef, myset_lst As Recordset
    Dim Filter$, source$, ftype%, fsize%, rname$, sel$, lst$, tname$, fname$
    Dim counter%, violation_count%, comp$, TableSize%, i%, j%, tbl1$
    Dim func$, compsource$, comptname$, compfname$
    Dim x%, y%, lsttname$, lstfname$, constant As Variant
    Dim multiplier As Double, fieldconst As Double, Negation As Variant

    rname = myset!RuleName
    Filter = myset!Filter
    x = InStr(1, Filter, " ")
    If x <> 0 Then     'Negation
       Negation = True
       Filter = Right$(Filter, Len(Filter) - x)
    Else
       Negation = False
```

```
    End If

    source = myset!SourceData
    i = InStr(1, source, ".")
    tname = Mid(source, 2, i - 3)
    fname = Mid(source, i + 2, Len(source) - i - 2)
    lsttname = tname
    lstfname = fname

    Do
        x = InStr(lsttname, " ")
        If x > 1 Then
            lsttname = Left$(lsttname, x - 1) & "_" & Mid$(lsttname, x + 1)
        End If
    Loop Until (x = 0)

    lst = prefix$ & level% & "_" & lsttname
    Operate = lst

    Set mydb = DBEngine.Workspaces(0).Databases(0)
    If Filter <> "CompositeRule" Then
        ftype = mydb.TableDefs(tname).Fields(fname).type
        fsize = mydb.TableDefs(tname).Fields(fname).size
    End If

    sel = "SELECT DISTINCTROW [" & tname & "].* INTO [" & lst
    sel = sel & "] FROM [" & tname & "] WHERE (((0)=1));"
    DoCmd RunSQL sel
    Set myset_lst = mydb.OpenRecordset(lst, DB_OPEN_TABLE)


    Select Case Filter
       Case "Uniqueness"
         If Negation = False Then
            sel = "SELECT DISTINCTROW " & source & ", Count(" & source
            sel = sel & ") AS CountOfUnique FROM [" & tname & "] GROUP BY "
            sel = sel & source & "HAVING (((Count(" & source & "))>1));"
            Set MyQuery = mydb.CreateQueryDef("QD_temp_subunique", sel)
            sel = "SELECT DISTINCTROW [" & tname & "].* FROM QD_temp_subunique INNER
JOIN ["
            sel = sel & tname & "] ON QD_temp_subunique.[" & fname & "] = " & source
             sel = sel & ";"
         Else
            sel = "SELECT DISTINCTROW [" & tname & "].* FROM [" & tname & "]"
         End If

         sel = "INSERT INTO [" & lst & "]" & sel
         DoCmd RunSQL sel
         violation_count = myset_lst.RecordCount
         mydb.DeleteQueryDef "QD_temp_subunique"

         vio_summary = vio_summary & "  # of Uniqueness violations in " & source & ":
" & violation_count & Chr(13) & Chr(10)

       Case "Null Value"

            sel = "SELECT DISTINCTROW [" & tname & "].* FROM ["
         If Negation = False Then
            sel = sel & tname & "] WHERE ((" & source & " Is Null));"
         Else
            sel = sel & tname & "] WHERE (Not(" & source & " Is Null));"
         End If

         sel = "INSERT INTO [" & lst & "]" & sel
         DoCmd RunSQL sel
         violation_count = myset_lst.RecordCount
```

```
        vio_summary = vio_summary & "  # of Null Value violations in " & source & ":
" & violation_count & Chr(13) & Chr(10)

      Case "Zero Value"
        If ftype > 0 And ftype < 8 Then
            sel = "SELECT DISTINCTROW [" & tname & "].* FROM ["
            If Negation = False Then
                sel = sel & tname & "] WHERE ((" & source & " = 0));"
            Else
                sel = sel & tname & "] WHERE ((" & source & " <> 0));"
            End If

            sel = "INSERT INTO [" & lst & "]" & sel
            DoCmd RunSQL sel
            violation_count = myset_lst.RecordCount
            vio_summary = vio_summary & "  # of Zero Value violations in " & source &
": " & violation_count & Chr(13) & Chr(10)
        Else
            MsgBox "The field " & source & " is not a number field.  Zero value
testing will be skipped."
        End If

      Case "Format"
        ftype = mydb.TableDefs(tname).Fields(fname).type
        fsize = mydb.TableDefs(tname).Fields(fname).size

        If (ftype = 10) Or (ftype = 12) Then
            sel = "SELECT DISTINCTROW [" & tname & "].* FROM ["
            If Negation = False Then
                sel = sel & tname & "] WHERE (Not(" & source & myset!FormatComp
                sel = sel & myset!FormatSize & " ));"
            Else
                sel = sel & tname & "] WHERE ((" & source & myset!FormatComp
                sel = sel & myset!FormatSize & " ));"
            End If

            sel = "INSERT INTO [" & lst & "]" & sel
            DoCmd RunSQL sel
            violation_count = myset_lst.RecordCount
            vio_summary = vio_summary & "  # of Zero Value violations in " & source &
": " & violation_count & Chr(13) & Chr(10)
        Else
            MsgBox "The field " & source & " is not a string field.  Format testing
will be skipped."
        End If

