2013

The Use of Relation Valued Attributes in Support of Fuzzy Data

Larry Ritchie Williams Jr.
Virginia Commonwealth University

THE USE OF RELATION VALUED ATTRIBUTES IN SUPPORT OF FUZZY DATA

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

LARRY R. WILLIAMS, JR.
Bachelor of Arts, History, Virginia Military Institute, 1982
Master of Science, Systems Management, Viterbi School of Engineering, University of Southern California, 1988
Doctor of Philosophy, Computer Science, Virginia Commonwealth University, 2013

Director: LORRAINE M. PARKER, Ph.D.
ASSOCIATE PROFESSOR, DEPARTMENT OF COMPUTER SCIENCE

Virginia Commonwealth University
Richmond, Virginia
2013

Acknowledgements

With Special Thanks and Appreciation to

My wife Lynne J. Williams whose gentle patience during my frequent far off stares and quiet support throughout the years would have tested anyone's patience. "You're thinking about it again, aren't you?"

Dr. Lorraine M. Parker whose extraordinary guidance and support through out this lengthy, involved and time consuming process has been instrumental in returning purpose to a young Cadet's life.

My research partner Dr. Marion 'Bob' Morrissett '13, the veteran graduate student who contributed so much to my research and to whose advice I should have paid far more attention.

My dissertation advisory committee for their significant time, contributions and guidance through the course of my research.

-- And, of course --

To my ‘Phone a Friend’

Charles A. Bell, Ph.D. 2005
Senior Software Developer
Team Lead MySQL Utilities
Oracle Corporated

Who always answered.

Table of Contents

Acknowledgements i

Table of Contents i

Abstract 9

Chapter 1 - Introduction 11

1.1 Objective Statement 11

1.2 Overview of the Approach 13

1.3 Contribution 14

Chapter 2 - Foundation Principles of Fuzzy Set Theory 15

2.1 Background 15

2.1.1 Logic 15

2.1.2 Law of Identity 15

2.1.3 Law of Contradiction 15

2.1.4 Law of Excluded Middle 16

2.2 Crisp Sets Compared to Fuzzy Sets 16

2.2.1 The Universe of Bottles using Fuzzy Sets 17

2.3 Relational Database Model 19

2.4 Fuzzy Database Model 19

2.5 Fuzzy Concepts 21

2.5.1 Crisp and Fuzzy Sets 21

2.5.2 Universe of Discourse 21

2.5.3 Crisp Set Membership 21

2.5.4 Fuzzy Set Membership 22

2.6 Fuzzy Set Operations 22
2.6.1 Equality 22
2.6.2 Inclusion 22
2.6.3 Complement 23
2.6.4 Intersection 23
2.6.5 Union 23
2.7 T-norm and T-conorm 24
2.7.1 T-norm 24
2.7.2 T-conorm 25
2.8 The Intrinsic Nature of Fuzzy Sets 26
2.8.1 Law of Contradiction 27
2.8.2 Law of Excluded Middle 27
2.9 Representations Using Fuzzy Sets 27
2.10 Linguistic Labels 28
2.10.1 Convexity 28
2.10.2 Height 28
2.10.3 Normal 29
2.10.4 α-cut and Strong α-cut 29
2.10.5 Support 30
2.10.6 Kernel 30
2.10.7 Linguistic Modifiers 30
2.10.8 Alternative to Linguistic Modifiers 34
2.11 Possibility Theory 36
2.12 Vocabulary of Fuzziness 38
2.12.1 Imprecise and Vague 40

2.12.2 Subjective and Unclear 41
2.12.3 Ambiguous and Uncertain 41
2.12.4 Incomplete and Inconsistent 42
Chapter 3 - Relational Database Systems 43
3.1 Problems Inherent to Relational Database Systems 43
3.1.1 Relvar Data Type 43
3.1.2 Attribute Data Type 43
3.2 Fuzzy Relational Database 44
3.2.1 Extending RDB with Fuzzy Types 45
3.2.2 Representation of Unknown and Uncertain Data 45
3.2.3 A Fuzzy Weight Attribute 46
3.2.4 Fuzzy Data Categories 46
3.3 Setting Expectations 47
3.4 Four Kinds of Database Update 49
3.4.1 Crisp Data to Crisp Domain 50
3.4.2 Fuzzy Data to Crisp Domain 51
3.4.3 Crisp Data to Fuzzy Domain 52
3.4.4 Fuzzy Data to Fuzzy Domain 55
3.5 Four Kinds of Database Query 59
3.5.1 Crisp Query of Crisp Data 60
3.5.2 Fuzzy Query of Crisp Data 60
3.5.3 Crisp Query of Fuzzy Data 63
3.5.4 Fuzzy Query of Fuzzy Data 67
Chapter 4 - Modeling Fuzzy Relations 69
4.1 Overview of Entity Relationship Modeling 69

4.1.1 Entity or Entity Set 69
4.1.2 Weak Entities 70
4.1.3 Relationship or Relationship Set 70
4.1.4 Weak Relationships 70
4.1.5 Attributes 70
4.1.6 Derived Attribute 71
4.1.7 Functionality 71
4.1.8 Entity Relationship Component Representation 72
4.2 Extending the Entity Relationship Diagram 73
4.2.1 Fuzzy Entity Relationship Considerations 74
4.3 Aggregation 83
4.3.1 Aggregation in Extended Entity Relationship Models 83
4.3.2 Extending Aggregation to Accommodate Fuzzy Data 84
4.4 Relation Valued Attributes 85
4.4.1 Crisp RVA 85
4.4.2 Using RVA's to Represent Fuzzy Data 87
4.4.3 Fuzzy Attributes as Relations 90
Chapter 5 - RVA as a Constraint to the Representation of Fuzzy Data 94
5.1 Introduction 94
5.2 Design Considerations 95
5.2.1 Metadata Considerations 95
5.2.2 Creation of a Nesting Function Between Tables 105
5.3 Design Features of the RVAChar 109
5.3.1 The 'How' and 'When' of it 109
5.3.2 The Knowledge Contained Within the RVAChar 109

5.3.3 How the 'Knowledge' Works 110
5.4 Maintenance Benefits of the 'Knowledge' Approach 113
5.4.1 Create Table 114
5.4.2 Drop Table 114
5.4.3 Add, Delete and Update Fuzzy Data Values 115
5.4.4 Modifying the Knowledge 115
Chapter 6 - Implementation of a Relation Valued Attribute in MySQL 116
6.1 Steps to Implementation 116
6.1.1 Implementation Considerations 116
6.1.2 Steps to Implementation 117
6.1.3 Modifying the MySQL Source Code 119
6.1.4 Distributing the Data Type through the Server Processes 120
Chapter 7 - Validation 142
7.1 The Approach to Validation 142
7.2 Data Used in Validation 143
7.2.1 Cartesian Product 145
7.2.2 Restrict 148
7.2.3 Project 150
7.2.4 Join 152
7.2.5 Intersection 156
7.2.6 Non-RVA Tests 158
7.3 Metrics 159
7.3.1 Accuracy 159
7.3.2 Completeness 160
7.4 Conclusions 161

Chapter 8 - Non-Fuzzy Application 162
8.1 Benefit of an RVA in a Non-Fuzzy Application 162
Chapter 9 - Representation of Fuzzy Data 165
9.1 Types of RVA Representation 165
9.1.1 Standard Grid Representation 166
9.1.2 Grouped Grid Representation 167
Chapter 10 - Formal System Study 168
10.1 Overview 168
10.2 Preparation 168
10.2.1 Application to the Institutional Review Board 169
10.2.2 Recruitment 169
10.2.3 Preparation 169
10.2.4 Heuristic Tests 170
10.2.5 Training 172
10.2.6 Overview of Fuzzy Data and the Role of the RVA 172
10.2.7 Working Systems 172
10.3 Study Results 173
10.3.1 System Functionality 173
10.3.2 Adherence to the Relational Model 173
10.3.3 User Acceptance 174
10.3.4 Enforced Constraint on Ordered Pair 174
10.3.5 RVA Representation 175
10.3.6 Consistency of Use 178
10.3.7 Use of RVA's Outside of Fuzzy Data 179
Chapter 11 - Conclusion 182

11.1 Restating the Problem 182
11.2 The Use of 'Knowledge' 183
11.3 The *build_query()* Method 183
11.4 Not Just For Fuzzy Data 184
11.5 Conclusions 184
Chapter 12 - Future Research 186
12.1 Introduction 186
1.1 Use of Fuzzy Data Equality in SQL Operations 186
1.2 Representation of a Relation Valued Attribute 188
1.3 Use of a Derived Fuzzy Value in an RVA 189
1.4 Incorporation of Ambiguity in Database Systems 190
1.5 Sensitivity Analysis 190
1.6 Beneficial Enhancements to the Man-Machine Interface 191
1.7 Nested RVA's 192
Appendix A – The Database Management System 195
Appendix B – Overview of Issues Concerning Data Types 196
1 Representation of Unknown or Missing Data 196
2 Attribute Domains and Domain Types 196
Appendix C – The *val_str()* Method 198
Appendix D – The *build_query()* Method 202
Appendix E – The Test Script 207
Appendix F – System Study Presentation Slides 212
Appendix G – Vita 223

This Page Intentionally Left Blank

Abstract

THE USE OF RELATION VALUED ATTRIBUTES IN SUPPORT OF FUZZY DATA

By Larry R. Williams, Jr., Ph.D. Student

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2013

Director: Lorraine M. Parker, Ph.D.
Associate Professor, Department of Computer Science

In his paper introducing fuzzy sets, L.A. Zadeh describes the difficulty of assigning some real-world objects to a particular class when the notion of class membership is ambiguous. If exact classification is not obvious, most people approximate using intuition and may reach agreement by placing an object in more than one class. Numbers or 'degrees of membership' within these classes are used to provide an approximation that supports this intuitive process. This results in a 'fuzzy set'. This fuzzy set consists any number of ordered pairs to represent both the class and the class's degree of membership to provide a formal representation that can be used to model this process.[2]

Although the fuzzy approach to reasoning and classification makes sense, it does not comply with two of the basic principles of classical logic.[3,p.36] These principles are the laws of contradiction and excluded middle. While they play a significant role in logic, it is the violation of these principles that gives fuzzy logic its useful characteristics.[4]

The problem of this representation within a database system, however, is that the class and its degree of membership are represented by two separate, but indivisible attributes. Further, this representation may contain any number of such pairs of attributes. While the data for class and membership are maintained in individual attributes, neither of these attributes may exist without the other without sacrificing meaning. And, to maintain a variable number of such pairs within the representation is problematic. C. J. Date suggested a relation valued attribute (RVA)[5] which can not only encapsulate the attributes associated with the fuzzy set and impose constraints on their use, but also provide a relation which may contain any number of such pairs.

The goal of this dissertation is to establish a context in which the relational database model can be extended through the implementation of an RVA to support of fuzzy data on an actual system. This goal represents an opportunity to study through application and observation, the use of fuzzy sets to support imprecise and uncertain data using database queries which appropriately adhere to the relational model. The intent is to create a pathway that may extend the support of database applications that need fuzzy logic and/or fuzzy data.

Chapter 1 - Introduction

1.1 Objective Statement

A fuzzy data value is represented by a set consisting of ordered pairs of values (c,w)[2]. The first member of this pair c represents the data's *class* consisting of any number of enumerated characteristic values within a domain such as *haircolor* where C = *haircolor* = {'blonde', 'brown', 'black', 'red', 'grey'}. As such, $c \in C$. The other value, w, is in the interval $W = [0,1]$ which is the fuzzy data value's *weight* or *degree of membership* and indicates the membership associated with c. $w = 0$ indicates that the fuzzy data has no degree of membership to c. $w = 1$ indicates that the fuzzy data has total membership with c. Any other value represents the degree to which the data can be said to be a member of c.

Each part of (c,w), however, is indivisible from the other. A *class* without an associated *degree of membership* is meaningless just as a *degree of membership* without an associated *class* is meaningless. Further, a fuzzy data value consists of any number of such pairs. To represent such a fuzzy data value in a database it is necessary to represent (c,w) in such a way that neither c nor w can be retrieved without the other. Further, it is necessary that a system consisting of fuzzy data provide the means to support any number of such pairs, (c,w) for an associated object. Figure 1.1 illustrates a possible fuzzy data value for an individual's hair color using the values using the domains C and W described previously:

```
Fuzzy Data Value = { ('Brown', 0.80),
                     ('Blonde', 0.75),
                     ('Red', 0.55)}
```

Figure 1.1 - Fuzzy Data Value for Hair Color

It can be seen that the fuzzy data value in Figure 1.1 is composed of three ordered pairs which, when seen from the perspective of the relational model, can be viewed as a relation comprised of two attributes from the domains *C* and *W* and three tuples. This value represents the ambiguous nature associated with the color of an individual's hair which can be said, to the various degrees specified, to be 'Brown', 'Blonde' and/or 'Red'.

Current database management systems cannot enforce a constraint to ensure the requisite indivisibility of the ordered pair associated with the fuzzy data value. Nor can they accommodate the variable number of such pairs potentially contained within the value. Within the confines of such systems the number of pairs is therefore limited and either member of (*c*,*w*) can easily be excluded by a query rendering the fuzzy data value limited and meaningless.

By way of a solution to this problem, this dissertation presents two primary accomplishments. The first is the design, development and validation of a new data type, a Relation Valued Attribute (RVA), that can be included in the relational database management system MySQL to support fuzzy data values. The RVA provides a constraint that is used to enforce the atomicity of (*c*,*w*) in the representation of a fuzzy data value. As a relation, an RVA also provides the benefit of allowing a fuzzy data value to have any number of such pairs. Secondly, the dissertation describes the engagement of a panel of users to evaluate this new system with respect to the use, costs and benefits of using an RVA to represent the fuzzy data values. This research effort contributes to a greater understanding of the benefits and use of relation valued attributes in support of fuzzy data.

1.2 Overview of the Approach

These accomplishments required a modification to the relational database server provided by the open source database management system (DBMS) MySQL be designed, implemented and validated. This design provides the ability to use and maintain an RVA to support the encapsulation of a complex fuzzy data value. The new RVA data type, for the first time, provides a constraint that ensures the atomicity of (*c*,*w*) which is necessary to the use and representation of fuzzy data within the relational model.

The system, as designed and implemented, demonstrated the feasibility of crisp data to coexist, in every respect, with fuzzy data. The system:

1. Appropriately maintains the relational model.
2. Does not affect existing or 'normal' relational functionality.
3. Appropriately supports the principle elements of relational algebra, specifically, the Cartesian Product as well as the Project, Restrict, Intersect and Join operations.[7]
4. Enforces a constraint on the atomicity of the ordered pair associated with fuzzy data.
5. Provides the flexibility to work not just with fuzzy data, but with nested relations of any size and cardinality.
6. Uses standard SQL syntax to obtain a result whether the query accesses crisp or fuzzy data.
7. Enforces the constraint on the representation of the fuzzy data value while also allowing the underlying supporting data to be accessed for modification or update.

1.3 Contribution

The unique contribution made as a result of this research is that for the first time, a relation valued attribute has been designed and implemented in an RDMS to support both crisp and fuzzy data values. The new data type not only provides a constraint that guarantees the atomicity of the ordered pair, but also allows the database administrator access to the underlying fuzzy data table for modification and update. This new system has been used to validate and study RVA's as an approach to the support of fuzzy data within the relational model.

The new system also allowed us to test the reactions and acceptance of users with respect to fuzzy data as well as see how quickly they came to adapt to the new concept.

Chapter 2 -Foundation Principles of Fuzzy Set Theory

2.1 Background

The relational data model with its strict application and adherence to two valued logic, recognized data dependencies, normalized decompositions and straight forward query language has, for many years, provided practitioners with an extremely logical and well-ordered representation of their data environment. The concepts inherent to fuzzy set theory can be used to expand this well ordered environment to reflect a more uncertain real world while still maintaining some sense of order.

2.1.1 Logic

The laws of identity, contradiction, and excluded middle form the basis of 2-valued logic in which statements must be either true or false within the context a particular logical argument.[5] These laws can be expressed using classical logic and classical or 'crisp' set theory.

2.1.2 Law of Identity

The law of identity states that 'whatever is, is' meaning a proposition is equivalent to itself and likewise a real-world object is identical to itself (i.e. both $(P \Leftrightarrow P)$ and $A = A$ are true). [5] Fuzzy set theory is not in conflict with this law and a fuzzy object is always itself.

2.1.3 Law of Contradiction

The law of contradiction states "nothing can both be and not be" meaning a proposition cannot be both true and false at the same time, within the same context (i.e. both $\neg(P \wedge \neg P)$ and $A \cap \bar{A} = \emptyset$ are true). [5] 2-valued logic does not allow a real-world object to be placed in more than one class at a time. It is not possible for the same bottle to be both empty and not empty at the same time. Fuzzy set theory allows a real-world object to have membership in one or more classes. A single bottle may be judged to be empty to some extent and not empty to some extent.

2.1.4 Law of Excluded Middle

The law of excluded middle states ‘everything must either be or not be’ meaning either a proposition is true or its opposite is true; both cannot be true (i.e. both ($P \vee \neg P$) and $A \cup \bar{A} = U$, are true). [5] 2-valued logic allows a real-world object to be in a single class and to only have the traits of this class. A bottle is full or not full, empty or not empty. Fuzzy set theory allows a real-world object to have partial membership in one or more classes and to have the traits of each class in varying degrees.

2.2 Crisp Sets Compared to Fuzzy Sets

Zadeh describes crisp set membership as ‘precise’ and fuzzy set membership as ‘imprecise’.[2,p.338] Crisp sets classify an object as either in the set or not in the set. A characteristic function evaluates an object's set membership and returns 1 if the object is in the set or 0 if the object is not in the set. Fuzzy sets allow classification of objects using a varying scale of set membership. A membership function returns a value from the interval [0,1] to indicate the degree of membership. This allows an object to have partial membership in one or more sets. Zadeh is clear that fuzzy set degree of membership is not the probability of the object being present in the fuzzy set, but rather it is the extent to which the object actually belongs to the classification represented by the set.[2,pp.339-40][6] Degree of membership allows the fuzzy set to be imprecise. Because of this imprecision, there is ambiguity in interpreting the meaning of degree of membership.

The imprecision of fuzzy sets can originate from more than one source during data entry and/or data querying. There may be uncertainty about the accuracy of the data used as the value for a fuzzy set element. The classification may be an opinion or a subjective point of view. The

criteria used to search a fuzzy set may be vague. The person searching may not have a clear understanding of match criteria or may not be aware that a fuzzy set represents a category of great breadth. In these cases, a data element's degree of fuzzy set membership provides a measure of breadth and depth for its classification and retrieval.

If there is a need for fuzzy sets, the meaning of a real-world object's membership in one or more of these sets must be clearly established and stated as a requirement of the implementation. This is included in documentation to set the scope of the user's expectations. Once the semantics of a fuzzy set are clearly stated, the benefits of intuitive understanding are available to those using the set.

2.2.1 The Universe of Bottles using Fuzzy Sets

In the universe of bottles, assume there are three fuzzy sets that describe the fullness or emptiness of bottles. Each of these sets is a fuzzy classification. It is possible for a bottle to have some degree of membership in one or more of these classes depending on the content of the bottle.

The membership function for the fuzzy set of empty 20 ounce bottles, as shown in Equation (2.1), is plotted over the scale of bottle contents in fluid ounces with two other fuzzy classes in Figure 2.1. The empty bottle degree of membership at 0 ounces is 1, at 10 ounces it is 0.5, and at 20 ounces it is 0.

$$\text{empty}(x) = \begin{cases} (20 - x) / 20 & \text{when } 0 \leq x \leq 20 \end{cases} \qquad (2.1)$$

The membership function for the fuzzy set of half-full bottles is given in Equation (2.2).

$$\text{half\_full}(x) = \begin{cases} 0 & \text{when } x = 0 \\ x / 10 & \text{when } 0 < x \leq 10 \\ (20 - x) / 10 & \text{when } 10 < x < 20 \\ 0 & \text{when } x = 20 \end{cases} \quad (2.2)$$

The membership function for the fuzzy set of full bottles is given in Equation (2.3).

$$\text{full}(x) = \begin{cases} x / 20 & \text{when } 0 \leq x \leq 20 \end{cases} \quad (2.3)$$

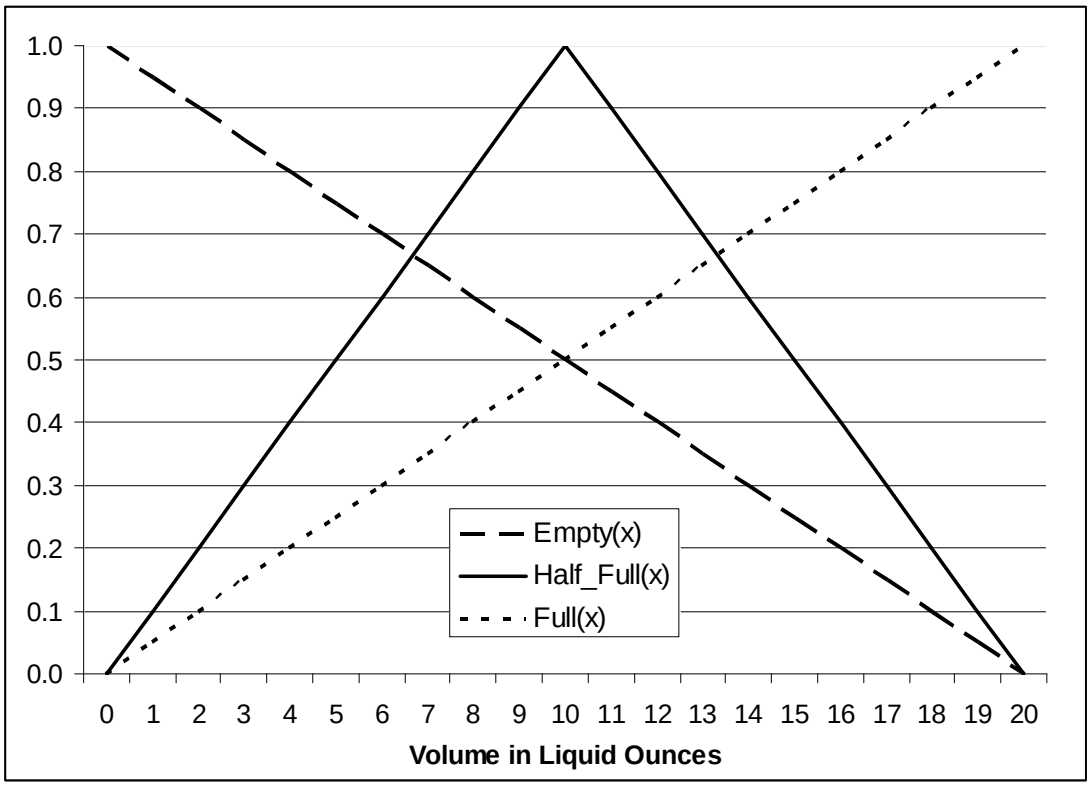


Figure 2.1: Fuzzy Sets (Universe of 20 oz. bottles: empty, half−full, full)

These three membership functions each define a fuzzy set that intuitively makes sense. This makes it possible to describe a bottle that contains 5 fluid ounces as 'partially full' and 'almost empty' or 'not very full' based its contents. In precise terms, it means that the membership degree of a bottle that contains 9 fluid ounces is 0.55 in a fuzzy set defined by the function empty, 0.9 in a fuzzy set defined by the function half-full, and 0.45 in a fuzzy set defined by the

function full. While the fuzzy sets defined by these three functions provide an intuitive model of bottles, the actual data used as arguments to these functions is from a crisp set VOLUME which contains the number of fluid ounces in each bottle.

2.3 Relational Database Model

E.F. Codd based the relational model on first-order predicate logic.[7] In the relational model the database is a collection of predicates over a finite set of predicate variables and the data is represented by n-ary tuples created from the Cartesian product of these n sets. This mathematical model of data makes it possible to create a consistent collection of information and to perform operations using relational algebra and relational calculus.[7]

Relational databases are collections of relational variables (relvars) that present the user with tables (relations) of data values in columns (attributes) and rows (tuples). A relation is composed of two parts. The first is the heading, which is a set of attribute names and domain type pairs. The heading is a predicate or truth-valued function in which the attribute names represent a set of parameters that range over the attribute domains of the specified relation. The second part of the relation is its body, which is a set of tuples. Each of these tuples is a proposition, which is true for its attribute values within the database.[8,pp.67-68] A tuple not present in the database is a proposition which is false. Thus a query which returns an empty result relation, shows no data that can indicate a true proposition.

2.4 Fuzzy Database Model

Support for fuzzy logic can be added to the relational model by the addition of fuzzy attributes. To do this, a domain is taken as the universe of discourse for a fuzzy set. A fuzzy attribute may

be of the same type as any crisp attribute with the addition of a degree of membership, called the *membership weight*.

Each tuple in the body of a query results relation is a proposition and true to the extent determined by the combined membership weights of its attributes. A crisp attribute value found in a relation is true with a weight of 1. A crisp attribute value not found is false with a weight of 0. Fuzzy attributes are true with a membership weight in the interval (0,1]. A fuzzy attribute value not found is false with an attribute weight of 0. Methods of combining weights of n attributes in a tuple are addressed by fuzzy operations for intersection and union using t-norm and t-conorm functions. These functions are described in Section 2.7 .

When the relational database model is extended to include attribute types based on fuzzy set theory, the foundation of logic and mathematics is retained and the model is extended to include features that support applications using fuzzy logic.

A database management system which uses fuzzy attributes to represent missing information and multi-valued logic is possible. Fuzzy sets suggest a potential alternative to null as a representation of missing data and attribute weights offer increased granularity for attribute classifications while supporting the recommended 3-valued Boolean data type described by Codd.[9]

Database applications able to match vague queries to ambiguous data in an intuitive way should be possible. Fuzzy sets offer techniques such as linguistic labels, modifiers, and variables that can support features of natural language. This capability is described in Section 2.10.

2.5 Fuzzy Concepts

2.5.1 Crisp and Fuzzy Sets

Fuzzy set theory is a generalization of crisp set theory. *U* is a space containing points that represent real-world objects. A generic element of *U*, denoted by *u*, ranges over this domain of discourse.[2, p.339] The elements used to define and create both crisp and fuzzy sets are from this universal set *U*, which is a crisp set.

2.5.2 Universe of Discourse

In the context of a fuzzy relational database model, the universe of discourse represents a domain of all attribute values. The attribute domain is the set of values from the universe that a specific attribute may take. LASTNAME is a character string variable which could contain the value 'Smith', but it also has a domain that includes 'Smith' and all other valid last names.

This latter use models logical concepts within the semantics of both model-theoretic and proof-theoretic database as described by Date.[8,p.776] This supports a notation in which attributes have a type that is defined by a domain which is a subset of values from the universe of discourse and referenced using a variable that ranges over this subset.[10,p.104]

2.5.3 Crisp Set Membership

Membership of an element in a crisp set C is determined by the set's characteristic function which returns a value from the set {0,1}. A 0 indicates that the element is not in the set. A 1 indicates that the element is a member of the set.[3,p.6]

$$\text{Char}_C(u) : U \rightarrow \{0, 1\} \text{ where } C \subseteq U \quad (2.4)$$

The domain of the characteristic function in Equation (2.4) is the crisp set *U* and the range is the crisp set {0,1}.

2.5.4 Fuzzy Set Membership

The degree of membership for an element in a fuzzy set F is a value determined by its membership function over the interval [0,1]. Fuzzy sets are created either by elements from crisp sets made fuzzy by the association of a membership weight with the element, or by combining elements from existing fuzzy sets. The membership function takes an element value as its argument and returns this value's membership weight for a specific fuzzy set. [3,p.11]

$$\mu_F (u) : U \rightarrow [0; 1] \quad (2.5)$$

The domain of the membership function in Equation (2.5) is the crisp set *U* and the range is the closed interval [0,1]. A fuzzy set *F* is defined over the universe of discourse *U*, and composed of pairs of values, where u is a value from the crisp set *U* and $\mu_F(u)$ is the degree of membership or the membership weight of u determined by the function μ_F , the membership function for fuzzy set *F* as in Equation (2.6).[2,p.339]

$$F = \{ \mu_F(u)/ u \mid u \in U, \mu_F(u) \in [0, 1]\} \quad (2.6)$$

2.6 Fuzzy Set Operations

2.6.1 Equality

Two fuzzy sets A and B defined over U are equal when:[11,p.7]

$$\forall u \in U, \mu_A(u) = \mu_B(u) \quad (2.7)$$

2.6.2 Inclusion

For two fuzzy sets *A* and *B* defined over *U*, *A* is determined to be included in *B* when: [11,p.7]

$$\forall u \in U, \mu_A(u) \leq \mu_B(u) \quad (2.8)$$

2.6.3 Complement

The complement, defined in Equation (2.9), is the relative complement of set with respect to the universe of discourse defined by set *U*. *U* is a crisp set so that the weight of $\mu(u)$ for an element *u* in *U*, but not in is equal to 0. [3,p.7]

$$= \{ \mu(u)/u \mid \mu_F(u) = 1 - \mu(u), u \in U\} \quad (2.9)$$

The complement, defined in Equation (2.10), is the active complement of set *F* with respect to itself. [3,p.7] This is the case for a fuzzy relational database where the set *F* is the 'active domain' as defined by Codd [7, p.380] and *F* is the 'active complement' of *F*.

$$= \{ \mu(u)=u \mid \mu(u) = 1 - \mu(u), u \in U, \mu(u) > 0\} \quad (2.10)$$

2.6.4 Intersection

The intersection of two fuzzy sets requires a binary function to combine membership weights when an element in set A matches an element in set B. This function, fuzzy intersection in Equation (2.11), takes two membership weights as its argument and returns an appropriate membership weight for the fuzzy set intersection operation. The class of t-norm functions meets these requirements so that t-norm and fuzzy intersection are interchangeable terms.[3,pp.61-2]

$$\textit{fuzzy intersection}: [0, 1] \times [0, 1] \rightarrow [0, 1] \quad (2.11)$$

Fuzzy intersection is often determined using the min t-norm function.

$$A \cap B = \{\mu_{A\cap B}(u)/u \mid \mu_{A\cap B}(u) = \min[\mu_A(u), \mu_B(u)], u \in A \wedge u \in B\} \quad (2.12)$$

2.6.5 Union

The union of two fuzzy sets is similar to fuzzy set intersection. A binary function is required to combine membership weights when an element in set A matches an element in set B or to return the membership weight of elements from either set that do not have a match in the other set. This

function, fuzzy union in Equation (2.13), takes two membership weights as its argument and returns an appropriate membership weight for the fuzzy set union operation. The class of t-conorm functions meets these requirements so that t-conorm and fuzzy union are interchangeable terms. [3,pp.76-7]

$$\textit{fuzzy union} : [0, 1] \times [0, 1] \rightarrow [0, 1] \qquad (2.13)$$

Fuzzy union is often determined using the max t-conorm function.

$$A \cup B = \{\mu_{A \cup B}(u)=u \mid \mu_{A \cup B}(x) = \max[\mu_A(u), \mu_B(u)], u \in A \vee u \in \vee B\} \quad (2.14)$$

2.7 T-norm and T-conorm

Operations that combine sets choose elements to be included in the results set. The characteristic function of crisp sets is used with logical operators to create logic functions to choose set elements. Logic functions that perform the choice admit a truth value of 0 or 1.

Operations that combine fuzzy sets use two kinds of operators to determine the degree of membership for each element selected for inclusion in the combined fuzzy set. The t-norm and t-conorm are logic functions that generalize 2-valued logic by admitting truth values on the interval [0, 1]. This value is taken as the degree of membership for the chosen element in the combined fuzzy set. The operands used to compute t-norm and t-conorm are the values from fuzzy set membership functions.[12,p.45]

2.7.1 T-norm

The t-norm operator, $\top$, is used by the fuzzy set intersection function. There are many functions generically represented using $\top$ that satisfy the criteria of the t-norm. [3,pp.74] All t-norm functions are defined by the properties given in Table 2.1 such that these functions maintain the

relationship in Equation (2.15). [12,p.46] The min function meets these criteria, is simple and is considered the standard t-norm operator for fuzzy sets. [3,pp.50] It is proven that min represents the upper limit for all t-norm functions. [3,pp.63] While either prefix (i.e. min(a,b)) or index (i.e. a $\wedge$ b) notation is correct, the prefix notation is used here to avoid confusion with the logical and $\wedge$ symbol.

$$\top(x, y) \leq \min(x, y) \qquad (2.15)$$

Property	Equation
function	$\top : [0, 1] \times [0, 1] \rightarrow [0, 1]$
arguments	$\forall a, b, x, y, z \in [0, 1]$
boundary conditions	$\top(x, 0) = 0, \top(x, 1) = x$
commutativity	$\top(x, y) = \top(y, x)$
monotonicity	$(x \leq a, y \leq b)) \Rightarrow \top(x, y) \leq \top(a, b)$
associativity	$\top(\top(x, y), z) = \top(x, \top(y, z))$

Table 2.1: Definitions for t-norm

2.7.2 T-conorm

The t-conorm operator, $\perp$, is used by the fuzzy set union function. There are many functions generically represented using $\perp$ that satisfy the criteria of the t-conorm. [3,pp.82] All t-conorm functions are defined by the properties given in Table 2.2 such that these functions maintain the relationship in Equation (2.16). [12,pp.46] The max function meets these criteria, is simple and is considered the standard t-conorm operator for fuzzy sets. [3,pp.50] It is proven that max represents the upper limit for all t-conorm functions. [3,pp.77] While either prefix (i.e. max(a, b)) or index (i.e. $a \vee b$) notation is correct, the prefix notation is used here to avoid confusion with the logical or $\vee$ symbol.

$$\perp(x, y) \leq \max(x, y) \qquad (2.16)$$

Property	Equation
function	$\perp : [0, 1] \times [0, 1] \rightarrow [0, 1]$

arguments	$\forall a, b, x, y, z \in [0, 1]$
boundary conditions	$\perp (x, 0) = x, \perp (x, 1) = 1$
commutativity	$\perp (x, y) = \perp (y, x)$
monotonicity	$(x \leq a, y \leq b)) \Rightarrow \perp (x, y) \leq \perp (a, b)$
associativity	$\perp (\perp (x, y), z) = \perp (x, \perp (y, z))$

Table 2.2: Definitions for t-conorm

2.8 The Intrinsic Nature of Fuzzy Sets

Some of the differences between crisp and fuzzy sets are significant. In particular, the law of contradiction (Equation 2.17) and the law of excluded middle (Equation 2.18) are applicable to crisp sets, but do not apply to fuzzy sets.

$$C \cap = \emptyset \tag{2.17}$$

$$C \cup = U \tag{2.18}$$

It is possible for $u \in U$ to have membership in both fuzzy set F and its complement . The complement of a fuzzy set has this property because fuzzy set theory allows objects to be placed into more than one classification as determined by the membership function associated with each fuzzy set.

If $0 < \mu_F(u) < 1$, then u is included in $\overline{F}$. This is because u has partial membership in F and a corresponding partial membership in its complement, .

If $\mu_F(u) = 0$, then u is also included in $\overline{F}$. This is because u has no membership in set F and the corresponding membership of $u \in$ is $\mu(u) = 1$.

An element in a crisp set *C* cannot also belong to its crisp complement, $\overline{C}$. This characteristic of crisp sets is the basis of the laws of contradiction and excluded middle which are central to classical logic and crisp set theory.

Multiple classifications allow fuzzy sets to represent real-world classification in an intuitive way and suggest techniques that use fuzzy logic to create multi-valued logic.

2.8.1 Law of Contradiction

Equation (2.17) defines the law of contradiction for crisp sets, but does not apply to fuzzy sets. [3,pp.25] The following example using fuzzy set *F* defined over the universe *U* demonstrates that the intersection of *F* and its complement is not equal to the empty set.

$$U = \{a, b, c, d, e, f, g, h, i, j\} \quad (2.19)$$

$$F = \{1.0/a, 0.3/b, 0.8/c, 0.6/d, 0.2/e, 0.1/j\} \quad (2.20)$$

$$F = \{0.7/b, 0.2/c, 0.4/d, 0.8/e, 1.0/f, 1.0/g, 1.0/h, 1.0/i, 0.9/j\} \quad (2.21)$$

$$F \cap = \{0.3/b, 0.2/c, 0.4/d, 0.2/e, 0.1/j\} \quad (2.22)$$

2.8.2 Law of Excluded Middle

Equation (2.18) defines the law of excluded middle for crisp sets, but does not apply to fuzzy sets. [3,pp.25] The example using fuzzy set *F* defined above demonstrates that the union of *F* and its complement is not equal to the universe *U*.

$$F \cup = \{1.0/a, 0.7/b, 0.8/c, 0.6/d, 0.8/e, 1.0/f, 1.0/g, 1.0/h, 1.0/i, 0.9/j\} \quad (2.23)$$

2.9 Representations Using Fuzzy Sets

Fuzzy sets model notions of set membership in a way that is compatible with common sense. Part of this model of reasoning is the idea of natural language where subtle differences in the

strength of descriptive terms are generally understood. This approach is used to query and match elements in a fuzzy set using match criteria with an expectation of more-or-less close matches.

2.10 Linguistic Labels

Linguistic labels are natural language words (e.g. large, tall) that provide a method of expressing membership in a fuzzy set using human terminology. [11,pp.3] Each linguistic label represents a classification defined as a fuzzy set or a subset of a fuzzy set. A fuzzy set which is convex (Equation 2.24) and normal (Equation 2.26) is also defined to be a fuzzy number representing a range of close values. [13,pp.50] A linguistic label can now represent a range of values as represented as a fuzzy number. In terms of natural language, modifying the breadth of a classification to a greater or lesser extent is achieved, theoretically, by using adjectives (e.g. *very* large). In terms of fuzzy set definition, such modifications must be made to the membership weights in queries of the set or through 'filters' applied to fuzzy set membership weights. While synonyms may be implemented using linguistic labels, adjectives and filters are implemented using α-cuts and/or functions known as linguistic modifiers.

2.10.1 Convexity

The fuzzy set F defined over U is convex when: [13, pp.50-1]

$$\forall x,y \in U,\ \forall \lambda \in [0, 1],\ \mu_F(\lambda \cdot x + (1 - \lambda) \cdot y) \geq min(\mu_F(x), \mu_F(y)) \qquad (2.24)$$

2.10.2 Height

The height of fuzzy set F is the greatest membership degree for any element in set F (i.e. the supremum α when ${}^{\alpha}F \neq \emptyset$).[3, p.21]

$$\text{Height}(F) = \sup_{u \in U} \mu_F(u) \qquad (2.25)$$

2.10.3 Normal

A fuzzy set *F* is normal if its height is equal to 1 and subnormal when its height is less than 1. [3, p.21]

$$\text{Normal(F)} = \begin{cases} 0 & \text{Height(F)} < 1 \\ 1 & \text{Height(F)} = 1 \end{cases} \quad (2.26)$$

$$\text{Subnormal(F)} = \begin{cases} 0 & \text{Height(F)} = 1 \\ 1 & \text{Height(F)} < 1 \end{cases} \quad (2.27)$$

2.10.4 α-cut and Strong α-cut

The α-cut of a fuzzy set *F* is a crisp set, denoted by ${}^{\alpha}F$ is:

$${}^{\alpha}F = \{u \mid u \in U, \mu_F(u) \geq \alpha, \alpha \in [0, 1]\} \quad (2.28)$$

The strong α-cut of a fuzzy set *F* is a crisp set, denoted by ${}^{\alpha+}F$ is:

$${}^{\alpha+}F = \{u \mid u \in U, \mu_F(u) > \alpha, \alpha \in [0, 1]\} \quad (2.29)$$

The concepts of α-cut and strong α-cut have a significant role in the relationship between fuzzy sets and crisp sets. Because these sets are crisp sets, certain properties and operations of crisp sets can be extended to sets defined using either an α-cut or a strong α-cut. [3, p.19] α-cuts can be used to define classifications used to search fuzzy sets using linguistic labels.

