APPROXIMATE RETRIEVAL IN DATABASES CONTAINING INCOMPLETE INFORMATION

by

Suresh Ramaswamy

Bachelor of Technology
Regional Engineering College Warangal, 1982

Submitted in Partial Fulfillment of the
Requirements for the Degree of Master of Science
in the Department of Computer Science
University of South Carolina
1991

Department of Computer Science
Director of Thesis

Department of Computer Science
Second Reader

Department of Computer Science
First Reader

Dean of the Graduate School

TABLE OF CONTENTS

ABSTRACT

Database systems help us to organize and manage information in an efficient way. Invariably all database systems provide some form of querying facility to assist us in retrieving the desired information. Most database systems handle only crisp data and provide only crisp querying of such data. However, it is evident that in normal life there are many instances when decision making involves approximate reasoning using imprecise or approximate data.

The study of approximate reasoning encompasses the issues of handling approximate data and providing approximate querying of the database. Various approaches have been adopted to study the problem though not all of them address both of these issues. In this thesis an attempt is made to reasonably address both these issues from the perspective of end user ease.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to Dr. Caroline Eastman, for the support and guidance during the course of my thesis. I would also like to thank Dr. Robert Trueblood and Dr. Robert Oakman for their support and suggestions.

I take this opportunity to also thank the staff of the Computer Science department and in particular Dr. Manton Matthews for making my stay here a pleasant one.

I would also like to thank my parents and family members in India for their constant guidance and support. Finally I am at a loss of words to express my gratitude to my wife Jaya without whose help this task would have been a difficult one.

CHAPTER 1

INTRODUCTION

1. MOTIVATION FOR WORK

Database systems have helped use and manage information more efficiently than traditional file systems of the past. Since their advent in the late sixties, they have become increasingly popular, particularly in the last decade with the rampant use of personal computers. Suitable querying of the database produces specific information, which has become an essential part of the decision making process in most organizations. Query formulation and retrieval of specific information has been one of the major areas of interest in database research. As of now many commercial database systems are available and are being used successfully in different aspects of industry. These systems are basically designed to handle precise information or data, and the numerous query languages available are designed to handle precise information and/or formulation of precise queries. However it is evident that in normal life there are many instances when decision making involves approximate reasoning using imprecise or approximate data. The above requirement, coupled with the fact that there is a large usage of database systems, has highlighted the need for an efficient system to handle approximate retrieval

in databases.

As indicated by Eastman [EAST 87], the study of approximate retrieval in databases involves addressing two issues: approximate data and approximate retrieval. Several models have been proposed to consider approximate retrieval with only crisp data. The models that have considered approximate retrieval and incorporate approximate data are generally based on fuzzy set models, and the approximate data stored are in the form of fuzzy values. Not much has been done to handle retrieval in a DBMS where the data values stored may be approximate but nonfuzzy.

The aim of this research is to extend the relational model

- to incorporate nonfuzzy approximate data
- to provide some simple schemes of retrieval for both crisp
- and approximate data in such a model.
- to provide a simple user interface and address some feedback issues.

2. ORGANIZATION OF THE THESIS

Chapter 2 presents a brief overview of related material. Chapters 3, 4 and 5 cover the basic design aspects of the model. Chapter 3 discusses the handling of approximate data in the data base. Chapter 4 deals with approximate querying of the data base. Chapter 5 addresses some user feedback issues. Finally Chapter 6

summarizes the work and presents some conclusions; in addition, some possible future extensions to the model are discussed.

Appendix I presents a sample employee data base. Most of the examples discussed are based on this sample data base. Appendix II gives a formal representation of the query language syntax.

CHAPTER 2

LITERATURE REVIEW

1. INTRODUCTION

During the last decade considerable work has been done in the area of approximate retrieval. The study of approximate retrieval encompasses the two issues of approximate querying of the database and handling of approximate data in the data base. Various approaches have been adopted to study the problem though not all of them address both of these issues.

Flexible user interfaces present a simplistic approach to query the data base, the focus mainly being ease of use. The user is relieved of the burden of knowing a query language and information is extracted from the data base either by browsing or by interactively building a query in stages. Though desirable from the end user point of view, these models do not permit any approximate querying and neither do they handle approximate data.

Various data models have been proposed to handle and consider the effect of incomplete values in databases. Nulls can be considered as a special case of incomplete data, and significant work has

been done regarding the treatment and handling of nulls [CODD 86, ZANI 82]. Crisp incomplete data in other forms such as range values and possible options from a list of values has also been considered. In the area of approximate retrieval much work has been done to represent several forms of incomplete data within the fuzzy framework.

The major thrust of work addressing the problem of approximate retrieval has been addressed by several systems under the fuzzy approach and nonfuzzy approach categories. Detailed descriptions of these systems are presented later. The models discussed under both these categories address the problem of approximate retrieval by computing the similarities between the different attribute values in the various tuples. Most of the nonfuzzy models handle only crisp data, whereas many of the fuzzy models handle approximate data within the fuzzy framework.

As our model aims to handle nonfuzzy approximate data, we also present some material regarding the handling and representation of such data.

We provide a brief overview in the following areas:

- Approaches to handling missing data
- Approximate retrieval models
- Flexible user interfaces.

2. APPROACHES TO HANDLE INCOMPLETE DATA WITHIN THE RELATIONAL FRAMEWORK

2.1 CODD'S MODEL FOR MISSING INFORMATION (APPLICABLE AND INAPPLICABLE)

This model proposed by Codd [CODD 86, CODD 87] gives a detailed analysis of handling null values, whether applicable or inapplicable for a given attribute. Two types of nulls are defined, and they are identified in the DBMS by use of appropriate semantic markers. One type of marker for a null, the A-mark, indicates that a possible data value could replace the A-mark at a later point of time. The other mark, the I-mark, indicates that an object cannot have any value designated for that attribute. The notion of three-valued logic is introduced, and the use of three-valued logic in evaluating a query is discussed. Some update issues involving markers and extensions to the query language are also presented.

2.2 LIPSKI'S MATHEMATICAL MODEL

In this model Lipski [LIPS 79] presents a mathematical model to handle incomplete information in a database. He discusses two different interpretations of a query -- the external and internal interpretation. The external interpretation refers to the interpretation as viewed in reality or by the real world. The information contained in the system may not be sufficient to

perceive this interpretation. As opposed to this, the internal interpretation is the interpretation of the query as viewed by the system with the help of the knowledge contained in it. Various axioms are presented to evaluate the external and internal interpretations of a query. A simple query language and its associated semantics are presented to manipulate the incomplete information database. The model has not been directly presented within the relational framework, though the author indicates that a major part of it could be rephrased within the relational model.

2.3 GRANT'S APPROACH TO HANDLE PARTIAL VALUES IN A DATABASE MODEL

Grant [GRAN 79, GRAN 77] proposes an extension to the relational model in which the attribute domains can contain sets of intervals to indicate range values in addition to the set of singletons and the null value. He presents some simple propositions to interpret the effect of range values for query evaluation. Some discussion is also presented regarding the effect of nonatomic attribute values and functional dependencies.

2.4 DATABASE RELAXATION AND INCOMPLETE VALUES

Shen [SHEN 88] proposes a model designed to serve as a front end query processor for a database containing incomplete values. The objective of this model is to structure an incomplete database to

facilitate query processing for a given query language. To achieve this facility, the author proposes the notion of a relaxed database which will serve to narrow down the objects to be processed in a query. The actual database then needs to focus only on objects derived from the relaxed database. The discussion here is oriented towards efficient query processing in an incomplete database in terms of restricting the number of objects to be processed and also in providing a better interpretation of the query. Detailed definitions and algorithms are presented and the discussion is in terms of a general query language.

3. APPROXIMATE RETRIEVAL MODELS

The problem has been mainly handled by two approaches:

- Fuzzy approach
- Nonfuzzy approach.

3.1 NONFUZZY APPROACH

Approximate retrieval is handled by applying some kind of similarity function to attribute values. In this approach, the question of measurement scale for an attribute must be addressed, and the similarity function applied depends on the scale of the domain.

3.1.1 THE ARES SYSTEM

Ichikawa and Hirakawa [ICHI 86, HIRA 84, ICHI 80] describe a relational database system with the capability of performing flexible interpretation of queries. The authors define a general purpose comparison operation to handle similarities between attribute values in a domain.

In their model they classify domains as Type I, Type II and Type III.

- Type I domain elements do not permit any similarity function to be defined
- Type II domain elements are ordered and permit similarities defined to reflect differences in attribute values
- Type III domain elements are not ordered but permit similarity measurements on attribute values. The similarity between the elements of this type of domain is represented by a graph. The authors provide a graphical tool in their implementation to represent this kind of similarity.

Normal relational algebra operations of selection, restriction and join are extended to handle the comparison operator introduced for approximate retrieval. Some additional features such as specifying ambiguity allowances for attributes restricting the number of output tuples are also discussed. To obtain a high degree of user friendliness, the system has been implemented with the QBE user interface.

3.1.2 THE VAGUE SYSTEM

Motro [MOTR 88] describes this system as a user interface to a relational database that permits vague queries. Data metrics which define the distance between attribute values of a domain are discussed. A similar-to comparison operator is defined to handle similarities between attribute values of a domain.

Three types of data metrics are discussed:

- computational metrics where distances between attribute values of a domain are derived by computation only
- tabular metrics where distances between attribute values are stated in a table and obtained by retrieval
- referential metrics where the distance between key attribute values are computed using the distance values of the non key attributes. To accommodate for the fact that different domains with different metrics may be involved, scaling is done. User supplied weighting factors are also considered in the evaluation of referential metrics.

Additional features such as a radius for each attribute value are discussed; the radius is primarily used as a threshold to select attributes with distances smaller than a threshold. Some suggestions are also presented regarding the selection of suitable metrics for domains of a database. Finally some material is presented regarding the handling of incomplete information in the system.

3.1.3 IMPLEMENTATION OF TWO APPROXIMATE OPERATORS

In this approach Mc Clelland, Trueblood and Eastman [McCL88] define two new operators CLOSE_TO and SOUNDS_LIKE to address the problem of approximate retrieval. The CLOSE_TO operator is defined to handle numeric domains and SOUNDS_LIKE is for handling character strings.

The SOUNDS_LIKE operator is based on a modified Soundex algorithm and is used to retrieve character strings that phonetically sound like a given character string. The CLOSE_TO operator combines statistical techniques of standard deviation and relative similar distance for a set of data to determine the selection of tuples close to a desired value. The two operators have been implemented in the BDB system, a research system which runs on Burroughs equipment.