      Case "FormatRestriction"
        If ftype > 0 And ftype < 8 Then
            sel = "SELECT DISTINCTROW [" & tname & "].* FROM ["
            If Negation = False Then
                sel = sel & tname & "] WHERE (Not((" & source
            Else
                sel = sel & tname & "] WHERE (((" & source
            End If

            If (Not (IsNull(myset!LowerRange)) And Not (IsNull(myset!LowerRange)))
Then
                sel = sel & " >= " & myset!LowerRange & ") And (" & source & " <= "
                sel = sel & myset!UpperRange & ")));"
            ElseIf (Not (IsNull(myset!LowerRange))) Then
                 sel = sel & " >= " & myset!LowerRange & ")));"
            Else
                 sel = sel & " <= " & myset!UpperRange & ")));"
            End If

            sel = "INSERT INTO [" & lst & "]" & sel
            DoCmd RunSQL sel
```

```
            violation_count = myset_lst.RecordCount

            vio_summary = vio_summary & "  # of Zero Value violations in " & source &
": " & violation_count & Chr(13) & Chr(10)
        Else
            MsgBox "The field " & source & " is not a number field.  Zero value
testing will be skipped."
        End If


      Case "CompConstant"
        If (ftype > 0 And ftype <= 8) Or (ftype = 10 Or ftype = 12) Then
            comp = myset!Compare1
            If ftype > 0 And ftype < 8 Then
                constant = CDbl(myset!constant)
                sel = "SELECT DISTINCTROW [" & tname & "].* FROM ["
                If Negation = False Then
                    sel = sel & tname & "] WHERE (NOT(" & source & comp & constant
                    sel = sel & "));"
                Else
                    sel = sel & tname & "] WHERE ((" & source & comp & constant
                    sel = sel & "));"
                End If
            ElseIf (ftype = 8) Then
                constant = CStr(myset!constant)
                sel = "SELECT DISTINCTROW [" & tname & "].* FROM ["
                If Negation = False Then
                    sel = sel & tname & "] WHERE (NOT(" & source & comp & "#"
                    sel = sel & constant & "#));"
                Else
                    sel = sel & tname & "] WHERE ((" & source & comp & "#" & constant
                    sel = sel & "#));"
                End If
            Else
                constant = CStr(myset!constant)
                sel = "SELECT DISTINCTROW [" & tname & "].* FROM ["
                If Negation = False Then
                    sel = sel & tname & "] WHERE (NOT(" & source & comp & """
                    sel = sel & constant & """));"
                Else
                    sel = sel & tname & "] WHERE ((" & source & comp & """ & constant
                    sel = sel & """));"
                End If
            End If

            sel = "INSERT INTO [" & lst & "]" & sel
            DoCmd RunSQL sel
            violation_count = myset_lst.RecordCount
            vio_summary = vio_summary & "  # of CompConstant violations in " & source
& ": " & violation_count & Chr(13) & Chr(10)
        Else
            MsgBox "The field " & source & " is not a number or date field.  Zero
value testing will be skipped."
        End If

      Case "CompField"
        If (ftype > 0 And ftype < 8) Or (ftype = 8) Then
            comp = myset!Compare2
            compsource = myset!compsource
            multiplier = myset!multiplier
            fieldconst = myset!FieldConstant
            i = InStr(1, compsource, ".")
            comptname = Mid(compsource, 2, i - 3)
            compfname = Mid(compsource, i + 2, Len(compsource) - i - 2)

            If IsNull(myset!Function) Then
               sel = "SELECT DISTINCTROW [" & tname & "].* FROM ["
```

```
                If Negation = False Then
                     sel = sel & tname & "] Where NOT ([" & tname & "].[" & fname &"]"
                     sel = sel & comp
                Else
                     sel = sel & tname & "] Where ([" & tname & "].[" & fname & "] "
                     sel = sel & comp
                End If
                sel = sel & " " & multiplier & "*[" & tname & "].[" & compfname & "]+("
                sel = sel & fieldconst & "));"

                sel = "INSERT INTO [" & lst & "]" & sel
                DoCmd RunSQL sel
            Else
                func = myset!Function
                sel = "SELECT DISTINCTROW [" & tname & "].* FROM ["
                If Negation = False Then
                     sel = sel & tname & "] WHERE NOT (" & source & " " & comp
                     sel = sel & " (SELECT "
                Else
                     sel = sel & tname & "] WHERE (" & source & " " & comp
                     sel = sel & " (SELECT "
                End If
                sel = sel & multiplier & "*" & func & "(" & compsource & ")+("
                sel = sel & fieldconst & ") FROM [" & comptname & "]));"

                sel = "INSERT INTO [" & lst & "]" & sel
                DoCmd RunSQL sel
            End If

            violation_count = myset_lst.RecordCount
            vio_summary = vio_summary & "  # of CompField violations in " & source &
": " & violation_count & Chr(13) & Chr(10)
        End If