The α-cut and strong α-cut define fuzzy set match criteria in terms of membership thresholds. An α-cut of a fuzzy set is the subset of elements whose membership weights lie above a threshold established by the value α. Adjusting the threshold will expand or contract the subset in a way

that will focus its membership on elements which have lessor or greater membership weights. [3, p.19-21]

2.10.5 Support

The support of a fuzzy set *F* is a crisp set of the element values from *F* that have a membership weight greater than zero. This is the equivalent of the strong α-cut where α is equal to zero (i.e. ${}^{0+}F$). [3, p.21]

$$\text{Support}(F) = \{u \mid u \in U, \mu_F(u) > 0\} \qquad (2.30)$$

2.10.6 Kernel

The kernel of a fuzzy set *F* is a crisp set of the element values from *F* that have a membership weight equal to one. These are the elements that completely belong to *F*. [13, p.51]

$$\text{Kernel}(F) = \{u \mid u \in U, \mu_F(u) = 1\} \qquad (2.31)$$

2.10.7 Linguistic Modifiers

Linguistic modifiers can be used with fuzzy set membership functions and linguistic labels to shrink or expand the notion of fuzzy set membership by decreasing or increasing area under the membership functions graph. This is similar to how adjectives modify the meaning of nouns in natural language. This technique is referred to as linguistic hedging. Linguistic hedges are intentionally ambiguous statements. Fuzzy set modifier functions shift or squeeze the membership weights from a fuzzy set to create a set that can be referenced by an appropriate linguistic label. The linguistic label of the modified set is composed by concatenating the fuzzy set identifier with a descriptive label (i.e. an adjective) associated with the set modifier. [14, p.54-5]

From the example in Section 2.2 the three membership functions (i.e. EMPTY, HALF-FULL or FULL) can be used with the crisp set VOLUME to create fuzzy sets. In Table 2.3 the

membership weights for full bottles of a particular VOLUME and the effects of weights for linguistic modifiers (i.e. concentration, dilation, and intensification) as defined in the following sections are shown. A 20 ounce bottle that contains 15 ounces is full with a weight of $\mu_{FULL}(15) = 0.75$, but it is very full with only a weight of $\mu_{VERY\text{-}FULL}(15) = 0.562$. The bottle belongs in the set FULL more than it belongs in the set VERY FULL. If the same bottle is described as somewhat full and contains 15 ounces, it has a weight of $\mu_{SOMEWHAT\text{-}FULL}(15) = 0.866$ which emphasizes the fact that a bottle this full is, to a greater degree, somewhat full.

Crisp set of bottle content in fluid ounces.

VOLUME = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}

Example, linguistic label creates a synonym for a fuzzy set:

FULL-SODA-BOTTLE = full(VOLUME)

Example, linguistic labels and modifiers create new fuzzy sets:

VERY-FULL = CON(full(VOLUME))

SOMEWHAT-FULL = DIL(full(VOLUME))

2.10.7.1 Concentration

Concentration decreases membership weights of all elements in a fuzzy set, but proportionally more for elements of lessor membership weight. Concentrating the membership weights of a fuzzy set decreases the spread of the set element weights. This linguistic modifier hedges a query as if it were an adjective such as “very." [14, p.54]

$$\mu_{CON(F)}(u) = (\mu_F(u))^2 \qquad (2.32)$$

The effect of concentration is shown visually in Figure 2.2 where the dotted plot lines cover a more narrow range of higher membership weights.

2.10.7.2 Dilation

Dilation increases membership weights for all elements in a fuzzy set, but proportionally more for elements of lessor membership weight. Dilating the membership weights of a fuzzy set increases the spread of the set element weights. This linguistic modifier hedges a query as if it were an adjective such as “more-or-less." [14, p.54]

$$\mu_{DIL(F)}(u) = (\mu_F(u))^{1/2} \qquad (2.33)$$

The effect of dilation is shown visually in Figure 2.2 where the dashed plot lines cover a wider range of higher membership weights.

2.10.7.3 Intensification

Intensification combines the effects of dilation and concentration. Membership weights are increased (i.e. dilated) for elements of greater membership weight and decreases (i.e. concentrated) for elements of lessor membership weights. This change takes place at the mid-point of the set membership weights in the interval [0, 1]. [14, p.54]

$$\mu_{INT(F)}(u) = \begin{cases} 2 \quad (\mu_F(u))^2 & \text{if } 0 \le \mu_F(u) \le 0.5 \\ 1 - 2\,(1 - \mu_F(u))^2 & \text{if } 0.5 \le \mu_F(u) \le 1 \end{cases} \qquad (2.34)$$

The effect of intensification is shown visually in Figure 2 where the dot-dash plot lines shift from concentration to dilation at the mid-point of the membership weights range.

Volume	Full	Concentrated	Dilated	Intensified
0	0.00	0.000	0.000	0.000
1	0.05	0.003	0.224	0.003
2	0.10	0.010	0.316	0.010
3	0.15	0.022	0.387	0.022
4	0.20	0.040	0.447	0.040
5	0.25	0.062	0.500	0.062
6	0.30	0.090	0.548	0.090
7	0.35	0.122	0.592	0.122
8	0.40	0.160	0.632	0.160
9	0.45	0.202	0.671	0.202
10	0.50	0.250	0.707	0.250
11	0.55	0.303	0.742	0.595
12	0.60	0.360	0.775	0.680
13	0.65	0.423	0.806	0.755
14	0.70	0.490	0.837	0.820
15	0.75	0.562	0.866	0.875
16	0.80	0.640	0.894	0.920
17	0.85	0.722	0.922	0.955
18	0.90	0.810	0.949	0.980
19	0.95	0.902	0.975	0.995
20	1.00	1.000	1.000	1.000

Table 2.3: Full Bottles Membership Weights with Modifier Weights

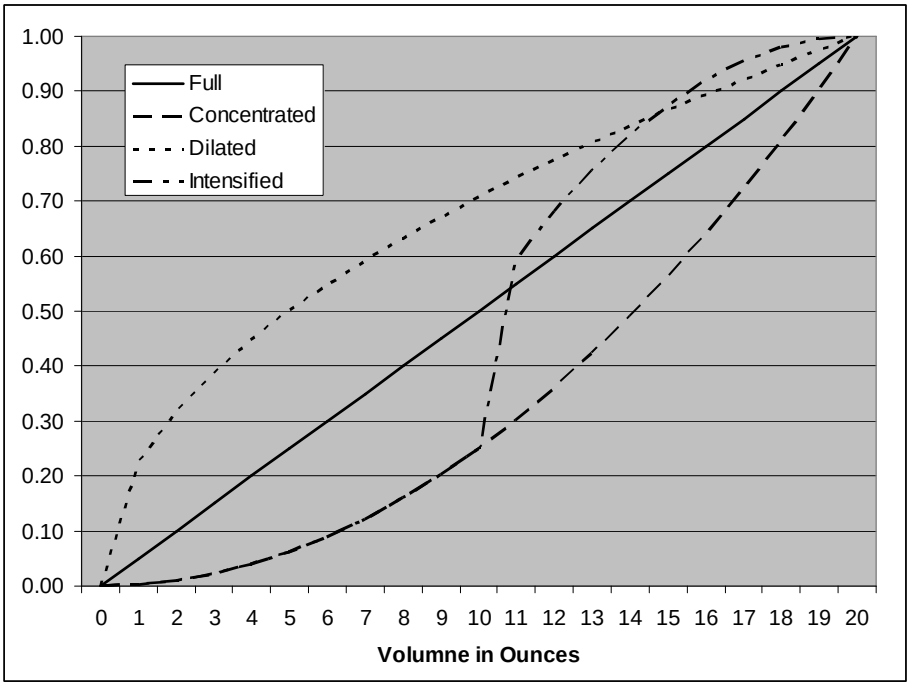


Figure 2.2: Full Bottles Membership Weights with Modifier Weights Full Fuzzy Set (concentrated, dilated and intensified)

2.10.8 Alternative to Linguistic Modifiers

The problems inherent in the concept of linguistic based modifiers as proposed by Petry and others [14] lay in the fact that words are cumbersome, unique to the society using the system and cannot reasonably, accurately and consistently be interpreted as quantitative values. It can easily be argued that any attempt to quantify language in any precise way is an exercise in futility. Even if such linguistic/value associations are agreed to by the user community or a team of experts, adjectives do not represent values. The community cannot reasonably interpret the word 'very' consistently as 0.8, which is essentially the concept behind linguistic modifiers; words and their synonyms that represent quantitative values that will have a consistent modifying effect on the fuzzy number.

Consider an even simpler example. Linguistic modifiers consist of words and their synonyms. These words are nothing more than character strings which themselves are nothing more than symbols that have a conceptual meaning to users who are familiar with the language being used. As such, the word 'very' is no more meaningful than '%xy&' if the community determines that '%xy&' has a linguistic meaning. It follows then that if a character or string of characters has a linguistic meaning to the community, then nothing more than the letter *x* is necessary to play the role of linguistic modifier. If it can be said that 'very' = 0.8 than it can just as reasonably be said that $x = 0.8$ along with its 'synonyms' *y* or *z*.

While a user *may* recall the quantitative value associated with 'very', there is no assurance that they will. In short, words as modifiers are themselves fuzzy with meanings only determined and maintained by a community.

As an alternative, graphic modifiers can be substituted for linguistic modifiers. Consider the example of an implementation of a linguistic modifier (Figure 2.3) where the user selects from a list of words to communicate the higher or lesser intensity of certainty.

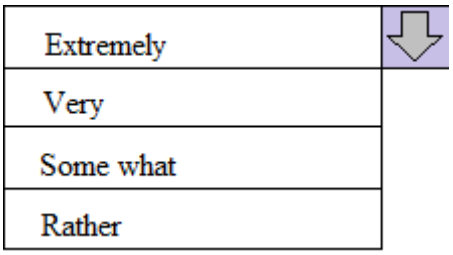


Figure 2.3: Use of Linguistic Modifier

Compare the example in figure 2.3 with the examples in figure 2.4 which represent a graphic modifier to accomplish the same objective.

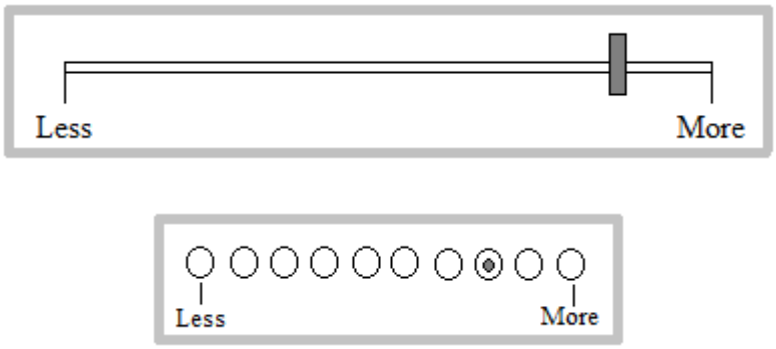


Figure 2.4: Use of Graphic Modifiers

A graphic modifier is not only more intuitive, but also does not require the user or the community to determine the significance or quantitative value of the modifier. In fact, the graphic modifier translates very easily into a quantitative value as the ‘less’ end of the spectrum logically represents a value of 0 while the other extreme logically represents a value of 1.

Furthermore, graphics are independent of any human language which eliminates the need to maintain a dictionary.

Given the fact that the purpose of a modifier is to communicate the level of intensity associated with a fuzzy variable, a graphic modifier has clear advantages over the use of linguistic modifiers and the associated dictionary of words, synonyms and their associated values. Further, the use of and results emanating from a graphic modifier translate directly into a quantitative value which represents the same information a community and system would struggle with in their attempt to quantify language.

From this point forward, any reference to modifiers will refer to values obtained through the use of quantitative graphic modifiers.

2.11 Possibility Theory

Zadeh built upon the concept of the fuzzy set and proposed possibility theory. Possibility theory is an abstraction that uses fuzzy logic to construct a semantic model of the logical relationships and patterns found in data. Zadeh's goal is a representation of information that accounts for the intrinsic imprecision, vagueness, and ambiguity in human reasoning and natural language. Possibility theory has a relationship with probability theory and includes concepts which parallel probability.[6]

Possibility theory references the same universe of discourse, *U*, as fuzzy set theory. The variable *X* takes the value of *u* from *U* as shown in Equation (2.35).

$$X = u \tag{2.35}$$

Let fuzzy set F be a subset of U with a membership function μ_F. If F acts as an elastic constraint to determine the values assigned to X as in Equation (2.36) and F is associated with X as a fuzzy restriction, the degree to which F satisfies the constraint on X is measured as $\mu_F(u)$ which also indicates the compatibility of u with the concept represented by F. The degree by which the constraint is stretched to allow this assignment is measured as 1 - $\mu_F(u)$ and is the distance between u and the concept represented by F.

$$X = u \mid \mu_F(u) \tag{2.36}$$

Let $R(X)$ be defined as a fuzzy restriction on X. An assignment equation of the form in Equation (2.37) associates the fuzzy set F with the restriction on values for the variable X.

$$R(X) = F \tag{2.37}$$

The fuzzy restriction applied to variable X in Equation (2.37) can be translated as the proposition that "X is F." Zadeh postulates that this proposition associates a possibility distribution, Pos_X, with X as a fuzzy restriction on X as given in Equation (2.38).

$$Pos_X \triangleq R(X) \tag{2.38}$$

The possibility distribution function of Pos_X is defined as pos_X and is the numeric equivalent of the membership function of X as shown in Equation (2.39).

$$pos_X \triangleq \mu_F \tag{2.39}$$

X is now a fuzzy variable associated with the possibility distribution Pos_X in the same way a random variable may be associated with a probability distribution. The possibility distribution function $pos_X(u)$ is the possibility that $X = u$ and is postulated to equal $\mu_F(u)$.

The relational assignment in Equation (2.37) which associates F with variable X may be restated using Equation (2.38) as follows in Equation (2.40).

$$Pos_X = F \qquad (2.40)$$

Equation (2.39) defines $pos_X(u)$ to be equivalent to $\mu_F(u)$ and the possibility that $X = u$, is postulated to be equal to $\mu_F(u)$. [6, p.12] In the context of a fuzzy relational database, the membership weight of a fuzzy attribute represents the extent to which it is possible that the attribute value belongs in the classification represented by its associated fuzzy attribute. This allows $pos_X(u)$ to be used to refer to the possibility of u.

An Example

Let X be a variable that may take the value of $u \in U$ and let *FULL* be a fuzzy subset of U consisting of bottles that are full to some degree (refer to Table 2.3). $R(X) = FULL$ associates the fuzzy set *FULL* defined by the function full in Equation (2.3) as a fuzzy restriction on X.

If X is *FULL*, the association of *FULL* with a possibility distribution for X may be expressed as $Pos_X = FULL$. Now the possibility of X taking the value of u is $pos_X(u)$ or $\mu_{FULL}(u)$. If the value of u is 15, the possibility of X taking this value is 0.75. It is possible that a 20 ounce bottle containing 15 ounces is full with a degree or weight of 0.75.

2.12 Vocabulary of Fuzziness

Imprecise is defined as not exact; vague or indefinite in nature. Imprecise was the first term used by Zadeh to describe fuzzy data. [2,p.338]

Vague is defined as being stated in general or indefinite terms; not having an exact or precise meaning. There are other definitions implying that the cause of vagueness is a lack of understanding, clouded thoughts or a hazy mental state. While the first definition describes fuzzy data, the latter describes the mental state of the user rather than the data.[6][13]

Subjective is defined as being determined from opinions, intuition, or feelings rather than observation and reason. Subjectivity represents preconceptions derived from within the observer; not necessarily based on the external environment.[15] This is similar to the idea that vagueness may originate in the mind of the user rather than within the data. In the context of a database query, this suggests that the user may have expectations that acceptable results must match.

Unclear is defined as being not explicitly defined, or indecipherable. Undefined data are similar to missing or not yet complete data. [15] Data that are indecipherable are perhaps incomplete, but may also indicate that there is confusion on the part of the person interpreting the data.

Ambiguous is defined as capable of being classified in two or more categories, which reflects the concept of partial membership in more than one fuzzy set. [13] Ambiguity is closely related to subjectivity, opinion, and perception.

Uncertain has a number of relevant definitions that depend on context. A fact may be uncertain if there is doubt or the fact is not known. A numeric value is uncertain if it cannot be accurately measured or determined. Places and things are uncertain if not identified or located. Events in the past may be uncertain if they are of an indefinite date and time, as are events in the future that may not occur.[2]

Incomplete is defined as not being finished or not having all of its components. Missing data, unknown data, not applicable, or data not yet available are all meaningful descriptions of incomplete data.[9]

Inconsistent is defined as showing contradiction; a proposition with parts that cannot all be true. Data integrity rules are intended to enforce data consistency requirements when a relational database is modified. If inconsistent data can be represented using fuzzy sets, the necessary membership weights may be set using database integrity rules.[13]

Ma presents five classifications of imperfect data (i.e. inconsistent, imprecise, vague, uncertain, and ambiguous) for use when modeling fuzzy data. [13,p.47] But a careful analysis of these five classifications and the eight commonly used terms to describe fuzzy data suggests four kinds of fuzzy data described in the sections below. These types of fuzzy data are characterized by data structures of one or more related values and operators that manipulate these data as necessary to represent imperfect fuzzy data at both a syntactic and logical level.

2.12.1 Imprecise and Vague

Both imprecise and vague describe values that may be known approximately, but not exactly. Imprecise numeric input is best represented as a data range that is likely to contain the accurate, but unknown value. A membership weight must be calculated for the approximate numeric value. Alternately, the input value may be a natural language term that is mapped by a process that determines an approximate numeric value and its ranges as well as the fuzzy weight that indicates how much the numeric value represents the natural language term. Vague queries with arguments defined in terms of natural language are the corresponding search for this approximate

value. The query term should map to the approximate value. An alternate search technique for approximate values is the use of α-cuts to select those values at and above a specified membership weight.[6][13]

The database user intuitively knows if an answer is close or acceptable and can judge the query results. These imprecise numeric values are fuzzy numbers that are represented as weighted approximations mapped within a minimum/maximum range.

2.12.2 Subjective and Unclear

Application design and queries must encompass user subjectivity and data clarity. While unclear data may be incomplete data, it is also possible that lack of clarity is related to the data collection process or intrinsic data ambiguity and thus related to user subjectivity.[15] The database user's point of view is the focus of subjectivity and is represented by expectations for query results. If it is not clear how the database user will perceive the data, accurate query results depend on good database design and a clear understanding of application requirements specification. Queries derived from application requirements can be anticipated during application design and this awareness may be used to create stored query procedures that take advantage of this knowledge. Ad hoc queries may require a different strategy by using α -cuts to constrain searches. A solution may be to return a broadly defined result relation and allow the user to refine the ad hoc query interactively to seek more accurate results.

2.12.3 Ambiguous and Uncertain

Ambiguity may require multiple data classifications and membership weights. An ambiguous domain is represented by a data value of some appropriate type and one or more fuzzy

membership weights associated with the attribute as classification. Each classification is a set in which the data value has partial membership.[13]

A common partial membership classification for fuzzy data is uncertainty. Application requirements should anticipate the potential of uncertainty and database design can then include a certainty membership weight. The certainty of an attribute value may vary over time (e.g. a value that is certain with a low membership weight at data entry may be updated with greater certainty later). Queries that answer questions can use the certainty weight when evaluating the truth of propositions in the search results.

2.12.4 Incomplete and Inconsistent

Incomplete data is a characteristic of change. Incomplete data must be stored as is until missing information is available. This is a known issue for the relational database model.[9] A fuzzy classification similar to data certainty may solve this problem.[6] An incomplete data value may have partial membership in a classification named complete. A search for data that is not complete can be used by a data validation process. A query that answers questions can use the weight of completeness to determine data reliability and to evaluate the truth of propositions in the search results.

Inconsistent data is another issue and is related to database design rather than the relational database model. This is the case where different data values for the same attribute of the same object occur in different locations within the database. Inconsistent data indicates incorrect database design which can be corrected without using fuzzy logic.

Chapter 3 - Relational Database Systems

3.1 Problems Inherent to Relational Database Systems

There are unsolved problems in database systems that implement the relational model. The relational model was extended [9] to address the missing data problem and the extension was partially implemented in many relational database management systems. Imprecise and uncertain data remain an unsolved problem that may be solved by extension to the relational model and enhancements to existing database systems.

3.1.1 Relvar Data Type

The relational model allows a relation to be nested within an attribute as a value (i.e. a relation valued attribute). [16,p.23-24] Relvars have a type defined by their heading and an existing relvar may be used as a template to create an identical relvar, but there is no declared relvar data type. [17,p.28] For this reason, a relation valued attribute (RVA) has an implied data type of “relation” or RVA and its domain is all relations.

3.1.2 Attribute Data Type

The data types in the relational model are restricted to the domain types used to declare attributes. Current relational database management systems (RDBMS) do not support user defined data types that can be integrated into domain definitions. Current RDBMS domains allow Boolean, character, string, integer, floating-point and unstructured binary data types. What is typically considered a user defined data type in a current RDBMS is nothing more than a restriction on the range of numerical values allowed.

3.2 Fuzzy Relational Database

Database applications may be more intuitive and usable when queries are allowed to be vague and results are enhanced with information derived by inference. Using fuzzy set theory to extend the relational model may provide a uniform method of support for vague queries.

The relational model has been extended[9] and continues to be refined.[16] Database systems based upon the relational model have changed a great deal and in some cases are not consistent with the model. The use of null to represent missing data and 3VL are extensions to the relational model. Null was added to the SQL standard, but no RDBMS products[i] implement a corresponding Boolean data type that includes the “unknown" value to support 3VL.[18,pp.13-4]

An extension to the relational model based on fuzzy set theory is acceptable as long as the model is not violated. The model was based on crisp sets which are a specific case subsumed by fuzzy set theory. A fuzzy extension must be developed with an understanding of relational database theory, formal logic systems, and fuzzy set theory so that database domains can be implemented as fuzzy sets where attributes have names, types, and membership weights. This must be done in a way that makes sense and follows Codd's Twelve Rules, Date's “Rule Zero," (i.e. requirement that a relational database management system use its relational management facilities to manage the database), and the “Golden Rule" which requires that the set of relvars that constitute a database remain consistent with its integrity constraints at all times). [8]

[i] In 2013, a search of the world wide web, product documentation and experiments using DBMS products indicate that Microsoft SQL Server[17], Microsoft Access[18], Oracle[19], Ingres[20], PostgresSQL[21] and MySQL[22] either implement a Boolean type capable of no more than 2-valued logic or that have no Boolean type at all. This requires the user to designate some other data type and a set of values to represent false, true, and unknown.

3.2.1 Extending RDB with Fuzzy Types

The relational model can be extended using fuzzy set theory to allow relvar attributes which have domains that are fuzzy sets. To make this extension, the appropriate distinction must be made between domains based on ordinary or crisp sets and those based on fuzzy sets. Such a distinction has been included in experiments using RDBMS products such as Microsoft SQL Server and Access, Oracle, Ingres, Postgres, and MySQL which either implement a Boolean type capable of no more than 2-valued logic or have no Boolean type at all. This requires the database user to designate some other data type and a set of values to represent false, true and unknown.

3.2.2 Representation of Unknown and Uncertain Data

Codd added null and 3VL to the relational model to represent unknown or uncertain data. [9] Fuzzy sets are able to represent uncertainty. Zadeh gives an example application for fuzzy sets which suggests an implementation of 3VL using a fuzzy set using two value levels α and β with the relationships defined in Equation (3.1).

$$\begin{gathered} 0 < \alpha < 1 \\ 0 < \beta < 1 \\ \beta < \alpha \end{gathered} \qquad (3.1)$$

A 3VL is defined using the attribute membership weights over the interval [0,1] with an application specific definition of the meaning for the α and β value levels. [2,pp.341-2] The formula for this logic is shown in Equation (3.2).

$$\text{3VL for set F} = \begin{cases} \text{false} & \text{when } \mu_F(u) = [0, \beta] \\ \text{unknown} & \text{when } \mu_F(u) = (\beta, \alpha) \\ \text{true} & \text{when } \mu_F(u) = [\alpha, 1] \end{cases} \qquad (3.2)$$

3.2.3 A Fuzzy Weight Attribute

A fuzzy domain type must represent the membership weight constrained to (0,1] for compatibility with crisp data types. A crisp data type could be extended by the addition of a membership weight. Simply stated the fuzzy-typed attribute is both a membership weight in (0,1] and a crisp attribute. There must be a mechanism to associate the weight with a crisp value. This will create a (*c*,*w*) pair which satisfies first normal form. It must be possible to associate multiple (*c*,*w*) pairs to an attribute of another data type to allow database values membership in more than one classification. These features will allow fuzzy data types to be used within a DBMS implementation.

3.2.4 Fuzzy Data Categories

Not every type of data is suited to a fuzzy interpretation. If the effort necessary to support a fuzzy-typed attribute is to be worthwhile, the use of fuzzy data must enhance the application. In Section 2.12 our categories of fuzzy data were defined. Each of these categories requires the selection of an appropriate scale of measurement before the classifications for fuzzy-typed attributes can be determined. There are five scale types which may apply to an attribute that is a candidate for fuzzy interpretation. These are the nominal, ordinal, interval, ratio, and absolute scales.[26,p.53] The data represented using these scales can be either numeric or a string used as a descriptive label.

Imprecise and vague data is from the ratio scale and is typically a floating-point value represented by a fuzzy number and/or a linguistic variable. Ratio data may be used to calculate totals and averages. Subjective and unclear data typically requires one or more partial classifications using fuzzy-typed attributes and may be from the nominal, ordinal, interval, or ratio scale. Ambiguous and uncertain data always requires partial classification using fuzzy-typed attributes and may be from the nominal, ordinal, interval, or ratio scale. Incomplete data attributes may be from the nominal, ordinal, interval, or ratio scale and must be associated with a partial classification using a fuzzy-typed attribute that represents the level of data completeness. Inconsistent data indicates a flawed database design and cannot be fixed by fuzzy logic.

Fuzzy classifications are given adjectives for names because of the necessary descriptive nature of attribute properties. The attribute and classification in combination describe some salient trait of an entity. The classification must be defined so the expected weight is greater than zero and if there is a default value, it is one. For example, the uncertainty of most data may be zero. If the uncertainty of an attribute value is a weight of 0.2, its certainty is 1 – 0.2 and the classification associated with the attribute is then represented as CERTAIN with a weight of 0.8.

3.3 Setting Expectations

A challenge to working with fuzzy data is the management or setting of user expectations. Using classic 2 valued logic, the response to a query returns data based on the evaluation of the predicate used to obtain that data as being true. This response reflects the classic two-valued logic that have come to expect. For example, users requesting a list of students enrolled in a specific class, say CS 301, expect the query to return any and all students who are enrolled in CS

301. The expectation is further confined to only those students enrolled in CS 301. The user would not expect students from another class to be included. In fact, such a response would be an error. The user's expectations follow the same logic as the query. If the student is enrolled in CS 301 the predicate evaluates as true. If the student is not in CS 301 the predicate evaluates as false and the result will reflect this fact by the exclusion of that student from the response set. But fuzzy sets expand the capability of classical logic beyond the traditional two values. By necessity it also expands the user's expectations concerning the response. Because fuzzy set theory allows a real-world object to have membership in one or more classes, user expectations must be adjusted to expect and benefit from this outcome.

As an illustration, consider a search engine on the Internet. The inner workings of a modern search engine are well known and no implication is being made that a search engine operates according to the tenets of fuzzy set theory. This example is intended only to highlight the importance and the role of user expectation in the query and response operation where the input and response is inherently uncertain. When a user enters a word or phrase into the search field, the engine returns associated web links that most closely match the user's input. For example, someone might enter "Three brave xxwy# men" in the search field of the search engine. Obviously, or at least it is assumed, "xxwy#" is meaningless and no sites exist on the Internet for this value. Using precise, two-valued classical logic, no web sites should be returned. But users have come to expect the search engine to return those sites that most closely match the selection criteria and to do so in descending order of the closeness of the match. There is any number of web sites containing the words `three', `brave' or `men'. There are probably a number of sites that have the phrase 'Three brave' or 'brave men' which, depending on the algorithm used would

either have greater or lesser prominence than the individual words. But this is not a fuzzy result set. Modern search engines may not assign membership weights to the set returned by the search query and so the concept of weighted membership has no role in this process. In fact, if none of the four words entered into the search field were on the Internet, nothing would be returned. In order to get a response set, some crisp value from the query must be present in the data. So there is an appearance of imprecision and uncertainty, but this imprecision is restricted to the presence of crisp data in some form. The search predicate evaluates as 'true' for some word, words or phrase contained in the query. And yet, despite this fact, this example serves to illustrate a situation where the user expects a result even if that result is not precisely or completely what they asked for.

With the shift from classical two-valued logic to logic that accepts an object existing in more than one class and to varying degrees, the paradigm with respect to user expectations must shift as well, but only in logical ways. The user looking for students enrolled in class CS 301 should still expect a response that contains all and only those students enrolled in CS 301. In this situation, adding students enrolled in CS 312 and assigning a degree of membership would not make sense. In such a case, any application developed to accommodate fuzzy set theory should still operate in a way appropriate to the situation and in accordance with the user's expectations. The user, on the other hand, will need to expand those expectations to include either a crisp set or a fuzzy set response as logically appropriate.

3.4 Four Kinds of Database Update

The motivation for using fuzzy data is driven by application requirements and must be supported by a combination of database capability and design. When data is fuzzy it is necessary to

determine the nature and extent of this fuzziness using a consistent analytical process. The complexity of data entry and update is increased by the need to set a fuzzy-typed attribute weight. The data entry process must employ techniques that interpret the values entered for database update and map membership weights from these values. The semantic content of fuzzy data is stored in a relational database using crisp values that populate data structures designed to represent the components of fuzzy types (i.e. partial classification, multiple classification, fuzzy number, or linguistic variable). The impact on data entry applications that use these mapping techniques is significant, but data entry should be assisted by using parameters stored in the database catalog instead of being encoded in the application.

3.4.1 Crisp Data to Crisp Domain

A crisp data value updated and stored in the database is the typical case. The expectation of the database designer and the application developer is an accurate transfer of data values from data entry forms or measurement instruments to database storage.

3.4.1.1 A Simple Example

A database of business contacts is an example of a simple crisp database application. Each tuple includes a contact name, organization name, mailing address, email address and phone number. Salesmen collect business cards from potential customers and someone enters the data from each card into the database. The data source for this application is certain and crisp. The database domains are crisp.

3.4.1.2 Height Example

Data entry accepts the height in inches for a male subject age 20 and over. There is no expectation of error or subjective opinion. The measurement process is documented for a precise

measurement device operated by a skilled and trained technician who transcribes the measurement to a data entry form.

3.4.2 Fuzzy Data to Crisp Domain

This is the case where the opinions, point of view, and judgment of the person who collects and/or enters the data are used to adjust a value before it is added to the database. A mapping from a fuzzy concept to an appropriate precise value in a crisp domain may be needed, but it is also likely that data is missing and requires processing as an exception. If there is no default value, data entry must either follow some instruction set or enter a sentinel value that can be corrected later. It is likely uncertainty remains at the end of this process and information is lost.

3.4.2.1 A Simple Example

The business contacts application is enhanced to include estimated sales data. These estimates are developed by the salesman after meeting with a customer. Salesmen are expected to estimate sales for current year using their expert opinion. This makes the data fuzzy, but some salesmen take it a step further and estimate a minimum and a maximum for expected sales as the estimate. They argue that the best estimate is somewhere in this range. The data entry operator enters the contact information and must include the estimated sales. If the estimate is a range, the data entry operator calculates the mid-point of the minimum and maximum and enters it as the estimate. Later the database administrator adds attributes for the minimum and maximum to the database with a function that calculates the sales estimate as the mid-point. If the salesman includes an estimate with a range, the calculated estimate can be overwritten. This creates a crisp version of a fuzzy number without a membership weight to measure the expert's belief in the approximated number. The expected sales data is fuzzy and the domain is crisp, but information is lost.

3.4.2.2 Height Example

Data entry accepts height in descriptive terms using natural language for a male subject age 20 and over. The expectation of a subjective opinion is accepted. The measurement may be reported as an estimate by the subject who does not know his height or by another person. The height as a descriptive term may be short, average, tall, very short, normal, and “I am tall or almost tall.” Data entry is allowed to ask about the estimate and use the replies to enter a precise and specific height value in inches. Data entry must consider the height of the individual who estimated the subject's height. If the estimator believes that he and the subject are tall, but appears to be 5 feet in height, the estimation should be adjusted downward. If the estimator believes he is average height, but appears to be over 6 feet in height, and estimates the subject is very tall, the estimation should be adjusted upwards. If the estimator reports his height in inches and confirms that the subject was about the same height, this height can be entered. Using these techniques to map fuzzy data to a crisp domain will result in some instances where information is lost because there is no associated fuzzy-typed attribute to indicate certainty.

3.4.3 Crisp Data to Fuzzy Domain

Crisp data may be used to update fuzzy domains designed for partial membership and/or multiple membership classifications. Fuzziness originates from the relationship between the crisp value and the fuzzy category. The membership weight is calculated based on this relationship which is an aspect of the application for which the database must be designed. The information needed to classify data values may be included on a data entry form or be available as part of the data collection process in a point-of-entry system. A well designed user interface for data entry is necessary to guide the process of category selection.

An example of crisp data update to a fuzzy domain is data collection under circumstances that indicate uncertainty about the accuracy of the data as it is collected or measured. In this case the crisp data values are entered with a weight of certainty. The alternative is to enter this data into a crisp domain with a qualification warning that the data is as accurate as possible, but the level of this accuracy is not known.

Data can be missing because it is not yet available, does not exist, or is not applicable. In this case, a data entry option that allows a selection among classifications is necessary. The categories are represented by fuzzy-typed attributes and should be positive (i.e. with a weight of 1 as the typical case) such as PRESENT vs. missing, KNOWN vs. unknown, AVAILABLE vs. unavailable, or APPLICABLE vs. not applicable with the choice of category used to calculate membership weight indicating the state of data completion as COMPLETE. An alternate approach is to rely on a sentinel value entered in place of the missing data and used to determine status of data completion.

3.4.3.1 A Simple Example

The business contacts application is modified to require the data entry operator to classify the business using the information on the business cards both printed and written by the salesman. The database administrator decides the need for an opinion from the data entry operator requires that fuzzy data be stored in the database. A fuzzy classification to measure the uncertainty of this opinion is associated with the kind-of-business attribute. Expectations are that in most cases there is no uncertainty so the default membership weight for uncertainty is 0. The classification is changed to measure certainty and named CERTAIN. The default membership is 1.0, but the data entry operator may adjust this to any value on the interval (0,1]. There is no list of

appropriate categories for the attribute named KIND-OF-BUSINESS so the database administrator makes a snap decision to add another fuzzy classification. The prevailing opinion of the data entry operator is accepted, but if necessary, a sentinel value of unknown or missing can be entered for the kind of business. This fuzzy classification for unknown or missing data and is associated with KIND-OF-BUSINESS using the name UNKNOWN with a default membership weight of 0. The data entry application sets UNKNOWN to 1 if the KIND-OF-BUSINESS value is unknown or to 0.5 for the value missing. If the missing data becomes available, a search for unknown values can match on the UNKNOWN membership weight 1 and data is updated. The data source is crisp, but the domain is made fuzzy by the addition of subjective data to the database.

3.4.3.2 Height Example

Data entry accepts the height in inches for a male subject age 20 and over from a verbal report. The report may be from a technician who used a precise measurement device to measure the individual or from clerical staff who asked the individual about his height. It is necessary to ask about measurement accuracy and include an accuracy membership weight with the height data.

Data entry is required to ask a series of questions and set the accuracy membership weight according to the answers and several rules. “Did you measure the individual with an instrument?", “If not, did you estimate height or ask the individual?", and “If you asked the individual his height, was he standing up and does he appear to be that tall?" An accuracy weight in the interval (0,1] is determined from these questions and used to valuate the CERTAIN fuzzy-typed attribute associated with the precise height value in the database.

3.4.4 Fuzzy Data to Fuzzy Domain

Fuzzy domains are designed to store fuzzy data in a way that preserves the meaning of the data and minimizes loss of information. A database designed for fuzzy domains that are compatible with the application requirements for data representation can support any and all fuzzy data types including partial classification, multiple classifications, fuzzy numbers, and linguistic variables. Fuzzy data entry requires methods that can both store values in fuzzy type data structures and map these values to fuzzy membership weights.

3.4.4.1 A Simple Example

The business contacts application is enhanced to reduce the number of business classification errors, correct a fuzzy database design flaw related to unknown data, and improve the representation of sales estimates using a genuine fuzzy number in place of crisp values.

The data entry operator now classifies the business type using information from the business card and a pull-down list of acceptable classifications derived from the database administrator's domain analysis of customers. This list of categories eliminates confusion created by typing errors, synonyms, and conflict between the singular and plural. The certainty of a classification for KIND-OF-BUSINES selected from the pull-down list is a default CERTAIN weight of 1.0, but the data entry operator may adjust this to any value in the interval (0,1]. If the kind of business is unclear or the information is missing, radio-buttons are used to select a either unknown or missing instead of a classification from the pull-down list.

Implementing this change allows the database administrator to correct a database design flaw introduced with the fuzzy typed-attribute UNKNOWN. The membership weight for an unknown

data value is 1 which means that the membership weight for a known data value is 0. While this seems logical, in the context of the relational model and crisp set theory, a value that is in the database has a weight of 1 and values not present have a weight of 0. To correct this problem, the UNKNOWN category is replaced with a fuzzy-type attribute named KNOWN and the database weights are changed. If a KIND-OF-BUSINESS is selected from the pull-down list the default KNOWN weight is 1.0. If the unknown button is selected the KNOWN weight is set to 0.1, the database administrator chose this value because it is a low value in the interval (0,1]. If the missing button is selected the KNOWN weight is set to 0.7. These weights are less than one and can be matched by queries and updated if the missing information becomes available.

The estimated sales data introduced to the business contacts application is enhanced to use a fuzzy-number attribute in place of the crisp attributes representing the estimate. To management's surprise the salesmen were correct and the best estimate of sales for the years is an approximate value between the estimated minimum/maximum. The database administrator replaces the attributes for minimum expected sales, maximum expect sales, and estimated sales with a fuzzy number that uses a triangle function to represent the sales estimate. The triangle function for each customer in the contact database uses the minimum and maximum range as the base of the triangle on the x axis and is normalized such that the estimated sales has a membership weight of 1. The salesmen are asked to enter all three values on the business card, but if the range is the best estimate, the mid-point value is used as the estimate. Using a fuzzy number supports sales queries using approximate numbers and fuzzy arithmetic that can sum the estimated sales for multiple customers giving another fuzzy number.

The business contacts database is easy to use and provides unexpected useful information. The data source and the domain are both fuzzy.

3.4.4.2 Height Example

Fuzzy sets are defined using crisp data from the National Health Statistics Reports[27,p.16] to develop a fuzzy height domain for men age 20 and over, the fuzzy sets defined from this data are used to derive partial classifications, multiple classification, fuzzy numbers and linguistic variables that support fuzzy data update for precise data from overlapping ranges.

Once defined, the meaning of fuzzy terms or values and the significance of fuzzy set membership weight must be understood in the context of the domain defined for the fuzzy data. This understanding can be established by a meaningful crisp attribute name (i.e. HEIGHT), an association with relevant fuzzy-typed attributes (i.e. CERTAIN and KNOWN), and metadata that defines the fuzzy domain (i.e. as a linguistic variable). The attribute HEIGHT is a crisp attribute on the domain of positive floating point numbers in the range of possible measurements of stature for adult men. Additionally, it is a fuzzy number based on the range of heights from the linguistic labels (i.e. short, average, and tall) from Table 4 and a linguistic variable defined by the membership functions for short in Equation (3.2), average in Equation (3.4), and tall in Equation (3.5). These functions are virtual fuzzy sets represented as metadata and stored in the database system catalog. The use of linguistic labels for data entry requires the HEIGHT be associated with a fuzzy-typed attribute CERTAIN with partial membership in this category to measure any uncertainty implied by imprecise input and to support vague query. If it is possible for HEIGHT to a valid unknown value, it must also be associated with a fuzzy-typed attribute named KNOWN with partial membership.