3.1.4 DEGREES OF ITEM INTEREST

This model is based on decision theory and is described by Rowe [ROWE 84]. The model attempts to help the user in choosing from a set of alternatives from a data base system. The output selected is ranked in the order of utility to the user. According to the author the model is also aimed at improving the efficiency and cooperativeness of the interface between the decision maker and a

data base system. The author points out that standard query languages are typically suitable for absolute conditions or operations and in general will be unsuitable for handling the problem of choosing alternatives from a data base. He also discusses some of the problems associated with the use of weighted terms and its impact on the user community.

The model basically includes two components :

- decision theory approach to identify and represent information
- an inference mechanism used in querying.

In the first component the author introduces the notion of utility and suitability (which indicates the probability that a given feature is suitable for a user choice). Discussion is also presented regarding choosing utilities and suitabilities for a domain, the need to differentiate separate utilities, and the effect of combining utilities and suitabilities. Using an AI approach the information is stored in slots under frames, and the process is governed by domain specific rules. According to the author the basic idea for this kind of "slot filling" is to incorporate a certain degree of customization for users and usages in the model.

The second component discusses an inference mechanism with the aim of improving the flexibility in the model. Inferences can be drawn for any query based on the information in the various

frames. As the frames are designed to reflect user behavior, this can be viewed as a form of indirect user feedback.

The model is explained in terms of an example regarding handling cargo transport in the merchant shipping industry.

3.2 FUZZY APPROACH

3.2.1 MODELS BASED ON CRISP DATA AND EMPLOYING ONLY LANGUAGE EXTENSIONS

Tahani [TAHA 77], and Kacprzyk and Ziolkowski [KACP 86] discuss models which primarily permit extending the query language to accommodate fuzzy linguistic predicates. The data considered in both these models are crisp. The basic idea is that permissible fuzzy linguistic terms are defined for the various attributes and that these terms can be used for querying the database. Some techniques for adding suitable modifiers to the terms and creating composite terms [TAHA 77] are also discussed.

3.2.2 FUZZY RELATIONAL DATABASE MODEL

Zemankova and Kandel [ZEMA 85] present a model which is an extension of a relational model to incorporate fuzzy set concepts. The authors state that the basic aim of the model is to handle imprecise data values and cater to individual profiles. In this

model the domains of various attributes are extended. In addition to normal scalar/numeric values, attribute values could be sequence lists of scalar/numeric quantities or hold possibility distributions. In addition to handling of imprecise information, considerable discussion is provided on fuzzy terms such as labels, modifiers, fuzzy relationships and fuzzy connectives. The paper also presents an excellent overview of fuzzy set theory.

3.2.3 THE SIMILARITY RELATION BASED MODEL

The model presented by Buckles and Petry [BUCK 84] tries to address the dual issues of handling imprecise information and approximate querying of the database. They permit domains in their model to be extended to include sequence lists of scalar and numeric values in addition to singleton values. They also propose that a similarity relation should be defined for each domain indicating the degree of similarity between every pair of elements in the domain within the interval [0,1]. Threshold values can be specified for any domain, and the similarity relation can be used for merging of tuples provided that the similarity levels stated in the similarity relation do not fall below the threshold value specified for that domain.

4. FLEXIBLE USER INTERFACES

4.1 THE RABBIT INTERFACE

Williams [WILL 84] describes his model to be a database retrieval interface based on the paradigm of retrieval by reformulation. The basic idea proposed is that each querying session can be an extended interaction between the user and the system where the user is permitted to refine his/her query stage by stage depending on the responses and options provided at each stage. This approach is particularly beneficial to casual users who have very little knowledge of what they want or what are the exact contents of a large database. Rabbit uses a semantic net to represent the database, and some functions are provided to map the tuples from the underlying relational database using the distances in the semantic net. One major feature of Rabbit is that the user can initiate a query session by first describing one of the basic objects of the database. There is no need for the user to be burdened with the knowledge of a query language. The response generated at every stage provides six critiquing options to facilitate further refinement of the query. Some other features such as dynamic perspectives and analysis questions are also discussed by the author.

4.2 BAROQUE: A BROWSER FOR RELATIONAL DATABASES

Motro [MOTR 86] describes a user interface where the user can either browse through the database by using navigation or probing styles. It initiates an exploratory search of the database in which information about the various objects are classified hierarchically, extending from the name of the database to the various data values. It provides a network view of the relational database and is equipped with four different functions that scan the network in four different ways. One of the major advantage of this model is that it is targeted for the use of `casual users who may not be familiar with the structure or contents of the database and/or have only a vague idea of the retrieval target. The browsing interface also does not burden the user with the knowledge of any query language. As described by the author, it seems that the model can also be easily implemented on top of any existing relational database system.

CHAPTER 3

HANDLING APPROXIMATE DATA

1. INTRODUCTION

The domains of various attributes in our model are extended to include three types of incomplete information. The information, though incomplete, is nonfuzzy and is as indicated below;

- range values for numeric domains. e.g. attribute age can be assigned a value {30, 40} indicating that the age of the person can lie within the range having end points 30 and 40.

- sequence list of numeric or scalar values where the attribute could take the value of any one element from the list. e.g. attribute city = {Columbia / Atlanta / Orlando} implies that city could be either Columbia or Atlanta or Orlando.

- undefined and inapplicable values of an attribute. Any object for which an attribute value is missing is viewed as an undefined value for that attribute at that instant of time. In general, this can be interpreted as a special case of

the above two types of incomplete information as in this case the user does not even have partial knowledge to restrict the value of the attribute to some fragment of the domain. Instances when a value is not permitted for the given attribute are also covered by this kind of specification.

In our model we restrict the primary key attribute(s) of a relation to contain only crisp values.

Significant work has been done to handle incomplete data within the fuzzy approach to approximate retrieval. Some of these approaches extend the attribute domains to handle fuzzy representations of data as discussed in Zemankova and Kandel [ZEMA 85], while models such as those presented by Buckles and Petry [BUCK 84] discuss the notion of similarity relations, which indicate the fuzzy similarities between the various attribute values for a given domain. Our approach does not adopt the fuzzy framework to represent incomplete data. The approach to handling missing or unknown values is very similar to the conventional treatment of nulls in relational data base systems. Instead of specifying null for an unknown value we identify that particular attribute value to be of status U (undefined). The main purpose for adopting this approach is to maintain a uniform basis for the treatment of incomplete values as well as unknown values.

Other than the treatment of nulls we have included only two major types of incomplete data, namely range values and sequence lists. Though it may be possible to investigate other types of representation of incomplete data, we restrict our study to the types discussed above for the following reasons :

- The types we are considering seem to be the kind one may frequently encounter in real life situations.
- Each type of incomplete data considered may require special techniques for query processing and in addition possibly some preprocessing prior to storage or retrieval. This implies that the greater the number of incomplete data types handled, the higher the likelihood of the system being very complex.

The following topics are discussed in detail with regard to handling approximate data :

- representation of approximate data
- functional dependency and normal form issues
- operational issues.

2. REPRESENTATION OF APPROXIMATE DATA

Attributes of a relation which may hold incomplete values are identified in the catalog. A suitable status indicator precedes the values of these attributes and could be one of the following;

- C --> crisp value
- R --> range value
- S --> sequence list
- U --> undefined

It is important to note that, unlike conventional data bases, the values of the various attributes could be nonatomic, particularly those attributes containing range values and sequence list values. Furthermore the various status indicators are used mainly by the DBMS for interpretation purposes. These are not to be treated as normal attribute values of a data base. Whenever a crisp data value is entered for an attribute, the DBMS places a status indicator C along with the actual value in the data base. Likewise R and S indicators are used with range values and sequence lists, and a U indicator is used with undefined values of the attribute. It is envisaged that, when the precise value of a data element is known, the DBMS will suitably change an R or S status indicator to C. It is also possible that when more information is known about a particular undefined attribute value, then the U indicator can be modified to R or S, depending on the case (still incomplete) or it

could change to a C if a crisp value is entered. Likewise it is also possible to change a crisp value to an incomplete value and the status indicator will change from C to either R, S, or U, as the case may be. It may also be noted that this aspect of the design could be extended to represent other forms of incomplete data in the future.

The DBMS could also use the status indicators to retrieve tuples selectively based on the type of incomplete information contained in them provided the query language can be extended to express the same. More material on this subject is presented under the topic of query processing.

3. FUNCTIONAL DEPENDENCY AND NORMAL FORM ISSUES

As our data model permits attributes to contain either crisp or incomplete values, it may be not be feasible to verify the functional dependency constraints between the attributes for the data base as a whole. This implies that functional dependency constraints will not be verified for tuples containing incomplete attribute values. However, when an attempt to change an incomplete value to a crisp value is made for an attribute, then the various functional dependency constraints for that attribute will be verified.

From our definition of incomplete attribute values, it may be

possible to verify a functional dependency in some cases; but in some others it may not be possible. A typical example is indicated below:

Suppose A -> B and the values of A and B (consider single attributes) are as shown:

1)

A	B
5	{30, 40}
5	{35, 45}

2)

A	B
5	{30, 40}
5	{50, 60}

If the second tuple is entered in case (2), then it clearly violates the functional dependency constraint A -> B. The same cannot be said of case (1) as the ranges of the B attribute indicated in the two tuples overlap and the functional dependency constraint can be verified only when the precise values of attribute B are known in both tuples. Even though the functional dependency constraints are checked and partial verification is possible, it must be remembered that further verification may need to be done if the incomplete values are finally changed to crisp values. Keeping this in mind we will defer the checking of functional dependencies for incomplete valued attributes until they are amended to contain crisp values. However, it will be possible to specify the various functional dependency constraints for the entire data base and verify them for tuples containing crisp values.

Since some of the attribute values can be nonatomic, the

relations in our model may not be in first normal form. One approach to handle non first normal form attribute values (mainly repeating groups) is to permit the decomposition of tuples containing these values by defining some kind of suitable operator. This kind of approach is well described in Jaeschke and Schek [JAES 82]. By virtue of our interpretation of incomplete values, we will not permit the decomposition of any tuple containing nonatomic attribute values.

We need to revise the definitions of the higher normal forms such as second normal form, third normal form etc. to exclude the requirement of atomic valued attributes. Our aim is to design relations in third normal form. In doing this we must bear in mind that if we try to losslessly decompose a relation in second normal form, then the decomposition must be done along the attribute(s) with only crisp values. The above is essential as the attribute(s) along which decomposition is done may constitute the primary key of the newly formed relation. Hence a better approach would be to first get the relations of our model in third normal form and then decide which non key attributes can have incomplete values.