    Case "IsIn"
        compsource = myset!compsource
        i = InStr(1, compsource, ".")
        comptname = Mid(compsource, 2, i - 3)
        compfname = Mid(compsource, i + 2, Len(compsource) - i - 2)

        sel = "SELECT DISTINCTROW [" & tname & "].* FROM ["
        If Negation = False Then
            sel = sel & tname & "] Where NOT EXISTS (SELECT * FROM [" & comptname
        Else
            sel = sel & tname & "] Where EXISTS (SELECT * FROM [" & comptname
        End If
        sel = sel & "] Where [" & comptname & "].[" & compfname & "] = ["
        sel = sel & tname & "].[" & fname & "]);"

        sel = "INSERT INTO [" & lst & "]" & sel
        DoCmd RunSQL sel
        violation_count = myset_lst.RecordCount
        vio_summary = vio_summary & "  # of IsIn violations in " & source & ": " &
violation_count & Chr(13) & Chr(10)

    Case "CompositeRule"
        tbl1$ = EvaluateNewRule(CStr(myset!CompositeRule), prefix$ & level% & "_")

        If tbl1$ = "" Then
            MsgBox ("Rule '" & myset!CompositeRule & "' was not found!")
        Else
            myset_lst.Close
            Call RenameTable(tbl1$, lst)
            Set myset_lst = mydb.OpenRecordset(lst, DB_OPEN_TABLE)
        End If

    End Select
```

```
    myset_lst.Close
    Operate = lst
End Function


Function Position (start%, s$, i%) As Integer
' Purpose: Return the position of the (i+1) field in string s

  Dim j%, k%

  k = start
  For j = 1 To i
   k = InStr(k, s, ";") + 1
  Next j

  Position = k

End Function

Sub RenameTable (oldtable$, newtable$)
' Purpose: Rename table oldtable to newtable

    DoCmd SetWarnings False

    Dim sel$

    sel = "SELECT [" & oldtable & "].* INTO [" & newtable & "] FROM [" & oldtable
    sel = sel & "];"
    DoCmd RunSQL sel
    Call DeleteTable(oldtable)

End Sub

Sub RuleReport (tblname$, RuleName$, ddate$, ttime$)
' Purpose: Create a report based on table tblename

    On Error GoTo Err_RuleReport

    DoCmd SetWarnings False
    DoCmd Hourglass False

    Dim mydb As Database, MyQuery As QueryDef, myset As Recordset, R As Report
    Dim sel As String, y%, x%, Count%, F As Form, table_name$, dummy As Variant
    Dim ExceptList$, ExceptArray() As String, myrec As Recordset

    Set mydb = DBEngine.Workspaces(0).Databases(0)
    sel = "SELECT DQ_NewRulesLog.* From DQ_NewRulesLog Where (DQ_NewRulesLog.RuleName
= """
    sel = sel & RuleName & """ And DQ_NewRulesLog.Date = """ & ddate & """ And
DQ_NewRulesLog.Time = """
    sel = sel & ttime & """);"
    Set MyQuery = mydb.CreateQueryDef("QD_temp_RuleReport_header", sel)

    Set myrec = MyQuery.OpenRecordset(DB_OPEN_DYNASET)
    myrec.MoveFirst
    ExceptList = myrec.ExceptList

    If ExceptList = ";" Then
        dummy = Createfrm1(tblname$)
    Else
        x = 0
        Count = 0
        Do
            Count = Count + 1
```

```
            x = InStr(x + 1, ExceptList, ";")
        Loop While x <> 0

        ReDim ExceptArray(Count - 2)
        For x = 0 To Count - 2
            ExceptArray(x) = GetStringField(ExceptList, x + 1)
        Next x
        dummy = Createfrm2(tblname, ExceptArray())
    End If

    On Error GoTo Err_RuleReport
    SendKeys "DQ_temp_SubRuleform{enter}", False
    DoCmd DoMenuItem 3, a_file, a_saveformas, , a_menu_ver20
    DoCmd Close A_FORM, "DQ_temp_SubRuleform"
    DoCmd OpenReport "DQ_report", a_preview
    mydb.DeleteQueryDef "QD_temp_RuleReport_header"
    DoCmd DeleteObject A_FORM, "DQ_temp_SubRuleform"
    DoCmd SelectObject a_report, "DQ_report", False

Exit_RuleReport:
    DoCmd Echo True
    DoCmd SetWarnings True
    DoCmd Hourglass False
    Exit Sub

Err_RuleReport:
    MsgBox Error$, , "RuleReport"
    Resume Exit_RuleReport

End Sub

Function SetStringField (list$, fld$, Fnum%) As String
' Purpose: Change the field specified by Fnum in string list to fld

    Dim x%, y%

    x = Position(1, list, Fnum - 1)
    y = InStr(x, list, ";")
    SetStringField = Mid$(list, 1, x - 1) & fld & Mid$(list, y)

End Function
```

7398-47