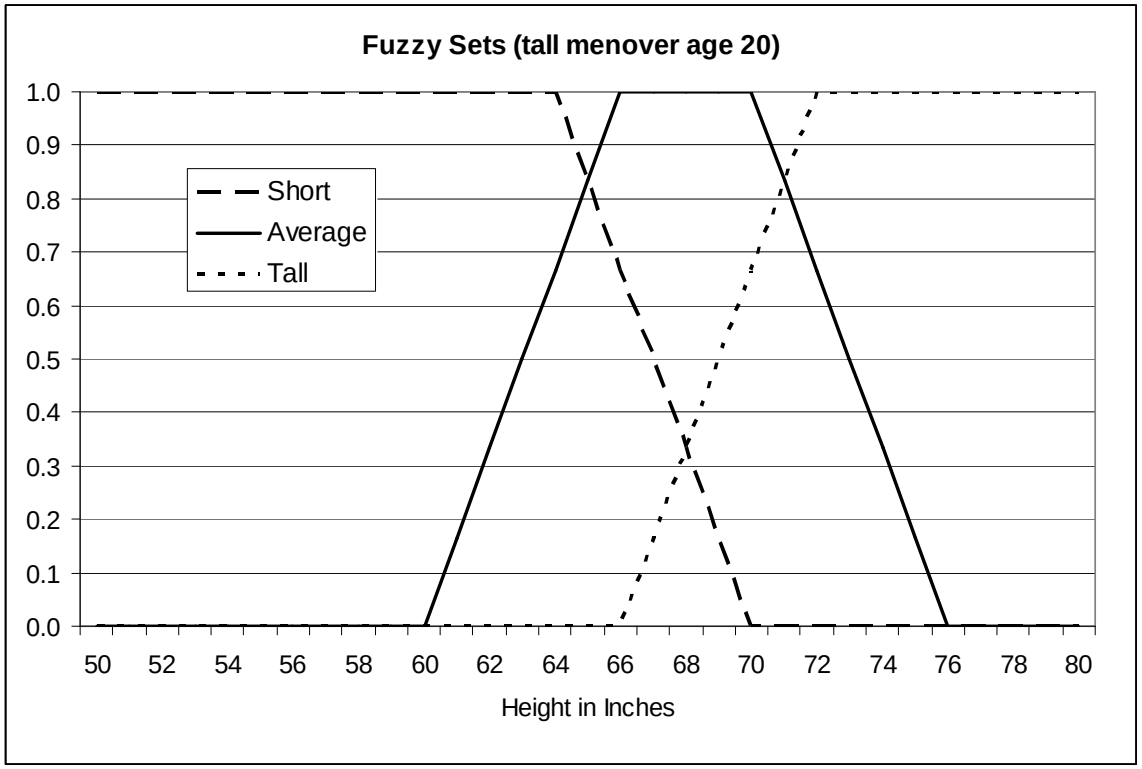


Figure 3.3: Height Men over Age of 20.

The membership function for the fuzzy set of short men over the age of 20 shown in Equation (3.2) is plotted as half of a trapezoid over the scale of height in inches in Figure 3.3. The short membership weight at 60 inches is 1 and it falls to 0 when height reaches 69 inches.

$$\text{short}(x) = \begin{cases} 1 & \text{when } x \leq 64 \\ (70 - x) / 6 & \text{when } 64 < x < 70 \\ 0 & \text{when } x \geq 70 \end{cases} \qquad (3.2)$$

The membership function for the fuzzy set of men of average height over the age of 20 shown in Equation (3.3) is plotted as a trapezoid over the scale of height in inches in Figure 3.3. The average height membership weight from 69 to 71 inches is 1 and it falls to 0 as height either decreases or increases.

$$\text{average}(x) = \begin{cases} 0 & \text{when } x \leq 60 \text{ or } x \geq 76 \\ (x - 60) / 6 & \text{when } 60 < x < 66 \\ (76 - x) / 6 & \text{when } 70 < x < 76 \\ 1 & \text{when } 66 \leq x \leq 70 \end{cases} \quad (3.3)$$

The membership function for the fuzzy set of tall men over the age of 20 shown in Equation (3.4) is plotted as half of a trapezoid over the scale of height in inches in Figure 3.3. The tall membership weight at 80 inches is 1 and it falls to 0 as height falls to 71 inches.

$$\text{tall}(x) = \begin{cases} 0 & \text{when } x \leq 66 \\ (x - 66) / 6 & \text{when } 66 < x < 72 \\ 1 & \text{when } x \geq 72 \end{cases} \quad (3.4)$$

3.5 Four Kinds of Database Query

The impact of fuzzy search on the data query processor may be significant. The complexity of data query is increased by the need to evaluate fuzzy-typed attribute weights in the results. The semantic content of fuzzy data stored in a relational database using data structures designed to represent the components of each fuzzy type (i.e. partial classification, multiple classification, fuzzy number, or linguistic variable) must be evaluated by the query processor and compared to both precise and vague search terms. Search effectiveness and consistency can be enhanced by using parameters stored in the database catalog instead of being encoded in the application. These search techniques must interpret database attribute values and associated membership weights for presentation. An alternate approach is to let the database user rely on intuition to evaluate results. For example, membership weights for data certainty and completeness may be self-evident.

3.5.1 Crisp Query of Crisp Data

A exact match of data query arguments to precise data stored in the database is the typical crisp case. If the search criteria are accurate and complete, a corresponding crisp result set is expected. If the search criteria are terms that the user believes may match in some cases, the search criteria are incomplete, but a useful match on one or more term is possible.

3.5.1.1 A Simple Example

The business contacts database from Section 3.4 is queried using a value to match an attribute of interest. This could be the last name of the contact or the name of the organization. The query and results are crisp. If there is not an exact match, there are no tuples in the result.

3.5.1.2 Height Example

The user queries a database of males age 20 and over for those who are 72 inches or greater in height. There is no expectation of error in the database. A result set of those men who were precisely measured at 72 inches or taller are founds and included in the query results.

3.5.2 Fuzzy Query of Crisp Data

The goal to support fuzzy query of crisp data requires a retrofit to the classical database. The capability of vague search depends on an understanding of the nature of the data and if this has not been built into the database by design, it must be implemented in the application's query programs. This may be done by adding fuzzy-typed attributes to represent classification with partial membership for categories such as certainty of data accuracy or data completeness. Another approach is to create tables of linguistic labels that represent ranges of data values to support a more user friendly interface. In both cases the association between the crisp data and the fuzzy retrofit must be implemented in the application.

3.5.2.1 A Simple Example

The business contacts database from Section 3.4 is queried using a stored procedure. This procedure is a search for approximate estimated sales as a match to the value of its argument. The query compares the argument value to the minimum sales, estimated sales, and maximum sales in the business contact database and returns all tuples where the argument falls within a value range. The results set is ranked using the range of estimated sales to maximum sales expected and then using the range of minimum expected sales to estimated sales. This is a simulated fuzzy query that returns all estimates that are close to the search argument. Most of the sales estimates are mid-points between the minimum and maximum and give very rough estimates. Totaling or averaging sales gives approximate results. The crisp data in the database represents fuzzy data.

3.5.2.2 Height Example

A person's height can be measured precisely and verified. If they turn out to be taller or shorter, there may be error in the measurement process rather than the measurement system. There is no judgment involved in determining a person's height when a calibrated measurement device is used.

There is uncertainty when the issue is whether a person who measures 6 feet in height is tall. The answer varies widely given a person's perspective. A community may have a consensus, but this is only a combination of individual opinions. A universal consensus is unlikely because the local definition of tall will depend on the average height of the individuals in the community. If the community is short then “average” height may be considered tall.

It is possible for the members of the community to change. If the members of the original community tend to be short, the community conception of tall may increase as the population of the community becomes taller. It is reasonable to expect the community to change its opinions as its membership changes; therefore it is futile to define a static definition for tall.

It is important to include the perspective of the observer in the process that defines what is considered tall. It is also important to respect the lower and upper limits of human height. In Figure 3.3, it can be seen that a male with a height of 67 inches or greater is considered to be tall. At a height of 72 inches a male is tall with a membership weight of $\mu_{TALL}(x) = 1.0$, but those of average height may have some degree of membership in the tall classification. The cross-over point for the average height and tall membership functions is at 74 inches where $\mu_{AVERAGE}(x) = 0.83$ and $\mu_{TALL}(x) = 0.83$. This means that at a height of 71 inches a male may be considered to be of average height with a weight of 0.83 and tall with a weight of 0.83. Membership is fuzzy at the cross-over point allowing membership equally in multiple classifications.

To allow a search using a descriptive phrase such as a “tall adult male," a synonym table must be provided for the user interface so that fuzzy query terms can be selected and used to create a phrase that can map to a crisp value. The synonym table includes search terms and α-cuts which are values calculated from the height values close to the meaning of the term. Once an appropriate α-cut is identified the database may be search to find values equal to or greater than the α-cut which will be included in the results relation.

Additionally, modifiers such as “very” could be factored into the degree, modifying the value of membership function of height at its extremes, to further refine the weighted response. For example, VERY(x) = $\mu_{INT(HEIGHT)}$(x) uses an linguistic modifier to concentrate height towards the greater values of μ_{height}(x). This technique allows “very tall” to be included in the synonym table and associated with an α-cut of greater value that that for “tall.”

3.5.3 Crisp Query of Fuzzy Data

A precise query for fuzzy data may be the case of using an ad hoc query to examine data and resolve an anomaly in the database or a problem in the application. It may also be the case of a user checking a fact with a general purpose data query tool. The search criteria are precise terms the user expects to match a particular tuple in the database. Retrieving this data value for interpretation is the purpose of this kind of query.

3.5.3.1 A Simple Example

The business contacts database from Section 3.4 (Crisp Data to Fuzzy Domain) is queried using general purpose tool included with the database management system. The database administrator needs a report of the KIND-OF-BUSINESS attribute values to conduct a domain analysis of customers and wants to evaluate the data entry operator's level of certainty associated fuzzy-typed attribute CERTAIN. The crisp query is ordered by the values in KIND-OF-BUSINESS and the fuzzy membership weights are interpreted by the database administrator as values on the interval (0.1].

3.5.3.2 Height Example

The user queries a database of males age 20 and over for those who are 72 inches or greater in height. There is no expectation of error in the database. A results set of those men who were

precisely measured at 72 inches or taller is found and included in the query results. But the attribute HEIGHT in the database is fuzzy and includes a fuzzy attribute UNCERTAINTY because the data entry process allowed either height in inches or a descriptive phrase as input. The data entry user interface relied on fuzzy constraints to enter the data and determine its membership weight in UNCERTAINTY. If the user query requests all attributes, there is a column in the results table indicating the certainty that the height is accurate.

3.5.3.3 Issues and Expectations

By its very nature, a precise query set expects a precise response set to be returned. If a user asks for students taking Course 310, the user will expect to see these students and only these students contained in the response set. It would be inappropriate to also include students enrolled in course 311 and associate these students with a weighted possibility.

Nor would it be appropriate to return a set containing a value of anything other than 'Large' if the user selects boxes where size is equal to 'Large'. While the characteristic 'Large' may be seen as subjective and its application uncertain, it would not be stored or maintained as a subjective value in a set. Seemingly subjective values such as Large, Medium and Small would be assigned within the set based on certain quantifiable and community agreed upon criteria. While the concept of size is thus 'fuzzy', its application within the set is not. Size is, in fact, given certainty by its assignment of a value. In its use, the query for a box where the seemingly subjective size is 'Large' would be just as definitive as a query where the course equals '310'. While the data is inherently uncertain in its definition, that definition is given crisp form by its application as a value in a set.

Another example of this type might be that of an asset class. An asset class might contain a value of `Boat', `Car' or `Truck'. Individuals can debate what characteristics uniquely and indisputably qualify an asset to be a `Car', but the fact that the characteristics are debatable implies uncertainty. For example, there is a vehicle called a 'Duck' that was designed to go off of the road and into the water and propel itself like a boat. Then there is the popular American classic automobile, the El Camino, which was distinguished by appearing to have all the characteristics commonly associated with a `Car' but possessing a truck bed. So the question arises, is the Duck a `Car' or a `Boat'? Is the El Camino a `Car' or a `Truck'? Well, when a user extracts from a set all assets of class `Car', the system will return all those assets which have been assigned a class value of 'Car'. The uncertainty is within the user's mind only. The uncertainty generally does not exist within the elements of the set. But what if the system were to take the preceding example and assign a membership weight to the value contained in the CLASS attribute? If it were determined that the El Camino is 0.6 a `Truck' class asset and 0.9 a `Car' class asset, and request all assets where the class is equal to `Car', one might get something of a fuzzy set as represented in Table 3.5.

Asset	$\mu_{car}(x)$	Classification
Bentley	1.0	Car
Thunderbird	1.0	Car
El Camino	0.9	Car
Duck	0.6	Car
F-150	0.0	Car

Table 3.5: Cars

Would this result be meaningful? Would it have a practical application? In such a case, the possibilistic value would need to be assigned by the community up front. The decision would need to be made and a value assigned that an El Camino has a 0.9 membership in the `Car' class.

Or, taken from the other side, there is a 0.1 possibility that it is not a `Car'. The result set would then be ranked by its membership weight, this time in decreasing order of likelihood.

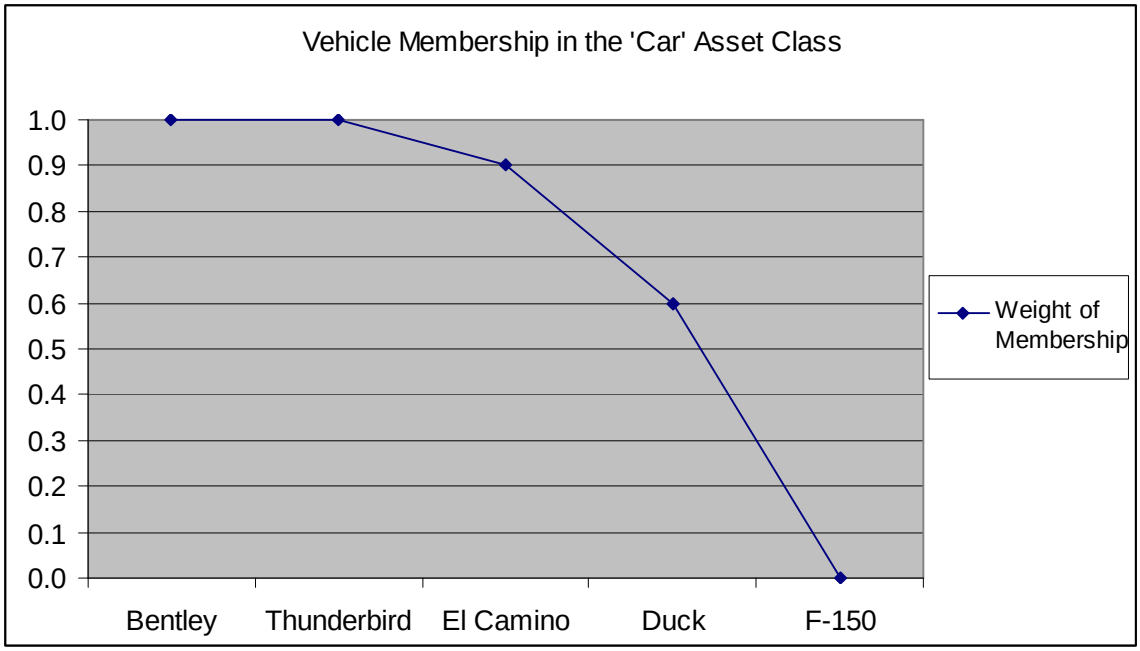


Figure 3.4: Classification of Cars and Trucks

While the graph shows the F-150 to have a membership within this set of 0.0, it could be argued and agreed to by the community that the F-150, though a 'Truck', does transport passengers and so is, to some degree of truth, a car.

So there is a challenge with assigned and static membership weightings relating to the complexity of maintaining the data. This example is a fairly easy one. The task of establishing and maintaining a 'possibility of truth' for all values and all permutations at first blush seems unreasonable. Suppose there are asset classes for car, truck, boat, machine, computer (automobiles have computers contained within them) and then these are categorized as light

weight, heavy weight, medium weight and the scope of the exercise and the system gets a little out of hand, particularly when these weights are reviewed and reevaluated.

3.5.4 Fuzzy Query of Fuzzy Data

Fuzzy queries are designed to match the information encoded in structures designed for fuzzy data types including partial classification, multiple classifications, fuzzy numbers, and linguistic variables. Fuzzy queries require methods that can map search criteria to values in fuzzy data structures, retrieve these data, and interpret fuzzy membership weights in a meaningful way. The fuzzy database and query application interface must be designed to be compatible with application requirements. Adequate data representation is a prerequisite for good data presentation.

3.5.4.1 A Simple Example

The business contacts database from Section 3.4 (Fuzzy Data to Fuzzy Domain) is searched using a query program developed for the business contacts application. The database user interface form includes each database attribute, but matches using only the query arguments entered in this form. A sample query returns the contact name and phone number of a specific type of organization (i.e. KIND-OF-BUSINESS) selected from the pull-down list if the expected sales for the year are approximately a million dollars or more with a possibility of 0.9 or greater.

The search returns three tuples that represent the proposition interpreted as “the business contact NAME can be reached at PHONE NUMBER where approximately ESTIMATED SALES worth of sales with a possibility of SALES-WEIGHT are expected.” These three tuples are ranked using an aggregation of the membership weights for all crisp and fuzzy attributes in the search.

This aggregation is interpreted as the overall truth represented its associated tuple (i.e. proposition).

3.5.4.2 Height Example

Fuzzy sets are defined using crisp data from the National Health Statistics Reports [27,p.16] to develop a fuzzy height domain for men age 20 and over, the fuzzy sets defined from this data are used to derive partial classifications, multiple classification, fuzzy numbers and linguistic variables that support fuzzy queries such as “above average height” and “maybe tall” for use with precise and fuzzy data that overlaps in ranges. This example may be identical to the Height Example in 4.4.2 (Fuzzy Query of Crisp Data).

Chapter 4 - Modeling Fuzzy Relations

4.1 Overview of Entity Relationship Modeling

A relational database is composed of individual entities and the relationships that exist between them. Each entity and relationship is defined by the attributes contained within the entity or relationship which will evolve into the relational variable or 'relvar' and the data model associated with the entity or relationship. In order to communicate the characteristics of a relational database, the entities, the relationships and their attributes, a modeling methodology is useful. Various symbologies have been devised with the first being proposed by Dr. Peter Chen.[28] Together these symbols are used to construct an 'Entity-Relationship' (E-R) Diagram. While the specific characteristics and symbols used can have a rather personal styling, there are some common elements essential to all E-R diagrams as described below.

4.1.1 Entity or Entity Set

An entity is something that exists and has properties such as an employee, a supplier or an asset. Similar entities form an entity set. There is a predicate associated with entity sets that we can use to test whether the entity belongs in a set. For example, we know that if a supplier is in the entity set 'Supplier' that it contains all the essential properties associated with the entity supplier. Let E_i denote an entity set and e_i an instance of an entity within the set. A supplier may be contained within E_i as a parts supplier or as a supplier of some other good or service that will be represented in the database.[28] But both suppliers have common properties and both, despite their different role, would be contained in the entity set E_i. Such a representation might be described using an 'ISA' representation where PARTS_SUPPLIER is a SUPPLIER and MATL_SUPPLIER is a SUPPLIER and both share common essential attributes.

4.1.2 Weak Entities

A weak entity is an entity that contains information that can not exist without the existence of in another, *owner* entity. For example, an entity 'Child' would be weak, since without a related 'Parent' owner entity no instances of children in the Child entity could exist. In all other respects, however, weak entities maintain the same characteristics as any other entity.

4.1.3 Relationship or Relationship Set

Relationships consider the associations between entities.[28] The *role* of a relationship is the function that that relationship plays between other entities. A supplier 'supplies' parts or an employee 'works in' a department. Formally, let E_i again denote an entity set. Let e_i denote an instance of E_i. The role of an entity can be stated as r_i / e_i where r_i is the role of e_i in the relationship.

4.1.4 Weak Relationships

Weak relationships associate weak entities with their owner entity.

4.1.5 Attributes

Attributes and their values are represented by attribute/value pairs. The attributes themselves describe the essential information specific to an entity or a relationship such as 'supplier number', 'supplier name' or 'supplier address'. The values are the values associated with these attributes in an instance of the entity or relationship. As Peter Chen puts it, "An attribute can be formally defined as a function which maps from an entity set or a relationship set into a value set or a Cartesian product of value sets."[28]

$$f\text{: } E_i \text{ or } R_i \rightarrow V_i \text{ or } V_{i1} \text{ x } V_{i2} \text{ x } \ldots V_{in} \qquad (4.1)$$

4.1.6 Derived Attribute

A derived attribute is one which does not formally exist as an attribute associated with an entity or a relationship and whose value is not physically stored in a table. It can be represented as an attribute, but its value will be derived from another attribute or attributes. For example, a person's age can be derived from subtracting a person's birthday from a given date. As such, there would be no need to for the database to store 'age' as an attribute within a table. The benefit of such a derived attribute is that its value is not necessarily static. As with age, if a person was born on August 26, 1960, they would be 50 years old on August 26, 2010. On that day, the derived attribute 'age' would return the value '50'. If the system maintained the age as a static value, the age would not be correct in 2011 unless updated. Using a derived attribute contributes to the accuracy and the efficiency of the attribute 'age' in the system at both the logical and the physical levels.

4.1.7 Functionality

The functionality of a relationship between entities relates to the relative number of instances that these entities can share. For example, many employees may work in many departments resulting in a many to many functionality. In another example, one department may be managed by only one employee. Such a one to one circumstance is denoted with a functionality of 1:1. In yet another example, employees may be assigned to work in accordance with a set number of occurrences, for example, 1, 2 or 3 departments. This circumstance would be denoted as n:m. Within the entity relationship diagram, functionality can be represented in many different ways. However, for the purposes of this dissertation, the following designations will be used to represent functionality:

1 – One and only one instance within an entity.
n – A number > 0.
m – An indeterminate number from 0 to *x*.

4.1.8 Entity Relationship Component Representation

Each of the components described above are represented in their most basic form as shown below.

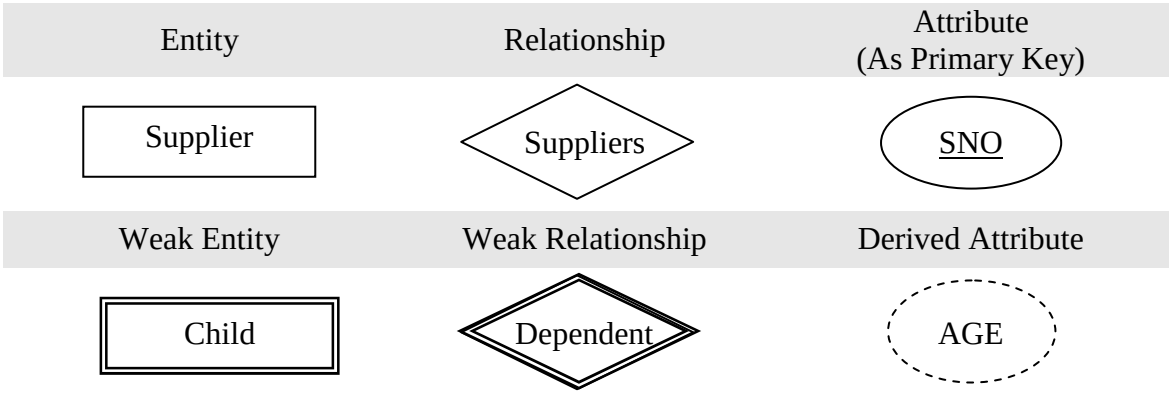


Figure 4.1

Putting these components together, a simple E-R diagram can be illustrated as follows.

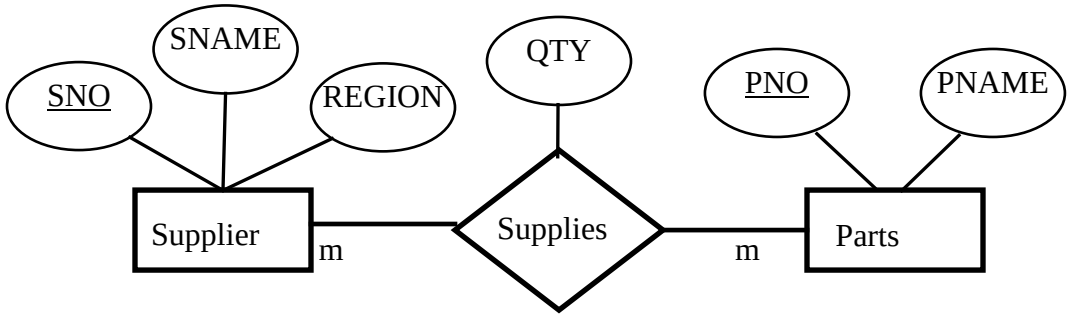


Figure 4.2

The diagram represented above is composed of

- two entities, ‘Supplier’ and ‘Parts’,
- their keys are denoted by an underlined fields, SNO and PNO,
- the relationship between them is ‘Supplies’ which is the role that the supplier has with the parts,
- the attributes of each relation and
- the functionality of the relationship such that ‘many’ suppliers supply ‘many’ parts.

The relation for the relationship ‘Supplies’ contains two implied attributes SNO and PNO which are traditionally not shown on the diagram and one additional attribute that describes the quantity (QTY) of the PNO supplied by the particular SNO. The implied attributes would combine to represent a composite key {SNO, PNO} within the relation.

It should be noted that the process of designing a well thought out E-R diagram will generally result in a database whose relations will be fully normalized and the relvars that eventually evolve from the entities and their relationships can be joined losslessly.

4.2 Extending the Entity Relationship Diagram

One advantage of the relational database model as represented in the E-R diagram is that it clearly relates the most salient aspects of the entities, relationships and attributes that make up the database being modeled. A key element of the E-R diagram is that it can be used to communicate the database being modeled not just to data administrators and practitioners, but to end users as well. There is a tendency to over complicate the E-R diagram by injecting more and more information so that it becomes extremely complicated and confusing. So there is a question as to what is appropriate and what is too much. Is the symbology clear and intuitive and does it represent the essential elements of the database design?

When the relational model is extended to include the concepts associated with fuzzy set theory to create an Extended Entity Relation (EER) diagram, this question must be revisited and given serious consideration. Much has been written and proposed as to how best to represent a fuzzy database in the entity relation diagram. The sections below describe some of these approaches and concludes with the approach that seems most appropriate for this research.

4.2.1 Fuzzy Entity Relationship Considerations

An attribute may be inherently fuzzy or an attribute may be crisp, but treated and used as a fuzzy attribute. If an entity contains a fuzzy attribute, is the entity itself fuzzy or is the 'fuzziness' limited to the attribute? There are a number of particular characteristics associated with a fuzzy data object. How can these characteristics be communicated clearly?

4.2.1.1 Zvieli and Chen Approach

Zvieli and Chen [29] offered the first approach to the representation of fuzzy data. This approach consists of three basic levels and is simple. It applies a fuzzy notation at the entity, relationship and attribute levels as appropriate to differentiate them from crisp components. The main criticism of this initial approach is that it fails to take into account the various 'meanings' of fuzziness such as membership, importance and so on which will be discussed in Section 4.2.1.3.

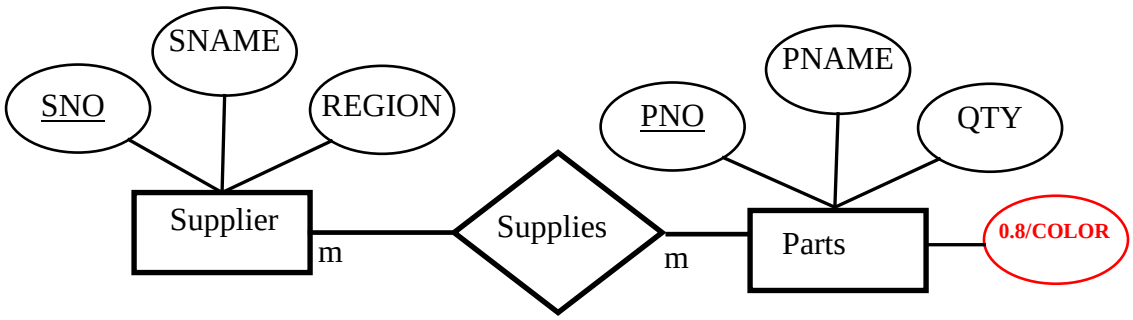


Figure 4.3. The Zvieli and Chen Approach

As can be seen in Figure 4.3, fuzzy attributes or other EER components are recognized by the inclusion of a fuzzy weight paired with the component name. The attribute 'COLOR', for example, is shown as an attribute with a 'weight' of 0.8 in the entity 'Parts'. Such a weight could represent the level or degree of importance of the COLOR attribute in the entity Parts. In this instance, the weight of importance would apply to all values contained in the COLOR attribute rather than be individually applied to each tuple instance within the relation as would otherwise be the case.

4.2.1.2 Chaudhry, Moyne and Rundensteiner Approach

Chaudhry, Moyne and Rundensteiner [30] proposed an extension to the E-R model that incorporates the conversion of crisp data into fuzzy data by defining linguistic labels as *n* fuzzy sets over the universe of an attribute. Data contained within a crisp tuple is transformed into a fuzzy tuple with a linguistic modifier applied to the value and the degree of membership associated with it. The, now fuzzified, entity is represented as a separate entity with its data values stored or represented separately from the crisp representation.

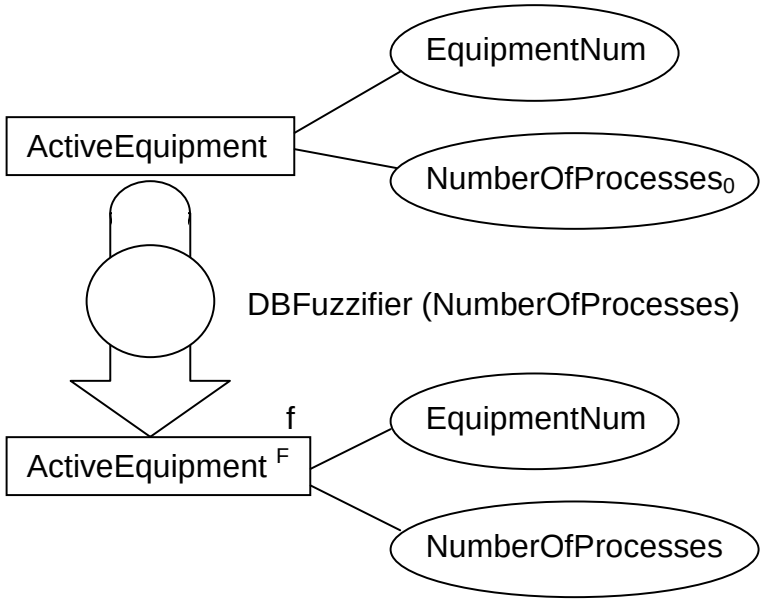


Figure 4.4 – DB Fuzzifier Transformation

As shown in Figure 4.4, the specific number of processes may be known. A piece of equipment may have twenty steps to an assembly process and a factory may have any number of such machines. At any point in time, the database may store the actual number of processes taking place. In a crisp database, a report may return this information to the operator. A point in time query specifying the predicate WHERE NUMBEROFPROCESSES = 10 might return only one equipment number. But if the query were 'fuzzified' to communicate the request as 'WHERE NUMBEROFPROCESSES <is about> 10', the query might return 3, 4 or any number of

equipment numbers whose current number of processes was ‘about 10’. The question, then, is what does ‘about 10’ mean? How would the result set be arrived at? And this would be a question that the implementation team would need to address. For purposes of an extended E-R diagram, however, all that is necessary is to communicate the fact that there is an expectation that crisp data will be fuzzified. Questions of ‘how’ and ‘to what extent’ would be left to the implementation phase.

This particular contribution holds significant promise as research into the access and use of crisp data within a fuzzy context is important.

4.2.1.3 Galindo, Urrutia and Piattini Approach

In 2006, Galindo, Urrutia and Piattini not only addressed the confusion introduced by Ma, Zhang, Ma and Chen, but introduced several new concepts inherent in the use and application of fuzzy data. [31]

4.2.1.3.1 Fuzzy Attribute Types

Galindo et al proposed four fuzzy attribute types, each with differing characteristics. This data, as represented in an EER diagram as attributes, can be broken down as follows.

4.2.1.3.1.1 Type 1

Precise Data: Precise data is crisp data that can be fuzzified. For example, the crisp value held by an attribute Height = 72 might be made fuzzy by the application of the linguistic label ‘Tall’ and assigned a weight of membership. Another example might be a search for a 2 inch bolt. The predicate ‘Where length = 2’ against a crisp data set would return only bolts where length is, in fact, ‘2’. But the expectations and the concept might be extended to ‘about 2 inches’ which might return not only bolts with a length of 2 inches, but those that were ‘about’ 2 inches with an

appropriately applied weight of membership. This type of data can be used to extend traditional relational databases as fuzzy databases through the use of crisp data as fuzzy.

4.2.1.3.1.2 Type 2

Imprecise Data Over an Ordered Referential: This data type can contain both crisp and fuzzy data but is essentially an extension of Type 1 in the form of a possibility distribution over an ordered domain. For example, “The vehicle weighs approximately 3,500 lbs.” While the actual weight may be known, the concept of approximation, particularly when grouped with other vehicles of approximately the same weight is significant.

4.2.1.3.1.3 Type 3

Data of Discreet, Non-Ordered Domains With Analogy: This data type represents the more common concept of fuzzy data by dealing with data values that are similar, but by their imprecision, not the same. For example, ‘Blonde’ or ‘Brown’ hair each describes a similar color, but while variations make the colors similar, they are not necessarily equal to one another. The attribute type also lends itself to the possibility distribution or degree of membership within the set as μ_{COLOR}(‘*Blonde’*) → 0.6/Blonde.

4.2.1.3.1.4 Type 4

This attribute type is similar to Type 3, but removes the requirement that the attributes be evaluated based on their similarity. For example, a vehicle may have a particular role within a fleet of vehicles, but its role is not evaluated with respect to its similarity or dis-similarity in relation to other vehicles.

4.2.1.3.2 Representing Fuzzy Attributes

The confusion inherent in the EER symbology provided by Chen can be addressed by Galindo's suggested alternatives for the representation of fuzzy attributes and the type with which they are associated.[11] The following table combines these two aspects of fuzzy data representation.

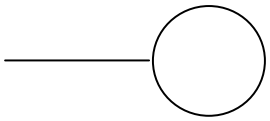
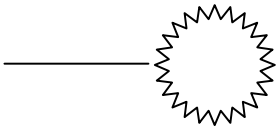
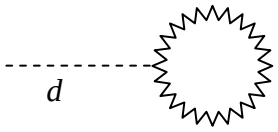
Type	Representation
Type 1 Simple	
Fuzzy attribute, Type in were $n \in \{2, 3, 4\}$ simple.	
Derived fuzzy attribute	

Figure 4.5

The following illustration presents a simple example of these symbols in use with an entity. The diagram illustrates an example of an employee entity composed of a crisp employee identification number 'EmplID' attribute, a crisp 'Height' attribute and three fuzzy attributes, 'Age', 'Hair' and 'Eyes' with their associated fuzzy classification. Age is of fuzzy type T2, imprecise data over an ordered referential. Hair is of fuzzy type T3, which represents data of discreet, non-ordered domains with analogy. And eye color is of type T4, which is similar to type T3, but removes the requirement that the attributes be evaluated based on their similarity.

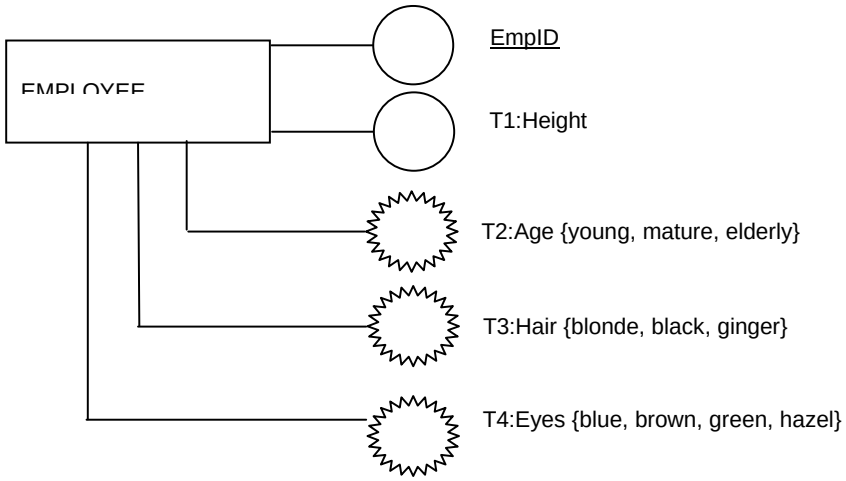


Figure 4.6

4.2.1.3.3 Representing Fuzzy Degrees

To this point, fuzzy attributes have been considered to represent the degree of membership within a fuzzy set. This representation is valid but may be too general. It should be noted that there are other, more granular representations of this degree.[13]

Membership Degree The membership weight designated as G^0.

Fulfillment Degree The property that a certain attribute can fulfill a requirement to a degree between two extremes is designated G^1.

Uncertainty Degree The property that we are certain that the value represented is accurate. This is designated G^2.

Possibility Degree The measurement or degree that the value represented is possible is designated G^3.

Importance Degree Different attributes can have differing levels of importance. This importance is designated as G^4.

While the common expectation and standard is Membership Degree, it should be considered significant that if one degree type can be represented, other degree types can be represented for the same attribute to give the representation even more depth and meaning.

4.2.1.3.4 How Degrees Are Assigned

A degree, whether it be membership, possibility, or importance must be assigned if the attribute is going to have any meaning. This degree can be arrived at in one of two ways.

Derived – A degree that is derived is determined based on a function. For example, if the a bottle is said to be 'half empty', the degree of membership for the volume contained within a bottle is determined based on the parameters established for 'half-full' and the volume contained within the bottle. This degree of membership can be calculated and a value assigned. For example:

$$\text{half\_full}(x) = \begin{cases} 0 & \text{when } x = 0 \\ x / 10 & \text{when } 0 < x \leq 10 \\ (20 - x) / 10 & \text{when } 10 < x < 20 \\ 0 & \text{when } x = 20 \end{cases} \qquad (4.3)$$

Non-Derived – A degree that is non-derived is one that has been assigned. The degree is typically maintained as a crisp data value within the database and is returned with the corresponding attribute value as its degree of membership.

4.2.1.3.5 Fuzzy Entities

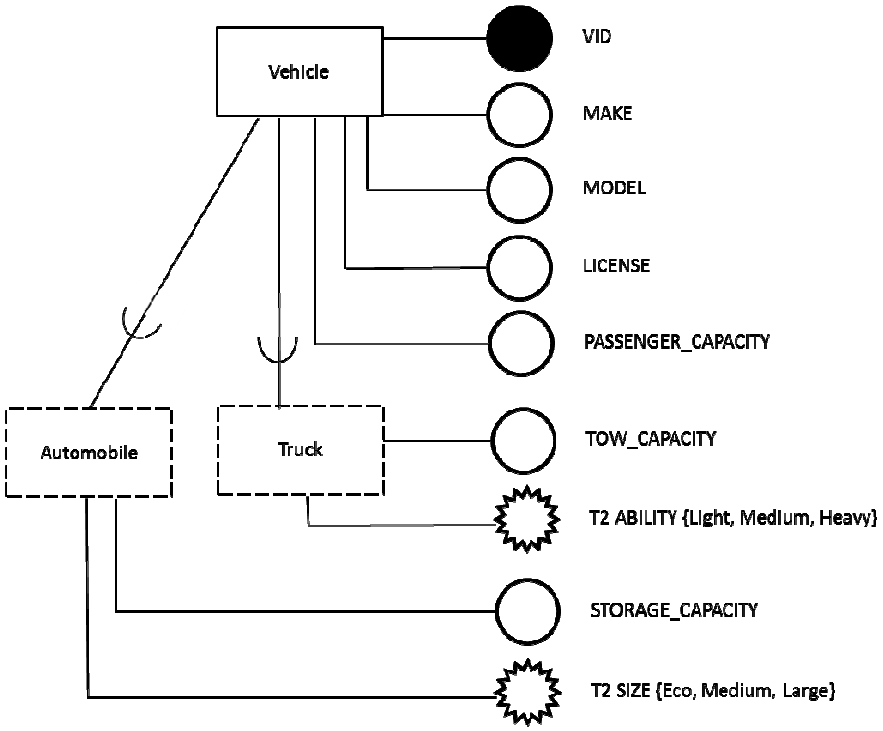
Entities can be fuzzy objects in much the same way that an entity can represent crisp objects. The distinction lies in the fact that a fuzzy entity will possess a degree of membership as an entity whether its attributes are crisp or fuzzy. For example, a sub-type of 'Vehicle' may be 'Truck'. This sub-type may be a fuzzy data object as the actual 'Truck' may possess a weight of membership in the truck class of 0.6. Formally, let E be a fuzzy entity and e be an instance of E.

$$\forall e_i \in E \text{ with } i = 1, 2, \ldots, n, \mu_E(e_i) \in [0, 1]\} \qquad (4.4)$$

A fuzzy entity is represented by a box framed by a dashed line rather than a solid line. To continue with the vehicle entity just described, the representation looks like Figure 4.7 as the relationship model shows vehicle types ‘Automobile’ and ‘Truck’. There are a number of notable aspects to this representation.