4. OPERATIONAL ISSUES

4.1 THREE VALUED LOGIC

As some of our attribute values can be incomplete, it is evident

that we cannot verify the values of predicates in a query to be only TRUE or FALSE. We adopt the methodology of three valued logic as described by Codd [CODD 86], where the predicates can take the values TRUE or FALSE or MAYBE. The truth value MAYBE may result whenever any predicate involves an attribute containing incomplete values. Evaluating a condition clause in a query in our system would result in one of the truth values TRUE, FALSE or MAYBE. As indicated by Codd [CODD 86], if we denote P and Q to be two propositions, then the following are truth tables for the logic operations involving NOT, OR and AND operations.

P	NOT P
T	F
M	M
F	T

P OR Q		Q		
		T	M	F
	T	T	T	T
P	M	T	M	M
	F	T	M	F

P AND Q		Q		
		T	M	F
	T	T	M	F
P	M	M	M	F
	F	F	F	F

4.2 EFFECT OF RELATIONAL ALGEBRA OPERATIONS

The standard relational algebra operations, such as *Select*, *Project*, *Join,* etc., can be applied to our model. It must be kept in mind that, irrespective of the nature of the operation, if two attribute values in our model are compared as part of the operation, then the resulting relation may contain the

corresponding tuple(s) only if the comparison results in a TRUE value. It may also be possible to consider the output relation to contain tuples which yield the truth value MAYBE. This can be achieved by suitably extending our query language.

4.3 USE OF BUILT IN FUNCTIONS

With the exception of *COUNT*, the aggregate functions such as *SUM*, *AVERAGE*, etc., depend on the actual attribute values under consideration. This is due to the fact that *COUNT* just returns the number of tuples retrieved and is independent of the actual attribute values of any tuple. If the attribute value in question contains an incomplete value, then it may not be possible to use that in the evaluation of these various functions. Instead of restricting the user to use these functions only when querying on crisp data, it would be more reasonable to permit them for all queries in general and omit incomplete values in the evaluation of these functions. Depending on the nature of the attribute, it may also be possible to maintain default values for the attribute in the catalog which could be used for the evaluation of these functions. A similar problem as indicated above will also be experienced in the case of user defined functions involving incomplete valued attributes.

4.4 ORDERING OF VALUES IN THE RESULT

Ordering is significant only for crisp queries, that is queries having no fuzzy predicates. It is reasonable to assume that the user will be more interested in crisp data values rather than incomplete values. Hence we can adopt an ordering system where the incomplete values are arranged after the crisp values, whenever an ordering is indicated as part of the query.

Two other possible approaches to decide the order of the output are as indicated below :

- list in order based on the lowest value and have range values and sequence lists components sorted with least element first.

  e.g. 30, 31, {33, 37}, 34, {36/39}, 38.

- list in order based on the lowest value for crisp and range values and repeat the tuple containing a sequence list as many times as the number of elements of the list, in the respective places.

  e.g. Atlanta, {Atlanta/Columbia}, Boston, Columbia, {Atlanta/Columbia}

4.5 DELETION OF DUPLICATE TUPLES

The deletion of duplicate tuples has to be addressed for tables. These tables may not have any primary key to uniquely identify the

tuples in the table and are typically the outcome of a projection operation. To determine whether two tuples are duplicate, the corresponding attribute values are tested for equality using three valued logic. When we are comparing two incomplete attribute values for equality, such as AGE1 = {30, 40} and AGE2 = {30, 40}, the outcome will always be MAYBE even though the two incomplete values appear to be syntactically equal. Assuming that all other corresponding attributes are semantically equal, these two tuples will not be deemed duplicate even though the values are identical.

CHAPTER 4

APPROXIMATE QUERYING OF THE DATABASE

1. INTRODUCTION

Our data model contains both crisp and incomplete data values, and in general querying our data base could result in tuples containing a combination of crisp and incomplete data values. However we permit the user to query on crisp data only if required. The querying process itself can be in terms of crisp queries or approximate queries or compound queries containing a combination of crisp and approximate predicates. The major emphasis on approximate querying is by way of using fuzzy linguistic terms. We discuss several simple extensions to the query language SQL to help us achieve our goals.

When resolving a fuzzy predicate in our model, all tuples qualify to be retrieved, even those that match the predicate only to a remote degree of similarity. We aim to make the output manageable and meaningful to the user by providing suitable user feedback facilities. However, we permit the user to state the number of tuples desired to be presented from a possibly large output. Unlike approximate retrieval models such as those presented by Zemankova and Kandel [ZEMA 85] and Chawla [CHAW

89a, CHAW 89b] we do not define any threshold or cutoff values to be associated with the fuzzy predicates in a query. However, we achieve the same effect by permitting the user to specify a suitable range of similarity values as part of our feedback process.

The following topics are discussed in detail as part of the approximate querying process :

- propositions to evaluate incomplete valued predicates
- specification of fuzzy linguistic terms
- extensions to the query language
- handling compound queries

2. PROPOSITIONS / STRATEGIES TO EVALUATE INCOMPLETE VALUED PREDICATES

In addition to atomic valued attributes, our data model includes nonatomic values mainly in the form of sequence lists and/or range values. Any query in our model is resolved using three valued logic, resulting in one of the truth values TRUE, FALSE or MAYBE. One or more incomplete valued attributes encountered in the data base could result in a truth value MAYBE. In evaluating one or more predicates of a query, we need to study the effect of the various comparison operations on incomplete valued attributes. In addition, we need to investigate the effect of logical operations.

2.1 RANGE ATTRIBUTE VALUES

Consider two attribute values 'a' and 'b'. These could either be single valued attributes or range valued attributes. If either 'a' or 'b' is single valued then $a_{min} = a_{max} = a$ and $b_{min} = b_{max} = b$.

1) Equality test:

a=b is TRUE — if a and b are single valued attributes with the same value.

a=b is MAYBE — if either a or b or both are range valued attributes and the intersection of a and b is not null.

a=b is FALSE — if a and b are single valued attributes with different values.
if either a or b or both are range valued attributes and the intersection of a and b is null.

2) Greater than test:

a>b is TRUE — if a and b are single valued attributes and a is greater than b.
if $a_{min} > b_{max}$ and the intersection of a and b is null where a_{min} is the lower

bound for value a and b_{max} is the upper bound for value b.

a>b is MAYBE — if either a or b or both are range valued attributes and the intersection of a and b is not null and a_{max} NOT = b_{min}

a>b is FALSE — if a and b are single valued attributes and a is less than or equal to b.

if $a_{max} <= b_{min}$ where a_{max} is the upper bound for value a and b_{min} is the lower bound for value b.

3) Less than test:

a<b is TRUE — if a and b are single valued attributes and a is less than b.

if $a_{max} < b_{min}$ and the intersection of a and b is null where a_{max} is the upper bound for value a and b_{min} is the lower bound for value b. if either a or b is single valued then $a_{min} = a_{max} = a$ and $b_{min} = b_{max} = b$.

a<b is MAYBE — if either a or b or both are range valued

attributes and the intersection of a and b is not null and a_{min} NOT = b_{max} .

a<b is FALSE — if a and b are single valued attributes and a is greater than or equal to b.
if $a_{min} >= b_{max}$ where a_{min} is the lower bound for value a and b_{max} is the upper bound for value b.

2.2 SEQUENCE LIST VALUES

With some amount of preprocessing, the elements of a sequence list can be arranged in ascending order. The equality, greater than, and less than tests can be evaluated as indicated above where the min and max values of the set refer to the first and the last values of the set.

2.3 SPECIAL PROCESSING OF ORed PREDICATES FOR SEQUENCE LISTS

Some strategy needs to be developed to handle situations where two or more predicates of the same attribute are connected by the logical connective OR in a query. An example will highlight the requirement.

If a query requests employees where city of residence is Atlanta

or Columbia then an attribute with a value as a sequence list, such as {Columbia/Atlanta} will in terms of three-valued logic be resolved as

MAYBE OR MAYBE

(in case of Columbia) (in case of Atlanta)

In terms of three-valued logic, the truth value of 'MAYBE OR MAYBE' will result in the truth value MAYBE. But from a real world interpretation the attribute value {Columbia/Atlanta} should have satisfied this query, as the sequence list attribute value of city implies that the employee could be either from Columbia or Atlanta. This implies that some additional checking will have to be done for such queries whenever we encounter attribute values expressed as sequence lists.

3. SPECIFICATION OF FUZZY LINGUISTIC TERMS

3.1 DEFINING SIMPLE TERMS

We primarily provide approximate querying of the database by suitable definition of fuzzy linguistic terms. As described in Tahani [TAHA 77], every fuzzy linguistic term t is defined using a compatibility function μ : U--> [0,1] which associates with each object u, an element in the Universe of our discourse U, its compatibility with t. Applying this definition to our model it means that the various values which an attribute can take from its

domain will be suitably mapped by the compatibility function into the range [0,1], indicating its compatibility with the definition of a term defining the mapping function. The distances between the various attribute values in a numeric domain reflect the degree of similarity/dissimilarity between them. When these various attribute values are mapped by the compatibility function into the domain [0,1], then these distances are still reflected by the corresponding varying degrees of membership. Attributes deriving values from domains not belonging to a linear scale do not permit the definition of fuzzy linguistic terms. Each value in such a domain needs to be first mapped on to a numeric domain; such a transformation may not be easy.

In our model we expect the users to define a few (typically one or two) fuzzy linguistic terms for numeric attributes. As indicated in the models described by Zemankova [ZEMA 85], Tahani [TAHA 77] and Chawla [CHAW 89], if two terms are defined as stated above to represent the two ends of the scale then they are related as shown below :

μ(term2) = 1 - μ(term1) = μ(NOT term1)

We wish to state that there is no restriction to the number of fuzzy linguistic terms that can be defined for a given attribute in our model. Our model will in fact permit any number of fuzzy terms for a given attribute. If a large number of fuzzy terms are defined for a given attribute, then the likelihood of the user getting confused in selecting an appropriate term is high. By

defining fewer terms and providing suitable feedback options, we hope to avoid the above problem and still aim to retrieve the desired tuples. An example of a fuzzy linguistic term is defined below.

Example

If one of the attributes in our model is age, then it is possible that the users may be interested in querying the database with the fuzzy linguistic terms *old* or *young*. If we consider an employee database for an organization, then typical age attributes could be between 65 and 20 years. We could define a fuzzy linguistic term *old* as follows:

a) for age > 60 , μ(old) = 1

b) for age between 51 and 60,

μ(old) = 1 - (0.1 * (60 - age)/1).

c) for age < 51, μ(old) = 0

This implies that for a decrease in age from 60 by one year, the certainity of membership in the fuzzy set for that attribute value decreases by 0.1.