1. A vehicle has an ‘ISA’ (casually thought of as ‘is a’) relationship with the TRUCK class (‘truck’) to a degree of membership, but the same vehicle could also have an ISA relationship as an AUTOMOBILE (‘automobile’) with an appropriate weight of membership. Simply put, a vehicle could be an automobile *and* a truck, but to the same or different weights of membership. This ability to exist in both domains at the same time is significant.
2. It is further significant that an object’s existence in one domain will generally have no effect on the object’s existence in another. In other words, a vehicle’s existence as an ‘automobile’ is independent of a vehicle’s existence as a ‘truck’.
3. A weight of membership could appropriately be assigned to the vehicle entity as a whole.
4. Both automobile and truck can have fuzzy attributes contained within their list of attributes. It should be noted that these fuzzy attributes do not represent the fuzzy weight of membership of the entity but only the weight of memberships for the specific attribute data type.
5. Lastly, attributes considered to be specific to one class of asset may contain an appropriate value or a fuzzy weight/value pair for that asset class. Those attributes that are *not* associated with the asset class may contain a ‘null’ value for these same attributes. For example, attribute TOW_CAPACITY contains a value of 5,000 to reflect the asset’s towing capacity. Towing capacity is particular to trucks and generally not to automobiles.

On the other hand, STORAGE_CAPACITY, or storage capacity, is considered specific to automobiles rather than trucks and so no value for STORAGE_CAPACITY would be assigned to the TRUCK class of this asset. The presence of nulls in any relation is a potential issue that must be considered.



VID	MAKE	MODEL	LICENSE	P_CAP	T_CAP	ABILITY	S_CAP	SIZE
0.9/Truck/**3324**	Ford	F150	243-FRG	3	5000	0.9/Heavy		
0.3/Automobile/**3324**	Ford	F150	243-FRG	3			350	06/Medium

Figure 4.7

4.2.1.4 Fuzzy Relationships

A fuzzy relationship describes a relationship between entities to a fuzzy degree of membership. Figure 4.8 describes the relationship of properties near one another.

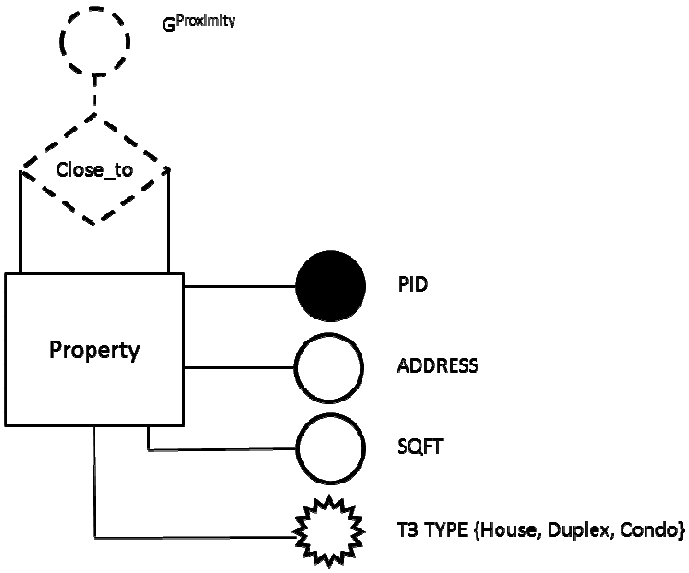


Figure 4.8

In Figure 4.8, a property is a crisp entity with a fuzzy attribute 'Type' which determines the weight of membership of the property in the Type fuzzy class. The properties, however, are situated in proximity to one another to varying degrees. This proximity can be in 'Proximity' to any number of properties with possible values of 'near', 'far' or 'adjacent'.

4.3 Aggregation

The definition of the word 'aggregate' is to 'group'. Yet, the word has several very specific meanings with respect to modeling entities, their relationships and queries.

4.3.1 Aggregation in Extended Entity Relationship Models

Aggregation in EER modeling is used to simplify or generalize the model by grouping related items together. At a high level, and within a complex relational model, related entities might be aggregated and represented by a single, generalized entity. For example, an entity 'Vehicle' may contain its own attributes, but also be comprised of any number of 'sub entities' such as 'engine', 'radio' or 'chassis'. This complex object might only be represented as the 'Vehicle' entity within the relational model. At the lower level, attributes might be aggregated for a similar reason. For

example, a person's street, city, state and zip code might be aggregated or generalized as 'address' in the model while the actual tables derived from this model, would hold the specific attributes that make up the address.

4.3.2 Extending Aggregation to Accommodate Fuzzy Data

Aggregation within the data model is particularly important for the communication of those aspects that comprise the fuzzy attribute or entity.

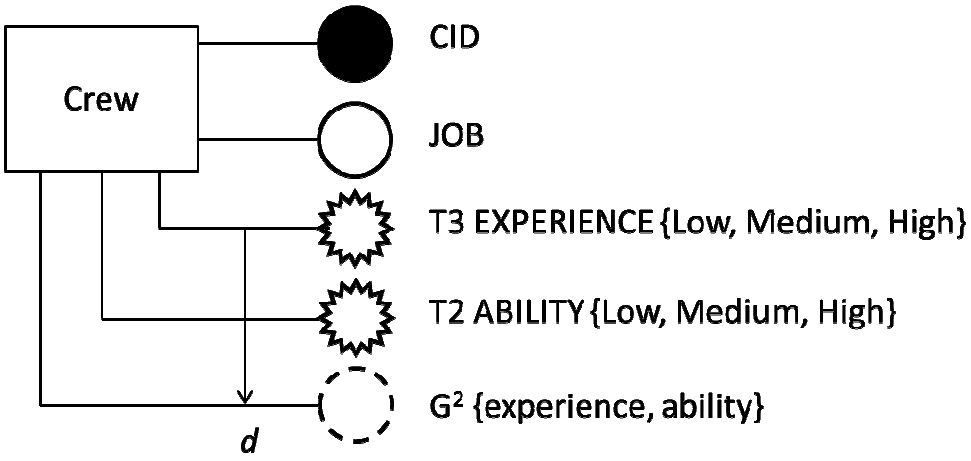


Figure 4.9.

Figure 4.9 illustrates an example of an entity relating a flight crew member's Crew ID, their job and the aggregation of two fuzzy attributes, Experience (type 3) and Ability (type 2) along with a third, derived attribute that constitutes the 'grade' of the crew member's qualifications. Here, grade is degree G^2, uncertainty. How certain is the assessment of the crew member's experience and ability? An additional degree could be applied for type G^4, importance. How important is the degree of experience or ability of the individual and their position? One might assume that a pilot's ability and experience might be of a higher degree of importance than the steward staff crew member.

The grade for experience and ability is an aggregated grade. The grade, G^2, represents the combined fuzzy weight of membership between experience and ability giving the entity a '*tuple* weight of membership' in relation to other Crew tuples.

As can be seen, the representation of a fuzzy attribute or attributes can get quite complex and convey significant information from a number of different perspectives. The tools illustrated here provide an example of how this information can be conveyed. It is up to the database designer to determine which tools to use and to what extent based on the needs of the system under development.

4.4 Relation Valued Attributes

In 1989, C.J. Date proposed that a relation could contain attributes that are, themselves, relations.[32] He called these attributes Relation Valued Attributes (RVA). Typical attributes possess relatively simple characteristics such as 'integer' or variable length character strings, but a relation valued attribute can be comprised of integers *and* variable length strings within the same attribute. In fact, it can contain any combination of base attribute types. Representing this attribute type in a relational model is relatively simple as relations have already been defined. It is a fairly straight forward extension of the entity relation diagram to incorporate the symbology of the RVA into the more complex entity relation.

4.4.1 Crisp RVA

Like any relation, an RVA is composed of attributes. These attributes are then consolidated into a single relation valued attribute. Figure 4.10 below shows an RVA labeled SUPPLIED_PART which is composed of two attributes, PART_NO and QUANTITY. As an RVA,

SUPPLIED_PART represents a single attribute within the entity Supplier_Part. As a result, the entity Supplier_Part has a degree of five, not six.

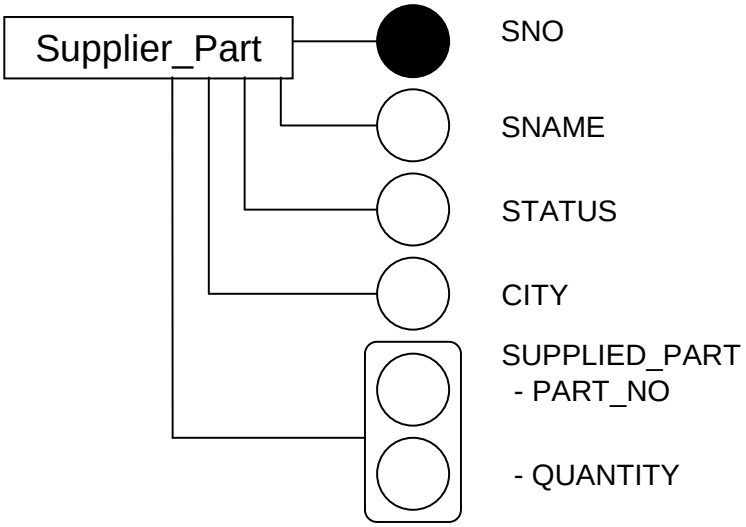


Figure 4.10

Table 4.2 below illustrates what the data would look like if the entity were developed within a relational database. Note that the cardinality of the table represented by this example is 2 and, again, the degree is 5. The RVA represents a single instance of the attribute contained within one relation whose relvar is itself composed of two attributes. The tuples contained within each of these two records attribute exist only within their respective tuple instance attributes.

SNO	SNAME	STATUS	CITY	SUPPLIED_PART	
S1	Acme	10	London	**PART_NO**	**QUANTITY**
				P1	200
				P3	150
				P4	300
S2	Breeze Master	30	Athens	**PART_NO**	**QUANTITY**
				P1	200
				P2	180
				P5	200
				P6	145

Table 4.2

Date argues that because the relation contained within the RVA is *encapsulated* within a single attribute, the attribute is merely of an arbitrarily complex domain and as such satisfies the requirements of First Normal Form.[32]

4.4.2 Using RVA's to Represent Fuzzy Data

One of the criticisms of the representation of fuzzy data as individual attributes within a relation is that the fuzzy degree of membership is not physically 'bound' to the value to which they are associated. In other words, if a value's degree of membership is contained in one attribute and the value itself is contained in another, it is possible for one, the other or both to be excluded from a result set or ordered in such a way that there is no obvious association between degree of membership and value to the user. Because a relation valued attribute contains all of its associated attributes within its well defined structure, RVA's are well suited to the representation of fuzzy data. A fuzzy attribute represented by an RVA contains not only the value itself, but any number of fuzzy degree of membership. And, because the attributes contained within the RVA are all 'bound' within the RVA, they cannot easily be separated or disordered. The following example of an RVA containing fuzzy data involves a threat assessment database with multiple threats and multiple sources of information. In this example, the entity THREAT is fuzzy while having a crisp primary key as the threat identification number or TID. Two crisp attributes

provide basic information about the threat such as NAME and the GPS_COORD as the location of the threat. The overall threat level of the entity is maintained at the entity level, not at the attribute level. The threat level in this example is ‘derived’ or determined to be a value calculated through a function or aggregation using other system data. As such, it is not a physical attribute within the tuple and is not shown as an attribute in the relation’s ER diagram. This being the case, the overall level of the ‘Threat’ is HIGH, MEDIUM or LOW to a determined degree. This point is significant. Up to this point, a weight of membership has been significant within a tuple or used to discriminate among the same attribute in other tuples of the relation. This example, however, illustrates how a tuple can have a ‘tuple wide’ weight of membership within the relation and sorted based on this tuple weight of membership among other tuples. The TID is assigned and crisp. The relation valued attribute ASSESSMENT has one crisp attribute to represent the groups of general information sources as SOURCE_TYPE and three fuzzy degree of membership types, MEMBERSHIP, CERTAINTY and IMPORTANCE. By including an RVA rather than storing the information in separate tables, two important things are accomplished. First, a degree of membership for this threat can be associated with the threat at the entity level. In this way, each threat could be displayed in order of the threat’s membership in any of the threat classes. The second benefit of this approach is that multiple perspectives of ‘membership’ such as degree of membership, certainty and importance can be related through the data. The EER diagram might look something like the one shown in Figure 4.12.

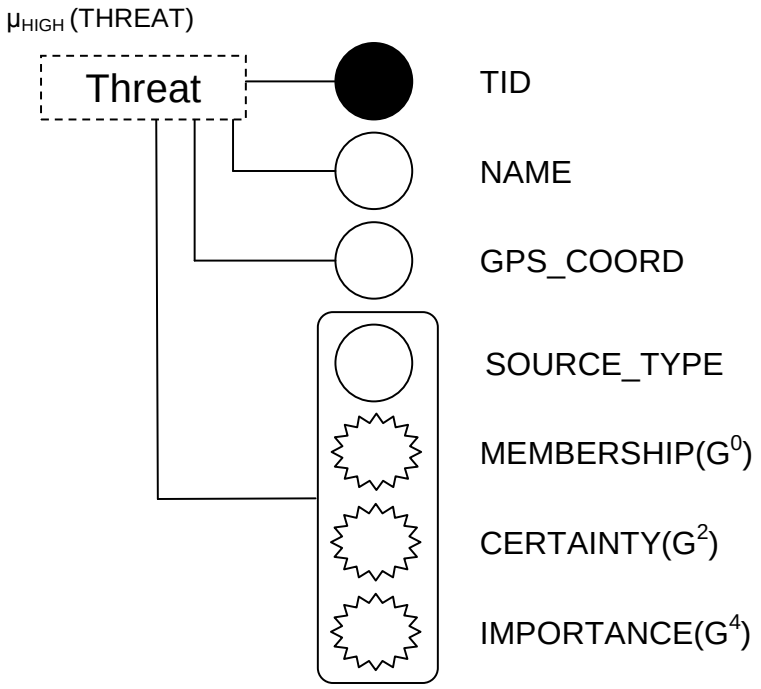


Figure 4.11

A representation of the table structure and some sample data for the entity 'THREAT' is shown in Table 4.3. TID, NAME and GPS_COORD are represented as one might expect. The relation valued attribute ASSESSMENT, however, contains all four 'sub-attributes' described by the EER diagram. The cardinality of this relation is 2 as there are only two threats reported in the result. The degree of membership of this relation is 5 since the threat level represented by μ_{HIGH} is derived and displayed as an attribute or column although it has no physical domain in the underlying table. ASSESSMENT is represented as an RVA and is composed within a single attribute.

μ_{HIGH}	TID	NAME	GPS_COORD	ASSESSMENT			
0.54	003	A Threat	35 51.052/ 78 53.163	**SOURCE_TYPE**	**MEMBERSHIP**	**CERTAINTY**	**IMPORTANCE**
				Informant	0.60	0.90	1.00
				Email Traffic	0.48	0.75	0.70
				Phone Traffic	0.39	0.68	0.75
0.48	004	B Threat	45 32.016/ 81 30.094	**SOURCE_TYPE**	**MEMBERSHIP**	**CERTAINTY**	**IMPORTANCE**
				Informant	0.54	0.83	0.94
				News	0.49	0.84	0.83
				Outside Analysis	0.46	0.84	0.83
				Phone Traffic	0.41	0.63	0.70

Table 4.3

Notice that because the attribute ASSESSMENT is an RVA, the attribute can contain any number of records, SOURCE_TYPES, for the base TID tuple. Records contained within the RVA must adhere to the same relational constraints as any base relation.

4.4.3 Fuzzy Attributes as Relations

As can be seen in the preceding section, it could be argued that as relation valued attributes, a fuzzy attribute is itself an entity or a relationship and could very possibly be represented as a relation. Consider the following example based on the Threat Assessment example above. The RVA composed of the attributes SOURCE_TYPE, MEMBERSHIP, CERTAINTY and IMPORTANCE can be mapped in an E-R diagram as a relationship between the entities Threat and SourceType. The weight of membership between the two entities is contained within their relationship. As a result, it could be argued that the threat assessment example shown in Table 4.3 is nothing more than the join of the two entities via their relationship as described by the E-R diagram shown in Figure 4.12.

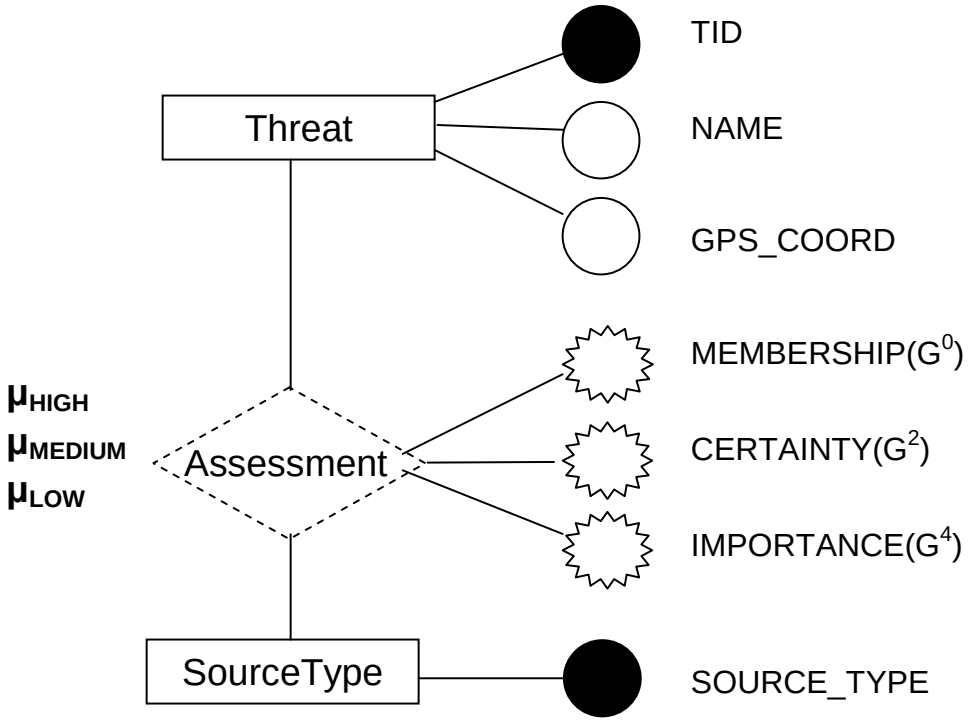


Figure 4.12

The tables resulting from this approach are shown below in Tables 4.4a through 4.4c using the same sample data provided in Table 4.3 above.

μ_{HIGH}	TID	NAME	GPS_COORD
0.54	003	A Threat	35 51.052/ 78 53.163
0.48	004	B Threat	45 32.016/ 81 30.094

Table 4.4a

TID	SOURCE_TYPE	MEMBERSHIP	CERTAINTY	IMPORTANCE
003	Informant	0.60	0.90	1.00
003	Email Traffic	0.48	0.75	0.70
003	Phone Traffic	0.39	0.68	0.75
004	Informant	0.54	0.83	0.94
004	News	0.49	0.84	0.83
004	Outside Analysis	0.46	0.84	0.83
004	Phone Traffic	0.41	0.63	0.70

Table 4.4b

SOURCE_TYPE
Informant
Email Traffic
Phone Traffic

News
Outside Analysis

Table 4.4c

In order to provide the data in a structure similar to that shown in Table 4.3, however, these three tables must be joined. Such a representation is shown in table 4.5.

μ_{HIGH}	TID	NAME	GPS_COORD	SOURCE_TYPE	MEMBERSHIP	CERTAINTY	IMPORTANCE
0.54	003	A Threat	35 51.052/ 78 53.163	Informant	0.60	0.90	1.00
0.54	003	A Threat	35 51.052/ 78 53.163	Email Traffic	0.48	0.75	0.70
0.54	003	A Threat	35 51.052/ 78 53.163	Phone Traffic	0.39	0.68	0.75
0.48	004	B Threat	45 32.016/ 81 30.094	Informant	0.54	0.83	0.94
0.48	004	B Threat	45 32.016/ 81 30.094	News	0.49	0.84	0.83
0.48	004	B Threat	45 32.016/ 81 30.094	Outside Analysis	0.46	0.84	0.83
0.48	004	B Threat	45 32.016/ 81 30.094	Phone Traffic	0.41	0.63	0.70

Table 4.5

Note that the representation shown in Table 4.5 contains a great deal of redundancy. Such redundancy is common when tables are joined, but the representation in Table 4.3 eliminates this redundancy. Further, the encapsulation of the fuzzy attributes within the RVA attribute ASSESSMENT is lost along with the direct and intuitive association these attributes had with their source type in Table 4.5. It should also be noted that Table 4.5 has a degree of 7 and a cardinality of 7 as compared with the RVA representation in Table 4.3 that has a degree of 5 and a cardinality of 2. Given these factors, the data representation illustrated in Table 4.5, as is often the case with joining tables, is not in third normal form (3NF) where the RVA representation illustrated in Table 4.3 is giving the implementation the option of storing the data as the RVA representation or within separate tables depending on the access and maintenance preferences inherent in the system's design. Table 4.5 does not have this luxury of choice and would, out of responsible necessity, exist only as separate tables.

And yet, it could be further argued that any fuzzy weights of membership were nothing more than the representation of a relationship between one entity and another entity representing a characteristic in question. Consider a ‘Suspect’ entity and any number of descriptive characteristics such as HairColor, EyeColor or Height with the various relationships respecting weight of membership between them. The question then is, should fuzzy attributes structured as RVA’s be included as attributes within a single relation or broken out as a separate relationship and maintained as its own relation? The answer, it seems, is; it depends. The point is, there are options.

Chapter 5 - RVA as a Constraint to the Representation of Fuzzy Data

5.1 Introduction

The preceding chapters discussed the theory behind both fuzzy data and relation valued attributes. This discussion has described the requirements necessary for the representation of fuzzy data. This representation requires both a class and a weight of membership to represent the fuzzy data value. Both of these attributes *must* be provided in a result or neither should be provided. In other words, the atomicity of fuzzy data requires both attributes or neither.[37] The relation valued attribute (RVA) provides a solution. The RVA is an attribute and as an attribute can be included or excluded from the selection predicate of an SQL query. But the domain of an RVA is a relation that is well suited to contain the required ordered pairs of attributes associated with a fuzzy data value. Encapsulating the relation within the attribute of an RVA data type without direct access to the relation's members ensures that neither the class nor the weight of measurement can be addressed specifically for inclusion or exclusion from an SQL query and thus ensures the atomicity of the fuzzy data value.

The purpose of this research was to design, implement and study a database system that, for the first time, not only supports the use of fuzzy data within a relational database system, but does so through the use of relation valued attributes which will enforce not only the generally accepted capabilities and constraints established for an attribute by the relational model, but also provides additional capabilities and constraints specific to the use and maintenance of fuzzy data.

This chapter will address the considerations and approach taken to design and implement a proof of concept implementation of a relational database using an RVA data type to support fuzzy data.

5.2 Design Considerations

5.2.1 Metadata Considerations

Implementations of the relational model store metadata about tables, rows, columns, indexes, types, and constraints in system tables that can be queried.[33] Extending the relational model to include fuzzy data requires that fuzzy metadata be included and accessible in some form within the system. The fuzzy RDBMS, the database administrator, and the application developer will all use the metadata to work with defined RVA data types.

5.2.1.1 Database Design Considerations

The implementation of a relation valued attribute as an extension to the structured query language (SQL) requires an approach that will allow the RVA to be maintained at the physical level and a modification to the associated data definition language (DDL) to support the creation of an RVA data type. At the logical level, consideration must be given to the data manipulation language and whether it will require modification to provide the user with the ability to retrieve and manage the data within the database.

5.2.1.2 Approach Considerations

Two basic approaches were considered in the design of the RVA. Each approach was considered given the constraints and capabilities of the chosen development environment provided by the open source database system MySQL.

The first approach was to create a table within a table. In theory, this approach seemed plausible. In fact, the database system and the database engines provided in MySQL already had many of

the features thought necessary to the implementation of this approach.[38] The second approach considered was the creation of a new type of attribute, an RVA attribute that would contain the metadata associated with the relation valued attribute such as the table, the key to the table and the attributes desired within the relation. Using the information contained within this attribute, the relation associated within the RVA could be identified and the data associated with this relation nested within the result of the calling query.

The following sections discuss the these two approaches in detail and conclude with the reasons why the second approach was chosen for implementation.

5.2.1.2.1 Creation of a Table Within a Table

Creating a table within a table to support the implementation of RVA's adds a particular challenge to the storage, organization, indexing and management of the underlying files. RVA's require their own storage, indexing and query strategies just as is the case with the higher level nesting relation. To compound the complexity, each of these housekeeping tasks must be inextricably associated with and linked to the nesting relation. For example, in order to access a value from a component contained with an RVA, the nesting relation must first be considered so that the query can efficiently access the components of the associated nested relation. Inefficient storage and indexing strategies would lead to poor database performance when using an RVA of this type. This section looks at some of the more common file storage, index and query methods and how they might have been applied to attributes containing relations.

There are a number of different approaches to the storage and indexing of database objects at the physical level. Given the nature of RVA's, however, extra consideration needs to be given to

such a design. While there are certain circumstances where an RVA will have one and only one record, the likelihood is that it will have more than one. Not only would the base table be expanding and contracting as records are inserted and deleted, but in the case of RVA's, would the nested relation. So, as a relation within a relation, the RVA may be seen as separate from its base table, but inextricably linked to the base table and at the tuple level! Given these characteristics, reading, writing and maintaining a table containing an RVA in secondary memory would require special consideration and even some limitations. The following section reviews the more common approaches to database files and their associated page abstraction at the physical level and looks at some of the advantages and disadvantages of each.

5.2.1.2.2 Fixed Length Records

The simplest database file format is a fixed length record written to disk using a collection of pages.[34] Using a fixed length file format for the base table *or* the RVA would be an appropriate approach if these two were separate tables, but they are not. The RVA is contained within the base relation and both the base relation and the RVA (or multiple RVA's) may have a variable number of records inserted into and deleted from them which, while fixed in length, will likely result in a true record of frequently changing length. As a result, this approach was dismissed as a viable option.

5.2.1.2.3 Variable Length Records

If the tuple containing an RVA is a variable length record, the page contained on the disk cannot be divided into a fixed collection of slots. Instead, the database's disk space manager will need to find free space of an appropriate length to accommodate the complex tuple.[34] As data is added to this record within the complex data structure, the allocation of space becomes of critical

importance. The task of disk management of variable length records can be accommodated using a slot directory containing record offset and record length pairs for each slot.

The offset is essentially a pointer to the start of the data record and, as records are deleted the offset is set to -1 to indicate free space. Maintaining the variable length records laying end to end, a pointer maintained by the slot directory provides the offset of the area of free space where new records may be written. If a record is too large to fit into the remaining free space, the page may be reorganized and defragmented to free up a contiguous block of free space sufficient for the record's needs or the record would be written to the next page in sequence.

5.2.1.2.4 Variable Attribute Records

The challenge with RVA's is that they are relations. As relations, rather than individual 'simple' attributes, they can be composed of a relvar of a significant degree and cardinality. In a sense, a relation valued attribute is 'variable' in both length and width as well as composition. As a result, neither the simple fixed length record layout nor even the variable length record layout is appropriate to the requirements of an RVA.

In order to illustrate the physical implementation of an RVA within a base table, consider a simplified version of the supplier/parts database shown in Table 5.1. There is a base table consisting of three attributes, SID, NAME and SPART. SID and NAME are standard data types. Attribute SPART is an RVA and as such consists of a nested relation consisting of its own attributes, PID and QTY.

SID	NAME	SPART	
011	Acme	**PID**	**QTY**
		010	200
		023	150
		030	253
025	Breeze Master	**PID**	**QTY**
		010	200
		049	180
		060	206
		072	145

Table 5.1 – SUPPLIER

Query 5.1 below outlines how the query language might be modified and used to create the structure of the SUPPLIER relvar with the nested structure SPART:

```
Create Table SUPPLIER
        (
        SID         Integer             Primary Key
        NAME        Char(25)
        SPART       RVA (Type SPART
                        PID     Integer Primary Key
                        QTY     Integer
                        )
        );
```

Query 5.1 – SUPPLIER Data Definition

The structure of the SUPPLIER table contains the field SPART that is typed as an RVA referencing the structure SPART that was previously defined in the database. SPART is defined as a relation and with the part identification number PID designated as the primary key. Once the table has been defined, the technical characteristics of the table provide the structure to the RVA. A potential issue with this approach is that data dictionary structures are subject to change. If, for example, a part NAME were added to the SPART table structure, how would this change affect existing instances of SPART that were based on the original incarnation of the SPART structure? In theory, every instance of SPART associated with a SUPPLIER tuple will need to be modified

to accommodate the underlying structural change. MySQL provides such a tool in its ALTER TABLE function, but this tool would need to be modified to accommodate not just the nesting table alteration, but the nested RVA alteration as well. This task would not be insignificant. While this approach shows some promise as it gives a well defined and re-useable structure to the RVA attribute, there is a necessary requirement to modify not just the table structure and the attributes, but to modify the underlying SQL functions as well.

Figure 5.1 shows how the supplier table structure might be created with reference to an associated RVA table structure SPART. The B-Tree example and the sample data representation shown suggests only one method of how the data might be accessed using a sample query.[35] Note that a pointer from the base table SUPPLIER is used to access the SPART data area. RVA's used in this way will consist, essentially, of many instances of their referenced table. In other words, there may be many instances of the SPART table existing in memory with each instance potentially consisting of many records. So the question might be, how will the SPART instance associated with each tuple be accessed? One obvious approach is to include a pointer to each nesting table's defined RVA instances. In fact, no other approach seems plausible. Further research and analysis, however, may uncover other possible approaches.

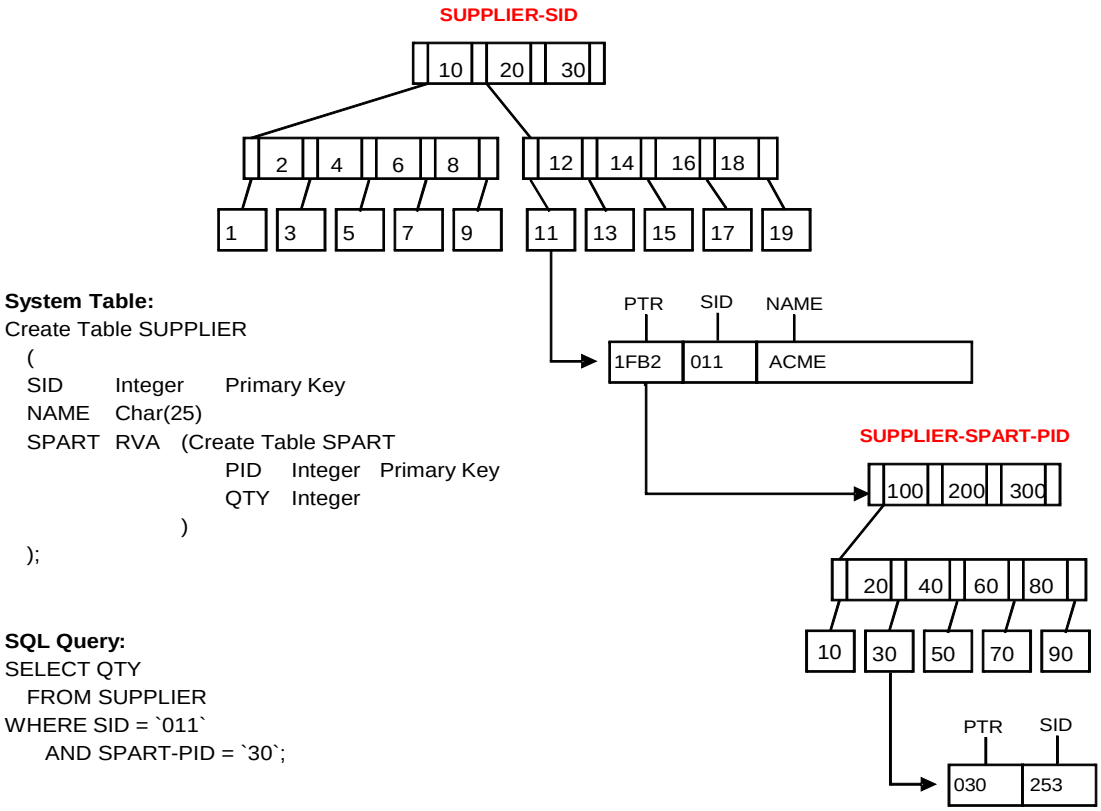


Figure 5.1

In another slightly more complex example, reconsider the threat assessment database from Section 4.4.2. Table 5.2 below provides a review of this table.

μ_{HIGH}	TID	NAME	GPS_COORD	ASSESSMENT			
0.54	003	A Threat	35 51.052/ 78 53.163	**SOURCE_TYPE**	**MEMBERSHIP**	**CERTAINTY**	**IMPORTANCE**
				Informant	0.60	0.90	1.00
				Email Traffic	0.48	0.75	0.70
				Phone Traffic	0.39	0.68	0.75
0.48	004	B Threat	45 32.016/ 81 30.094	**SOURCE_TYPE**	**MEMBERSHIP**	**CERTAINTY**	**IMPORTANCE**
				Informant	0.54	0.83	0.94
				News	0.49	0.84	0.83
				Outside Analysis	0.46	0.84	0.83
				Phone Traffic	0.41	0.63	0.70

Table 5.2

The dependencies between the attributes contained within the RVA relvar are inexorably linked and have been minimized. Each tuple instance of data contained within the attributes SOURCE_TYPE, MEMBERSHIP, CERTAINTY and IMPORTANCE cannot be separated

without losing information and there is no logical 'break' between them. As a result, the data model defined to contain ASSESSMENT might exist at a minimum as follows:

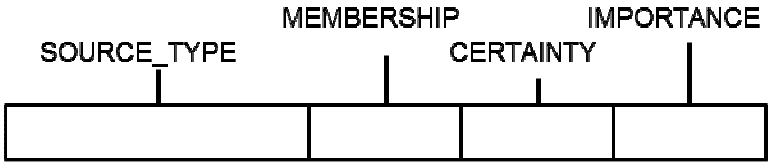


Figure 5.2

At this point, each additional instance of ASSESSMENT could either be set end to end within contiguous blocks in memory as fixed length records or a pointer might be assigned a value to the next instance within the physical structure of the attribute. But this is a design question that is answered for standard tables as well as for RVA instances. As was the case with the supplier database discussed earlier, a pointer from the base relation would likely be a requirement. Putting an example together, the data in threat TID 003 might look something like the following:

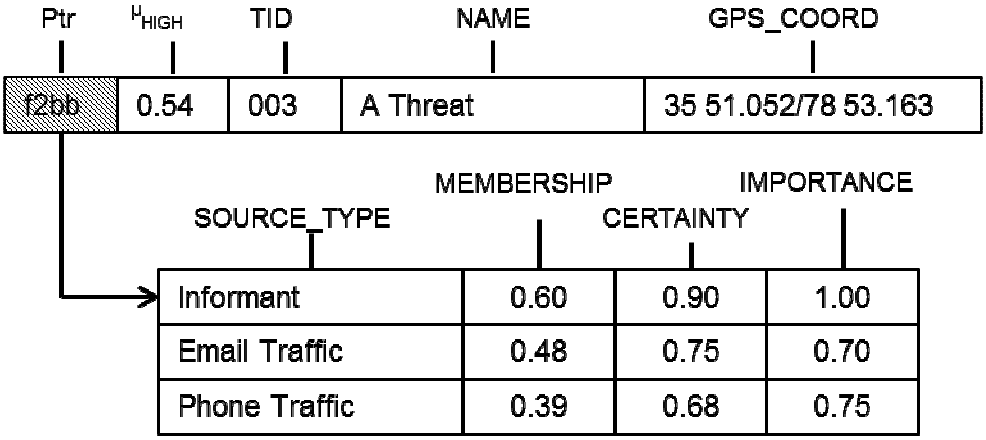


Figure 5.3.

The data dictionary tables contained within the SYSTEM table space would provide the address of the THREAT table. The database would then access a particular record by the primary key. All ASSESSMENT records would be accessed using the pointer contained within the THREAT record. The challenge to using RVAs, even in this seemingly straight forward way, is how the

data dictionary table would need to be modified to reflect the added structural features contained within the RVA.

5.2.1.2.5 Indexes and RVAs

Performance of database queries and operations are greatly improved through the use of indexes. Indexes are files containing 'data entries' and their particular key search criteria *k* denoted as *k**. Each data entry contains enough information to efficiently locate an associated record's location given its record identification number or 'RID'. Ramakrishnan and Gehrke suggest that there are three general options for indexes to be structured and used.[36] Specifically:

1. A data entry *k** is an actual data record with a search key value *k*.
2. A data entry is a <*k, rid*> pair, where rid is the record id of a data record with search key value *k*.
3. A data entry is a <*k, rid-list*> pair, where rid-list is list of record ids of data records with search key value *k*.

If a data file for employees called, *employee*, contains four fields, EID (for 'employee identification number), NAME, AGE and SALARY where EID is a key, an index file may be created for the employee field EID to support efficient query and data access to the records held within the table's data file. This file would have an entry for every employee identification number with information related to the EID record given one of the three options outlined above. It should be noted, however, that the employee table would only have one index file on field EID. And, repeating the point above, this field would contain a column for every EID in the employee table. RVA's, however, are themselves relations. Each instance of an RVA within a nesting table is a table unto itself. If one were to follow the index file strategy outlined so far, each RVA instance would have its own index file on key attribute(s). In the threat assessment

example outlined in table 5.2 above, an index file may be created on SOURCE_TYPE so as to increase query and table maintenance efficiency on this key attribute of the threat assessment's RVA ASSESSMENT. Within the limited confines of the example, maintaining these index files and their data may not be a significant task. But real world tables are far larger, containing many records and the maintenance of each index created in support of each RVA instance would be overwhelming.

5.2.1.2.6 Data Definition Language for RVAs

Creation of a relation containing one or more RVA's is a multi-step process and the data definition language needs to be extended to allow for the complex metadata inherent in an RVA. The first step is to create the relvar for the RVA. Creating the structure of this relvar will include the data structure of the relvar in the data dictionary to include any other existing table or RVA. By creating a reference structure, all attributes, data types and constraints applicable to the structure will be assigned and enforced on the complex RVA type attribute. For the purposes of this example, create an assessment structure called ASSESS_STRUCT. This structure is an included structure and will not, itself, contain any data. As in included structure, tables can be provided with their complex characteristics much the same way that an attribute derives its characteristics from a domain.

```
CREATE TABLE ASSESS_STRUCT
        (
        SOURCE_TYPE     CHAR(25)  PRIMARY KEY,
        MEMBERSHIP      DECIMAL(3,2),
        CERTAINTY       DECIMAL(3,2),
        IMPORTANCE      DECIMAL(3,2)
        );
```

Query 5.2 – ASSESS_STRUCT Data Definition

Once the RVA's relvar is created, it can be included as the structural relvar for an RVA in the table THREAT. The structure field SOURCE_TYPE is designated as the Primary Key. Each RVA structure, like any base relation structure, must have a designated primary key. The values contained in any table created from this structure would have this key attribute and must be unique within the RVA for each base relation tuple. For example, the ASSESSMENT RVA can only contain one 'Informant' source type for each associated THREAT record. As with any base relation, dependencies and normal form constraints would be adhered to. Using data type 'RVA' includes the referenced structure as the underlying format for the attribute.

```
CREATE TABLE THREAT
        (
        TID             INT(4) ZEROFILL  PRIMARY KEY,
        NAME            CHAR(30) NOT NULL,
        GPS_COORD       CHAR(30),
        ASSESSMENT      RVA(ASSESS_STRUCT)
        );
```

Query 5.3 – THREAT Data Definition

By assigning the data type RVA to the base table, two things occur. First, the RVA takes on the complex structure of the data object it references without having to define the object within the 'CREATE TABLE' SQL. Secondly, the use of the RVA data type tells the data dictionary to include a pointer from the base relation to the RVA's associated data.

5.2.2 Creation of a Nesting Function Between Tables

When Date introduced his theories concerning relation valued attributes, he described the state of the tables comprising the RVA as being in a nested or an unnested state.[37] In their nested state, the data and the comprised tables combined to create a single, nested relation similar to that shown in Table 5.1. In their unnested state, the comprised tables exist as independent entities.

They are related, but exist independent of each other. For example, consider the supplier/parts relation illustrated here again as query 5.1 shown below:

```
Create Table SUPPLIER
        (
        SID     Integer         Primary Key
        NAME    Char(25)
        SPART   RVA (Type SPART
                    PID     Integer Primary Key
                    QTY     Integer
                    )
        );
```

Query 5.1

The result of this query is the creation of a table within another table to create a single, inseparable entity. Compare this to the following representation of an unnested version of this entity consisting of two independent tables:

```
Create Table SUPPLIER
        (
        SID     Integer         Primary Key
        NAME    Char(25)
        SPART   RVAChar(125)
        );
```

Query 5.4 – SUPPLIER Data Definition

```
Create Table PART
        (
        SID     Integer     Primary Key
        PID     Integer     Primary Key
        QTY     Integer
        );
```

Query 5.5 – PART Data Definition

Just as the case of the threat assessment example described in 2.1.2.4, two individual tables are created.