Based on the above example we could define a fuzzy term *young* as

μ(young) = 1 - μ(old).

For simplicity's sake the fuzzy term *young* is defined in terms of the fuzzy term *old* . An alternative definition of *young* can be stated as indicated below:

a) for age < 25 , μ(young) = 1

b) for age between 25 and 34,

μ(young) = 1 - (0.1 * (age - 25)/1).

c) for age > 34 , μ(young) = 0

3.2 LINGUISTIC HEDGES

Based on the above example, it is clear that we could possibly define a variety of fuzzy terms for a permitted attribute. We could have for attribute age another term defined as *medium age* or extend the definition of *old* and *young* by linguistic hedges such as *very old*, *approximately old*, *very young* etc. However, we refrain from defining too many linguistic terms and extensions of definitions by using linguistic hedges. The definition of many such terms may confuse the user rather than help him/her. We hope to circumvent the above requirement by providing suitable user feedback options. For example, a user needing to determine *old* employees will be provided options to choose a suitable range of employees, which would have otherwise been obtained by defining a term *medium age* or *not so old*.. One important thing we must bear in mind is that in our design we are not specifying any threshold or cutoff limits and permit selection and viewing of all tuples irrespective of their degrees of membership in the fuzzy set.

3.3 DEFINING COMPLEX FUZZY LINGUISTIC TERMS

We can permit the definition of complex fuzzy linguistic terms which are dependent on two or more predefined fuzzy terms. The new fuzzy term can be expressed as a combination of the component fuzzy terms with suitable weighting factors to indicate their degree of importance.

We can formally define a complex fuzzy term F as

$$F = \frac{\sum w_i \mu_i}{\sum w_i}$$

w_i = importance of attribute (weight) in the definition of the new term.

μ_i = membership value of the attribute in the fuzzy set as defined by the component fuzzy term.

Example : Consider the EMPLOYEE relation as shown below:

EMPLOYEE

EMP-NO	AGE	YEARS-WORKED
001	34	8
002	50	2
003	45	14

If we are interested in looking for *Senior* employees, then the term *Senior* could be defined as

$$Senior = \frac{(1.5 * Tenure + 1 * Old)}{(1.5 + 1)}$$

Where the fuzzy term *Tenure* is defined as

= 1, If years of service > 10

= 1 - 0.1 * (10 - years of service) for all other cases

and the fuzzy term *Old* is defined as

= 1, If age > 60

= 1 - 0.1 * (60 - age) / 1, if age between 51 and 60

= 0, if age < 51

In our model the definition of this complex term in terms of its

component terms is decided by the data base administrator. As our model is mainly oriented towards casual users, it does not make much sense to permit users the flexibility of manipulating the weights of the various component terms. Even for a fairly experienced user of the data base, it may not be easy to perceive the nature of the output when querying with fuzzy terms, thus making it extremely difficult to visualize the effect of changing the term weights on the final result.

4. EXTENSIONS TO THE QUERY LANGUAGE

4.1 INTRODUCTION OF AN APPROXIMATE OPERATOR

4.1.1 DEFINITION OF THE APPROXIMATE OPERATOR

We introduce an approximate operator ' ≈ ' to permit us to define fuzzy predicates as part of our query. Since our fuzzy querying will be restricted to numeric domains, the ' ≈ ' operator will be used only for numeric data. The ' ≈ ' operator can be a part of a fuzzy predicate in one of two ways:

TYPE 1) attribute ≈ fuzzy linguistic term
e.g. age ≈ old

TYPE 2) (a) attribute ≈ numeric constant
e.g. age ≈ 30

(b) attribute ≈ attribute instance
e.g. employee age ≈ age of Tom

The TYPE (2) {(a) and (b)} definitions are supposed to serve the same functionality as the CLOSE_TO operator defined in systems such as that described by Chawla [CHAW 89], where tuples which are close to a desired numeric value of an attribute need to be retrieved.

Strategy for resolving TYPE (1) predicates :

The system looks in the catalog to determine whether the fuzzy linguistic term used is a valid one for the specified attribute. The attribute value of each tuple is mapped by the compatibility function defined for the term into the domain [0,1].

Strategy for resolving TYPE (2) predicates :

Case (a) :

The objective of this kind of search is to present to the user tuples with attribute values close to the desired value. The compatibility function could be defined for an attribute age such as

1 - (m * Abs (age - X)) ; for (m * Abs (age - X)) > 0

0 ; otherwise

where m is some kind of variance measure indicator.

If X = 30 as in our above example it becomes

1 - (m * Abs (age - 30)).

The appropriate value of 'm' for a given attribute is decided by the Database administrator and a suitable selection of 'm' would typically depend on the spread of the data values for the attribute.

If the values for the attribute are spread further apart, a lower value of 'm' could be chosen to get a reasonable mapping of the various attribute values into the domain [0,1]. On the other hand if the values are repetitive and close to each other, it would be better to have a higher value of 'm'. In any case 'm' cannot be greater than 1. In order to prevent the function from containing negative values, any computation resulting in a negative value is viewed as zero for the attribute value in consideration.

An alternative approach is to permit the system to pick an 'm' value for a given attribute based on the distribution of its values in the database. We could then permit this chosen value of 'm' to be overridden by the data base administrator if desired.

Case (b) :
A strategy similar to case (a) is adopted. However, in this case the value of the attribute instance needs to be evaluated first.

4.1.2 USE OF ' ≈ ' OPERATOR ON INCOMPLETE DATA

The response to any query in general could be a combination of tuples containing both crisp and incomplete data. It would be reasonable to assume that the user will be more interested in tuples containing crisp values of the desired attribute rather than those containing incomplete values. The incomplete data involved would be numeric and could be either range values or sequence

lists. On this line of reasoning we interpret the various cases of fuzzy querying predicates as indicated below.

For a fuzzy predicate, e.g., age $\approx$ old

If a particular attribute has a value such as {52,55} then we would try to map this range into our fuzzy domain of [0,1] by applying the compatibility function on the lower and upper bounds and selecting the value which indicates the lower value of similarity. This type of worst case analysis of the incomplete valued attribute enables us to map any crisp attribute value in the database, say 53, to a higher degree of similarity in our fuzzy set. In case an attribute value is a sequence list such as {52/53/54} a similar approach is adopted.

For a fuzzy predicate, e.g., age $\approx$ 30

The same approach as indicated in the example above is adopted.

We would expect most compatibility functions to be monotonic. For non monotonic compatibility functions, one of the two approaches indicated below can be adopted :

- It is reasonable to assume that our range values or sequence lists will be of limited length. If the user cannot narrow down an attribute value to be within a limited range or a limited length sequence list, it would have been better to have entered the attribute value as undefined. For reasonable length range values or sequence lists, it would be possible to evaluate the compatibility function for various possible

values and pick the value which indicates the lowest degree of similarity.

- An alternative approach is to consider piecewise segments of the range or sequence list options and evaluate the fuzzy membership values for the lower and upper bounds for each segment. This will then help us determine an overall match exhibiting the least value of similarity with the stated predicate. This approach may involve less computation using the compatibility function as compared to the above mentioned approach but may not guarantee whether the least similarity value chosen is actually precise.

4.2 QUERYING ONLY ON CRISP DATA OF THE DATA BASE

Though our model permits incomplete values to be handled in the database, it would be desirable to permit the user to query only on crisp values if required. This option can be provided irrespective of whether the query is a crisp query or contains fuzzy predicates. We permit the user to include the keyword exact as part of the query to indicate that only exact values are desired. Two examples of querying on attribute age in a employee database are indicated below.

i) Select Name, Age, Department

from Main

where Age > 25 (Exact)

ii) Select Name, Age, Department

from Main

where Age ≈ old (Exact)

Explanation: The exact keyword implies that only crisp values of the attributes are tested (indicated in the where clause) and further only those tuples containing crisp values of the selected attributes qualify for output.

4.3 QUERYING ON DATA VALUES BASED ON THEIR STATUS

Our DBMS recognizes a data attribute to be one of C (crisp), R (range), S (sequence list) and U (undefined) value. There may be instances when a user needs to retrieve data values pertaining to a particular status or one of a few statuses. We permit the user to additionally indicate as part of the query the status of data values desired. This is an optional facility; by default, all data values, whether complete or incomplete, that satisfy the query will be retrieved. The examples indicated below highlight some of the possible options.

Select Name, Age (Status = R, S)

from Main

Such a query retrieves tuples for all employees which have age as incomplete values either as range or sequence lists. This could be used to decide whether suitable action should be taken for these cases to ascertain a crisp value for age.

Select Name, Age (Status = U)

from Main

This query results in tuples having attribute age with no values defined. The user can probably try to ascertain some more information about these employees and try to get crisp values of age or represent age as a incomplete value wherever possible.

As a special case, if we query on the employee age as

Select Name (Status = C), Age (Status = C)

from Main

then the output generated is the same as that resulting from the query

Select Name, Age

from Main (Exact)

We provide querying of the database to retrieve tuples with only exact data by using the keyword (Exact) and feel that this option is more appropriate to users who are not generally aware of the various indicators of data values and are not concerned with incomplete attribute values. It may be noted that the status clause needs to be indicated along with the attribute for which data values pertaining to those statuses are needed. All other attributes mentioned in the query for which no such clause is indicated may contain an attribute value of any status in the output.

An alternative approach is to provide the users with logical views containing only crisp data. This way the users who normally deal only with crisp data need not be burdened with the knowledge of the various extensions to the query language.

For the main relation in our employee database, we can define a view to enable the user to access only crisp data as:

```
Create View Main-Crisp
As Select Emp-No, Name, Age, Department, Sex
From Main (Exact)
```

4.4 USE OF JOIN OPERATION ON INCOMPLETE VALUED DATA

Whenever two attribute values from two different relations are compared during a join operation, the two corresponding tuples

qualify for the output only if the result of the comparison operation is TRUE. There may be instances when a user needs the tuples to appear in the output relation even if the result of the comparison operation is a MAYBE. The following example will highlight this requirement :

Suppose there are two relations, Employee and Bonus, as indicated below.

EMPLOYEE

EMP-NO	NAME	DEPARTMENT
001	Tom	D10
002	Bill	{D10/D11}

BONUS

DEPT.	EMP-BONUS
D10	500
D11	750

If we wish to determine the amount of bonus for each employee then the following two outputs, indicated by relation EMPLOYEE-BONUS, may result, depending on the two types of

queries in case 1 and case 2.