SID	NAME	SPART
011	Acme	@Table:SUPPLIER@Key:SID@Fields:PID, QTY
025	Breeze Master	@Table:SUPPLIER@Key:SID@Fields:PID, QTY

Table 5.4 – SUPPLIER

SID	PID	QTY
011	010	200
011	023	150
011	030	253
025	010	200
025	049	180
025	060	206
025	072	145

Table 5.5 – PARTS

Unlike the approach shown in Section 5.2.1.2.4, both tables have a common field SID which acts as a primary key in Table SUPPLIER and a component of the composite key in Table PARTS. There is no pointer in this table structure. Each table is created, populated and maintained independently of the other with the exception of a foreign key relationship to ensure that no parts are added to the PARTS table for which there is no supplier. The value shown in RVA field SPART is explained in detail in Section 5.3.1.2. Everything in the example thus far is accomplished using existing SQL functionality. No new index algorithms or storage functionality is required. Standard SQL constraints are enforced without any system modification. It is only when the two tables are nested that new functionality is introduced in this approach.

The only changes to the SQL processing at this point are to create the new data type and the nesting process along with the process strategy necessary to initiate nesting of the two separate tables. In this way, the vast majority of MySQL's native functionality and features can be leveraged with a minimum of modification to the core processes.

5.2.2.1 Considerations for Selecting the Normal State

If the normal state of the relation was to be the nested state, then the development efforts associated with the RVA type would need to focus on the creation and maintenance of a nested relation within an attribute. Additional functionality would need to be developed that would provide the ability to unnest the relation from the attribute to create an independent relation. In fact, early development efforts pursued this approach and an effort was made to physically create a table within a table. It was soon learned, however, that MySQL was designed and written to prevent developers from doing this very thing. When a table is being created in MySQL a lock is placed on the table being created that precludes any other tables from being created within it at this point.[38] To change this functionality would have far reaching and potentially unpredictable ramifications within the MySQL process.

On the other hand, if the normal state of the relation were to be the unnested state then the relation could be created and maintained outside of the nesting table using existing MySQL functionality and the development efforts could focus on the nesting functionality necessary to encapsulate the values contained in the relation within the attribute. In this way, a table could be created with attributes and constraints with both primary and foreign keys assigned just as with any other table in MySQL. This approach was seen as being highly advantageous because there was no reason to reinvent functionality that already existed in MySQL. The problem, then, resolved to be only one; how to nest the nested relation as an attribute within the nesting relation?

5.3 Design Features of the RVAChar

Given the advantages of maintaining the RVA as a separate table, it was determined that this approach was to be taken. As separate tables, both the nesting and the nested table could be created, maintained and populated separately, utilizing MySQL's native capabilities. But still, the problem remained; how and when should nesting take place.

5.3.1 The 'How' and 'When' of it

There are two steps to solving the problem of when and how to nest the RVA's relation within the result set of a query. The first step is to create a new data type called an 'rvachar'. This new data type has knowledge embedded within it such that, when used in a query, it provides the information required to access and nest an RVA within the nesting relation. Once the state of the relation has been determined the second step is to figure out how this nesting will take place, under what circumstances and when will the nesting take place?

5.3.2 The Knowledge Contained Within the RVAChar

The rvachar data type has knowledge embedded within it. This knowledge is composed of a string of three parts which act as tokens when the string is parsed. For example:

@Table:v_type@Key:vid@Fields:class,weight;

The 'Table' value is the name of the nested table associated with the attribute. The 'Key' value is the common key shared with the nesting table. The key in the nested table may have a foreign key relationship with the nesting table but the RVA may only require an abbreviated version of the nesting table's key which is why it is specified within the knowledge string. The 'Fields' component in the value specifies the attributes within the RVA that will be contained in the result. In this way, the RVA is suitable not only for use with the specific requirements of fuzzy data, but of any relation that the database designer chooses to include in the RVA's result. In this

way, the designer is free to specify only those attributes she determines to be necessary to her needs. When declaring an attribute of the rvachar data type, the attribute is created as a field that is not null and with a default value consisting of the knowledge string that is set at the time of creation. Because this value is the default, the value will be set each time a new row is added to the table and this value will be the same for all instances of the field. Once the field has been created within the table and a default value assigned for all instances, it can be maintained as any other table. The field can be deleted, created and the default value can be viewed or changed by using MySQL's standard 'modify' or 'describe' command. But the 'knowledge' value of the rvachar data type field cannot be returned as a result using a select statement. This prohibition is because data types in MySQL are defined as objects with methods. When an attribute defined as an RVA data type is included in the selection of a query, the knowledge contained within the attribute is first selected using the data type's *val_str()* method. This method extracts the knowledge contained within the initial value retrieved and uses this knowledge to nest the relation specified within the base nesting table. As a result, the relation contained within the RVA attribute is returned with the result rather than the knowledge value used to obtain and nest it. The *val_str()* method operates as the nesting function for the RVA.[37]

5.3.3 How the 'Knowledge' Works

When a query is made against a table containing an RVA attribute, the MySQL parser builds a complex tree structure called a 'thread' to contain all of the tokens extracted from the query as well as the network, client and buffer specifications required to successfully execute and return a result to the client. Once the thread is populated with the query's tokens, MySQL begins the task of collecting the data that satisfies the query. One of the first steps in the collection of data is to obtain the value of the RVAChar specified in the SELECT predicate. The first time that the

thread attempts to obtain the value associated with the RVAChar attribute the process enters the RVAChar class method '*val_str()*' which accepts as parameters values containing the original query string, the knowledge contained within the RVA attribute and all the information necessary to take control of the original query. The following is a summary of the steps taken by this method to obtain and return a result satisfying the requirements of the original query as well as the specified relation nested within the result[ii]:

1. The query processing enters rvachar method *val_str().*
2. The original query string is saved in a variable.
3. The rvachar value is saved in a variable.
4. Using the knowledge contained in the rvachar value, the original query string is syntactically modified to join the nesting and the nested tables.
5. A new thread is created and initialized.
6. The new query is assigned to the new thread element using the *build_query()* method. This method takes the knowledge contained in the RVA attribute's value and, using the information contained in the original thread's parsing structure creates a new query that will be sent back through they MySQL parser in the new thread.
7. The new thread's memory root variables are initialized with appropriate block and memory size allocations.
8. The network or 'net' values from the original thread are assigned to the new thread. This step is important because the network and portal settings are the door through which the original query came and where the result must be returned.

[ii] Full code for the *val_str()* and *build_query()* methods are provided in Appendices C and D.

9. The query buffers are initialized with the new query being assigned to the ‘buff’, ‘read_pos’ and ‘write_pos’ variable assigned the new query value.
10. The security profile settings contained within the original thread are assigned to the new thread so that both threads now have the same profile and authorizations.
11. The parser state of the query is reset for the new thread and the cache settings are assigned for the new result.
12. A new packet is created for the new thread to create the memory for, and to hold the results of, the query.
13. The MySQL method *my_pthread_setspecific_ptr(THR_THD, select_thd)* is called. This step is a very important one as it sets the value of the global MySQL variable current_thd. A great many methods in MySQL check the value of the thread entering as a parameter to the value of current_thd to ensure that ‘this’ thread is the current thread. If not, both processing and the server connections will fail.
14. With the new thread fully initialized and assigned, MySQL method *mysql_parse()* is called:

```
mysql_parse( select_thd,                  ← New thread
             select_thd→query(),          ← New query string
             select_thd→query_length(),   ← New query length
             my_parser_state   );         ← Parser state of the new thread
```

    mysql_parse() will validate the new query, parse the query into the tables, commands and fields (items) contained within it, validate the ‘user’ authorizations, check and lock tables as necessary and, if all validations are passed, will execute the query placing the result in query buffer.

15. *Mysql_parse()* returns no value and but when it returns to the calling process within the *val_str()* method a select_send variable is initialized. The *select_send* variable is responsible

for closing the query, cleaning the server and sending the ‘End Statement’ to the server which will send the result to the client through the original query’s port.

16. At this point, the result is or has been sent to the client.
17. Call an ‘error’. This error does not specify a thread and so will be called on the original thread. Calling an error inserts an artificial error into the original thread.
18. Call MySQL method *my_pthread_setspecific_ptr(THR_THD, table→in_use)* to reset the current thread back to the original thread. The variable table→in_use contains the structure and values of the original thread.
19. Set the *is_sent* variable of both the original and the new thread to FALSE. In doing so, the error set earlier will be processed by the original thread.
20. Upon recognizing that an 'error' has occurred, the original thread will stop all further processing of the original query, back out and clean up all the original query buffers and return control to the client.

Using this design, MySQL was modified to create a relational database system in which the use of relation valued attributes is possible. This enabled the use of RVA's to implement the storage and use of fuzzy data values.

5.4 Maintenance Benefits of the 'Knowledge' Approach

There are a number of benefits to using an approach with knowledge embedded in the field's value. As Section 5.3 mentioned, both the nesting and the nested tables are maintained separately. By maintaining the tables separately, maintenance tasks are greatly simplified. This section discusses several of these tasks and the approach to their use with an attribute of the RVAChar type.

5.4.1 Create Table

Creating the nesting table containing the RVAChar and the nested table that the knowledge contained within the RVAChar attribute uses standard SQL.

First, the nesting table is created with the RVA attribute provided with a name, specified as an 'rvachar' of a length appropriate to the anticipated length of the knowledge string and provided with the specific knowledge string value as a default. Because the attribute is created with the knowledge value as the default value, no value need ever be written to the field again. When a record is added to the nesting table for the base attributes, the RVA field will be assigned the knowledge value by default for each new record.

Second, the nested table is created. This table, too, is created using standard SQL specifying a key that contains, at a minimum, the nesting table's key. The table structure must also contain the fields specified in the nesting table's RVA knowledge.

When the RVA is called in a query, it reads the knowledge provided in the RVA field as the default and uses this knowledge to access the nesting table's key and the fields configured for access by the knowledge.

5.4.2 Drop Table

Typically, the nested table will be created with a foreign key relationship to the nesting table. Using standard SQL the foreign key can be specified to cascade on delete. In this way, dropping

a table containing an RVA and the nested table it refers to is no different than dropping any other related tables in a database.

5.4.3 Add, Delete and Update Fuzzy Data Values

The fuzzy data value embedded in the RVA is maintained in a separate table. The attribute merely accesses this data when called for in a query. As such, adding, deleting or updating fuzzy data values is accomplished through the use of standard SQL just as it is with any other table. Maintaining the data in this way has no affect on the functionality of the RVA.

5.4.4 Modifying the Knowledge

The knowledge contained within the RVA attribute can also be maintained using standard SQL. Using the UPDATE command, the value of the knowledge can be set to what ever new default is desired. In this way, an RVA can easily be expanded to include other fuzzy data values such as 'importance' or 'certainty' for a particular object merely by adding these columns to the nested table and modifying the RVA's knowledge to access them using the UPDATE command.

Chapter 6 - Implementation of a Relation Valued Attribute in MySQL

6.1 Steps to Implementation

The client server SQL based database product MySQL was chosen for the implementation of a Relation Valued Attribute (RVA) because it is open source and readily available for download and modification. The following technical specifications are provided:

Product	Version	Comment
MySQL	5.5.20	Database System
Microsoft Visual Studio 2010	10	Development Environment
CMake	2.8.6	Development Product Build System
Bison	2.4.1	YACC Compatible Parser Generator

Table 6.1 - System Specification

6.1.1 Implementation Considerations

Because MySQL is open source, there were a number of issues that presented challenges to the implementation of a new attribute type and the process logic used to access and use it.

The primary issue encountered was the use of the query thread as a global variable in some classes and methods. There are methods in MySQL that use the FIELD class global variable 'table->in_use', 'current_thd' or 'thd' as a global representation of the active thread rather than passing the thread under consideration as a parameter. At times, the method logic would check the value of 'table->in_use' against the 'current_thd' via the following assertion:

```
DBUG_ASSERT(table->in_use == current_thd);
```

As a result, the RVA thread would enter a parsing method and fail due to the fact that global thread representation was still assigned to the value of the original thread. Encountering this circumstance is one reason that it is good practice to eliminate global variables by encapsulating

them within a method either by declaring them within the method or passing them to a method by reference.[39] This issue was overcome by assigning the new RVA thread 'select_thd' to 'current_thread' through the use of the method *my_pthread_setspecific_ptr(THR_THD, select_thd)* prior to calling the parsing method and setting it back to the original thread using the same method after the result was sent to the client.

The second significant issue encountered resulted from the inherent nature of object oriented programming and the method construct used. Because the RVA and the nesting functionality created to accommodate it accessed methods common to other data types and SQL functionality, the optimal approach recommended that any change to the method functionality be avoided if possible. If changes were necessary, they must be implemented in such a way as to have no mal-affect on other functionality of other data types. Generally, the solution to this problem was to encapsulate any changes necessary by interrogating the data type under consideration by using the *item* class method *field_type()* to distinguish functionality specific to the requirements of the RVA from any others.

6.1.2 Steps to Implementation

The implementation of an RVA in MySQL had two parts. First, a new RVA data type was be created along with the process methods necessary to support the data type's functionality. Second, and most importantly, the underlying MySQL processes were changed to accommodate the functionality associated with an attribute that is also a relation.

6.1.2.1 Creation of an RVA Data Type

There are two steps to the creation of a new data type in MySQL. The first is to create the data type in the file sql_yacc.yy which creates a 'token' for the new type. The second step is to create the methods in MySQL to accommodate the characteristics that the data type will have.

6.1.2.2 Modifying the sql_yacc.yy File

The first step to creating a new data type within MySQL is to modify the file sql_yacc.yy. The purpose of the 'yacc' file is to define 'tokens' which are recognizable by the MySQL binary code. The yacc contains tokens for data types and all major MySQL functionality such as 'SELECT'. Even the 'EQ' symbol used in the standard SQL is defined as a function. Once defined in sql_yacc, a handle is associated with the token which passes the primary parameters into MySQL. For example, here is the code for 'SELECT':

```
/* Select: Retrieve data from table */
select:
        select_init
        {
                LEX *lex = Lex;
                lex→sql_command = SQLCOM_SELECT;
        }
        ;
```

First, the word 'select' is referenced and everything that follows the ':' symbol defines the essential variable components of the select command as it will exist in MySQL. Second, SQLCOM_SELECT is the hook into 'MySQL proper' that associates this token with the appropriate MySQL constant.

When a data type is defined, the token represented by the data type must be initialized. Here is the sql_yacc definition and initialization of the RVACHAR data type:

1. First define the token. Tokens are defined in sql_yacc in alphabetic order and the RVACHAR token is:

   ```
   %token     RVACHAR
   ```

2. Next, the type is defined. A 'type' in this instance is a command or object type, not a data type. Among the many 'types' are 'delete', 'drop' and 'insert' along with the actual data

types. Because the RVACHAR token is patterned off of a STRING data type, the RVACHAR token was placed among the string and character types.

…opt_delete_option **rvachar** varchar nchar…

3. The type is then further defined as follows. Again, because the STRING data type was used as the pattern for the creation of the rvachar data type, the same type definition was used.

type:

```
| rvachar field_length opt_binary
{
    $$ = MYSQL_TYPE_RVACHAR;
}
```

4. Finally, the rvachar is referenced to its internal MySQL type using the new token RVACHAR.

```
rvachar
        RVACHAR { }
    ;
```

These changes are all that were required within the sql_yacc file. After saving the file, the sql_yacc.yy file was run through bison to generate new sql_yacc.cc and sql_yacc.h files. This action generate two new files which were renamed in the source directory as shown:

y.tab.c must be renamed as **sql_yacc.cc**
y.tab.h must be renamed as **sql_yacc.h**

6.1.3 Modifying the MySQL Source Code

The MySQL source files mysql_com.h, mysql.cc, item.cc,[iii] field.h and field.cc are where the new data type is defined and its code embedded.

[iii] The term 'item' is synonymous with the term 'field' or 'attribute' within the MySQL context.

6.1.3.1 Defining the Data Type in mysql_com.h

All data types are 'registered' or defined within a list of enumerated constants and defined in MySQL source file mysql_com.h. Essentially, this creates a 'legal' value within the population of data type constants. Among enumerated data types *enum_field_types* the new field type *MYSQL_TYPE_RVACHAR* was added.

6.1.3.2 Linking the Data Type to its Defined Constant

The enumerated constant that is defined for any data type that is not specifically 'primitive' must be linked to an internal type. In this case, the constant MYSQL_TYPE_RVACHAR is linked to the 'actual' data type RVACHAR. Specifically, the method

```
static const char *fieldtype2str(enum enum_field_types type)
```

gets a new line

```
case MYSQL_TYPE_RVACHAR:        return "RVACHAR";
```

which associates the actual data type with the defined data type.

6.1.4 Distributing the Data Type through the Server Processes

Now that the data type has been defined, the token can now be defined as an object and, again following the lead of the basic STRING type, distributed through the MySQL server processes. The following are given in something of a 'priority' order, although that priority is subjective.

6.1.4.1 Modifying item.cc To Define the Data Type

Because the RVACHAR is a special purpose string, the RVACHAR behaves in most ways just like a string. To support this behavior, any features or functions that are specific to strings should also apply to the RVACHAR data type. Many of these features exist in the MySQL file item.cc. The specific changes made to this file are shown as they are appear in the string type. Specifically:

1. Create the RVACHAR as a string type. The new string type will be an RVACHAR, not a string because we have defined the data type to have a unique identifier as shown by the use of the method type( ) which pushes the field construction down to the more basic Field_string.
2. The second change is to the method tmp_table_field_from_field_type shown as defined below.

   Field *Item::tmp_table_field_from_field_type(TABLE *table, bool fixed_length)

   This method is used, essentially, to create an 'item' or attribute for any of the defined data types. It should be noted that the word 'item' is used within the MySQL code to imply 'field'. Once inside this method, the RVA data type once again follows the lead of the STRING type to create the field when this method is called.

```
switch (field_type( )) {
        case MYSQL_TYPE_RVACHAR:
        case MYSQL_TYPE_STRING:
          if (fixed_length && max_length < CONVERT_IF_BIGGER_TO_BLOB)
          {
            field= new Field_string(max_length, maybe_null, name,
                                    collation.collation);
            break;
          }
```

3. The next change to the item.cc file is to the 'send' method defined below.

   bool Item::send(Protocol *protocol, String *buffer)

   This method returns a Boolean result indicating success or failure and is extremely important. As the name implies, it is instrumental in the storage and sending of the field's result. If this method is not successfully accessed, the value of the data is not stored and will not be returned to the client. The structure surrounding the 'protocol' member will be discussed

shortly when the thread structure is discussed. For the moment, all that is required is to associate the RVA data type in line with the similar data types, particularly STRING.

6.1.4.2 Modifying field.h To Define the Data Type

All but the primitive data types in MySQL are created as classes. Each data type class contains all of the essential, and many very useful, methods necessary for the functions and features that support the data type. The MySQL type STRING was used as the pattern for the new RVACHAR data type. In fact, the string methods were copied in their entirety with very few changes. The only changes made to the class definition were to reference the new data type created earlier. Specifically:

1. The new class is defined with a unique identifier

   ```
   class Field_rvachar :public Field_longstr
   ```

2. The type( ) and real_type( ) method return the defined field type as enumerated

   ```
   enum_field_types type( )        const { return MYSQL_TYPE_RVACHAR; }
   enum_field_types real_type( )   const { return MYSQL_TYPE_RVACHAR; }
   ```

6.1.4.3 Modifying field.cc To Support Data Type Functionality

Now that the class and its methods have been defined in the header file, the methods must be provided with the logic required to function not only as a character string, but as the RVA data type it is intended to be.[iv] Fortunately, the logic associated with the new RVA data type is essentially the same as it is for the STRING data type. As a result, all but one of the methods associated with the Field_string class were replicated as Field_rvachar methods. That one

[iv] The methods discussed in these sections are available in their complete form in Appendices C and D.

exception was the '*val_str()*' method which is the key to the implementation of the relation valued attribute in MySQL.

Within MySQL, each data type has a method whose purpose is to return the value obtained for that specific type. While many data types have a '*val_str()*' method that returns a string representation of the value obtained for that data type, each instance of *val_str()* is specific to the data type. By creating a new data type rvachar, we are assured that when an attribute of the RVA data type is used in a query that MySQL will consistently access the *val_str()* method associated with the rvachar class to obtain the value appropriate to that query's requirements. When using an RVA attribute, however, the *val_str()* method is accessed only one time. The first time that the *val_str()* method is accessed during the course of processing a SELECT statement, a new query, based on the original query is created. The example shown below illustrates how the original query is used and modified within the *val_str()* method to create the relation valued attribute. The modifications are shown in red.

Original Query:

```
SELECT VID, TYPE
  FROM VEHICLE;
```

New Query:

```
SELECT A.VID, B.CLASS, B.WEIGHT
  FROM VEHICLE AS A
  INNER JOIN V_TYPE AS B
    ON A.VID = B.VID;
```

While still within the *val_str()* method, this new query is passed back through the MySQL parser. Notice that the VEHICLE field TYPE does not exist in the new query. As a result, the new query will never again enter the rvachar method *val_str()*. It is the result of this new query containing the RVA data nested inside the original base table that is returned to the user while an error is set in the original query that terminates it and returns control to the user.

6.1.4.4 Detailed Description of the *val_str()* Method Functionality

All of the processing logic necessary to nest the nested table within the nesting table is contained within the rvachar class' *val_str()* method. The following sections describe, in detail, the steps taken to produce a result containing an RVA.

6.1.4.4.1 Method Header

The *val_str()* method header is shown below.

```
String *Field_rvachar::val_str(String *val_buffer __attribute__((unused)),
       String *val_ptr)
```

The *val_buffer is unused, but the *val_ptr is intended to hold the result string that is passed back to the calling process. In fact, this method's return statement is, 'return val_ptr' and the val_ptr still returns this value. Because an error has been inserted into the original thread's structure, however, this value will essentially be discarded upon return and all further processing of the original query is halted.

6.1.4.4.2 Normal Processing: Checking the Active Thread

The first thing that this method does is to check that the active thread is the current thread. This is a very important check that occurs frequently in MySQL. The following statement compares the variable table→in_use to the global variable current_thd. In the *val_str()* method, table→in_use *is* the active thread. All references to the original query or the original thread can be seen as synonymous with the variable table→in_use.

```
DBUG_ASSERT(table->in_use == current_thd);
```

Under normal operations, when a query is sent to the server and begins processing, the pthread method *my_pthread_setspecific_ptr(THR_THD, thd)* is called. This method sets the value of

current_thd to the active thread. Many methods called within MySQL begin with the assertion shown above and if the active thread does not match the current thread all further processing stops and the connection to the server will be lost. It is imperative, then, that these two threads evaluate as equal.

6.1.4.4.3 Normal Processing: Saving Off the 'Knowledge'

The value contained within the RVA field TYPE contains the knowledge needed to nest the table such as the name of the table to be nested and the key to that table which must match a portion of the key in the nesting table. As a result, normal *val_str()* processing is initially still important. After some basic validation, the value contained in the attribute is assigned to val_ptr parameter using the 'set' string method.

```
val_ptr->set((const char*) ptr, length, field_charset);
```

The value passed to val_ptr is saved off to character attribute rva_val in the first step to utilizing this information for query string modification. If, at this point, the value contained by rva_val is NULL, the RVA table attribute has not been set up properly, there is no knowledge that can be used to nest the two tables and so RVA processing is skipped and the original query is processed with no further interruption.

6.1.4.4.4 RVA Processing: Saving off the Original Thread Query

If, on the other hand, the rva_val variable does contain the necessary knowledge, the next step is to save off the original query so that it can be modified to accomplish the nesting of the two tables in this example. This action is fairly straight forward as the query is contained in the 'query_string' member of the original thread. The following command accomplishes this task.

```
sql_select = select_thd->query_string.str();
```

6.1.4.4.5 RVA Processing: Evaluating the SQL Command

Theoretically, the only reason that a query should ever enter the *val_str()* method is to satisfy the requirements of an SQL select statement. To be on the safe side, however, the SQL command is evaluated. As with most other required information, the SQL command associated with the original thread is contained in the 'sql_command' member located off of the 'lex' member branch of the thread tree.

```
enum enum_sql_command sql_command= table->in_use->lex->sql_command;
```

If the SQL command evaluates as equal to SQLCOM_SELECT, the query is a 'select' query and the tables must be nested. If not, the original query proceeds without any interruption from the nesting process.

6.1.4.4.6 RVA Processing: Building the new thread

At this point, the process has obtained the original query, the knowledge necessary to modify it and has confirmed that this is a select statement. The process has everything that it needs to create the new query. The first step in this process is to create a new thread to contain the new query and the myriad details associated with it. It is at this point that the original query is 'hijacked' by the new query that will return the nested RVA results. The following commands create a new thread and initialize the base thread elements.

```
select_thd = new THD;
select_thd->init();
```

The new thread is now a basic thread with its elements ready to be populated with the values necessary to accomplish its purpose.

It bears repeating here that a 'thread' in MySQL is not a thread in the traditional or classic sense. It is, in fact, a *highly* complex data structure that contains all aspects required of any type of query in MySQL. In the process of returning an RVA within a result relation, it will not be

necessary to use even most of the elements and functions provided by the THD class. This dissertation addresses only those elements that are required of the RVA query and response. The reader is invited to peruse the internal structure of a MySQL thread, preferably within a debugging session, so that the data contained in the elements will provide insight into the use and purpose of the elements contained within the THD structure.

6.1.4.4.7 RVA Processing: Building the new query

When the process gets to the *val_str()* method, we extract two critical pieces of data; the original thread which contains the token assignments from the original query and the knowledge encased in the value of the RVA data type. To review the example:

Original Query:

```
SELECT VID, TYPE
  FROM VEHICLE;
WHERE VEHICLE.VID >= '2';
```

New Query:

```
SELECT VEHICLE.VID,
       V_TYPE.CLASS,
       V_TYPE.WEIGHT
  FROM VEHICLE AS VEHICLE
INNER JOIN V_TYPE AS V_TYPE
    ON VEHICLE.VID = V_TYPE.VID
WHERE VEHICLE.VID >= '2';
```

The original query shown above comes in as token assignments made during the parsing of the original query. An example of this token assignment structure is shown in Figure 6.1 below. The 'item_list' is the thread structure component that contains the fields contained in the thread.

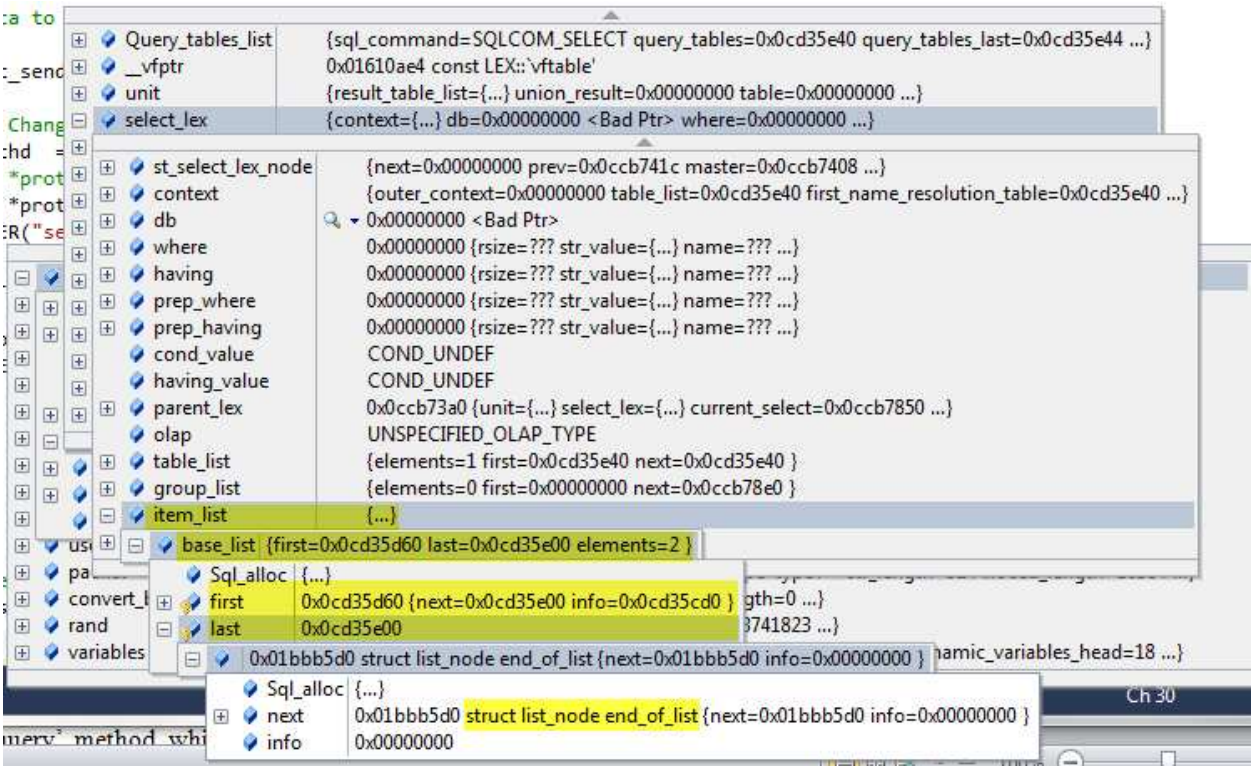


Figure 6.1 - Item_list Structure in thread.

The names of the fields are extracted from this structure using the MySQL object 'List_iterator_fast' and the <Item> structure shown here as it is used in the *build_query()* method:

```
List_iterator_fast<Item>  it(thread->lex->select_lex.item_list):
```

The knowledge contained in the RVA type field is similar to that shown here:

"@TABLE:V_TYPE@KEY:VID@FIELDS:WEIGHT,CLASS"

It contains the table to be nested, the key to the nested data and the fields to be nested. Note that there is no limit to the number of nested fields. The thread information and this knowledge are both passed to the '*build_query()*' method which returns a character string containing the new query to the *val_str()* method. The *build_query()* method is defined as (For the full implementation, please refer to Appendix D:

```
char *Field_rvachar::build_query(THD *thread, CHARSET_INFO *set, char *knowledge, char
*query)
```

Because the *build_query()* method uses the parsed version of the original query, accessing the query tokens is easy and reliable. The *build_query()* method uses these same tokens to build a new query. The first tokens that the query builder looks for are the fields contained in the SELECT statement.. In the case of the example above, these tokens contained in the item_list are VID and TYPE where TYPE is the RVA data type. Using this information, the builder will first append 'SELECT' to a new string buffer variable. It will then append the table name obtained (i.e. 'VEHICLE') from the table_list token and a period ('.') to the string 'VID,'. Thus modified, it will append the new string 'VEHICLE.VID' to the SELECT string buffer. Using the knowledge contained in the knowledge string, the builder will add the two fuzzy data attributes weight and class, to the string buffer with each proceeded, in this case, by a 'V_TYPE.'. With the result being:

SELECT VEHICLE.VID, V_TYPE.WEIGHT, V_TYPE.CLASS

We now have the SELECT portion of the new query. Using the tables obtained for the SELECT predicate described earlier, the builder then begins to build the 'FROM' clause. It does so by creating a JOIN using the two tables, VEHICLE and V_TYPE, and the value associated with the KEY found in the knowledge. In the case of this example, the FROM predicate would appear as:

FROM VEHICLE AS VEHICLE

INNER JOIN V_TYPE AS V_TYPE

ON VEHICLE.VID = V_TYPE.VID

This new FROM clause would be added to the SELECT string buffer. Specifically, 'VEHICLE' will follow the FROM clause and will be succeeded by the string 'AS VEHICLE' to become the string section 'VEHICLE AS VEHICLE'. This section will then be appended to the buffer string.

Again using the table passed by the knowledge in the method, the string V_TYPE will be preceded by 'INNER JOIN' and succeeded by the string 'AS V_TYPE'. Combined, this new string 'INNER JOIN V_TYPE AS V_TYPE' is appended to the string buffer. Finally, in this example, using the knowledge passed to the method, the value associated with '@KEY:' is evaluated on both sides of the equality. The table 'VEHICLE.' and 'V_TYPE.' precede the key attribute string VID to create a string 'ON VEHICLE.VID = V_TYPE.VID' with the whole being added to the query string. At this point, the *build_query()* method consist of the string,

"SELECT VEHICLE.VID, V_TYPE.WEIGHT, V_TYPE.CLASS

FROM VEHICLE AS VEHICLE INNER JOIN V_TYPE AS V_TYPE

ON VEHICLE.VID = V_TYPE.VID"

The last part of this example is the restriction found in the WHERE predicate. The restriction is, without a doubt, the most complicated of the query sections and this complexity is reflected in the thread structure. It is beyond the scope of this research to accommodate all WHERE predicate possibilities, but the example demonstrates the essential approach being used by MySQL. It is necessary here to briefly describe the WHERE component of the thread structure and how it works. To begin with, recall that the WHERE predicate was stated as:

WHERE VID >= '2';

The select_lex.where element of the thread is a linked list that prepends the tokens to the list in prefix order. The followings screens illustrate how VID >= '2' is maintained in the WHERE list.

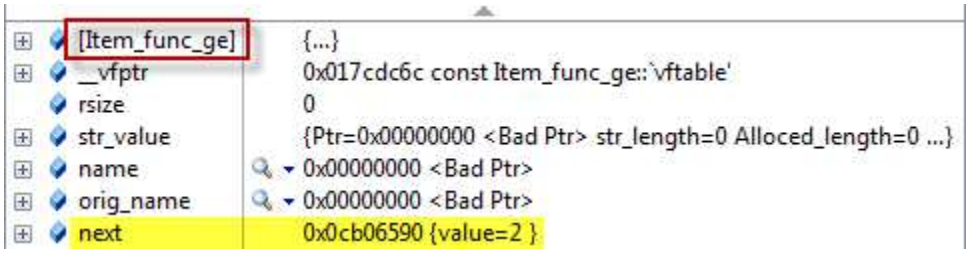


Figure 6.2 - Item Function

The head of the list is the ‘func_type’ which in this case is ‘greater than or equal to’ or ‘GE’. The pointer to the next element in the list shows the value = 2.

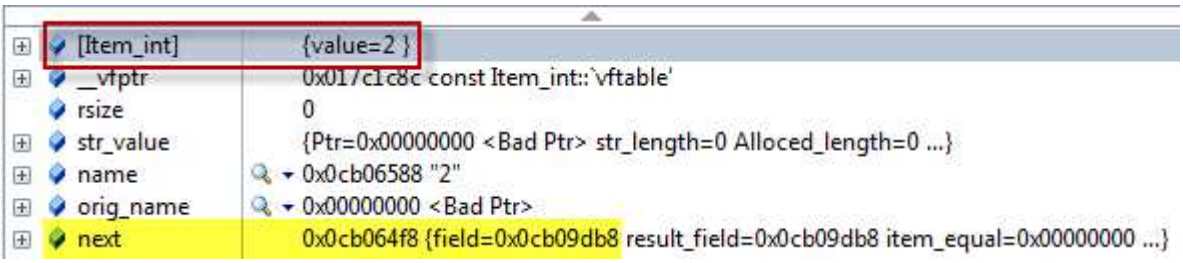


Figure 6.3 - Item Value

The func_type for this element is, essentially, ‘int’ which is the data type of the value. The field VID is of type integer and so it is only appropriate that an integer value be evaluated to it.

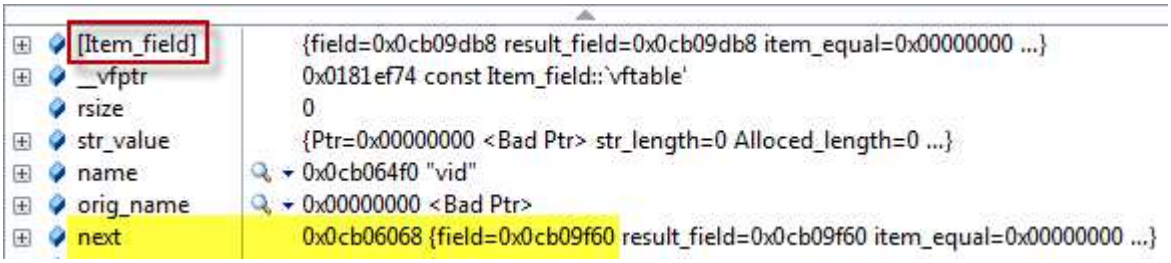


Figure 6.4 - Item Field

The last element of the WHERE list has the func_type ‘field’ and contains the value ‘VID’. So the list elements flow in prefix fashion as ‘GE’ → ‘2’ → VID which are the two values to be compared and the comparison operator between them. So the WHERE list has all of the elements contained in the WHERE clause. The job now is to access them and create a string with these elements to finish off the selection query.

Because the WHERE predicate can contain more than one comparison, the *build_query()* method provides an Item pointer to the next item in the where list, starting with the first. It is declared as: **Item *ptr_next**. This pointer allows the process to traverse the WHERE list and because it is of type Item, it contains all of the members inherent in the WHERE list which is of type Item.

The next object provided for use to the method is a variable called ‘my_func’ which is defined as: **Item_func::Functype my_func**. This variable and the method Functype contained with the object Item_func are of critical importance. In Figure 6.2, the element ‘func_item_ge’ is an element of an Item_func array. Each of these functions contain complex members appropriate to just that function type. Fortunately, all that the *build_query()* method requires from this complex structure is the comparison operator. The process obtains this value through the assignment:

```
my_func = ((Item_func*)ptr_next)->functype();
```

my_func will be assigned a number. This number will correspond to a member in the enumerated list of operators contained in Item_func’s functype variable. Below is the list of enumerated values. Those of particular interest to the *build_query()* method are shown below:

```
enum Functype { UNKNOWN_FUNC,EQ_FUNC,EQUAL_FUNC,NE_FUNC,LT_FUNC,LE_FUNC,
                GE_FUNC,GT_FUNC...};
```

Notice, however, that there are many very familiar SQL function types from OR, XOR, AND and IN to those used for nested and more complex queries.

The next token obtained is the value being compared, in this case ‘2’. Then, the field being compared is obtained which in this case is ‘VID’. Putting these three tokens together in a string with WHERE, the result is WHERE VID >= 2 which is what is needed. This string is appended to the whole to leave the *build_query()* with a final select statement of:

Query 6.1

```
SELECT VEHICLE.VID,
       V_TYPE.CLASS,
       V_TYPE.WEIGHT
  FROM VEHICLE AS VEHICLE
 INNER JOIN V_TYPE AS V_TYPE
    ON VEHICLE.VID = V_TYPE.VID
WHERE VEHICLE.VID >= ‘2’;
```

The final query string is then assigned to the character string ‘query’ and returned to the *val_str()* method from which it was called.

6.1.4.4.8 RVA Processing: Processing the new query

When the new query string is returned to the *val_str()* method, it is measured using the standard strlen(*String) method and saved to the thread ‘query_string’ element with the method calls

```
select_thd->set_query(my_sql_string, query_length );
select_thd->query_string = CSET_STRING(my_sql_string, query_length,field_charset);
```

This key section is critical. The new query must be a reliably well-formed statement. Once created, however, any errors in the statement will be returned to the user as though the query was passed to the server in its new form. For example, if the new query were malformed, an error returned may not reference any element in the original query. Such messages would be quite confusing to a user who would have no ability to affect the syntax of the ‘built’ query.

6.1.4.4.9 RVA Processing: Allocating Memory

Memory allocation and buffer size is critical to execution of a query. In some cases, memory is dynamically allocated during the query execution. In other cases it is not. But an initial value sufficient to the queries needs can be assigned at this point. The first step is to create a new mem_root element in the select_thd. Once created, the mem_root allocations in the *val_str()* method were set to default values.

Separate, but related, the select thread query allocation is set in the ‘alloc’ member of the select_thd to a length equal to the length of the new query built earlier.

```
unsigned int query_size = query_length;
select_thd->alloc(query_size);
```

6.1.4.4.10 Assigning the 'net' element to the thread

There are a number of critical elements contained within the original thread that must be assigned to the new thread. Perhaps the most critical of these elements is the 'net' element. The net element is the 'client connection descriptor' and contains both the network parameters as well as the client identifier. Without this information, the communication back to the client is lost. The only information that needs to be changed in the net element is the contents of the buffer, which will contain the new query and the lengths of these buffers to hold the new query header and result. These buffers will be addressed in a moment. For now, here is the assignment of the net element to the new thread.

```
select_thd->net             = table->in_use->net;
select_thd->stmt_da         = table->in_use->stmt_da;
```

6.1.4.4.11 RVA Processing: Preparing the query buffers

The query thread uses four 'working buffers' to hold the query, navigate the request and hold the result. These buffers are the buff, buff_end, write_pos and read_pos. Initially, the buff, write_pos and read_pos variables contain the query as submitted. As the parser moves through the query string and extracts the markers these buffers are consumed. During the actual data selection, these buffers will contain various result components which will then be written to the 'packet' which will be discussed in Section 6.1.4.4.12. In short, the buffer areas are an important element in the thread structure as these are the workhorse buffers that provide temporary storage for query operations.