Case 1

Select *

From Employee, Bonus

where Employee. Department = Bonus. Department

EMPLOYEE-BONUS

EMP-NO	NAME	DEPARTMENT	EMP-BONUS
001	Tom	D10	500

It is apparent that details for employee 002, (Bill) do not appear as the comparison of department value {D10/D11} from EMPLOYEE with the values D10 and D11 of relation BONUS results in MAYBE.

Case 2

Select *

From Employee, Bonus

where Employee. Department = Bonus. Department (Maybe)

EMPLOYEE-BONUS

EMP-NO	NAME	DEPARTMENT	DEPT	EMP-BONUS
001	Tom	D10	D10	500
002	Bill	{D10, D11}	D10	500
002	Bill	{D10, D11}	D11	750

The MAYBE extension provided helps us to gather more information in the output and to analyze a given problem in more detail even though the information is incomplete. As compared to the case 1 example, the bonus information of Bill in case 2 is indicated at least tentatively and may thus help the user in his/her decision making process. It is however apparent that we are unable to do a natural join in this case.

5. HANDLING COMPOUND QUERIES

In general queries may be simple or compound and predicates in a query may be crisp or fuzzy. Whether the query is a simple query or a compound query, if it contains only crisp predicates the query evaluation process is straightforward and tuples with attributes which match the predicates in the query to TRUE, qualify for the output. Whenever a query contains one or more fuzzy predicates,

then resolving those predicates does not result in one or more exact matches of the attribute values in the various tuples, but instead results in a selection of tuples which match the fuzzy predicate to varying degrees of similarity. In the case of simple queries containing fuzzy predicates the user is presented the various tuples (in our case all tuples) which satisfy the predicates in descending order of similarity. To be able to handle compound queries which may contain both crisp and fuzzy predicates, we need to develop a strategy to present the output in an order most beneficial to the user. We adopt a technique based on that described in Tahani [TAHA 77].

The condition in a compound query could be stated as

..... $predicate_1$ AND/OR $predicate_2$ AND/OR $predicate_3$ .....

(formal definition of the query stated in appendix)

We consider a query having two predicates as part of the condition clause and connected with the logical connectors AND/OR. The treatment we describe below can be extended for any number of predicates. We need to evaluate some kind of tuple compatibility value for the entire query and place the output tuples in descending order of this value. The tuple compatibility value is computed from the various component predicate compatibility values. Crisp predicates always have predicate compatibility values of one or zero, depending on TRUE or FALSE. Fuzzy

predicates have predicate compatibility values in the range [0, 1]. The computation of the tuple compatibility value depends on whether the logical connector connecting the predicates is AND or OR.

If we denote two predicates in a query by $predicate_1$ and $predicate_2$, then the following schemes indicate the evaluation of tuple compatibility value involving logical connectors AND and OR.

AND Operation

$predicate_1$ AND $predicate_2$

If we denote μ as the predicate compatibility value and K as the tuple compatibility value, then in this case

$K = \min(\mu_1, \mu_2)$

If $predicate_1$ is crisp ; $\mu_1 = 1$ or 0

if $predicate_2$ is crisp ; $\mu_2 = 1$ or 0

if either of the two predicates is a fuzzy predicate, then μ represents the membership value of that attribute in the fuzzy set as defined by the compatibility function of the corresponding fuzzy term.

OR Operation

$$\text{predicate}_1 \quad \text{OR} \quad \text{predicate}_2$$

If we denote μ as the predicate compatibility value and K as the tuple compatibility value, then in this case

$$K = \max(\ \mu_1, \mu_2\ )$$

If predicate_1 is crisp ; $\mu_1 = 1$ or 0

if predicate_2 is crisp ; $\mu_2 = 1$ or 0

if either of the two predicates is a fuzzy predicate, then μ represents the membership value of the attribute in the fuzzy set as defined by the compatibility function of the corresponding fuzzy term.

In both the AND and OR cases, when only crisp predicates are involved, the tuple compatibility value is always one or zero. This is in conformity with the normal querying process as the tuples are always retrieved under the identical match condition of TRUE. We may note that this strategy should not be viewed as a constraint on the user, as the user is presented the option of selecting suitable tuples from the output.

6. EXAMPLES OF QUERIES

a) Crisp query

(i) Select Emp-No, Name, Age, Department
From Main, Salary
Where Age > 53 AND
Age < 60 AND
Department = D10 AND
Basic-Salary > 2000

Emp-No	Name	Age	Department
002	Susan	{57,58}	D10
008	Eric	55	D10

Explanation: All four predicates in the query are crisp predicates. Only two tuples are retrieved from our sample database. The first tuple for employee Susan has an incomplete value of age, {57,58}. This value, though incomplete, satisfies the predicate for age in the given query.

(ii) Select Emp-No, Name, Age, Department

From Main, Salary

Where Age > 53 AND Age < 60 AND

Department = D10 AND

Basic-Salary > 2000 (Exact)

Emp-No	Name	Age	Department
008	Eric	55	D10

Explanation: This query is same as the query indicated in (i); the only difference is that the keyword Exact is included. So only crisp values are selected. The first tuple in the output of (i) is missing here for this reason.

b) Fuzzy query

Select Emp-No, Name, Age, Basic-Salary

From Main, Salary

Where Age ≈ Old AND

Basic-Salary ≈ High

RESET RANGE PREDICATE MANAGE

EMP-NO	NAME	AGE	BASIC-SAL	S. I.
001	Kevin	62	5000	1.0
004	Bill	{60,65}	4000	1.0
008	Eric	55	3000	0.9
007	Tom	59	2800	0.8
002	Susan	{57/58}	2500	0.7
010	Mike	31	{1800,2000}	

```
Select Emp-No, Name, Age, Basic-Salary
From Main, Salary
Where Age ≈ Old AND Basic-Salary ≈ High
```

Explanation : There are two fuzzy predicates in this query. Only six tuples are visible in the output window. The query itself is indicated in a separate window at the bottom of the screen. All the tuples in the view set can be viewed by choosing suitable options from the pull down menu. The tuples are listed in the decreasing order of their similarity value, which is indicated adjacent to each tuple.

c) Mixed query

```
Select Emp-No, Name, Basic-Salary, Competence
From Main, Salary, Performance
Where Age ≈ Young          AND
      Department = D10     AND
      Basic-Salary > 2000  AND
      Competence ≈ High
```

RESET RANGE PREDICATE MANAGE

EMP-NO	NAME	BASIC-SALARY	COMPETENCE	S. I.
008	Eric	3000	4	0.7
002	Susan	2500	3	0.6
001	Kevin	5000	4	0.45

```
Select Emp-No, Name, Basic-Salary, Competence
From Main, Salary, Performance
Where Age ≈ Young AND Department = D10 AND
      Basic-Salary > 2000 AND Competence = High
```

Explanation : Three tuples from the view set are visible in the output window. The query itself is indicated in a separate window at the bottom of the screen. All the tuples in the view set can be viewed by choosing suitable options from the pull down menu. The

tuples are listed in the decreasing order of their similarity value, which is indicated adjacent to each tuple.

CHAPTER 5

USER FEEDBACK

1. INTRODUCTION

Our model permits approximate querying of the database mainly by the suitable use of fuzzy linguistic terms. The definition of these terms and their interpretation could be very subjective, and they may provide very little clue to the users about the nature of output they could expect. In addition our model is directed towards the end user community, where users may have to frequently deal with more than one database and/or may not have sufficient knowledge about the stored data values in the database. To handle this problem, we provide the users with some feedback options in order to retrieve a set of responses close to that desired by the user. Unlike other models such as those discussed by Zemankova [ZEMA 85] and Chawla [CHAW 89] we do not expect our fuzzy predicates in a query to have any associated values of possibility or certainty, for the purpose of restriction in the output. Our model instead retrieves all the various values of the database which satisfy the fuzzy predicate to various degrees of similarity, including zero. This approach seems to be reasonable as it will be unfair to expect users to indicate possibility/certainty values for fuzzy predicates whose

interpretation they are unsure of.

As of now our model addresses the feedback issues only for fuzzy queries. We feel that the implementation of feedback for crisp queries may not be difficult but are unsure of its usefulness. The principle underlying our feedback strategy is to provide users with options to selectively retrieve desired tuples from an initially selected output set. Our technique of permitting users to focus on their desired set of tuples from a possible output could possibly be achieved by one of the two approaches indicated below:

- for a given attribute provide users with a larger number of fuzzy linguistic terms. This approach still does not eliminate the problem of the interpretation of the fuzzy terms and does not give any insight to the user as to what he/she can expect when using the fuzzy terms to query the database. It is possible that by trying the various fuzzy terms defined for the attribute and by a process of trial and error, the user could arrive at the desired result.

- a second alternative could be to provide several interpretations of a given fuzzy term and permit each user or group of users to use a specific interpretation of the term. This kind of user model approach can be ineffective at times when the user is looking for something different from the normal case.

After analyzing the above two alternatives, we adopt the user feedback strategy in our model and hope that our users would be able to converge on their desired output from an initially retrieved output. As part of our user feedback design, we discuss three major issues:

- User interface for feedback
- Feedback options and operations
- Integration of the feedback module with normal querying process.