The variable buff_end marks the limit of the query. This is a very important variable that must be set large enough to handle the result set and the 'header' which is written before the selection.

Here, the buff_end variable is set to the default value and should be more than sufficient to the need.

6.1.4.4.12 RVA Processing: The packet

The packet holds the entire result set. By the time the original query gets to the *val_str()* structure, its packet has been 'polluted' with results already obtained from the original query. As a result, the new thread is provided with a clean, new packet area with an initial default memory allocation. At the completion of the query process, the 'send' class object will access the new thread's packet rather than the original thread's packet and 'flush' it to the client as output.

6.1.4.4.13 RVA Processing: Security settings

When the new thread is created, it has no security value. Without a security value, the 'user' authorizations cannot be interrogated by the query execution to determine whether the user has authorization to this particular action or the specific tables involved. The simple thing to do here is to copy the security settings that came in the original query. The *val_str()* method does just that.

6.1.4.4.14 RVA Processing: The Parser State

When the new thread is created, it has no active parser state. The primary element of the parser state is the Lex input stream. The first step is to initialize a new parser state element for the new thread. Next, a new Lex input stream or 'lip' must be created, initialized and have its initial value assigned to it. The MySQL documentation provides a good summary of the Lexical input stream and is presented here.

Class: Lex_input_stream

This class represents the character input stream consumed during lexical analysis.

In addition to consuming the input stream, this class performs some comment preprocessing, by filtering out-of-bound special text from the query input stream. Two buffers, with pointers inside each buffer, are maintained in parallel. The 'raw' buffer is the original query text, which may contain out-of-bound comments. The 'cpp' (for comments preprocessor) is the pre-processed buffer that contains only the query text that should be seen once out-of-bound data is removed.

The parameters passed to the lexical input stream are the thread pointer, the query string and the query string's length. Once instantiated and with initial values assigned, the 'lip' is assigned to the parser state.

A new parser state is created for the new thread, too. But this is not the same parser state. In fact, all that is required for the thread is to instantiate the parser state and assign a value of NULL while a value of 0 is assigned to the 'safe_to_cache_query' element on the lex branch of the thread tree to signal that the query is safe to cache.

6.1.4.4.15 RVA Processing: Protocol

The 'protocol' element of the thread handles the status of the query processing and more importantly the 'send status' and the 'end_statement'. It is by calling the protocol's end statement method that the results contained within the packet are actually sent to the client. At this point, however, all that is required is that the protocol element be initialized to reference the new thread.

```
select_thd->protocol->init(select_thd);
```

6.1.4.4.16 RVA Processing: Client capabilities

The client_capabilities element of the thread variable express whether the client is capable of such things as multiple queries or multiple statements. It is not necessary to determine this value for the new thread. Rather, the client_capabilities value can be copied directly from the original thread.

```
select_thd->client_capabilities = table->in_use->client_capabilities;
```

6.1.4.4.17 RVA Processing: Setting the current thread

Because MySQL is a client server product and methods can be called through various client threads at any time, there are a number of checks to ensure that the thread entering a particular method is the 'current thread'. The current thread is represented by a value in the variable current_thd. When the original query enters the *val_str()* method, the value contained in current_thd is equal to the original thread. In fact, the evaluation of the thread within the methods is through the debug macro DEBUG_ASSERT(thd = = current_thd). If the assertion returns in the negative, the link between the client and server is dropped. It is therefore necessary to set the new thread to the current thread. This is done using the statement shown below.

```
my_pthread_setspecific_ptr(THR_THD, select_thd);
```

6.1.4.4.18 RVA Processing: Calling *mysql_parse()*

Everything that must be done and provided for the new thread has now been done. The query has been set, the network parameters have been established, the buffers have been instantiated and populated as necessary and the required memory has been allocated. The process now calls the MySQL method *mysql_parse()* as follows.

```
mysql_parse(select_thd,
            select_thd->query(),
            select_thd->query_length(),
            my_parser_state);
```

The four parameters are:

1. The new thread variable containing the basic, but essential elements necessary to parse and execute the query.
2. The query string.
3. The length of the query string.
4. The independent parser state

6.1.4.4.19 RVA Processing: Use of the *mysql_parse()* Method Rationale

There is a reason that the *mysql_parse()* method was used in this instance. Several methods and alternatives were considered and tried. As mentioned earlier, the THD or thread structure at the heart of MySQL is extraordinarily complex. Nearly all of the methods contained in MySQL either read data from or write data to this object. As a result, the problem with selecting the 'right' method had to do with finding the right 'entry point' for the new thread and making a determination as to which of these entry points would provide the new thread with the most MySQL functionality. Choosing the wrong method would have meant spending considerable effort in replicating the processes that already existed in MySQL and resulted in a product that was error prone and highly unreliable.

It was originally thought that the MySQL method '*mysql_select()*' was the appropriate entry point to use. The method's parameters appeared to have all the 'pieces and parts' necessary to proceed if only the process could be designed to appropriately assign a value to each of these parameters. Considerable effort was expended in an attempt to do this. Two facts eventually became apparent. First, given the wide variety of selections being made in a query, it would be almost impossible to correctly populate all of the associated thread elements and variables. More

importantly, however, attempting to do so would replicate, in most every respect, the job already being done by the parser method. It is the job of the parser to collect an input string and to break the tokens contained within the string down into its various components. In fact, the parser method, '*mysql_parse()*' required only four parameters; the thread to hold the communications information and the results of the parser's efforts, the query for the parser to 'parse', the length of the query to assist in string manipulation and a variable provided to hold the parser state of the object. By using the MySQL parser method, the entry point for the new method was far enough 'up stream' to utilize all of the MySQL functionality necessary to populate the thread with every piece of required information. In addition, if the query were a well formed and executable statement, it was the job of the parser method to execute the query and store the results of the query. Essentially, the input to the parser was the query to be processed. The output of the parser was the results of the query's execution.

Using the parser allowed all of the RVA's nesting functionality to be called from and encapsulated within the *val_str()* method. With rare and minor exception, no other MySQL process functionality required change. While the use of almost all of the MySQL functionality is an obvious benefit to this approach, the other key benefit was the fact that using this approach completely eliminated any potential and unforeseen mal-effect to changing MySQL code outside of the *val_str()* method.

6.1.4.4.20 RVA Processing: Sending the result to the client

The output of a successful call to the parser method is a packet variable containing the results of the query contained within the thread structure. The next step is to access this packet and send it

to the client as the result. To accomplish this task, a select_send object needed to be initialized and assigned the value of the thread and its essential elements.

Once prepared, the thread would update the server with the appropriate status flags and call the '*end_statement()*' method. It is the *end_statement()* method that actually sends the result to the client by flushing the results buffer. The result buffer contains the result header consisting of the column headings and the results consisting of the 'items' contained within each result record.

6.1.4.4.21 RVA Processing: Terminating the original query

As alluded to earlier, the *val_str()* method was called as part of the original query processing. The method that called the *val_str()* method expects a string to be returned to it and to then continue processing the original query. The RVA *val_str()* method, however, has already processed and returned a result from its query. As a result, the original query must be terminated and control must return to the user on the client side. To accomplish this, an 'error' is set. By setting an error, the calling method will receive a result, but that result will come with an error.

```
/* Set the error that stops further original SQL processing. */
       success.set("RVA Success", 11, field_charset);
       my_error(ER_CHECK_NOT_IMPLEMENTED, MYF(0), success);
```

Before the process returns, however, the global current_thd set to the new thread must first be reset back to the original thread. To do this, the original thread is simply passed as a parameter to the set specific method.

```
my_pthread_setspecific_ptr(THR_THD, table->in_use);
```

The is_sent element of both the original and the new thread must be set to FALSE. MySQL will drop the connection between the client and the server if a process tries to send an error to the client *after* the result has been sent. If an error occurs in processing the expectation is that it will

occur before a result is obtained and sent to the user. In testing, however, there has been no mal-effect from setting the ‘is_sent’ element to FALSE even though the result has, in fact, been sent. When the calling method is informed that an error has occurred, it will stop all further processing, free any allocated memory, unlock the tables used in the original query and return control to the user.

Chapter 7 - Validation

7.1 The Approach to Validation

A validation of the system developed for the use of RVA's in support of fuzzy data was performed to confirm and demonstrate the ability of the system to satisfy several primary goals. The first of these goals was to demonstrate that the system not only supports the necessary functionality of the relational database, but also adheres to the strict mathematical requirements of the relational model. Specifically, the use of RVA's must appropriately accommodate the five primitive operations established by Codd as required for the relational model; the Cartesian Product and the Restriction, Projection, Join and Intersection operations.[7]

The next goal was to demonstrate the ability of the system to accommodate the use of RVA's in the support of non-fuzzy data. While the focus of this research is the use of RVA's to represent fuzzy data, RVA's are not limited to this task. RVA's can also be used to represent other, more common, data requirements. To illustrate this additional role of relation valued attributes, a non-fuzzy data example is provided and discussed in Section 7.2.6.

Another important goal of this research is to consider the representation of an RVA within the result of a query. There are two general representations that have been used. Chapter 9 discusses these representations, their benefits and drawbacks. During the formal system study the opinions of users as to which representation is more meaningful and suitable to their needs was elicited.

In addition to the representation, the new database system provides the user with a new and unfamiliar tool. Queries are somewhat different and the results may not be as expected. In order

to help gauge the effect on the user resulting from these differences a number of heuristic questions were asked to gain insight into the user's reaction.

Finally, it was necessary to demonstrate and confirm that the introduction of the RVA within the database does not affect the normal functionality of the database to support and maintain non-fuzzy data.

7.2 Data Used in Validation

The tests used to validate the representation of fuzzy data using relation valued attributes consisted of a database comprised of tables VEH_TAMPA, VEH_MIAMI and V_TYPE. The structure of VEH_TAMPA and VEH_MIAMI are shown in Table 7.1. Table V_TYPE is shown in Table 7.2:

```
+-------+--------------+------+-----+-----------------------------------------+
| Field | Type         | Null | Key | Default                                 |
+-------+--------------+------+-----+-----------------------------------------+
| vid   | int(11)      | NO   | PRI | NULL                                    |
| loc   | char(25)     | YES  |     | Tampa                                   |
| name  | char(25)     | YES  |     | NULL                                    |
| manuf | char(25)     | YES  |     | NULL                                    |
| type  | rvachar(125) | NO   |     | @table:v_type@key:vid@fields:weight,class |
+-------+--------------+------+-----+-----------------------------------------+
```

Table 7.1 - VEH_TAMPA and VEH_MIAMI

```
+--------+--------------------------+------+-----+---------+
| Field  | Type                     | Null | Key | Default |
+--------+--------------------------+------+-----+---------+
| vid    | int(11)                  | NO   | PRI | NULL    |
| weight | decimal(4,2)             | YES  |     | NULL    |
| class  | enum('Car','Truck','Van') | NO   | PRI | Car     |
+--------+--------------------------+------+-----+---------+
```

Table 7.2 - V_TYPE

The data contained within these tables are shown in Tables 7.3, 7.4 and 7.5. The query to select the data from the tables is shown with the result of the query. VEH_TAMPA and VEH_MIAMI RVA attribute *type* is specified. When specified in the query, the relation specified by the RVA,

namely V_TYPE consisting of attributes containing the ordered pair (weight,class) necessary for the representation of fuzzy data is nested within the base table.[2][v]

```
mysql-VCU> select vid, loc, name, manuf, type from
veh_tampa;
+-----+-------+-----------+-----------+--------+-------+
| vid | loc   | name      | manuf     | WEIGHT | CLASS |
+-----+-------+-----------+-----------+--------+-------+
|   1 | Tampa | El Camino | Chevrolet |   0.90 | Car   |
|   1 | Tampa | El Camino | Chevrolet |   0.60 | Truck |
|   2 | Tampa | Camero    | Chevrolet |   1.00 | Car   |
|   3 | Tampa | F-150     | Ford      |   0.70 | Car   |
|   3 | Tampa | F-150     | Ford      |   1.00 | Truck |
+-----+-------+-----------+-----------+--------+-------+
5 rows in set (0.18 sec)
```

Table 7.3 - VEH_TAMPA Data

```
mysql-VCU> select vid, loc, name, manuf, type from
 veh_miami;
+-----+-------+---------+-------+--------+-------+
| vid | loc   | name    | manuf | WEIGHT | CLASS |
+-----+-------+---------+-------+--------+-------+
|   3 | Miami | F-150   | Ford  |   0.70 | Car   |
|   3 | Miami | F-150   | Ford  |   1.00 | Truck |
|   4 | Miami | Caravan | Dodge |   0.80 | Car   |
|   4 | Miami | Caravan | Dodge |   0.90 | Van   |
|   5 | Miami | Mustang | Ford  |   1.00 | Car   |
+-----+-------+---------+-------+--------+-------+
5 rows in set (0.01 sec)
```

Table 7.4 - VEH_MIAMI Data

The value in the RVA field *type* consists of the data knowledge required to nest table V_TYPE into the nesting tables as appropriate, assigning the various fuzzy data relations to the associated *vid*.

[v] Note: In order to maintain the format of the results, the fixed font 'Courier New' has been used to display the results of the queries which were copied directly from the system screens. In this way, the integrity of the result is best maintained.

```
mysql-VCU> select * from v_type;
+-----+--------+-------+
| vid | weight | class |
+-----+--------+-------+
|   1 |   0.90 | Car   |
|   1 |   0.60 | Truck |
|   2 |   1.00 | Car   |
|   3 |   0.70 | Car   |
|   3 |   1.00 | Truck |
|   4 |   0.80 | Car   |
|   4 |   0.90 | Van   |
|   5 |   1.00 | Car   |
+-----+--------+-------+
8 rows in set (0.09 sec)
```

Table 7.5 - V_TYPE Data

7.2.1 Cartesian Product

The Cartesian product consists of two vehicle tables, VEH_TAMPA and VEH_MIAMI, both of which contain an RVA attribute TYPE. In both cases, TYPE contains data consisting of the following knowledge as shown in the description in Table 7.1:

@TABLE:V_TYPE@KEY:VID@FIELDS:WEIGHT,CLASS;

The Cartesian product will return a set consisting of data from both tables, each containing the RVA data appropriate to both. The sequence of the SQL query entered by the user, the query as modified by the nesting method '*build_query()*' based on the thread elements involved and the final query used to return the result set is as follows:

Original Query:

```
Select *                                                    Query 7.1
  From veh_tampa, veh_miami;
```

Thread→Item:

```
select veh_tampa.vid, veh_tampa.loc,
      a.class, a.weight,        ←TYPE for tampa
      veh_miami.vid, veh_miami.loc,
      b.class, b.weight         ←TYPE for Miami
```

Thread→Tables:

```
 from veh_tampa as veh_tampa inner join v_type as a on veh_tampa.vid = a.vid,
```

veh_miami as veh_miami inner join v_type as b on veh_miami.vid = b.vid

Final Query:

```
Select veh_tampa.vid, veh_tampa.loc, a.class, a.weight
       veh_miami.vid, veh_miami.loc, b.class, b.weight

  From veh_tampa as veh_tampa inner join v_type as a on veh_tampa.vid = a.vid,
       veh_miami as veh_miami inner join v_type as b on veh_miami.vid = b.vid;
```

The result is shown in Table 7.6:

```
mysql-VCU> select * from veh_tampa, veh_miami;
+-----+-------+-----------+-----------+--------+-------+-----+-------+---------+-------+--------+-------+
| vid | loc   | name      | manuf     | WEIGHT | CLASS | vid | loc   | name    | manuf | WEIGHT | CLASS |
+-----+-------+-----------+-----------+--------+-------+-----+-------+---------+-------+--------+-------+
|   1 | Tampa | El Camino | Chevrolet |   0.90 | Car   |   3 | Miami | F-150   | Ford  |   0.70 | Car   |
|   1 | Tampa | El Camino | Chevrolet |   0.90 | Car   |   3 | Miami | F-150   | Ford  |   1.00 | Truck |
|   1 | Tampa | El Camino | Chevrolet |   0.60 | Truck |   3 | Miami | F-150   | Ford  |   0.70 | Car   |
|   1 | Tampa | El Camino | Chevrolet |   0.60 | Truck |   3 | Miami | F-150   | Ford  |   1.00 | Truck |
|   2 | Tampa | Camero    | Chevrolet |   1.00 | Car   |   3 | Miami | F-150   | Ford  |   0.70 | Car   |
|   2 | Tampa | Camero    | Chevrolet |   1.00 | Car   |   3 | Miami | F-150   | Ford  |   1.00 | Truck |
|   3 | Tampa | F-150     | Ford      |   0.70 | Car   |   3 | Miami | F-150   | Ford  |   0.70 | Car   |
|   3 | Tampa | F-150     | Ford      |   0.70 | Car   |   3 | Miami | F-150   | Ford  |   1.00 | Truck |
|   3 | Tampa | F-150     | Ford      |   1.00 | Truck |   3 | Miami | F-150   | Ford  |   0.70 | Car   |
|   3 | Tampa | F-150     | Ford      |   1.00 | Truck |   3 | Miami | F-150   | Ford  |   1.00 | Truck |
|   1 | Tampa | El Camino | Chevrolet |   0.90 | Car   |   4 | Miami | Caravan | Dodge |   0.80 | Car   |
|   1 | Tampa | El Camino | Chevrolet |   0.90 | Car   |   4 | Miami | Caravan | Dodge |   0.90 | Van   |
|   1 | Tampa | El Camino | Chevrolet |   0.60 | Truck |   4 | Miami | Caravan | Dodge |   0.80 | Car   |
|   1 | Tampa | El Camino | Chevrolet |   0.60 | Truck |   4 | Miami | Caravan | Dodge |   0.90 | Van   |
|   2 | Tampa | Camero    | Chevrolet |   1.00 | Car   |   4 | Miami | Caravan | Dodge |   0.80 | Car   |
|   2 | Tampa | Camero    | Chevrolet |   1.00 | Car   |   4 | Miami | Caravan | Dodge |   0.90 | Van   |
|   3 | Tampa | F-150     | Ford      |   0.70 | Car   |   4 | Miami | Caravan | Dodge |   0.80 | Car   |
|   3 | Tampa | F-150     | Ford      |   0.70 | Car   |   4 | Miami | Caravan | Dodge |   0.90 | Van   |
|   3 | Tampa | F-150     | Ford      |   1.00 | Truck |   4 | Miami | Caravan | Dodge |   0.80 | Car   |
|   3 | Tampa | F-150     | Ford      |   1.00 | Truck |   4 | Miami | Caravan | Dodge |   0.90 | Van   |
|   1 | Tampa | El Camino | Chevrolet |   0.90 | Car   |   5 | Miami | Mustang | Ford  |   1.00 | Car   |
|   1 | Tampa | El Camino | Chevrolet |   0.60 | Truck |   5 | Miami | Mustang | Ford  |   1.00 | Car   |
|   2 | Tampa | Camero    | Chevrolet |   1.00 | Car   |   5 | Miami | Mustang | Ford  |   1.00 | Car   |
|   3 | Tampa | F-150     | Ford      |   0.70 | Car   |   5 | Miami | Mustang | Ford  |   1.00 | Car   |
|   3 | Tampa | F-150     | Ford      |   1.00 | Truck |   5 | Miami | Mustang | Ford  |   1.00 | Car   |
+-----+-------+-----------+-----------+--------+-------+-----+-------+---------+-------+--------+-------+
25 rows in set (0.24 sec)
```

Table 7.6

7.2.2 Restrict

The restrict operation limits the result by the value specified in the WHERE predicate of the query. When used with non-fuzzy data, the restrict operation is straight forward. The example in Query 7.2 demonstrates this operation using non-fuzzy data both as an example and to confirm that the incorporation of an RVA within the relation database does not affect the standard functionality of the database. Query 7.2 selects the attributes *vid*, *loc*, *manuf* and *name* from table VEH_TAMPA where the manuf is equal to 'Chevrolet' and displays it in Table 7.7. The RVA attribute *type* was intentionally omitted.

Query 7.2

```
mysql-VCU> select vid, loc, manuf, name
             from veh_tampa
            where manuf = 'Chevrolet';
```

```
+-----+-------+-----------+-----------+
| vid | loc   | manuf     | name      |
+-----+-------+-----------+-----------+
|   1 | Tampa | Chevrolet | El Camino |
|   2 | Tampa | Chevrolet | Camero    |
+-----+-------+-----------+-----------+
2 rows in set (0.00 sec)
```

Table 7.7

The attribute *type* can also be selected in a restriction as any other attribute. For example, Query 7.3 can be modified to include attribute *type* and provide the result shown in Table 7.8.

Query 7.3

```
mysql-VCU> select vid, loc, manuf, name, type
             from veh_tampa
            where manuf = 'Chevrolet';
```

```
+-----+-------+-----------+-----------+--------+-------+
| vid | loc   | name      | manuf     | WEIGHT | CLASS |
+-----+-------+-----------+-----------+--------+-------+
|   1 | Tampa | El Camino | Chevrolet |   0.65 | Car   |
|   1 | Tampa | El Camino | Chevrolet |   0.75 | Truck |
|   2 | Tampa | Camero    | Chevrolet |   1.00 | Car   |
+-----+-------+-----------+-----------+--------+-------+
3 rows in set (0.06 sec)
```

Table 7.8

When the query includes an RVA and there is the necessity to specifically restrict on a value contained in the RVA, the restrict operation is somewhat different. For example, the user cannot directly access the RVA's attribute *class*. This is because *class* is not an attribute contained in table VEH_TAMPA. The attribute *class* is a member of the relation nested in the RVA attribute *type*. As a result, a user cannot simply specify *where class = 'Car'*. But using the intersection operator IN, the same result can be obtained. Query 7.4 demonstrates this feature by selecting vehicle data from table VEH_TAMPA where the class of the vehicle is 'Car'. In this case, all that is required is to transform the original selection to an inner join. The restriction provided by the inner join is sufficient to limit the result set to only those values appropriate to the request. Notice that in this case, the WHERE predicate references the IN operator and specifies the RVA restrictions within the nested SELECT of table V_TYPE. The sequence of the SQL query entered by the user, the query as modified by the nesting method '*build_query()*' based on the thread elements involved and the final query used to return the result set is as follows:

Query: **Query 7.4**

```
Select vid, manuf, name, type
 From veh_tampa
Where vid in (select vid from v_type
              Where class = 'Car');
```

Thread→Item:

```
select veh_tampa.vid, veh_tampa.manuf, veh_tampa.name,
```

a.class, a.weight, ←TYPE for tampa

Thread→Tables:

```
from veh_tampa as veh_tampa
inner join v_type as a
   on veh_tampa.vid = a.vid
  and a.class = 'Car' );
```

Final Query:

```
select veh_tampa.vid, veh_tampa.manuf, veh_tampa.name,
       a.class, a.weight,
from veh_tampa as veh_tampa
inner join v_type as a
   on veh_tampa.vid = a.vid
  and a.class = 'Car');
```

The Result appears in Table 7.9:

```
+-----+-----------+-----------+--------+-------+
| vid | manuf     | name      | WEIGHT | CLASS |
+-----+-----------+-----------+--------+-------+
|   1 | Chevrolet | El Camino |   0.90 | Car   |
|   2 | Chevrolet | Camero    |   1.00 | Car   |
|   3 | Ford      | F-150     |   0.70 | Car   |
+-----+-----------+-----------+--------+-------+
3 rows in set (0.12 sec)
```

Table 7.9

The functional operations, '=', '>', '<' and even 'IN' are contained within the 'WHERE' element of the query thread. The 'greater than' operation used to evaluate the attribute weight and the value under consideration are obtained through the use of the *func_type()* method.

7.2.3 Project

The project operation specifies attributes and the order in which they will appear in the result. Continuing the example from Query 7.3, the example shown in Query 7.5 selects the attributes *vid*, *manuf* and *name* from table VEH_TAMPA. The attribute loc, shown in Query 7.3 has been excluded from or 'projected out' of the selection and subsequent result. Query 7.5 demonstrates

this operation both as an example and to confirm that the incorporation of an RVA within the relation database does not affect the standard functionality of the database.

```
mysql-VCU> select vid, manuf, name                    Query 7.5
             from veh_tampa;
```

```
+-----+-----------+-----------+
| vid | manuf     | name      |
+-----+-----------+-----------+
|   1 | Chevrolet | El Camino |
|   2 | Chevrolet | Camero    |
|   3 | Ford      | F-150     |
+-----+-----------+-----------+
3 rows in set (0.00 sec)
```

Table 7.10

The purpose of the RVA, however, is to prohibit the exclusion of either element of the ordered pair necessary to the representation of fuzzy data. In fact, the primary objective of this research was to prohibit the individual exclusion of either class or weight of membership from the result. So, while it is possible and desirable to access the attributes WEIGHT and CLASS directly by selecting these values from the unnested table V_TYPE, it is not possible to access values from the WEIGHT or CLASS attributes *individually* from the RVA attribute TYPE. Values from these individual attributes cannot be directly accessed using the SELECT predicate against table VEH_TAMPA because the attributes contained within the relation nested inside the RVA cannot be accessed directly. Query 7.6 illustrates what happens when such an attempt is made.

```
mysql-VCU> select vid, manuf, name, type.class        Query 7.6
             from veh_tampa;
```

ERROR 1054 (42S22): Unknown column '*type.class*' in 'field list'

Selecting attribute TYPE from table VEH_TAMPA returns both WEIGHT and CLASS in the order specified by the RVA type without exception. Because table V_TYPE exists in its natural state as a standard table in MySQL, it can be maintained and assigned values just as any other

table using SQL. This point is a very important one as it allows the MySQL processes and the natural SQL language to affect table V_TYPE just as any other table in MySQL. That is because it *is* a table in MySQL. It is the RVA attribute TYPE that gives this table a special purpose and constrains the access to its data through the unique means.

7.2.4 Join

As the name suggests, the join operation joins two or more tables together to provide one result.

7.2.4.1 Natural Join

To demonstrate this operation, a new table MAINT was added to the database. The structure of this table and sample data is provided in Tables 7.11 and 7.12.

Field	Type	Null	Key	Default
vid	int(11)	NO	PRI	NULL
engine	char(20)	YES		NULL
trans	enum('Auto','Manual','Other')	YES		NULL
axles	int(2)	YES		NULL

Table 7.11 - Structure of Table MAINT

vid	engine	trans	axles
1	short block 350	Manual	2
2	323 horsepower V6	Manual	2
3	3.7-liter V6	Auto	2

Table 7.12 - Data in Table MAINT

Using the data in tables VEH_TAMPA and MAINT, an example of a join using non-fuzzy data is shown in Query 7.7. Query 7.7 provides as an example to confirm that the incorporation of an RVA within the relation database does not affect the standard functionality of the join operation as it is used in the relational database.

```
mysql-VCU> select a.vid, a.name, a.manuf,                    Query 7.7
                  b.engine, b.trans, b.axles
             from veh_tampa  as a
            inner join maint as b
               on a.vid = b.vid;


+-----+-----------+-----------+--------------------+--------+-------+
| vid | name      | manuf     | engine             | trans  | axles |
+-----+-----------+-----------+--------------------+--------+-------+
|   1 | El Camino | Chevrolet | short block 350    | Manual |     2 |
|   2 | Camero    | Chevrolet | 323 horsepower V6  | Manual |     2 |
|   3 | F-150     | Ford      | 3.7-liter V6       | Auto   |     2 |
+-----+-----------+-----------+--------------------+--------+-------+
3 rows in set (0.00 sec)
```

Table 7.13

This table contains a primary key VID which has a foreign key relationship with base table VEH_TAMPA as well as attributes containing ENGINE, TRANS (Transmission), AXLES (Number of Axles).

7.2.4.2 Join with Table Containing an RVA

The RVA is, first and foremost, an attribute. As such, it can be specified as any other attribute in the selection of data. The example shown in Query 7.8 demonstrates this operation on a selection containing an RVA.

The Query: **Query 7.8**

```
SELECT veh_tampa.vid,
       veh_tampa.name,
       veh_tampa.type,
       maint.engine,
       maint.trans,
       maint.axles
  FROM veh_tampa as veh_tampa inner join
       maint as maint
    ON veh_tampa.vid = maint.vid
 WHERE veh_tampa.manuf = 'Ford';
```

Thread → Item

```
SELECT veh_tampa.vid,
       veh_tampa.name,
       a.weight,
```

```
                a.class,
                maint.engine,
                maint.trans,
                maint.axles
```

Thread → Where

```
          FROM veh_tampa as veh_tampa
                inner join  v_type as a
             ON tampa.vid = a.vid,
                inner join maint as maint
             ON veh_tampa.vid = maint.vid
         WHERE tampa.manuf = 'Ford';
```

Final Query:

```
        SELECT veh_tampa.vid, veh_tampa.name,
                a.weight, a.class,
                maint.engine, maint.trans, maint.axles
          FROM veh_tampa as veh_tampa
                inner join v_type as a
             ON a.vid = veh_tampa.vid
                inner join maint as maint
             ON maint.vid = veh_tampa.vid
        WHERE veh_tampa.manuf = 'Ford';
```

The Result is shown in Table 7.14

```
+-----+-------+--------+-------+--------------+-------+-------+
| vid | name  | WEIGHT | CLASS | engine       | trans | axles |
+-----+-------+--------+-------+--------------+-------+-------+
|   3 | F-150 |   0.70 | Car   | 3.7-liter V6 | Auto  |     2 |
|   3 | F-150 |   1.00 | Truck | 3.7-liter V6 | Auto  |     2 |
+-----+-------+--------+-------+--------------+-------+-------+
2 rows in set (0.01 sec)
```

Table 7.14

As can be seen, the changes to the original query are relatively minor. All that is required is to insert the attributes WEIGHT and CLASS and the INNER JOIN on table V_TYPE into the query by the server.

```
select tampa.vid, tampa.name,a.weight,a.class,maint.engine,maint.trans
 from veh_tampa as tampa inner join v_type as a on a.vid = tampa.vid,
 inner join maint as maint on maint.vid = tampa.vid where tampa.manuf = 'Ford';
```

7.2.4.3 Join on Two Relation Valued Attributes

The ability to join two tables based on the values contained in a common RVA is theoretically possible. This functionality, however, is not yet available in the current version of MySQL but may be included in future research and a brief discussion of this operation is appropriate here. Because an RVA is an attribute, the precept shown in Query 7.9 is a valid SQL condition.

```
SELECT A.VID, A.LOC AS LOC1, B.LOC AS LOC2,
       A.NAME, A.MANUF, A.TYPE
  FROM  VEH_TAMPA AS A
 INNER  JOIN VEH_MIAMI AS B
    ON A.TYPE = B.TYPE;
```

Query 7.9

But because an RVA is also a relation, we are essentially saying that the SQL precept shown in Query 7.10 is the same as the contrived precept shown in Query 7.11:

```
ON <TABLE A> = <TABLE B>
```

Query 7.10

```
  ON A.MY_RVA.Attr1 = B.MY_RVA.Attr1
AND A.MY_RVA.Attr2 = B.MY_RVA.Attr2
AND A.MY_RVA.Attr3 = B.MY_RVA.Attr3
                   ▪
                   ▪
```

Query 7.11

And so on until all relation attributes and all attributes in the RVA relation are specified. This precept, too, is a valid SQL condition. If the SQL were modified to accommodate Query 7.11, this precept would also be valid. In fact, specifying all of a table's attributes in the join condition is not uncommon. The result of joining two tables on an RVA as shown in Query 7.10 is shown in Table 7.15.

VID	LOC1	LOC2	Name	Manuf	Type	
1	Tampa	Miami	El Camino	Chevrolet	**Membership**	**Class**
					0.90	Car
					0.60	Truck
3	Tampa	Miami	F-150	Ford	**Membership**	**Class**
					0.70	Car
					1.00	Truck

Table 7.15

The RVA field *type* behaves exactly as one would expect any attribute to behave. It is not necessary to specify the tables associated with *Type* contained in relations, namely VEH_MIAMI and VEH_TAMPA, because the knowledge and supporting table for both is V_TYPE. And, in the end, the individual relations contained in the RVA remain encapsulated within the attribute *Type*. Here again, the *Membership* and *Class* attributes contained within *Type* are inseparable.

7.2.5 Intersection

Unlike a number of other SQL products, MySQL does not support a true INTERSECTION set operator. The functionality behind the INTERSECTION operation can, however, be obtained using the 'IN' operator as shown in the example below:

```
SELECT <fields to be selected>
  FROM <table a>
WHERE (value) IN (SELECT value
                    FROM <table b>
                   WHERE <value predicate>);
```

In an example using the test database used to this point, in intersection could be created joining tables VEH_TAMPA and VEH_MIAMI on the value MANUF = 'Ford'. The intersection, then, is the records associated with VID = 3 as this is the only record present in both tables where the MANUF value = 'Ford' and the VID values are equal in both tables.

Query:

Query 7.12

```
SELECT veh_tampa.vid,
       veh_tampa.loc,
       veh_tampa.name,
       veh_tampa.manuf,
```

```
        veh_tampa.type,
        veh_miami.vid,
        veh_miami.loc
    FROM veh_tampa as veh_tampa inner join
        veh_miami as veh_miami
      ON veh_tampa.vid = veh_miami.vid
  WHERE veh_tampa.vid IN (SELECT vid
                            FROM veh_miami
                           WHERE manuf = 'Ford');
```

In some ways this selection is much more complex. Recall that attribute VEH_TAMPA-TYPE is a relation valued attribute and so, in many ways, this adds a third table to the selection. In other ways, however, the use of the RVA table V_TYPE simplifies the query. The 'IN' portion of the WHERE predicate is incorporated into a nested inner join just as was done in the restriction described in Section 7.2.2. The changes made to the selection in order to accommodate the RVA are shown below:

Thread→Item:

```
SELECT veh_tampa.vid,
       veh_tampa.name,
       veh_tampa.manuf,
       a.weight,
       a.class,
       veh_miami.vid,
       veh_miami.loc
```

Thread→Tables:

```
    FROM veh_tampa as veh_tampa
INNER JOIN v_type as a ON veh_tampa.vid = a.vid
INNER JOIN veh_miami as veh_miami
        ON veh_tampa.vid = veh_miami.vid
       AND veh_miami.manuf = 'Ford';
```

Final Query:

```
SELECT veh_tampa.vid,
       veh_tampa.name,
       veh_tampa.manuf,
       a.weight,
```

```
        a.class,
        veh_miami.vid,
        veh_miami.loc
    FROM veh_tampa as veh_tampa
INNER JOIN v_type as a ON veh_tampa.vid = a.vid
INNER JOIN veh_miami as veh_miami
        ON veh_tampa.vid = veh_miami.vid
       AND veh_miami.manuf = 'Ford';
```

The result is shown in table 7.16:

```
+-----+-------+-------+-------+--------+-------+-----+-------+
| vid | loc   | name  | manuf | WEIGHT | CLASS | vid | loc   |
+-----+-------+-------+-------+--------+-------+-----+-------+
|   3 | Tampa | F-150 | Ford  |   0.70 | Car   |   3 | Miami |
|   3 | Tampa | F-150 | Ford  |   1.00 | Truck |   3 | Miami |
+-----+-------+-------+-------+--------+-------+-----+-------+
2 rows in set (0.16 sec)
```

Table 7.16

7.2.6 Non-RVA Tests

In nesting the RVA relation within the base relation, the relation contained within the RVA attribute is joined to the base relation. As a result, the final queries shown in the previous sections are precisely what would be required if there were no relation valued attributes. But there is one major difference and that difference is a critical one. Section 7.2.3 describes the fact that when using an RVA any attributes contained within an RVA cannot be 'projected out' of the result. Through the use of RVA's all attributes contained within the nested relation will always be represented. This is particularly important when dealing with fuzzy data as a class without a weight of membership or a weight of membership without a respective class renders the result fairly meaningless. Query 7.12 demonstrates a query against tables VEH_TAMPA and V_TYPE without specifying the RVA attribute *type*. As a result, the vehicle's class can be projected out of the result. The query from section 7.2.2 is used out of interest. The attribute *a.class* is absent from the query used in section 7.2.2:

```
select veh_tampa.vid,
```

Query 7.12

```
        veh_tampa.manuf,
        veh_tampa.name,
        a.weight
from veh_tampa as veh_tampa
 inner join v_type as a
    on veh_tampa.vid = a.vid
   and a.weight > 0.6;
```

Without the use of an RVA, this sort of representation would be quite easy to achieve. With an RVA, however, this sort of representation would be impossible to achieve as was demonstrated in Section 7.2.3.

7.3 Metrics

To be acceptable to the relational model, the result of any query must be both accurate and complete. Using the RVA within MySQL satisfies both of these requirements.

7.3.1 Accuracy

As previously stated, a fuzzy data value is a set of (*c*,*v*) pairs. The RVA data type must both maintain the atomicity of the (*c*,*v*) pair and return the values contained within each pair accurately for the associated object as maintained for all pairs in the nested relation.

Let Accuracy = [0,1] where 1 is total accuracy and 0 is total inaccuracy. Equation 7.1 states that the ratio of complete (*c*,*v*) pairs returned to the number of possible (*c*,*v*) pairs existing for an object is 1 when all (*c*,*v*) pairs are accurately returned and 0 when none are correctly returned.

$$\text{Accuracy} = 1 - \frac{\text{The number of complete } (c,v)}{\text{The number of } (c,v) \text{ pairs}} \qquad \text{Equation 7.1}$$

7.3.1.1 Accuracy in Systems without RVA Data Types

Current systems that do not possess the constraint provided by the RVA can return an inaccurate result from a poorly written query. As Equation 7.1 illustrates, if a query were to neglect to

specify both attributes required for the (*c*,*v*) pair in the result for the associated object the fuzzy data values for each object can be incorrect.

7.3.1.2 Accuracy in Systems with RVA Data Types

It is not possible to return an inaccurate fuzzy data value when using an RVA. Through the use of an RVA, the accuracy of the fuzzy data value is guaranteed to be 1.

7.3.2 Completeness

All *requested* (*c*,*v*) pairs associated with an object in the nesting table must be returned when requested.

Let Completeness = [0,1] where 1 is complete and 0 is incomplete. Equation 7.2 states that the ratio of all (*c*,*v*) returned to the number of possible (*c*,*v*) pairs existing for an object is 1 when all (*c*,*v*) are returned and 0 when none are returned.

$$\text{Completeness} = \frac{\text{The number of } (c,v) \text{ pairs returned}}{\text{The number of } (c,v) \text{ pairs in fuzzy data value}} \qquad \text{Equation 7.2}$$

7.3.2.1 Completeness in Systems without RVA Data Types

In systems without an RVA data type, it is possible to write a query that excludes some (*c*,*v*) pairs from a result. Using the vehicle database discussed throughout this dissertation, it is possible to write a query that returns only Cars. It is a requirement of the user to know all classes associated with an object and to maintain any embedded queries in such a way that new classes, such as 'SUV', are returned with the result of a fuzzy data query.

7.3.2.2 Completeness in Systems with RVA Data Types

In systems using an RVA data type, it is not possible to omit a (*c*,*v*) pair unintentionally. The result can be restricted to only those desired, but if not specifically restricted, completeness using the RVA data type is guaranteed to be 1.

7.4 Conclusions

The system validation demonstrated that the system as designed, implemented and used satisfied all of the established goals. The system demonstrated that the incorporation of an RVA into a relational database system provided an enhanced capability while not affecting normal functionality. The system also appropriately adhered to the relational model through a demonstrated ability to obtain the Cartesian Product and perform the Restriction, Projection, Join and Intersection operations necessary to the use of the relational database.

Chapter 8 - Non-Fuzzy Application

8.1 Benefit of an RVA in a Non-Fuzzy Application

While the purpose of this research has been to demonstrate the benefits of relation valued attributes in support of the particular needs of fuzzy data, it should be noted that RVA's provide general purpose benefit as well. To illustrate the use of an RVA in a non-fuzzy data application, Tables DEPT and D_ROOM were created (See Tables 8.1 and 8.2):

Field	Description	Type	Key	Default
CODE	Dept. Code	Char(3)	Yes	
NAME	Dept. Name	Char(15)	No	
HEAD	Dept. Head	Char(15)	No	
ADMIN	Dept. Admin	Char(15)	No	
CONTACT	Dept. Contact	Char(8)	No	
ROOM	Room RVA	RVAChar(120)	No	@table:D_ROOM@key:CODE@fields: NUMBER,TYPE,AVAIL,NETWORK;

Table 8.1 - DEPT

Field	Description	Type	Key	Default
CODE	Dept. Code	Char(3)	Yes	
NUMBER	Room No.	Char(4)	Yes	
TYPE	Room Type	Enum('Meeting','Patient','Office','Labratory')	No	
AVAIL	Available?	Enum('Y','N')	No	'Y'
NETWORK	Networked?	Enum('Y','N')	No	'Y'

Table 8.2 - D_ROOM

Note that DEPT attribute *room* is specified as an RVAChar with the requisite knowledge provided as the default value. The data provided for the two tables is shown in the screen shots from the system below. The attribute *room* was left off of the query against table DEPT so that only the base data within table DEPT was displayed. Recall that any time an RVA field is selected, the nested data contained within the RVA's attribute will be displayed as a part of the result.

```
mysql-VCU> select code, name, head, admin, contact from dept;
+------+-----------------+---------+----------+----------+
| code | name            | head    | admin    | contact  |
+------+-----------------+---------+----------+----------+
| 001  | Oncology        | Jamison | Albright | 452-3354 |
| 002  | Obstetrics      | Steward | Simms    | 452-3376 |
| 003  | General Practic | Cho     | Johnson  | 452-3389 |
+------+-----------------+---------+----------+----------+
3 rows in set (0.00 sec)
```

Table 8.3 - DEPT Data

```
mysql-VCU> select * from d_room;
+------+--------+---------+-------+---------+
| CODE | NUMBER | TYPE    | AVAIL | NETWORK |
+------+--------+---------+-------+---------+
| 001  | 2101   | Patient | N     | Y       |
| 001  | 2102   | Patient | Y     | Y       |
| 001  | 3405   | Office  | Y     | Y       |
| 001  | 3406   | Office  | Y     | Y       |
| 002  | 2103   | Patient | Y     | Y       |
| 002  | 2104   | Patient | Y     | Y       |
| 002  | 4001   | Office  | N     | Y       |
| 003  | 2105   | Patient | N     | Y       |
| 003  | 2106   | Patient | N     | Y       |
| 003  | 3408   | Office  | N     | Y       |
+------+--------+---------+-------+---------+
10 rows in set (0.00 sec)
```

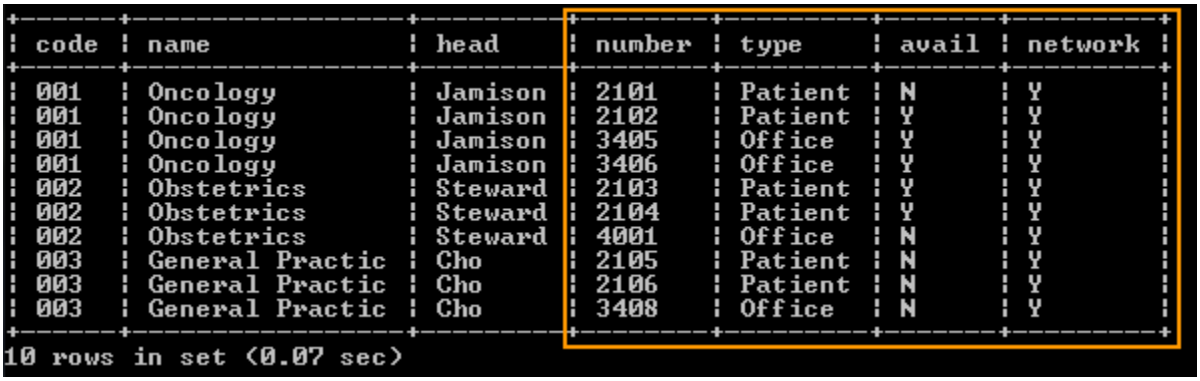
Table 8.4 - D_ROOM Data

When the table DEPT is queried with the RVA attribute included in the selection clause, the following example illustrates the result that would be returned. (NOTE: Due to the output and formatting limitations of the console, attributes *admin* and *contact* were excluded from this query.) As can be seen, the RVA attribute *room* returns the room data associated with each of the Departments as a relation containing attributes *number*, *type*, *avail* and *network*. As a result, this simple query returns a much more significant response, a response limited only by the knowledge value specified in the RVA. But, because the attribute *room* is an RVA, the cardinality of this result is still only 4 with an index of only 3 tuples.

The Query: **Query 8.1**

```
SELECT CODE, NAME, HEAD, ROOM
  FROM DEPT.
```

The Result is shown in Table 8.5:



code	name	head	number	type	avail	network
001	Oncology	Jamison	2101	Patient	N	Y
001	Oncology	Jamison	2102	Patient	Y	Y
001	Oncology	Jamison	3405	Office	Y	Y
001	Oncology	Jamison	3406	Office	Y	Y
002	Obstetrics	Steward	2103	Patient	Y	Y
002	Obstetrics	Steward	2104	Patient	Y	Y
002	Obstetrics	Steward	4001	Office	N	Y
003	General Practic	Cho	2105	Patient	N	Y
003	General Practic	Cho	2106	Patient	N	Y
003	General Practic	Cho	3408	Office	N	Y

10 rows in set (0.07 sec)

Table 8.5 - The Result

Chapter 9 - Representation of Fuzzy Data

9.1 Types of RVA Representation

There has been much debate as to the representation of fuzzy data in a result set. Below are the two representations under consideration using the data contained in sample table VEH_TAMPA. The query used to obtain this result is a simple one:

```
SELECT vid, loc, name, manuf, type                    Query 9.1
  FROM veh_tampa;
```

The result of this query could be provided to the user in one of two formats, either a standard grid or a grouped grid format as shown in Tables 9.1 and 9.2:

VID	LOC	NAME	MANUF	WEIGHT	CLASS
1	Tampa	El Camino	Chevrolet	0.90	Car
1	Tampa	El Camino	Chevrolet	0.60	Truck
2	Tampa	Camero	Chevrolet	1.00	Car
3	Tampa	F-150	Ford	0.70	Car
3	Tampa	F-150	Ford	1.00	Truck

Table 9.1 – Standard Grid Representation

VID	LOC	NAME	MANUF	TYPE	
1	Tampa	El Camino	Chevrolet	**WEIGHT**	**CLASS**
				0.90	Car
				0.60	Truck
2	Tampa	Camero	Chevrolet	**WEIGHT**	**CLASS**
				1.00	Car
3	Tampa	F-150	Ford	**WEIGHT**	**CLASS**
				0.70	Car
				1.00	Truck

Table 9.2 – Grouped Grid Representation

Table 9.1 is the format currently returned by the implementation in MySQL. As the MySQL server accumulates the data in response to a query, it stores the data in a buffer packet. At the conclusion of the query, the buffer is flushed and the result is sent to the client in its raw, structured format. As can be seen in this representation, each line of the RVA is associated with the non-RVA attributes which consist of the base table. The representation displayed in Table 9.2

shows the RVA as it is associated with a single ‘tuple’ of the base table attributes. There are clear advantages and disadvantages to both representations. In fact, the choice of representations may become merely a design issue based on user need. As a result, the following sections do not resolve this debate, but merely seek to discuss the various costs and benefits of each representation.

9.1.1 Standard Grid Representation

The primary benefit of the standard grid representation is the fact that is represents the data in a format that the user is accustomed to seeing. While the relation is obviously in First Normal Form only, each of the WEIGHT and CLASS tuples contained within the RVA are seen as associated with its base relation.

Another, not inconsiderable, benefit depends on how the user interacts with the MySQL server. If the user is accessing data through the console mode in a query → result fashion, the transaction is completed when the result from the query is returned to the user. If the user is accessing the server using the MySQL workbench, however, the interaction between client and server is more interactive. The result returned is a temporary and active relation consisting of the data resulting from a specific query with open links to both the tables and attributes involved. As a result, a change to the temporary relation is communicated to the involved tables and their attributes as a modification. In this way, the data comprising such representations of the RVA and its base relation could be maintained as a single relation rather than individually as would otherwise be the case.

9.1.2 Grouped Grid Representation

The grouped grid representation is one in which the RVA tuples are contained with a single tuple of the base relation. In such a case, there is a single instance of the base relation tuple and any number of tuples contained in the representation of the RVA nested relation. Date argues that such a representation is in 3rd Normal Form [37] given the structure of the representation consisting of a single normalized base table tuple and any number of tuples normalized within the RVA. The primary benefit here is the clear visual association of the RVA as a relation with a tuple with both an attribute heading (*type*) and attribute names displayed for all attributes within the relation. In using the standard grid representation, this visual association is potentially diminished depending on the perspective and acuity of the user.

It would be a fairly straight forward exercise to modify the database system to return this representation.

Chapter 10 - Formal System Study

10.1 Overview

A formal system study plan was developed and submitted to the University's Institutional Review Board (IRB) for exemption. The exemption was approved on 28 November 2012. This chapter details the goals of the study and the steps taken to achieve these goals.

The purpose of this study was to gain input from students in Computer Science, Information Systems and Bioinformatics. This input provided their insight into three major areas of interest in the use of RVA's in the support of fuzzy data. The first of these areas was the acceptance and understanding of fuzzy data by the user community. How intuitive was the concept of fuzzy data and did they feel it had relevance in the 'real world'. Secondly, the study sought to gain insight into two different representations of fuzzy data using RVA's. One representation was the standard grid. The other was a grouped grid. Thirdly, the study provided the users with an opportunity to try to break the association between an entity's class and its weight of membership. Perhaps the most important purpose of this research was to devise and demonstrate a means by which a constraint could be applied to the fuzzy data value representation that would preclude the separation of class from its associated weight of membership.

10.2 Preparation

An application for an exemption from the IRB was required to proceed with the study using students from the University as participants. Once the exemption was obtained, participants from the University's student body were recruited and arrangements made for University resources such as facilities, computers and presentation equipment. At this point, a presentation outlining

the subject matter of the research, research objectives and a script reflecting these objectives were developed.

10.2.1 Application to the Institutional Review Board

In October 2012, an application was made to IRB. The study would neither collect nor maintain any personally identifying information on the participants. Nor would the study be in any way invasive. In light of these points, the application to perform the study (Appendix E) was submitted and subsequently approved for exemption by the IRB on 28 November 2012.

10.2.2 Recruitment

The target demographic of participants for this study consisted of both graduate and undergraduate students from Computer Science, Information Systems, Business and Bioinformatics. It was believed that this demographic would possess the necessary level of education and familiarity with information systems and the requisite 'mind set' to provide meaningful feedback and insight into the system and research topic under review. In order to recruit and attract this demographic, a number of brief presentations were made in selected classes. Information sheets reiterating the study's purpose and the contact information necessary to reserve a place in the study were left with the students and posted in common areas.

10.2.3 Preparation

A study script was prepared that reflected the objectives of the operations to be performed. These objectives included information on functionality and provided a level of heuristic perspective. Using these study objectives as a guide, the next step was to prepare a lesson plan that provided a basic, minimum level of SQL knowledge to complete the steps contained within the study. Thirdly, an overview presentation was prepared to introduce the participants to the concepts

pertaining to fuzzy data and RVA's. Lastly, flash drives containing the working systems and the database to be used for the study were prepared for each user.

10.2.3.1 Study Objectives

The study being performed had the following primary objectives.

- Did the system function as designed and developed?
- Did the use of an RVA data type obey the dictates of the relational model?
- Was the representation of fuzzy data easily understandable?
- Was it possible to separate an entity's class and weight of measurement using an RVA?
- Which representation of the RVA was preferred and why?
- Was the use of an RVA type attribute in the SQL consistent with the use of other data types?
- What was the user's impression of using an RVA data type for purposes other than fuzzy data?

In support of these objectives, a test script was designed using a mix of exercises, open ended questions and closed questions where the participant was asked to rate a particular aspect on a scale of 1 to 5. The test script was loaded onto a flash drive and when complete, the script could be saved and sent by the participant using university email to the research team for review and analysis. Once received by the research team, the scripts would be downloaded, given generic names in numbered sequence and any identifiable association with a participant would be destroyed.

10.2.4 Heuristic Tests

While system testing is an important component in the analysis and validation of how relation valued attributes are used in the support of fuzzy data, heuristic testing can also provide valuable insight into user and community acceptance of this approach. In his paper describing the ten

most important heuristic system tests, Jakob Nielsen lists his top ten usability heuristics.[40] While not all of these heuristics apply to the use of RVA's, a number are applicable and useful in the evaluation of the finished system and user acceptance of the system. The following section lists the heuristics deemed pertinent to the study of the system and Nielsen's description of their use.

Visibility of system status

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Does the result coincide with user expectations?

- Is the feedback provided by the result, i.e. number of records and layout, intuitive and meaningful?

Match between system and the real world

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

- Does the user see a real world application for fuzzy data?
- Does the user see the benefit of using an RVA in support of fuzzy data?

Consistency and standards

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

- Is the request (query) of fuzzy data consistent and logical?

Error prevention

Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

- Are the error messages appropriate?
- Do the error messages make sense?
- Is the system reliable?

Aesthetic and minimalist design

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

- Are query constructs sufficient?
- Are query constructs excessive?

10.2.5 Training

While the participant population was well educated and familiar with a wide range of information systems, it was imperative that there be a degree of surety in at least a minimum level of competence using the database query language SQL. To ensure this competence, a brief, interactive SQL practicum would be held to impart the necessary level of knowledge and understanding to effectively participate in and complete the study.

10.2.6 Overview of Fuzzy Data and the Role of the RVA

In order to carry out a system study on the use of an RVA data type in support of fuzzy data, users needed to be provided with a certain level of knowledge and understanding of the what fuzzy data was, how it might be represented and the problems inherent in that presentation. It was important, however, that participants not be made 'expert' in either fuzzy data or relation valued attributes. The approach used for this study was to obtain the perspective of the participant and a fresh insight rather than have the participant merely parrot back the perspective of the presenter. In support of this approach, participants would be given a brief overview of fuzzy data, RVA's and the objectives behind the research. It was anticipated that as they progressed through their evaluation, the participant's grasp of both RVA's and fuzzy data would evolve.

10.2.7 Working Systems

Prior to the commencement of the study session, flash drives containing the modified MySQL database client/server system with an accompanying test database complete with tables containing RVA type data attributes were put on each computer in the laboratory utilized for the study. During the interactive SQL training session participants would be able to interact with MySQL as appropriate to reinforce the training concepts and eventually carry out the study steps.

During the training session, and presentations, proctors would be available to answer questions or address any technical issues that might arise. At the conclusion of the study, participants were allowed to take the flash drive with them in appreciation for their participation.

10.3 Study Results

Of the 55 participants, 31 completed or partially completed study scripts were returned. While this number is not statistically viable, it was sufficient to meet the study objectives by providing antidotal observation and insight.

10.3.1 System Functionality

The computers used for the study had significant security and 'terminal stay resident' (TSR) software running to ensure system protection and user validation. As a result, while the system functioned well and was sufficient for testing, it was not as stable as the system used for development which had greater memory, an updated operating system and fewer constraints on the operations. It was determined that reliability increased when the result returned from a query was limited through the use of a restriction in the 'WHERE' predicate. As a result, some participants were allowed to restrict their result which, while returning fewer records, accomplished the goal sufficient to the requirements of the study step.

Over all, however, the system functioned satisfactorily and returned the expected result for each step in the study.

10.3.2 Adherence to the Relational Model

The study further validated that the system as designed and developed adheres to the mathematical dictates of the relational model. To this end, the study included queries that required the participants to carry out a project, restrict and intersect operations on two related

tables using standard SQL. The results obtained when the RVA typed attribute was specified was presented accurately and appropriately in the result as the relation associated with each entity.

A separate step in the study challenged participants to try to write a query that would intentionally exclude either the class or the weight of membership from the RVA attribute's result. None of the participants were successful in accomplishing this objective despite quite a number of very serious attempts.

10.3.3 User Acceptance

The ability of the user to understand and grow accustomed to the use of an RVA had a number of factors of interest.

10.3.3.1 Displayed View

How the RVA containing the fuzzy data was represented in the result was a major factor for the users. While discussed in greater detail in Chapter 9, users seemed to prefer to see both the attribute name *Hair* as well as the RVA relation's attributes *Weight* and *Class* displayed in the heading of the result.

10.3.4 Enforced Constraint on Ordered Pair

The primary focus of this research was to devise a way in which a fuzzy data value consisting of the ordered pair class and weight of membership could be constrained in such a way as to be indivisible through selection. In keeping with this objective, the study challenged the participants to devise a way, using the SQL, to separate either class or weight of membership from the other member. This challenge was accepted to such a degree by the participants that, unfortunately, more than a few concentrated so heavily on breaking this pair that there was not time enough to finish the rest of the study questions. Below are some of the methods attempted

or comments describing how the pair might be separated? None, however, were successful in writing a query that overcame this challenge as it is not possible, using the RVA, to separate or exclude class or weight of membership from the result. Nor is it possible to order the relation returned in a way contrary to that specified in the RVA's knowledge.

- "The weight and class are meaningless if we exclude one to another." Which is both correct and concise.
- "I tried numerous ways to separate them, including selecting all attributes except weight (but of course that is part of hair). So, stumped." Note that this was only the second question in the study and already the realization into the critical importance of *both* attributes was taking hold.
- "Select ekey, last, first, hair from emp where hair != weight; This is just my suggestion. I really have no idea." A number of similar SQL approaches were attempted, but because neither *Class* nor *Weight* were attributes of table EMP, neither could be accessed nor excluded.
- "**mysql-VCU>**Select ekey, first, mi, last, dob, hair.class from EMP;

  To be more specific, suppose hair consists of weight, color, and length, the query excluding only weight would be written as

  mysql-VCU>Select ekey, first, mi, last, dob, hair.weight, hair.length from EMP;"

  This participant provided two example SQL statements. Both were actually tried and failed since class is an attribute containing a table, but is not itself a table. Still, it was impressive that this user was already considering an attribute with a compound structure and had even given the compound member a syntax.

10.3.5 RVA Representation

During the early stages of research there was some debate as to which representation was more appropriate for the use of an RVA. In the end, the decision as to how to display an RVA in a result was a design decision, one of preference. Once the result was obtained by the server, the representation of the result could be shown in, essentially, one of two different ways. In development, the representation shown in Table 4.16 was chosen. In order to gain insight and perspective from this user community, however, participants were given examples of the two

different representations of the RVA in an example result and asked which was more understandable and which was more accurate. The two representations shown below contain the exact same data, but in different formats.

EKEY	FIRST	MI	LAST	AGE	CLASS	WEIGHT
16	Charles	W	Backman	88	Grey	0.50
16	Charles	W	Backman	88	Black	0.60
17	Bill	H	Smith	52	Brown	0.75
17	Bill	H	Smith	52	Grey	0.25
17	Bill	H	Smith	52	Blonde	0.15

Table 10.1 – Standard Grid Representation

EKEY	FIRST	MI	LAST	AGE	HAIR	
16	Charles	W	Backman	88	CLASS	WEIGHT
					Grey	0.50
					Black	0.60
17	Bill	H	Smith	52	CLASS	WEIGHT
					Brown	0.75
					Grey	0.25
					Blonde	0.15

Table 10.2 – Grouped Grid Representation

It should be understood that neither representation is right or wrong. The objective here was simply to ascertain which representation made more sense to the users and why.

10.3.5.1 Understandability

By a fair margin, participants offered that the representation shown in Table 10.2 was the most understandable. The comments suggest that there were two reasons for this opinion. The first is that unlike the Standard representation, the Grouped representation actually shows the *Hair* attribute in the heading. Table 4.16 does not and so the user is left wondering why they selected Hair and got *Class* and *Weight*. Of course, *Class* and *Weight are Hair* and in time, with practice and experience, this new perspective would be accepted. But the representation in Table 10.2 requires no practice or experience. Hair is in the result and it is shown to consist of *Class* and *Weight* which, to most, made the most sense. The second reason was that the multiple lines of employee information in Table 10.1 seemed redundant to a number of participants. Table 10.2 did not contain this 'repetition' and so had a 'cleaner' appearance. To those who believed that Table 10.1 was the more understandable representation, however, it was this very repetition that seemed to appeal to them. To them, the repetition made it clear which *Class* and *Weight* went with which employee. These participants, now possessing some experience and a growing familiarity with an RVA and had no problem picking out the attributes which comprise the RVA *Hair*. Some of the relevant comments from both points of view are provided below.

- "<Grouped> is more understandable because without the sub label of hair, we have no idea what the .50 grey even means. The hair is 50% grey? We can't tell without using <Grouped>."
- "I think <Grouped> is more understandable because it doesn't repeat a lot of the same information from the two people."
- "<Grouped> is more understandable but I can see how <Standard> might be more understandable to someone who is more familiar with the computer programming. <Grouped> looks neater and the information is easier to pick out at first glance."
- "<Standard> because even though the names are repeated, it is easier to go line by line and read the data."

10.3.5.2 Accuracy

The information in Tables 10.1 and 10.2 are exactly the same. Neither is technically 'more accurate' than the other and yet because of impressions expressed in Section 10.3.5.1, there seemed to be a 'prejudice' toward a greater accuracy in Table 10.2. While the comments were not as verbose in this question, some of the comments are telling. Also telling is the concept of credibility. If a particular display gives even some of these more informed users more confidence in the accuracy of the data represented, this confidence is significant.

10.3.6 Consistency of Use

An RVA is an attribute with a more complex data structure, specifically a relation. So, when the RVA is included in a query, its use and operation is no different than any other attribute. When it is used to restrict the result as it is in the WHERE predicate of a query, the use of an RVA is not consistent. This inconsistency is the result of the fact that while primarily an attribute, it is also a relation. The characteristic that gives the RVA its strength in being used to encapsulate a fuzzy data value requires a somewhat complicated approach to its use in restricting the same fuzzy data value to a specific value. When asked to select employees whose hair was 'black', participants were required to write a query similar to that shown below:

Query 10.1

```
SELECT ekey, first, last, hair
  FROM emp
 WHERE ekey IN (SELECT ekey
                  FROM ehair
                 WHERE class = 'black')
```

Because the attribute *Class* is not accessible through the attribute *Hair*, the user is required to obtain the employee key *Ekey* from a nested select of the table EHAIR. In this way, the results returned by the query are restricted to only those employees whose hair is black. Note that this allows the selection of *Hair* to be restricted on an attribute of EHAIR and yet remain inaccessible

in the select clause. A number of participants made the valid point that it would be much easier to simply write the query to say:

```
SELECT ekey, first, last, hair                    Query 10.2
  FROM emp
 WHERE class = 'black'
```

This would be easier for the user. But *Class* is not an attribute of table EMP. *Class* is an attribute in the relation nested within the RVA *Hair*. But *Ekey* is an attribute of table EMP. It is also an attribute of table EHAIR and so *Class* can only be used to select the value of the common attribute Ekey while restricting the result in the query to only those employee's who have some membership in the *Class* = 'black'.

10.3.7 Use of RVA's Outside of Fuzzy Data

During the initial stages of research, the suggestion was made to expand the use of the RVA beyond its use solely for the representation of fuzzy data. Nothing had to change during the development of the RVA data type to accommodate this suggestion. When an RVA data type is included in the definition of a table, three pieces of knowledge are assigned as a default value to the attribute:

- Table: The name of the table that contains the data to be nested within the RVA.
- Key: The key of this table that is common to the table containing the RVA.
- Fields: Any number of fields within the nested table and in the order of their appearance.

In the case of fuzzy data, the knowledge assigned to the value of an RVA might appear as:

@Table:EHAIR@Key:EKEY@Fields:CLASS,WEIGHT;

This knowledge tells the server that the RVA will consist of data from table EHAIR which has the common key EKEY and will consist of the EHAIR attributes *Class* and *Weight*. As there is no restriction on the table, key or fields that can be specified in the RVA's knowledge string, there is no restriction on the size or scope of the data returned by the use of an RVA in a table

selection. To illustrate and confirm this functionality, two tables were included in the test database, DEPT and D_ROOM. Examples of these tables, their structure and the test data included appears below.

CODE	NAME	HEAD	ADMIN	ROOM
001	Oncology	Jamison	Albright	RVA: D_ROOM
002	Obstetrics	Steward	Simms	
003	General Practice	Cho	Johnson	

Table 10.3 - DEPT

CODE	NUMBER	TYPE	AVAIL	NETWORK
001	2101	Patient	N	Y
001	2102	Patient	Y	N
001	3405	Office	Y	Y
001	3406	Office	Y	Y
002	2103	Patient	Y	Y
002	2104	Office	N	N
002	4001	Patient	N	N
003	2105	Patient	Y	Y
003	2106	Office	Y	N
003	3408	Office	N	Y

Table 10.4 - D_ROOM

Unlike the RVA data type configured to support fuzzy data, the only change required of the RVA data type ROOM to configure it to support the requirements of the DEPT table was to include the desired fields from table D_ROOM in the RVA's knowledge as follows:

@Table:d_room@Key:code@Fields:number,type,avail,network

In this step, participants were asked what they believed would be the result of the following query against table DEPT that included the attribute *Room*.

```
SELECT code, name, head, room
  FROM dept
 WHERE code = '001';
```

Query 10.3

In this case, *Room* is an RVA which contains the knowledge necessary for all but the key field *Code* in table D_ROOM to be nested within it. The result is shown in the Table 10.5.

code	name	head	NUMBER	TYPE	AVAIL	NETWORK
001	Oncology	Jamison	2101	Patient	N	Y
001	Oncology	Jamison	2102	Patient	Y	Y
001	Oncology	Jamison	3405	Office	Y	Y
001	Oncology	Jamison	3406	Office	Y	Y

Table 10.5 - Query Result

By this time, participants were nearly through with the study questions. For as long as 30 to 40 minutes, participants had applied themselves to the task. They had been introduced to RVA's and had written queries to obtain RVA's consisting of fuzzy data. And they generally appeared to be flagging which may explain the inconsistency in the responses to the study questions. Almost uniformly, participants responded that a query against table DEPT, despite seeing an RVA attribute in the table structure and a table name associated with that attribute, responded that the query would return the data contained in table DEPT. Not a single mention was made of the data contained in table D_ROOM. And yet when asked if the result of this query met with their expectations, the response was almost uniformly high that the data contained in Table D_ROOM was as expected. "Matched perfectly with my expectations." Some responses, however, displayed some understandable confusion as the scores were either very high or very low with few in the middle. This was not fuzzy data. The impression had certainly been given that RVA's would be used for fuzzy data. There was no expectation on the part of the user that RVA's should or would be used for anything else. And yet the fact that so many accepted the presence of data from table D_ROOM in the result when selecting the attribute *Room* clearly indicates a growing understanding and acceptance of RVA's and their use in the support not only of fuzzy data, but of any data where encapsulation is an important factor.

Chapter 11 - Conclusion

11.1 Restating the Problem

The complex nature of a fuzzy data value requires a data structure and representation that supports a set of ordered pairs of values (c,w) [2]. The values contained in this pair represents the data's *class, c,* consisting of any number of enumerated characteristic values within a domain C = *{Universe of possible classes}*. As such, $c \in C$. The other value, w, is in the interval $W = [0,1]$ which is the fuzzy data value's *weight* or *degree of membership* and indicates the membership associated with c.

There are two significant challenges to the representation (c,w). The first is that one element without the other element of the pair is meaningless. The second challenge is that a fuzzy data value consists of a set of such pairs. To represent a fuzzy data value in a database it is necessary to represent the set of (c,w) in such a way that neither c nor w can be retrieved without the other and that a system supporting such data provide the means to support a set of such (c,w) pairs for an associated object.

Current database management systems do not have the ability to enforce a constraint which would ensure the requisite atomicity of the ordered pair associated with the fuzzy data value while at the same time accommodate the variable number of such pairs potentially contained within the value.

The goals of this research were two fold. The first goal was to design and engineer an RVA on the server of the open source DBMS MySQL that could provide the complex data structure

necessary for the representation and retrieval of fuzzy data. This design must allow fuzzy data to coexist in every respect with crisp data using the standard SQL language. The new data type needed to be flexible and easily configured using the standard SQL language and present the fuzzy data value to the user in an understandable and meaningful way. But most of all, the new data type must adhere to the tenets established for the relational model.

The second goal was to assess the understanding, usability and acceptance of fuzzy data on the part of users not involved with the research. Technology, after all, can be well designed and implemented, but if not well understood and useful to the community has little value.

11.2 The Use of 'Knowledge'

By using the 'knowledge' approach, the basic functionality of the server was not changed, but it was enhanced. The 'knowledge' approach provided a constraint on the data contained in the result by default. No new constraints needed to be created to accomplish this critical aspect of the design. Further, the data contained in the RVA was accessible outside the RVA. Its structure, the data contained within it, all could be changed outside the RVA or the nesting table without affecting base relation in any way. And lastly, the knowledge can be simply changed to specify new tables, attributes and order merely by configuring the default value of the RVA's knowledge.

11.3 The *build_query()* Method

It was decided that rather than reinvent the parser, the *build_query()* should access the thread structure and obtain the values that the new query would need and construct a new query which would then be submitted back into the parser, essentially letting MySQL do its job.

This approach proved quite useful in that the thread structure was well designed with any number of accessor methods available to extract the data and build the new query string.

11.4 Not Just For Fuzzy Data

At the onset of this research, the focus was on fuzzy data and the use of RVA's to support only fuzzy data values. Early on, the question was asked, "Can the RVA handle only fuzzy data or can it be used for any data requirement that would benefit from the unique characteristics of an RVA?" As the design phase became more detailed it became apparent that the design, without any change, should be able to handle a relation of any size or cardinality merely by specifying the table and the attributes to be included in the configuration of the 'knowledge'. This understanding turned out to be the case as shown by the examples in Chapter 8.

11.5 Conclusions

A relation valued attribute has, for the first time, been designed, engineered and integrated into a relational database management system. The design of this new attribute as been successfully implemented into the server of MySQL and has been used to support the complex data structure required of fuzzy data. Specifically, it maintains and guarantees the atomicity of the fuzzy pair (c,w) as attributes and provides the capacity to support any number of such pairs as tuples within the attribute's relation to represent the fuzzy data value.

Further, the design has been validated in that it satisfies the requirements of the relational model. Through examples using standard SQL with both crisp and fuzzy data, the design provides the user with the ability to obtain the Cartesian Product as well as to appropriately perform the project, restrict, intersect and join operations necessary to the use of a relational database system.

A database system now enhanced through the implementation of the RVA data type in support of fuzzy data is now available to provide a laboratory for further enhancement and research in the advancement of both fuzzy data and the use of RVA's in a relational database.

Finally, the study has shown that after a brief learning curve, users found the concept, use and results provided by the new RVA type to be intuitive and easy to understand.

Chapter 12 - Future Research

12.1 Introduction

One of the goals of this research was that a relational database system would be in place that would allow further research and study into the application, benefits and limitations of fuzzy data as supported by an RVA. This goal was achieved. The system now available provides the opportunity to do further research into this important aspect of Computer Science. The following sections suggest some research areas that show the most need and promise.

1.1 Use of Fuzzy Data Equality in SQL Operations

The value contained within an integer or character data type is easy to equate. If x = 1 and y = 1, queries can be written to query a result where x = y. Fuzzy data, however, and in particular fuzzy data encapsulated within an RVA presents a challenge. This challenge is comparable to the test of determining the equality between Table X to Table Y. But the challenge is compounded by the fact that the two tables being tested may not be the entire population of Table X and Table Y, but only the subset associated with a key in the nesting table. The data provided in Tables 12.1 and 12.2 provide the base data contained in a modified version of Tables VEH_TAMPA and VEH_MIAMI. Tables 12.3 and 12.4 provide the data with the fuzzy data nested within the base table for reference. Table 12.5 provides the source containing the fuzzy data contained in Table V_TYPE.

VID	LOC	NAME	MANUF	TYPE
1	Tampa	El Camino	Chevrolet	RVA Data Type Nested Table: V_TYPE
1	Tampa	El Camino	Chevrolet	
2	Tampa	Camero	Chevrolet	
3	Tampa	F-150	Ford	
3	Tampa	F-150	Ford	

Table 12.1 - VEH_TAMPA

VID	LOC	NAME	MANUF	TYPE
3	Tampa	F-150	Ford	RVA Data Type Nested Table: V_TYPE
3	Tampa	F-150	Ford	
4	Miami	Silverado	Chevrolet	
4	Miami	Silverado	Chevrolet	
5	Miami	Beetle	Volkswagen	

Table 12.2 - VEH_MIAMI

VID	LOC	NAME	MANUF	WEIGHT	CLASS
1	Tampa	El Camino	Chevrolet	0.90	Car
1	Tampa	El Camino	Chevrolet	0.60	Truck
2	Tampa	Camero	Chevrolet	1.00	Car
3	Tampa	F-150	Ford	0.70	Car
3	Tampa	F-150	Ford	1.00	Truck

Table 12.3

VID	LOC	NAME	MANUF	WEIGHT	CLASS
3	Tampa	F-150	Ford	0.70	Car
3	Tampa	F-150	Ford	1.00	Truck
4	Miami	Silverado	Chevrolet	0.70	Car
4	Miami	Silverado	Chevrolet	1.00	Truck
5	Miami	Beetle	Volkswagen	1.00	Car

Table 12.4

VID	CLASS	WEIGHT
1	Car	0.90
1	Truck	0.60
2	Car	1.00
3	Car	0.70
3	Truck	1.00
4	Car	0.70
4	Truck	1.00
5	Car	1.00

Table 12.5 - V_TYPE

Given the data contained in VEH_TAMPA and VEH_MIAMI, there is functionality currently does not exist within the database system that could compare the two tables for equality of each table's attribute *type*. Query 12.1 provides an example:

Query 12.1

```
SELECT VEH_MIAMI.vid,
       VEH_MIAMI.name,
       VEH_MIAMI.manuf,
       VEH_MIAMI.type
  FROM VEH_TAMPA as VEH_TAMPA,
```

```
        VEH_MIAMI as VEH_MIAMI
WHERE VEH_TAMPA.TYPE = VEH_MIAMI.TYPE;
```

This query would return all records in table VEH_MIAMI where the RVA relation's value in attribute VEH_MAIMI.type is equal to the RVA relation's value in VEH_TAMPA.type. The result is shown in Table 12.6.

VID	NAME	MANUF	WEIGHT	CLASS
3	F-150	Ford	0.70	Car
3	F-150	Ford	1.00	Truck
4	Silverado	Chevrolet	0.70	Car
4	Silverado	Chevrolet	1.00	Truck

Table 12.6

In order to achieve this result, all records in the relation nested in the VEH_TAMPA attribute *type* would need to be compared to all records in the relation nested in the VEH_MIAMI attribute *type* for each value in attribute *vid* as each relation contained in RVA attribute *type* is dependent on the base table's primary key *vid*. By extension, this means that the nested relation is dependent on the nesting table's primary key as well. But the query is not comparing the relations in attribute type, it is comparing the value in attribute *type* to the value in the other attribute *type*. That value is a relation.

1.2 Representation of a Relation Valued Attribute

There has been a great deal of discussion on the topic of how the data presented by an RVA should be represented. Tables 12.7, the standard grid representation, and 12.8, the grouped grid representation, provide examples of the two most discussed representations. The first, Table 12.7, is the representation used in the current system. During the system study, however, users felt that the representation shown in Table 12.8 was the more understandable and, by extension, accurate. Further research can modify the MySQL database management system to return a

result in the grouped grid representation. This change could then be used to further study impact on user perceptions and expectations.

EKEY	FIRST	MI	LAST	AGE	CLASS	WEIGHT
16	Charles	W	Backman	88	Grey	0.50
16	Charles	W	Backman	88	Black	0.60
17	Bill	H	Smith	52	Brown	0.75
17	Bill	H	Smith	52	Grey	0.25
17	Bill	H	Smith	52	Blonde	0.15

Table 12.7 - Standard Grid Representation

EKEY	FIRST	MI	LAST	AGE	HAIR	
16	Charles	W	Backman	88	CLASS	WEIGHT
					Grey	0.50
					Black	0.60
17	Bill	H	Smith	52	CLASS	WEIGHT
					Brown	0.75
					Grey	0.25
					Blonde	0.15

Table 12.8 - Grouped Grid Representation

1.3 Use of a Derived Fuzzy Value in an RVA

The focus of this dissertation was the implementation of an RVA data type to represent a fuzzy data value maintained in a table as a non-derived value. That is to say a generally static value maintained and stored in a permanent table created and accessed by the server. Derived data, however, is not stored in tables but is calculated in the select clause of a query. The example in

Query 12.2 illustrates such a derived fuzzy data value using the Bottle database discussed in Chapter 2.

```
SELECT bid, contents,                                    Query 12.2
     ((quantity/size) as MEMBERSHIP,
     'Full' as CLASS) as STATE ← 'As' an RVA labeled state.
  FROM BOTTLES
 WHERE bid = 'A'
   AND state = 'Full';
```

In this case, the RVA is *state* which consists of two attributes, *membership* and *class* which will contain the fuzzy data value {quantity/size, FULL}. The challenge here is that the RVA is created in the SELECT clause of the query. As such, and in theory, such a clause could mistakenly exclude either *class* or *membership*. It is this specific scenario which the new data type rvachar has been designed to prevent. It is believed that this challenge could be overcome by modifying the aggregating functionality possibly through the use of a key word such as "as ***rva*** STATE" that could perform a validation derived attributes and their values.

1.4 Incorporation of Ambiguity in Database Systems

'Incorporation of ambiguity' was not specifically considered in the goals of the study. There were no specific questions or steps to measure or give insight into this concept. During the system study on fuzzy data heads would 'nod in agreement'. As students answered the study questions or performed the exercises, questions were sometimes asked not about the study, but about fuzzy data. Is it used? Where is it used? Why isn't it used more? Little of this was captured on the studies returned, but if the study were revised and repeated, more concentration could be given to capture this perspective.

1.5 Sensitivity Analysis

Fuzzy data, by its very nature, reflects the many aspects of ambiguity that exists in the real world. As Ma points out, this ambiguity can take on and reflect many forms of fuzziness such as

membership within a class as has been primarily discussed within this dissertation but also other forms such as certainty and importance. (See Section 4.2.1.3.3)[13] Further, an object can possess many different fuzzy classifications. For example, a 'suspect' may have a fuzzy data value reflecting the ambiguity of hair color and another values reflecting eye color, skin tone and height. Given the many fuzzy attributes that can be represented by a fuzzy data value, the subjectivity of that value and the many perspectives that that value can represent, changes to one or more of these values can affect the ordinality of the result.

Applications using fuzzy data values of this sort will benefit from the ability to incorporate a degree of sensitivity modeling and analysis, particularly in the area of multi-criteria decision making. An application with the ability to easily modify a fuzzy data value and refresh the result to simulate an alternative result would add significant value to the usability of the underlying fuzzy data.

1.6 Beneficial Enhancements to the Man-Machine Interface

One of the criticisms in the use of fuzzy data has been the amount of data required to maintain and use fuzzy data. For example, within a crisp database, hair color might be assigned a value of 'blonde'. Entering a single such value requires a minimal and acceptable level of effort. A fuzzy data value, however, might have a value similar to the value shown in Equation 12.1.

Fuzzy Data Value = {('blonde', 0.00), Equation 12.1
('brown', 0.80),
('black', 0.00),
('red', 0.10),
('grey',0.00)};

Such a value would necessarily require far more data entry and data management. And this fuzzy data value is a very limited one! Fuzzy data within a enterprise system would contain many more such attributes and require a great deal of effort to populate and maintain the values contained

within them. The effort required to maintain fuzzy data could very easily undermine the value of fuzzy data.

It is possible, however, that the significance of this effort could be mitigated through the use of available tools to facilitate the man-machine interface. A significant area of future research should be focused on the design, implementation, use and analysis of various approaches toward the facilitation of the fuzzy data interface.

1.7 Nested RVA's

The current MySQL implementation of the rvachar data type nests only one RVA into a nesting table. Because the relation contained within an RVA is a relation, this relation must be able to contain an RVA as an attribute within its value. Effort should be made into the design and implementation of RVA values to be nested within other, superior, RVA values.

References

1 Larry R. Williams. “Considering a Database Application Using the Principles of Fuzzy Set Theory”. College of William and Mary. 2011 Graduate Research Symposium.

2 L. A. Zadeh. Fuzzy Sets. Information and Control, 8:338-353, 1965.

3 George J. Klir and Bo Yuan. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice Hall, Upper Saddle River, NJ, 1995.

4 Daniel McNeill and Paul Freiberger. Fuzzy Logic. Simon & Schuster, New York, NY, 1993.

5 C. J. Date. An Introduction to Database Systems. Pearson Education, Boston, MA, 2004.

6 L. A. Zadeh. Fuzzy Sets as a Basis for a Theory of Possibility. Fuzzy Sets and Systems, 100(Supplement):9{34, 1999.Reprinted from Fuzzy Sets and Systems 1 (1978) 3-28.

7 E. F. Codd. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, 13(6):377{387, 1970.

8 C. J. Date. An Introduction to Database Systems. Pearson Education, Boston, MA, 2004.

9 E. F. Codd. Extending the Database Relational Model to Capture More Meaning. ACM Transactions on Database Systems, 4(4):397- 434, 1979.

10 E. J. Lemmon. Beginning Logic. Hackett Publishing, Indianapolis, IN, 1978.

11 Jose Galindo. Introduction and Trends to Fuzzy Logic and Fuzzy Databases in Handbook of Research on Fuzzy Information Processing in Databases, ed. Jose Galindo. Information Science Reference, Hershey, PA, 2008.