2. USER INTERFACE

We will discuss two possible modes of user interface:

- Windowing environment
- Menu driven interface

2.1 WINDOWING INTERFACE

This interface uses underlying windowing software as part of the querying environment. It is essential that, in addition to the normal keyboard, an auxiliary input device such as a mouse is used. A sample screen is indicated below:

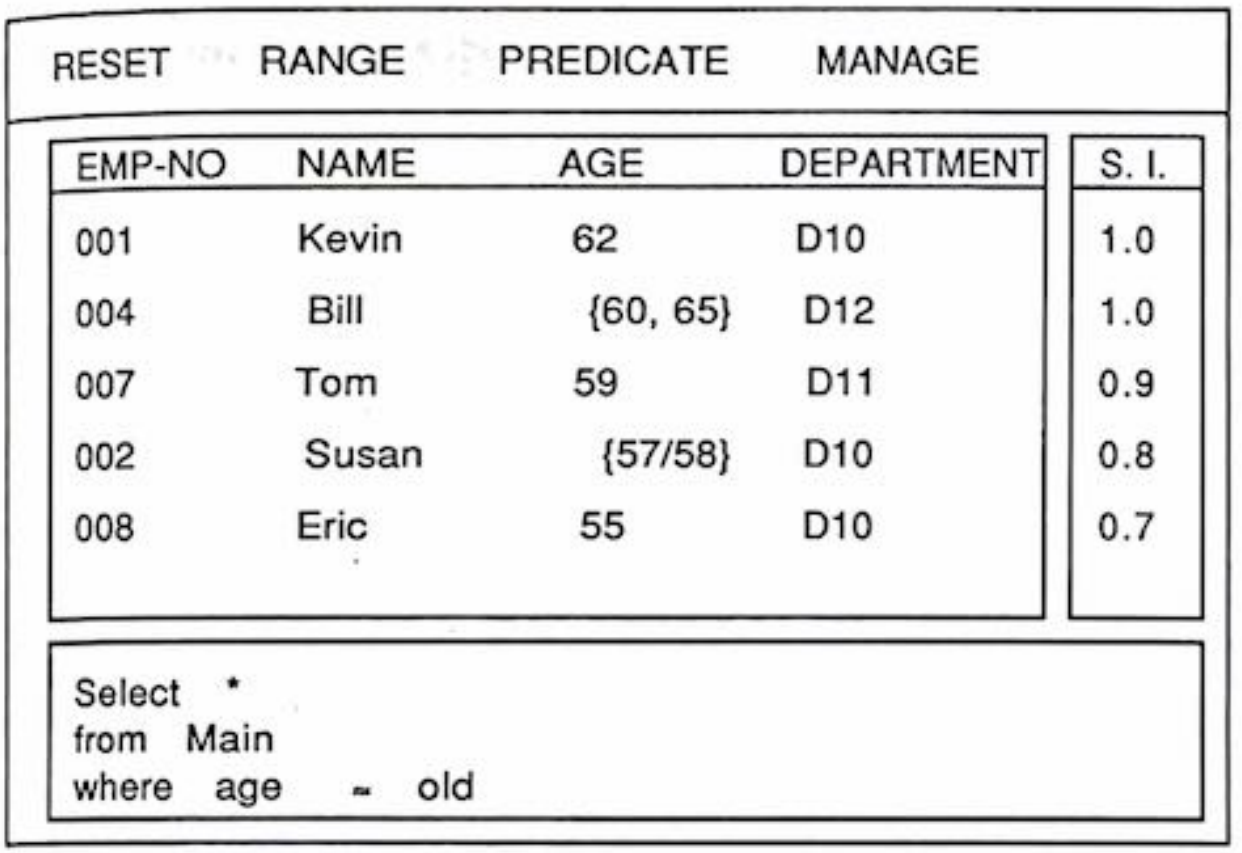

Figure 1.

The screen is normally occupied by three windows. The window in the top left hand portion displays the retrieved tuples as a result of evaluating the user query. The second window in the top right hand portion contains the tuple compatibility value. This window contains one value of the tuple compatibility value for each tuple in the adjacent window. The third window is located in the bottom of the screen and contains the original user query. In addition to the three windows, there is a menu bar located on the top of the screen. The menu bar contains four options, any of which can be selected by the user using a mouse. We will indicate now some of the screens resulting from the various options.

a) Reset: provides no suboptions.

b) Range: On selection, results in a new window as displayed on the screen. Both the beginning and ending values included under 'From' and 'To' respectively are also included in the selection.

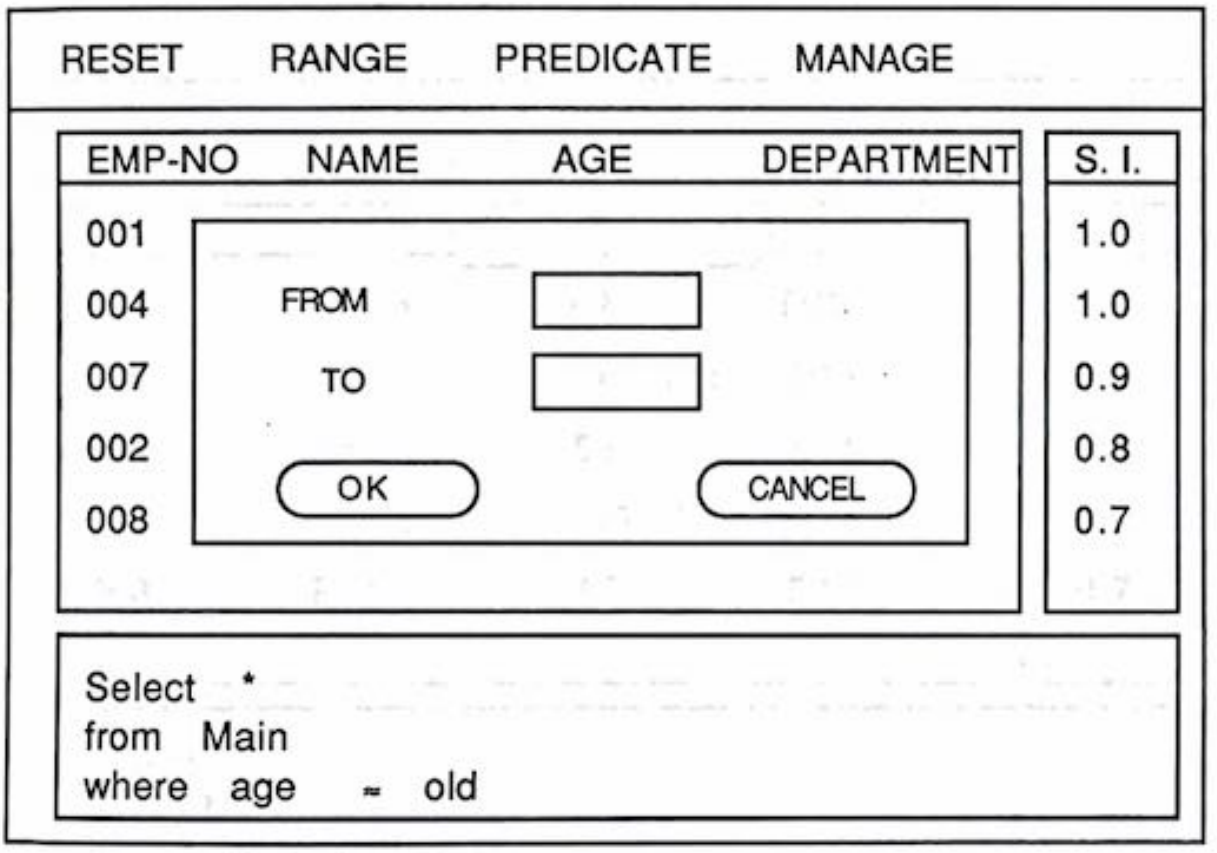

Figure 2.

c) Predicate : Results in the following two screens to be displayed consecutively.

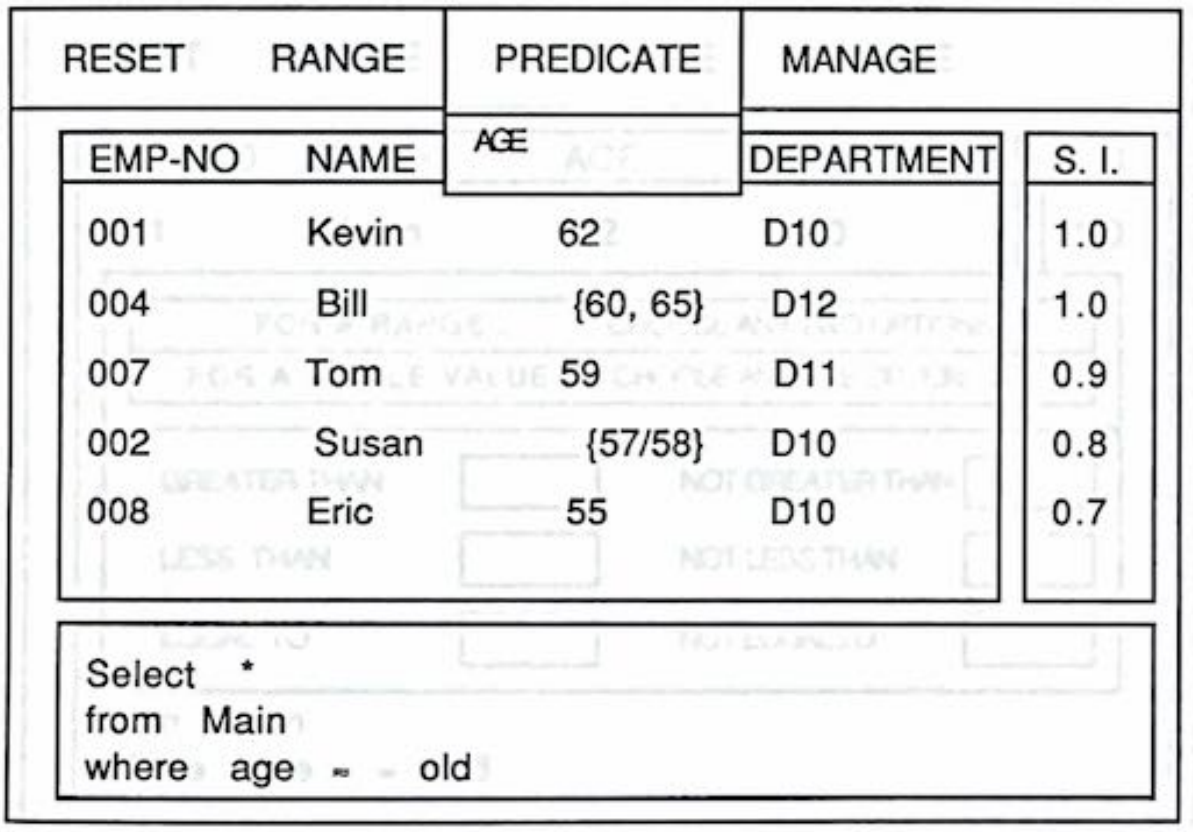

Figure 3.