12 Kwang H. Lee. First Course on Fuzzy Theory and Applications. Springer, Berlin, 2005.

13 Zongmin Ma. Fuzzy Database Modeling of Imprecise and Uncertain Engineering Information. Springer, Berlin, 2006.

14 Fredrick E. Petry. Fuzzy Databases, Principles and Applications. Kluwer Academic Publishers, Boston Massachusetts, 1996

15 C. J. Date. NOT Is Not “Not"! in Relational Database Writings 1985-1989. Addison-Wesley, Reading, MA, 1990.

16 C. J. Date and Hugh Darwen. Databases, Types, and the Relational Model: The Third Manifesto. Addison-Wesley, Reading, MA, 2007. 3rd edition.

17 C. J. Date. What is a Domain? in Relational Database Writings 1985-1989. Addison-Wesley, Reading, MA, 1990.

18 Technical Committee Group NCITS H2. ANSI/ISO/IEC 9075-1:1999 (SQL/Framework). ANSI/ISO/IEC International Standard (IS), Washington, DC, 1999.

19 Microsoft. Microsoft O ce Access 2003. Microsoft Corporation, Redmond, WA, 2003. O ce Help System.

20 Microsoft. Transact-SQL Reference. Microsoft Corporation, Redmond, WA, 2012. http://msdn.microsoft.com.

21 Oracle. Oracle Database SQL Reference 11g Release 2. Oracle Corporation, Redwood City, CA, 2012

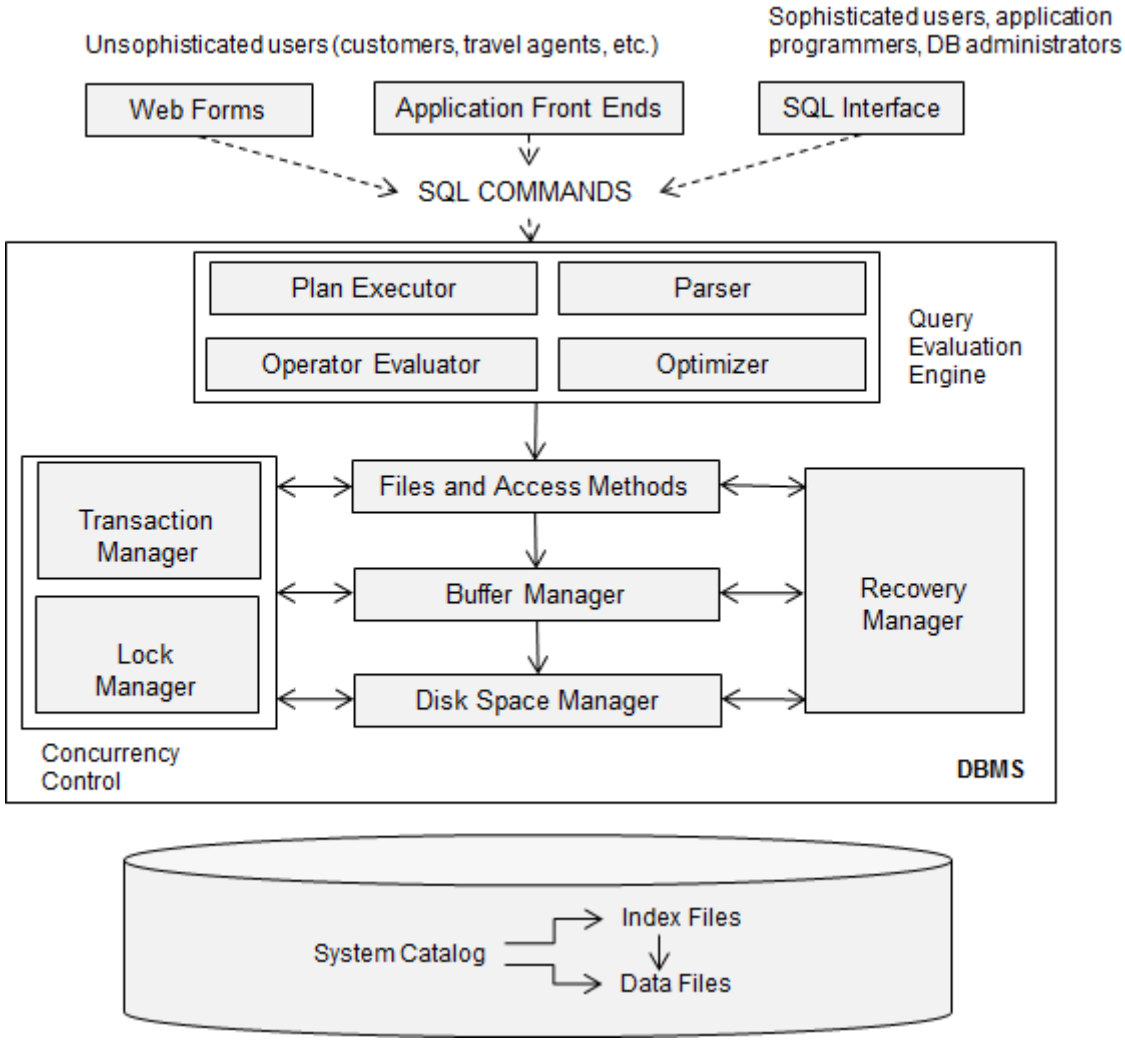
22 Oracle. ask Tom. Internet, January 2012. http://asktom.oracle.com.

23 Actian. Ingres 10.0 SP1 SQL Reference Guide. Actian Corporation, Redwood City, CA, 2012.

24 PostgreSQL Global Development Group. PostgresSQL 9.2.3 Documentation. PostgreSQL Global Development Group, Internet Community, 2013.

25 Oracle. MySQL 5.1 Reference Manual. Oracle Corporation, Redwood Shores, CA, 2012.
26 Norman E. Fenton and Shari Lawrence Peeger. Software Metrics: A Rigorous & Practical Approach. PWS, Boston, MA, 1997. 2nd edition (Revised Printing).
27 U.S. Department of Health and Human Services. National health statistics reports. Internet, October 2008. http://www.cdc.gov/nchs/data/nhsr/nhsr010.pdf.
28 Chen, Peter. "The Entity-Relationship Model--Toward a Unified View of Data". In: ACM Transactions on Database Systems 1/1/1976 ACM-Press ISSN 0362-5915, S. 9–36
29 Chen, Peter and Arie Zvieli. "ER Modeling and Fuzzy Databases." Proceedings of the Second International Conference on Data Engineering: 320-27. Print.
30 Chaudhry, N., J. Moyne and E. A. Rundensteiner. "An Extended Database Design Methodology for Uncertain Data Management." Information Sciences (1999): 83-112. Print.
31 Galindo, Jose, Angelica Urrutia, and Mario Piattini. "Fuzzy EER: Main Characteristics of a Fuzzy Conceptual Modeling Tool." Fuzzy Databases Modeling, Design and Implementation. London: Idea Group Publishing, 2006. 75-144. Print.
32 C. J. Date, and Hugh Darwen. Relational database writings, 1989-1991. Addison Wesley, 1992. 75-98. Print.
33 Guenther, Rebecca, and Jacqueline Radebaugh. "Understanding Metadata." Www.niso.org. National Information Standards Organization, n.d. Web. 12 Mar. 2013. <http://www.niso.org/publications/press/UnderstandingMetadata.pdf>.
34 Ramakrishnan, Raghu, and Johannes Gehrke. Database Management Systems. Boston [u.a.: McGraw-Hill, 2008. 327-333 Print.
35 Ramakrishnan, Raghu, and Johannes Gehrke. Database Management Systems. Boston [u.a.: McGraw-Hill, 2008. 342-343 Print.
36 Ramakrishnan, Raghu, and Johannes Gehrke. Database Management Systems. Boston [u.a.: McGraw-Hill, 2008. 226. Print
37 C. J. Date, and Hugh Darwen. Relational database writings, 1989-1991. Addison Wesley, 1992. 75-98. Print.
38 Chowla, Vaibhav, and Naveen Shetty. Implementation of RVA in MySQL Framework. Tech. Virginia Commonwealth University, Richmond, Virginia. Print. May 2012.
39 William Wulf and Mary Shaw, “Global Variable Considered Harmful”, ACM SIGPLAN Notices, volume 8, issue 2, 1973 February, pp. 28–34.
40 Nielsen, J. (1994b). Heuristic evaluation. In Nielsen, J., and Mack, R.L. (Eds.), Usability Inspection Methods, John Wiley & Sons, New York, NY. http://www.useit.com/papers/heuristic/heuristic_list.html
41 Technical Committee Group NCITS H2. ANSI/ISO/IEC 9075-2:1999 (SQL/Foundation). ANSI/ISO/IEC International Standard (IS), Washington, DC, 1999.

Appendix A – The Database Management System



Reference: Ramakrishnan, Raghu, and Johannes Gehrke. *Database Management Systems*. Boston [u.a.: McGraw-Hill, 2008.

Appendix B – Overview of Issues Concerning Data Types

1 Representation of Unknown or Missing Data

The relational model allows data to change over time, but does not account for unknown and uncertain data. A business process may result in data that is missing because it is not yet known. While expected to be a temporary database state, there is no assurance the missing data will be included once it is available. Other data values may be permanently missing because an attribute may not be applicable under circumstances not anticipated when the database was designed. The solution to both of these problems is to enter all data known or applicable and to mark attributes that do not have values. Null is such a mark, but it requires implementation of a 3-valued logic (3VL). This logic must use null correctly and include a Boolean type with logic operators that supports true, unknown, and false values.

Null is implemented in most significant RDBMS and the corresponding support for a 3VL Boolean type is required by the SQL standard.[41,p.24] Although Codd's 3VL is included in the relational model, it is difficult to use and has been challenged. [15] The problems caused by unknown and missing data are not fully solved.

2 Attribute Domains and Domain Types

C. J. Date advocates for domains created from user defined data types including complex types composed using existing domains.[17,p.27-53] A domain is a set of values from which an attribute in a relation may take its values. This concept is similar to programming languages which allow a variable to assume only values compatible with its declared type. The programming language type must either be built into the language or defined by the user of the language. Date argues

that a similar typing capability for the relational database attribute is consistent with the relational model.

Appendix C – The *val_str()* Method

```
String *Field_rvachar::val_str(String *val_buffer __attribute__((unused)), String *val_ptr)
{
        /* Various test variables */
        char            *rva_val;
        uchar           *dummy = new uchar();
        select_result   *my_result;
        Item            *test_item;
        String          rva_select;
        Query_arena     *set = new Query_arena;
        THD             *select_thd;
        char            *sql_select;
        String          test_query;
        int             test_val  = 0;
        int             my_strlen = 0;
        bool            my_res    = 0;
        int             rc        = 0;
        char            *my_sql_string;
        String          success;
        CSET_STRING     sql_statement;
        Statement       my_statement;
        uint32          query_length = 0;
        unsigned long   buff_length = 0;
        ulong           setup_tables_done_option = 0;

        DBUG_ENTER("Field_rvachar::val_str");
        /* Original val_str logic */
        ASSERT_COLUMN_MARKED_FOR_READ;
        /*************************************************************/
        /* PROGRAMMING NOTES AND REMINDERS                           */
        /* See the comment for Field_long::store(long long)          */
        /* See the comment for ha_innobase::open in ha_innodb.cc     */
        /* See the comment in ha_innodb.cc:                          */
        /*    #define EQ_MY_THD(thd)      ((thd) == thd)             */
        /*    check_trx_exists()                                     */
        /*    innobase_trx_allocate()                                */
        /*    innobase_trx_init()                                    */
        /*    ha_innobase::table_flags()                             */
        /*    ha_innobase::store_lock()                              */
        /*    ha_innobase::change_active_index()                     */
        /* sql_error.cc line 422 commented out.                      */
        /* sql_class.cc method select_send::send_data                */
        /* sql_class.h  method select_send::send_data                */
        /*************************************************************/
        DBUG_ASSERT(table->in_use == current_thd);
        uint length;

        if (table->in_use->variables.sql_mode &
                MODE_PAD_CHAR_TO_FULL_LENGTH)
                length= my_charpos(field_charset, ptr, ptr + field_length,
                field_length / field_charset->mbmaxlen);
        else
                length= field_charset->cset->lengthsp(field_charset, (const char*) ptr,
                field_length);

        val_ptr->set((const char*) ptr, length, field_charset);
```

```
/************************** RVA Value. We just need one! ***************************/
rva_val = new char[val_ptr->length() + 1];
// Save off the RVA string
rva_val = val_ptr->c_ptr();
/***********************************************************************************/

// If it's Null, ignore it. If not, we have some work to do.
if (rva_val == NULL)
        test_val = 2;
else
{
        test_val = 1;

        /* Save off the current state. We only do this once */
        /******************** The selection string coming into the query ****************/
        my_result  = table->in_use->lex->select_lex.join->result;
        my_result->cleanup();

        enum enum_sql_command sql_command= table->in_use->lex->sql_command;  // SQL Command

        /************************** Handle only if 'Select' ****************************/
        if(sql_command == SQLCOM_SELECT) {
                /* Create a new thread to send through the parser                    */
                /* See comment in sql_parse method check_stack_overrun               */
                select_thd                      = new THD;

                mysql_reset_thd_for_next_command(select_thd);

                THD *test_thd = current_thd;

                test_val = test_thd->id;

                select_thd->init();

                // Set up hard coded test query.
                query_length = strlen("Select a.vid, b.weight, b.type from vehicle as a inner join v_type
                                as b on a.vid = b.vid;");
                my_sql_string
                        = new char[query_length];
                my_sql_string
                = "Select a.vid, b.weight, b.type from vehicle as a inner join v_type as b on a.vid = b.vid;";
                select_thd->set_query("Select a.vid, b.weight, b.type from vehicle as a inner join
                                        v_type as b on a.vid = b.vid;", query_length);
                select_thd->stmt_arena->set_query_arena(set);

                MEM_ROOT my_root;
                Open_table_context my_context(select_thd, MYSQL_OPEN_TEMPORARY_ONLY);
                my_context.can_recover_from_failed_open();

                /* SQL Thread Statement assignment*/
                select_thd->db                  = "test";
                select_thd->db_length               = 4;

                select_thd->query_string=
                        CSET_STRING(my_sql_string, query_length, field_charset);

                select_thd->current_tablenr         = 2;
```

```
                        size_t test = 1;
                        unsigned int query_size = query_length;
                        select_thd->mem_root = new MEM_ROOT;
                        select_thd->mem_root->free             = 0;
                        select_thd->mem_root->used             = 0;
                        select_thd->mem_root->pre_alloc        = select_thd->mem_root->free;
                        select_thd->mem_root->min_malloc       = 32;
                        select_thd->mem_root->block_size       = 8164;
                        select_thd->mem_root->block_num        = 16;
                        select_thd->mem_root->first_block_usage = 0;
                        select_thd->mem_root->error_handler    = 0;

                        select_thd->alloc(query_size);

                        /* Thread Net Structure */
                        select_thd->net                  = table->in_use->net;
                        select_thd->stmt_da                = table->in_use->stmt_da;

/* Reset buffers */
        select_thd->net.buff     = new unsigned char[query_length];
                        buff_length = 16384;
                        select_thd->net.buff_end  = new unsigned char[buff_length];;
                        select_thd->net.pkt_nr    = 1;

                        strmake((char*) (*select_thd).net.buff, my_sql_string, query_length);
                        select_thd->net.write_pos = select_thd->net.buff;
                        select_thd->net.read_pos  = select_thd->net.buff;

                        select_thd->main_security_ctx          = *table->in_use->security_ctx;
                        select_thd->security_ctx             = &select_thd->main_security_ctx;

                        /* Parser State Required Variables */
                        Parser_state *my_parser_state          = new Parser_state;

                        Lex_input_stream *my_lip               = new Lex_input_stream;
                        my_lip->init(select_thd, my_sql_string, query_length);

                        my_parser_state->m_lip               = *my_lip;
                        my_parser_state->m_lip.m_thd = select_thd;
                        my_parser_state->m_lip.lookahead_token  = -1; // Less than zero

                        /* Select_thd initialization */
                        select_thd->m_parser_state           = new Parser_state;
                        select_thd->m_parser_state           = NULL;
                        select_thd->lex->m_stmt              = NULL;
                        select_thd->lex->safe_to_cache_query    = 0;

                        table->in_use->packet.free();
                        String new_packet;
                        select_thd->packet = new_packet;
                        select_thd->packet.set("", 0, field_charset);
                        select_thd->packet.alloc(16384);

                        select_thd->protocol->init(select_thd);
                        select_thd->client_capabilities = table->in_use->client_capabilities;

                        my_pthread_setspecific_ptr(THR_THD, select_thd);
  /*****************************************************************************/
  /* Send thread and parser state to mysql_parse                               */
```

```
        /* This is the key. We send a new thread back into the parser to be parsed, validated */
        /* and processed.                                                                     */
                        mysql_parse(select_thd,
                                    select_thd->query(),
                                                select_thd->query_length(),
                                                my_parser_state);
        /*****************************************************************************************/

                select_send my_send ;
                        my_send.set_thd(select_thd);
                        my_send.prepare(select_thd->lex->select_lex.item_list, &select_thd->lex->unit);

        /* Finalize server status flags after executing a command.                            */
                        select_thd->update_server_status();
                        select_thd->protocol->end_statement();

        /* Set the error that stops further original SQL processing.                          */
                        success.set("RVA Success", 11, field_charset);
                        my_error(ER_CHECK_NOT_IMPLEMENTED, MYF(0), success);

        my_pthread_setspecific_ptr(THR_THD, table->in_use);

                        goto end;
                }

        }
        return val_ptr;

end:
        /* Reset is_sent required to allow an error to be processed. MySQL will not process */
        /* any errors once a good result has been sent. The process would then fail.        */
        table->in_use->stmt_da->is_sent = FALSE;
        select_thd->stmt_da->is_sent    = FALSE;
        DBUG_RETURN(val_ptr);
}
```

Appendix D – The *build_query()* Method

```
char *Field_rvachar::build_query(THD *thread, CHARSET_INFO *set, char *knowledge, char
*query)
{
	List_iterator_fast<Item>  it(thread->lex->select_lex.item_list);
	SQL_I_List<TABLE_LIST> tab = thread->lex->select_lex.table_list;
	size_t s_pos = 0;
	size_t e_pos = 0;
	size_t s_len = 0;
	int found    = 0;
	int i        = 0;
	int length   = 0;
	int end      = 1;

	String tables;
	String base_tab;
	String rva_tab;
	String rva_key;
	String func_type;
	String key;
	String fields;
	String find;
	String ampersand;
	String value;
	String new_query;
	String old_query;
	String str_work;
	String str_knowledge;
	enum_field_types my_type;

	/* Make knowledge case uniform */
	while(knowledge[i])
	{
		knowledge[i] = toupper(knowledge[i]);
		i++;
	}

	i = 0;
	while(query[i])
	{
		query[i]     = toupper(query[i]);
		i++;
	}

	/* Get Key words into a string for manipulation */
	tables.copy("@TABLE:", strlen("@TABLES:"), set);
	key.copy("@KEY:", strlen("@KEY:"), set);
	fields.copy("@FIELDS:", strlen("@FIELDS:"), set);
	str_knowledge.copy(knowledge, strlen(knowledge), set);

	//Base Table
	base_tab.copy(thread->lex->select_lex.table_list.first->table_name,
			strlen(thread->lex->select_lex.table_list.first->table_name),
			set);
	//RVA Table
```

```
        ampersand.copy("@",1,set);
        s_pos = str_knowledge.strstr(tables, 0);
        s_pos = s_pos + tables.length();
        e_pos = str_knowledge.strstr(ampersand, s_pos);
        e_pos = e_pos - s_pos;
        rva_tab.set(str_knowledge,s_pos,e_pos);

        //RVA Key
        s_pos = s_pos + e_pos;
        s_pos = str_knowledge.strstr(key, 0);
        s_pos = s_pos + key.length();
        e_pos = str_knowledge.strstr(ampersand, s_pos);
        e_pos = e_pos - s_pos;
        rva_key.set(str_knowledge,s_pos,e_pos);

        old_query.append(query);

        /*** Create the SELECT segment ***/
        new_query.append("SELECT");

          for (Item *item= it++; item; item= it++)
          {

/* If this is the RVA field, we have to extract the attribute fields from the knowledge
*/
/* value passed to the method and append it as a 'b.' table to the selection.
Otherwise,*/
/* we extract the field value from the item list and append it as an 'a.' table to the
*/
/* selection predicate.
*/
                my_type = item->field_type();

                if (item->field_type() == MYSQL_TYPE_RVACHAR)
                {
                     s_pos = str_knowledge.strstr(fields, 0); // Get RVA field values...
                     s_pos = s_pos + fields.length();
                     s_len = str_knowledge.strstr(ampersand, s_pos);
                     find.copy(",", strlen(","),set);
                     do
                     {
                            e_pos = str_knowledge.strstr(find, s_pos);

                            if (e_pos == -1)
                            {
                                e_pos = str_knowledge.length() - s_pos;
                                   end   = 0;
                            }
                            else
                            {
                                   e_pos = e_pos - s_pos;
                                   end   = 1;
                            }

                            new_query.append(" ");
                            new_query.append(rva_tab);  // RVA Table
                            new_query.append(".");
                            value.set(str_knowledge,s_pos,e_pos);
```

```
                        new_query.append(value);
                        new_query.append(",");
                        s_pos = s_pos + e_pos + end;

                    }while(s_pos != strlen(knowledge));
                }
                else
                {
                        new_query.append(" ");
                        new_query.append(base_tab);  // Base Table
                        new_query.append(".");
                        new_query.append(item->name);
                        new_query.append(",");
                }

          }

        /* Chop off the last comma */
          length = new_query.length();
          new_query.chop();


        /*** Create the FROM JOIN segment ***/
          end = 1;
          new_query.append(" FROM ");
          new_query.append(base_tab);   // Base Table
          new_query.append(" as ");
          new_query.append(base_tab);
          new_query.append(" inner join ");

          new_query.append(rva_tab);   // RVATable
          new_query.append(" as ");
          new_query.append(rva_tab);
          new_query.append(" on ");
          new_query.append(base_tab);
          new_query.append(".");
          new_query.append(rva_key);
          new_query.append(" = ");
          new_query.append(rva_tab);
          new_query.append(".");
          new_query.append(rva_key);

          /** The nested include value...**/
          /*select vid from vehicle where vid in (select vid from v_type where class =
'Car');*/

          find.copy("(", strlen("("),set);
          found = old_query.strstr(find,0);
          if (found > 0)
          {
                 s_pos = found;
                 find.copy("WHERE", 5,set);
                 e_pos = s_pos;
                 if (old_query.strstr(find,s_pos) > 0)
                 {
                   s_pos = old_query.strstr(find,s_pos);
                 }
                 if (s_pos > e_pos)
```

```
        {
                s_pos = s_pos + 6;
                find.copy(")", strlen(")"),set);
                e_pos = old_query.strstr(find, s_pos);
                e_pos = e_pos - s_pos;
                value.set(old_query,s_pos,e_pos);
                value.strip_sp();
                new_query.append(" and ");
                new_query.append(rva_tab);
                new_query.append(".");
                new_query.append(value);
        }
}
else
{
        /*****************************/
        /*** Create the WHERE segment ***/
        if (thread->lex->select_lex.where != NULL)
        {
                Item *ptr_next; //Used to iterate throught 'where' tree.
                Item_func::Functype my_func;
                ptr_next = thread->lex->select_lex.where;
                new_query.append(" WHERE ");
                do
                {
          my_func = ((Item_func*)ptr_next)->functype();
                        if (my_func == 12) //Item_func[COND_AND_FUNC])
                                i = i;
                        else
                        {
                        new_query.append(base_tab);
                        new_query.append(".");
                        new_query.append(ptr_next->next->name);

                        switch(my_func)
                        {
                        case 1:
                                func_type.copy(" = ",strlen(" = "),set);
                                        break;
                        case 3:
                                        func_type.copy(" <> ",strlen(" <> "),set);
                                        break;
                        case 4:
                                        func_type.copy(" < ",strlen(" > "),set);
                                        break;
                        case 5:
                                        func_type.copy(" <= ",strlen(" <= "),set);
                                        break;
                        case 6:
                                        func_type.copy(" >= ",strlen(" >= "),set);
                                        break;
                        case 7:
                                        func_type.copy(" > ",strlen(" > "),set);
                                        break;
                        }

                        new_query.append(func_type);
                        //new_query.append(find);
```

```
                                find.copy("'", strlen("'"),set);
                                new_query.append(find);
                                new_query.append(ptr_next->name);
                                new_query.append(find);
                                }
                                ptr_next = ptr_next->next;
                        }
                        while (ptr_next != NULL);
                }
        }
        find.copy(";", strlen(";"),set);
        new_query.append(find);

        query = new char[new_query.length() + 1];

        strcpy(query, new_query.c_ptr());

    return query;
}
```

Appendix E – The Test Script

Test Script

When answering a question, please consider 1 being ‘low’ or ‘not very much’ and 5 being ‘high’ or ‘a great deal’.

If you have any questions, please do not hesitate to ask.

Have fun!

The working tables shown below are for your reference with respect to attribute headings, data and table names.

Table: **EMP**

EKEY	FIRST	MI	LAST	DOB	PT	HAIR
11	Edgar	F	Codd	1923	0	
12	Chris	J	Date	1941	0	
13	Hugh	NULL	Darwen	NULL	0	**R**
14	Andrew	NULL	Warden	NULL	0	**V**
15	NULL	NULL	Parker	1985	0	**A**
16	Charles	W	Bachman	1924	0	
17	Bill	H	Smith	1960	0	**E**
21	Jeffrey	D	Ullman	1942	1	**H**
22	Margo	I	Seltzer	NULL	1	**A**
23	Fabian	NULL	Pascal	NULL	1	**I**
24	David	?	McGoveran	-1	1	**R**
25	Chris	NULL	Date	1941	1	
26	Andrew	NULL	Warden	NULL	1	

Table: **EHAIR**

EKEY	CLASS	WEIGHT
11	brown	0.90
11	grey	0.20
12	blonde	0.60
12	blonde	0.60
12	brown	0.75
12	red	0.20
13	black	0.75
13	brown	0.85
14	brown	0.70
14	grey	0.40
15	blonde	0.70
15	brown	0.65
16	black	1.00
17	black	0.85
17	grey	0.65

1.a. Please write a SQL statement that chooses all data from EMP. Cut and paste your query and the result below.

1.b. Does the result you received meet your expectations? On a scale of 1=Very Little to 5=Very Much: _____

Comments:

2. The result in question 1.a. includes an RVA field *hair* which contains a relation consisting of two fields, *weight* and *class*.

Given what you have seen to this point, how do you believe that you would write a query that would exclude the attribute *weight* out of the query? Please write your query and share your comments in the space provided below.

3. Adapt the query in question 1, to choose all records from EMP where *ekey = 16*. Cut and paste your query and the result below.

4. Please choose all records from EMP **where ekey in (select ekey from ehair where class = 'black')**. Cut and paste your result below.

5. Please enter the following query into the system and record your results by cutting and pasting your result in the space below the query.

```
select emp.ekey, first, last,
   class, weight
 from emp, ehair
where emp.ekey = ehair.ekey;
```

Now, please enter the same query, but this time leave out the field **class** from the select clause and paste your result below.

Is this result meaningful to you? Why or why not?

Given the following tables:

Table: **DEPT**

CODE	**NAME**	**HEAD**	**ADMIN**	**ROOM**
001	Oncology	Jamison	Albright	**RVA: D_ROOM**
002	Obstetrics	Steward	Simms	
003	General Practice	Cho	Johnson	

Table: **D_ROOM**

CODE	**NUMBER**	**TYPE**	**AVAIL**	**NETWORK**
001	2101	Patient	N	Y
001	2102	Patient	Y	N
001	3405	Office	Y	Y
001	3406	Office	Y	Y
002	2103	Patient	Y	Y
002	2104	Office	N	N
002	4001	Patient	N	N
003	2105	Patient	Y	Y
003	2106	Office	Y	N
003	3408	Office	N	Y

6.a. What do you think the query in part 7.b. below does?

6.b. Please enter the following query into the system and paste the result in the space below the query.

```
SELECT code, name, head, room
 FROM dept;
```

6.c. Did the result you received meet with your expectations? On a scale of 1=Very Little to 5=Very Much: _____

7. Below are two somewhat different representations of fuzzy data. Both representations contain the same data.

EKEY	FIRST	MI	LAST	AGE	CLASS	WEIGHT
16	Charles	W	Backman	88	Grey	0.50
16	Charles	W	Backman	88	Black	0.60
17	Bill	H	Smith	52	Brown	0.75
17	Bill	H	Smith	52	Grey	0.25
17	Bill	H	Smith	52	Blonde	0.15

Table A – Standard Grid Representation

EKEY	FIRST	MI	LAST	AGE	HAIR	
16	Charles	W	Backman	88	CLASS	WEIGHT
					Grey	0.50
					Black	0.60
17	Bill	H	Smith	52	CLASS	WEIGHT
					Brown	0.75
					Grey	0.25
					Blonde	0.15

Table B – Grouped Grid Representation

7.a. Do you think Table A or Table B is more understandable?

Why?

7.b. Do think the information in Table A or Table B is more accurate? Why?

Observations

Please answer the following questions on a scale of 1 to 5 with 1 being ‘low’ or ‘not very much’ and 5 being ‘high’ or ‘a great deal’.

8.a. Were the results you received with respect to fuzzy data meaningful and understandable? _____

8.b. Do you see a real world benefit to the use of fuzzy data in database systems? _____

8.c. Do you see a real world benefit to the use of RVA’s in support of fuzzy data? _____

8.d. Do you think the queries used to access fuzzy data are logical? _____

8.e. If you received error messages, were they appropriate? Please enter 0 if this question is not applicable. _____

8.f. Was it easy to write queries to obtain the result desired? _____

8.g. Were the queries hard to write in order to obtain the result desired? _____

PLEASE LET US KNOW WHEN YOU FINISH

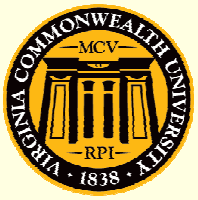
AND WE WILL SAVE THIS FILE

Thank you very much for your help!

Appendix F – System Study Presentation Slides

Slide 1

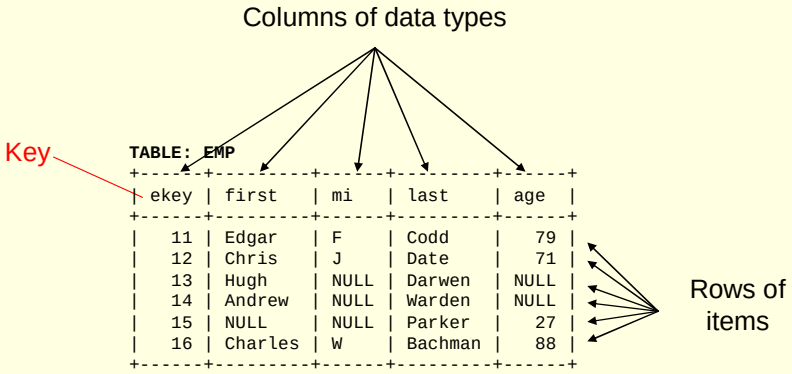
Structured Query Language (SQL)



Virginia Commonwealth University
School of Engineering
Department of Computer Science
Presented by
Larry R. Williams
And
Bob Morrissett

Slide 2

A Table



Slide 3

A Database

A database is a collection of related tables.

TABLE: EMP

ekey	first	mi	last	age
11	Edgar	F	Codd	79
12	Chris	J	Date	71
13	Hugh	NULL	Darwen	NULL
14	Andrew	NULL	Warden	NULL
15	NULL	NULL	Parker	27
16	Charles	W	Bachman	88

TABLE: ASSIGN

ekey	pkey	percent	eff_date
11	P1	0.50	2012-12-13
11	P3	0.50	2012-10-01
12	P1	1.00	2012-08-16
13	P1	0.75	2012-08-01
13	P2	0.25	2012-06-15
14	P2	1.00	2012-01-12
24	P3	1.00	2012-06-01
25	P2	1.00	2012-09-13

Slide 4

Starting MySQL

Change directory to F:

Start the server ‘mysqld’

Open a ***new*** window

Change the directory to F:

Start the client ‘mysql’

Slide 5

(SQL) select columns

Select EKEY, FIRST, LAST

Table: EMP

EKEY	FIRST	MI	LAST	AGE
11	Edgar	F	Codd	79
12	Chris	J	Date	71
13	Hugh	NULL	Darwin	NULL
14	Andrew	NULL	Warden	NULL
15	NULL	NULL	Parker	27
16	Charles	W	Backman	88

Slide 6

(SQL) from a table

Select EKEY, FIRST, LAST

From EMP;

Table: EMP

EKEY	FIRST	LAST
11	Edgar	Codd
12	Chris	Date
13	Hugh	Darwin
14	Andrew	Warden
15	NULL	Parker
16	Charles	Backman

Slide 7

(SQL) where something matches

```
Select FIRST, LAST, DOB
  From EMP
Where DOB = 1960;
```

Table: EMP

FIRST	LAST	DOB
Bill	Smith	1960

Slide 8

(SQL) select from two tables

Table: EMP

EKEY	FIRST	MI	LAST	DOB
11	Edgar	F	Codd	1923
12	Chris	J	Date	1941
13	Hugh	NULL	Darwin	NULL
14	Andrew	NULL	Warden	NULL
15	NULL	NULL	Parker	1985
16	Charles	W	Backman	1924

```
Select EMP.EKEY, FIRST,
       LAST, PKEY, PERCENT
  From EMP, ASSIGN
Where EMP.EKEY = ASSIGN.EKEY;
```

Table: ASSIGN

EKEY	PKEY	PERCENT	EFF_DATE
11	P1	0.50	12/13/2012
11	P3	0.50	10/01/2012
12	P1	1.00	08/16/2012
13	P1	0.75	08/01/2012
13	P2	0.25	06/15/2012
14	P2	1.00	01/12/2013
24	P3	1.00	06/01/2011
25	P2	1.00	09/13/2012

Slide 9

(SQL) create a result table

```
Select EMP.EKEY, FIRST,
       LAST, PKEY, PERCENT
From EMP, ASSIGN
Where EMP.EKEY = ASSIGN.EKEY;
```

EMP.EKEY	FIRST	LAST	PKEY	PERCENT
11	Edgar	Codd	P1	0.50
11	Edgar	Codd	P3	0.50
12	Chris	Date	P1	0.50
13	Hugh	Darwin	P1	0.75
13	Hugh	Darwin	P2	1.00
14	Andrew	Warden	P2	1.00

Slide 10

(SQL) connect the related rows in the tables

- Join

Table: EMP

EKEY	FIRST	MI	LAST	DOB
11	Edgar	F	Codd	1923
12	Chris	J	Date	1941
13	Hugh	NULL	Darwin	NULL
14	Andrew	NULL	Warden	NULL
15	NULL	NULL	Parker	1985
16	Charles	W	Backman	1924

Table: ASSIGN

EKEY	PKEY	PERCENT	EFF_DATE
11	P1	0.50	12/13/2012
11	P3	0.50	10/01/2012
12	P1	1.00	08/16/2012
13	P1	0.75	08/01/2012
13	P2	0.25	06/15/2012
14	P2	1.00	01/12/2013
24	P3	1.00	06/01/2011
25	P2	1.00	09/13/2012

Slide 11

(SQL) select from both tables

```
Select EKEY, FIRST, LAST
  From EMP;
```

Table: EMP

EKEY	FIRST	MI	LAST	DOB
11	Edgar	F	Codd	1923
12	Chris	J	Date	1941
13	Hugh	NULL	Darwin	NULL
14	Andrew	NULL	Warden	NULL
15	NULL	NULL	Parker	1985
16	Charles	W	Backman	1924

```
Select EKEY
  From ASSIGN
 Where PERCENT > 0.50;
```

Table: ASSIGN

EKEY	PKEY	PERCENT	EFF_DATE
11	P1	0.50	12/13/2012
11	P3	0.50	10/01/2012
12	P1	1.00	08/16/2012
13	P1	0.75	08/01/2012
13	P2	0.25	06/15/2012
14	P2	1.00	01/12/2013
24	P3	1.00	06/01/2011
25	P2	1.00	09/13/2012

Slide 12

(SQL) match values from one table to those in another table

```
Select EKEY, FIRST, LAST
  From EMP
 Where EKEY IN (Select EKEY
                  From ASSIGN
                 Where PERCENT > 0.50);
```

EKEY	FIRST	LAST
12	Chris	Date
13	Hugh	Darwin
14	Andrew	Warden

EKEY
12
13
14
24
25

Slide 13

Fuzzy Data

What color is this man's hair?

Slide 14

Fuzzy Data

What color is this man's hair?

0.75/Brown

0.25/Grey

0.15/Blonde

Slide 15

Fuzzy Data

What color is this man's hair?

EKEY	CLASS	WEIGHT
17	Brown	0.75
17	Grey	0.25
17	Blonde	0.15

Slide 16

Fuzzy Data

Table: EMP

EKEY	FIRST	MI	LAST	DOB	CLASS	WEIGHT
16	Charles	W	Backman	1924	Grey	0.50
16	Charles	W	Backman	1924	Black	0.60
17	Bill	H	Smith	1960	Brown	0.75
17	Bill	H	Smith	1960	Grey	0.25
17	Bill	H	Smith	1960	Blonde	0.15

EKEY	WEIGHT
16	0.50
16	0.60
17	0.75
17	0.25
17	0.15

Select EKEY, WEIGHT
From EMP_FT;

Slide 17

Fuzzy Data

HAIR

Table: EMP

EKEY	FIRST	MI	LAST	DOB	CLASS	WEIGHT
16	Charles	W	Backman	1924	Grey	0.50
16	Charles	W	Backman	1924	Black	0.60
17	Bill	H	Smith	1960	Brown	0.75
17	Bill	H	Smith	1960	Grey	0.25
17	Bill	H	Smith	1960	Blonde	0.15

Slide 18

Fuzzy Data

Relation Valued Attribute (RVA) 'HAIR'

Table: EMP

EKEY	FIRST	MI	LAST	DOB	HAIR	
16	Charles	W	Backman	1924	CLASS	WEIGHT
					Grey	0.50
					Black	0.60
17	Bill	H	Smith	1960	CLASS	WEIGHT
					Brown	0.75
					Grey	0.25
					Blonde	0.15

Slide 19

Fuzzy Data

Table: EMP

EKEY	FIRST	MI	LAST	DOB	HAIR	
16	Charles	W	Backman	1924	CLASS	WEIGHT
					Grey	0.50
					Black	0.60
17	Bill	H	Smith	1960	CLASS	WEIGHT
					Brown	0.75
					Grey	0.25
					Blonde	0.15

```
Select EKEY, HAIR
  From EMP
Where EKEY = '17';
```

Slide 20

Fuzzy Data

EKEY	HAIR	
17	CLASS	WEIGHT
	Black	0.85
	Grey	0.65

```
+------+--------+-------+
| ekey | WEIGHT | CLASS |
+------+--------+-------+
|   17 |   0.85 | black |
|   17 |   0.65 | grey  |
+------+--------+-------+
```

Actual Screen Display

```
Select EKEY, HAIR
  From EMP
Where EKEY = '17';
```

Appendix G – Vita

Larry Ritchie Williams, Jr. was born at the Naval Hospital at Naval Air Station (NAS) Mayport, Florida in 1960 to then Lieutenant Larry R. Williams, USMC and his wife Betty 'Whitt' Williams, RN.[vi] As the son of a Marine officer, Larry grew up living in many different States and Europe.

Larry attended the Virginia Military Institute in Lexington, Virginia where he received his BA in 1982. He received his MS in Systems Management in 1988 from the University of Southern California in Los Angeles, California. Prior to beginning his studies at Virginia Commonwealth University, Larry supplemented his education through various under graduate and graduate courses in Computer Science at George Mason University.

Larry's professional career has been spent primarily as a software engineer and information systems consultant working for such companies as KPMG Peat Marwick, the United States Department of the Navy, Dominion Resources, Altria, and Reynolds Metals with whom he spent two years working in Bunbury, Western Australia assisting in the effort to transition the information systems from a refinery's mainframe to an SAP enterprise system. In 2002, Larry founded Sterncastle Consulting, Incorporated and has been working since this time as a software engineering consultant to the United States Customs and Border Protection Service and the Department of Homeland Security.

He has been married to the former Lynne Jacobson for over 25 years. They have two children, Zachary and Joshua, both of whom they are justifiably proud.

---

vi Lt. Williams would later retire from the Marine Corps at the rank of Colonel and receive his PhD from George Washington University in 1994. He is currently a Collegiate Professor teaching in the Graduate School at University of Maryland, University College.