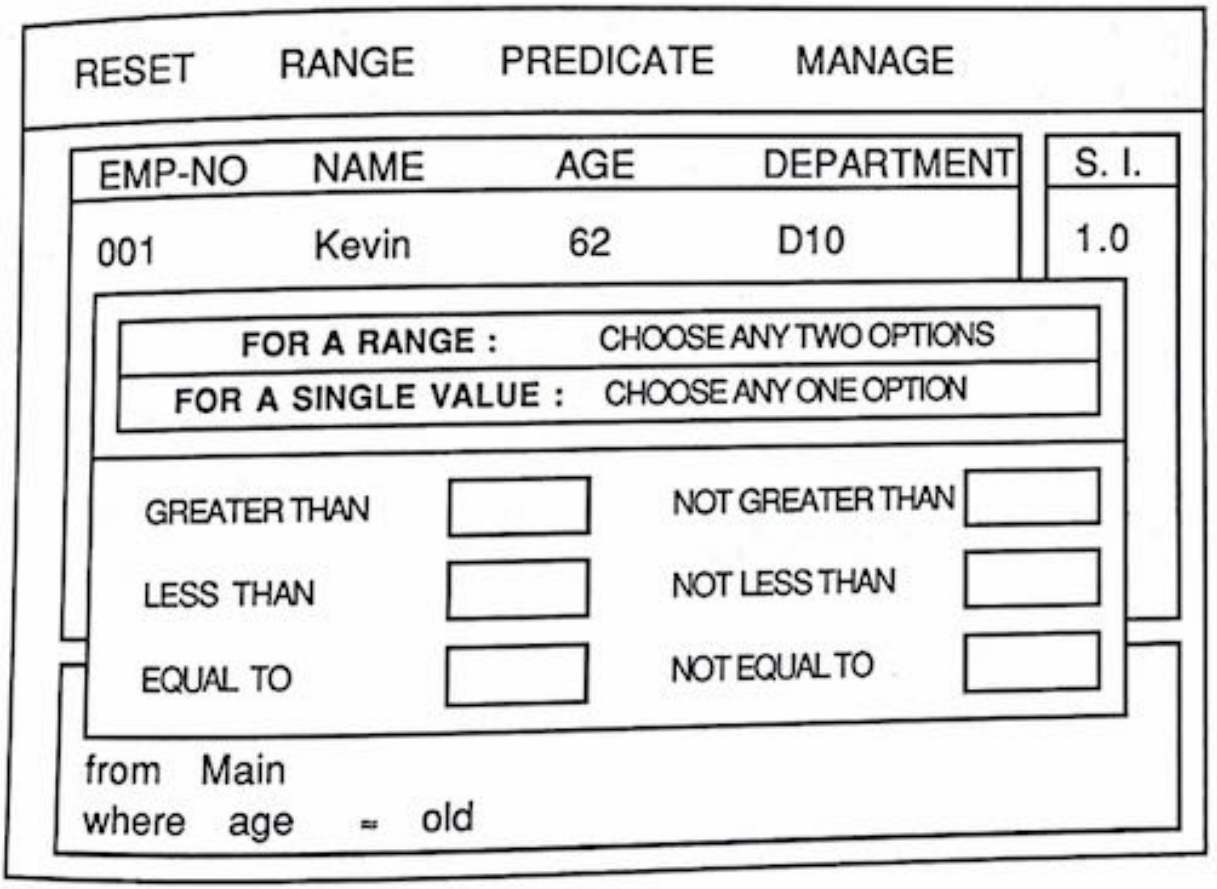
RESET
RANGE
PREDICATE
MANAGE
EMP-NO
NAME
AGE
DEPARTMENT
S. I.
001
Kevin
62
D10
1.0
FOR A RANGE : CHOOSE ANY TWO OPTIONS
FOR A SINGLE VALUE : CHOOSE ANY ONE OPTION
GREATER THAN
NOT GREATER THAN
LESS THAN
NOT LESS THAN
EQUAL TO
NOT EQUAL TO
from Main
where age = old

Figure 4.

d) Manage : Results in the following submenu options to be provided.

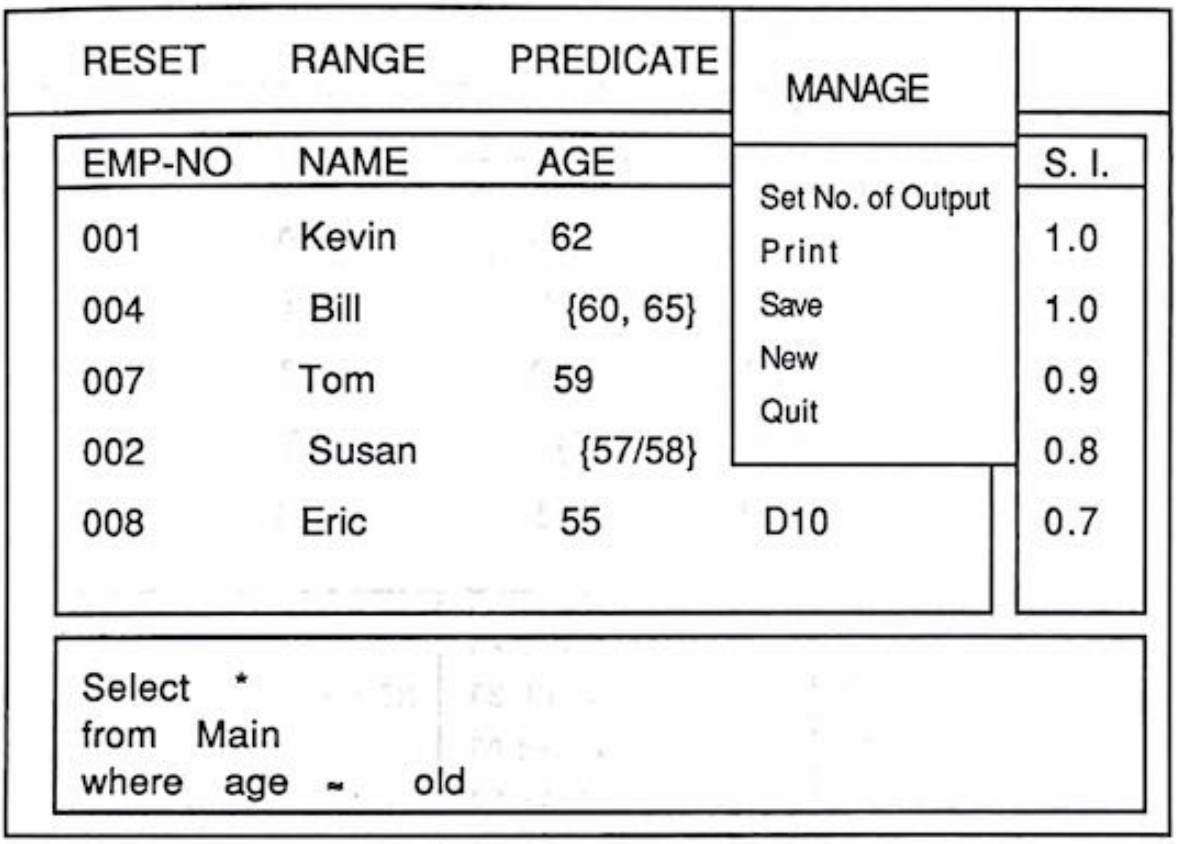

Figure 5.

In addition to the above options, scrolling up/down or across any window, opening and closing a window or adjusting sizes of the windows is provided as part of the normal operations of the underlying windowing software.

2.2 MENU DRIVEN INTERFACE

In a menu driven interface, the screen would appear as shown below.

EMP-NO	NAME	AGE	DEPARTMENT	S. I.
001	Kevin	62	D10	1.0
004	Bill	{60, 65}	D12	1.0
007	Tom	59	D11	0.9
002	Susan	{57/58}	D10	0.8
008	Eric	55	D10	0.7

F1 RESET	F4 SET OUTPUT NO.	F9 PREVIOUS SCREEN
F2 RANGE OF S.I. INDEX	F5 PRINT	F10 NEXT SCREEN
F3 PREDICATE	F6 SHOW QUERY	F11 QUIT
	F7 SAVE	
	F8 NEW	

Figure 6.

where the various 'F's indicate function keys to be used from the keyboard. The range and predicate options and set output no. options could lead to new submenu screens to be presented, and options as discussed above for the windowing environment could be presented.

3. FEEDBACK OPTIONS AND OPERATIONS

There are four major user feedback options available, and some of them are designed to have suboptions. The following options and their functionalities are discussed :

- Reset
- Range
- Predicate
- Manage
  - Set No. of output
  - Print
  - Save
  - New
  - Quit

3.0 GENERAL GUIDELINES

Before discussing the various feedback operations in detail, we would like to highlight some common features pertaining to the output displayed. The number of tuples displayed at any point of time is determined by the window (screen) size or the number of tuples selected for output, whichever is smaller. If there are more tuples selected for output than can be accommodated in the window, the remaining tuples can be viewed by suitably changing the window size and/or scrolling. In case of a screen, as in a menu

driven interface, a suitable option can be selected to see the next set of tuples. We wish to reiterate here that, for a given fuzzy query, all the tuples of the database are retrieved, including even those whose compatibility with the fuzzy definition is zero or very small. The output tuples retrieved are arranged in descending sequence of tuple compatibility value which is determined by the various predicate compatibility values. As the number of tuples retrieved for display can be very large, it will be impractical and unwieldy to display all of them. The initial number of tuples to be displayed as a result of a query can be set to some desired value by the user. This can typically be done by the user at the beginning of any query session. For sake of ease of explanation, we wish to refer to all tuples qualified for output at any instant as the output set and the number of tuples the user can display at any point of time as the view set. The size of the view set can be modified by the user.

Example: Number of tuples initially qualified for output is 300. This is the output set. The user could have set the number of output tuples to be displayed as 50; this is the view set.

3.1 RESET

The idea of user feedback is that the user can see a given set of tuples and may choose to refine the output displayed to suit his/her needs. By choosing one or more suitable operations, the user could eventually narrow down on a desired set of tuples.

Every feedback operation with the exception of reset is done on the previous output set. The reset operation always retrieves the initial output set, the one that is obtained as a result of the original query and prior to any feedback operations. An example will highlight this requirement.

Example: Suppose the initial output set consists of 150 tuples. The user, after viewing some of these tuples, decides to refine this output set by using a predicate operation, such as say age > 55 and age < 60. This operation of restriction is done only on the 150 tuples and could result in a new output set of 110 tuples. The user could refine this further by using a range operation to restrict the tuple compatibility value to greater than 0.85. This could further lead to a new output set of 70 tuples. This operation is done only on the previous output set, i.e. on the 110 tuples. If now the user is uninterested in the current output set of tuples, it is possible to revert back to the initial output set by using reset and to start refining the output once again. This process can be repeated again and again until the user gets his/her desired set of tuples.

3.2 RANGE

Each of the tuples displayed in our view set is displayed along with a tuple compatibility value and the tuples are displayed in the descending order of this value. After viewing the initially retrieved view set of tuples, the user could decide to restrict the output set to be within a range of tuple compatibility values. This

range could be specified as indicated in Figure 2 .

Example: Consider a fuzzy query such as

Select *

From Main

When Age ≈ old.

The output set = 400 tuples and view set = 30 tuples. After viewing some tuples in the view set, the user could use the range operation to specify tuple compatibility index to be from 0.8 to 0.7. This would provide a new output set of 80 tuples.

3.3 PREDICATE

If the user does not want to restrict the output set initially obtained by using the Range operation, then this operation is an alternative. Users could use this option to restrict the output set by choosing one of the attributes shown in the submenu in Figure 3. The attributes shown in the submenu are only those for which fuzzy predicates were specified in the initial query. On selection of any attribute, the user is provided with options as indicated in Figure 4. A single option can be chosen to define a predicate or two options can be chosen to indicate a range predicate. It must be noted here that at present this kind of restriction of output tuples can be done on the output set by choosing only one attribute at a time. It is however possible to do a sequence of such operations on more than one attribute or on the same attribute.

Example: Consider the fuzzy query as indicated below :

Select *

From Main

Where Age ≈ old.

The output set contains 400 tuples, and the view set contains 30 tuples. After viewing some tuples in the view set the user could use the predicate operation to further refine the output set. Here the submenu for predicate contains only one attribute, age, as this is the only attribute which is fuzzily queried.

Suitable selection of an age range, say from 47 to 55, in Figure 4 could possibly lead to an identical output as obtained from the Range operation.

3.4 MANAGE

As the name indicates this option is used to do some miscellaneous management tasks. (See Figure 5)

3.4.1 SET OUTPUT NO.

This option is used to set the number of output tuples to be displayed at any point of time from the output set. This implies that this operation determines the number of tuples in the view set. This could be done at any point of time and is a facility provided to enable the user to view more or fewer tuples from an

output set.

3.4.2 PRINT

This option is used to print the tuples contained in the output set.

3.4.3 SAVE

This option permits the user to save his/her query and output set into a file. This file could be saved in some temporary area of the data base and could be accessed by the user for later use.

3.4.4 NEW

This option is used to start a fresh query from the beginning.

3.4.5 QUIT

This option is used to exit from the query mode.

4. INTEGRATION OF THE FEEDBACK MODULE WITH THE NORMAL QUERYING PROCESS

On invoking the query processing software, the user is presented a screen to enter the initial query. We need to consider the following cases.

Case of crisp queries : In the case of windowing environments, the user is presented with a window of selected responses. In addition, a separate window is presented at the bottom to show the query. The options presented are PRINT, SAVE, NEW and QUIT. In the case of a menu based interface, the responses are shown on the screen, and the user is presented the options as indicated above. In addition the user is also presented options to advance to the next screen and also to see the initial query.

Case of fuzzy queries : After the initial query is entered, the user is presented with the screen as indicated in Figure 1 or in Figure 6 corresponding to windowing or menu driven interfaces.

In both the cases discussed above, selecting the NEW option results in getting back to the initial query processing environment.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

1. INTRODUCTION

In this final chapter of our thesis, we present some conclusions on our data model and draw some comparisons with other related systems. We also indicate some avenues of future research, which could probably result in a more enhanced and efficient model. The material is presented in the following sections:

- Future directions of work
- Comparisons with related work
- Conclusions.

2. FUTURE DIRECTIONS OF WORK

The following topics are discussed in this section:

- representation and handling of other forms of incomplete data
- storage and display related issues
- query processing and optimization issues
- user feedback related issues
- update of the various types of data values handled

A) *Representation and handling other forms of incomplete data:* We have restricted our attention to three major types of nonfuzzy incomplete data, mainly from the point of view of simplicity of design. It would be interesting to explore the possibility of representing other forms of incomplete data, both nonfuzzy and fuzzy. Of particular interest would be to represent fuzzy incomplete data as indicated in Zemankova [ZEMA 85] and to study the effect of such data on our system behavior. It would also be worthwhile to consider similarities for nominal data and even nonnumeric ordinal data.

B) *Storage and display related issues:* The physical model of our data base design has not been investigated. Work needs to be done to identify suitable and efficient means of storing our relations, particularly in view of the fact that our relations could contain nonatomic attribute values. Investigation also needs to be done regarding the presentation of such data for display purposes. Some analysis has to be done to present the various tuples in a manageable and appealing manner.

C) *Query processing and optimization issues:* The underlying query processing needs to be studied and suitable changes have to be made to accommodate the various extensions suggested in the query language. As indicated in Chapter 4, some special query processing strategy needs to be formulated to handle queries

containing ORed predicates in order to correctly resolve sequence list values. It has to be studied whether some change needs to be done at the query preprocessor level or during actual execution of the query to handle this problem. Significant work also needs to be done regarding the optimization of queries in our model as approximate queries are involved.

D) *User feedback related issues:* As our model includes an interface where user response is solicited as a means of feedback, it would be desirable to have an efficient and user friendly user interface. Alternative options could be investigated to improve on the current feedback strategy discussed. In addition, as indicated in Chapter 5, work needs to be done to study the effect of feedback on crisp queries.

E) *Strategies for proper update of the various types of data values:* Since our model handles crisp as well as different types of incomplete data, work needs to be done regarding the update of data values from one type to another. It needs to be seen whether any restrictions apply in changing data values of any particular type to another and if so under what conditions. Permitting incomplete values such as range and sequence lists to be entered in the data base presents a relaxed view of the data base for entry purposes. This may however necessitate some investigation of security and update issues.

3. COMPARISONS WITH RELATED WORK

The issue of approximate retrieval in terms of approximate data and approximate querying has been addressed by a number of data models. The depth in the study involved and the diverse nature of approaches has resulted in a fairly large number of models. Only a few models discuss both the issues of approximate data and approximate querying (mainly under the fuzzy framework), whereas a significant number discuss the issue of approximate querying involving only crisp data. Some models have discussed the issue of approximate data alone without considering the approximate querying aspects.

Our model addresses the problem of approximate retrieval from a perspective of end user ease and differs from the existing approaches in a number of ways.

Our model handles certain forms of incomplete data and also permits approximate querying of the data base, by using fuzzy linguistic terms. Models proposed by Codd [CODD 86], Grant [GRAN 79] and Lipski [LIPS 79] discuss the effect of handling only incomplete data and do not address the problem of approximate querying of the data base. Fuzzy models such as those presented by Zemankova and Kandel [ZEMA 85] and Buckles and Petry [BUCK 84] address both the issues of incomplete data and approximate

querying, but the incomplete data considered is only of the fuzzy form. Tahani [TAHA 77] and Kacprzyk and Ziolkowski [KACP 86] discuss approximate querying by using fuzzy linguistic terms, but do not address the issue of incomplete data. Chawla [CHAW 89] presents a detailed approach to approximate querying of the database using both fuzzy and non fuzzy concepts, but presents a very limited treatment of a particular type of fuzzy representation of approximate data. Shen [SHEN 88] presents a model suitable as the basis for a front end query processor and does not discuss approximate querying issues. A large number of nonfuzzy models such as those presented by Ichikawa and Hirakawa [ICHI 86], McClelland and Trueblood and Eastman [McCL 88], and Motro [MOTR 88] address only the problem of approximate retrieval and do not discuss the handling of approximate data.

In our model we provide for approximate querying of the data base by using fuzzy linguistic terms. The basic notion of defining the various fuzzy terms and strategies for resolving fuzzy queries involving logical operations are similar to those indicated in Tahani [TAHA 77], Zemankova and Kandel [ZEMA 85] and Chawla [CHAW 89]. We however differ from these models by providing a user feedback option in the case of fuzzy queries. We feel that this option will permit users to have a very flexible interpretation of the fuzzy terms and aid them in getting a more meaningful output. From this viewpoint, our model also differs from work discussed by Motro [MOTR 86] and Williams [WILL 84] where the idea is to

provide friendly and flexible options to the user for the purpose of querying in general.

4. CONCLUSIONS

We have presented a data model to address the issue of approximate retrieval in a data base including incomplete data. Our model is based on the relational data model, and we have attempted to keep the deviation from the relational framework as small as possible.

Our model, in addition to handling crisp data, is able to handle incomplete data in the form of sequence lists, range values and unknown values. Both crisp and approximate queries can be handled. We have tried to provide minimal extension to a language like SQL to facilitate our fuzzy querying. Wherever possible suitable alternative schemes are indicated for achieving a desired functionality.

Though prior models have discussed the problem of approximate retrieval in a data base using fuzzy linguistic terms, our model presents a different approach to the problem by incorporating user feedback. We feel that this could be a key issue in the approach and aim to help the user to get a better understanding of the fuzzy terms involved and thus serve to provide a more meaningful output. Another key issue we have addressed is to incorporate

some common forms of incomplete data in our model. To handle incomplete data, we have suggested a simple scheme of representation, investigated the effect on the relational model design requirements and also indicated some propositions for their treatment during query resolution.

We do not claim that the approach we have indicated in our data model is the most suitable one, but rather feel that this could serve as a good starting point to initiate further work in this area.

Emp-No	Name	Age		Department		Sex
001	Kevin	C	42	C	D10	M
002	Susan	S	{37,38}	C	D10	F
004	Bill	R	{40, 41}	C	D12	M
007	Tom	C	39	C	D11	M
008	Eric	C	55	C	D10	M
010	Mike	C	31	S	{D10,D11}	M
...						

APPENDIX I

Our Employee database consists of the following three relations :

MAIN : Contains employee details such as Emp-No, Name, Age, Department and Sex. Emp-No is the key of the relation.

Emp-No	Name	Age		Department		Sex
001	Kevin	C	62	C	D10	M
002	Susan	S	{57/58}	C	D10	F
004	Bill	R	{60, 65}	C	D12	M
007	Tom	C	59	C	D11	M
008	Eric	C	55	C	D10	M
010	Mike	C	31	S	{D10/D11}	M
.....						

SALARY : Contains details such as Emp-No, Basic-Salary, Allowances and Tax-Deduction. Emp-No is the key of the relation.

Emp-No	Basic-Salary		Allowances		Tax-Deduction	
001	C	5000	C	0	C	700
002	C	2500	C	200	C	300
004	C	4000	C	0	C	500
007	C	2800	C	240	C	350
008	C	3000	C	250	C	350
010	R	{1800,2000}	R	{110,150}	C	150
.....						

PERFORMANCE : This relation stores attributes regarding the employee's performance on the job. Emp-No is the key of the relation.

Emp-No	Mgr-No		Competence		Motivation	
001	C	075	C	4	C	3
002	C	075	C	3	R	{3,4}
004	C	099	C	4	C	4
007	C	045	C	3	C	3
008	C	075	C	4	C	4
010	S	{075,045}	U		U	
.....						

APPENDIX II

EXTENSIONS TO THE DDL AND DML STATEMENTS OF SQL

Notation: { }* : Zero or more occurences of the items enclosed

[ ] : Items enclosed are optional

DDL

CREATE TABLE rel-name

(attribute-defn {, attribute-defn}*)

attribute-defn := attribute-name, data-type [NOT NULL]

[{, fuzzy-term}*]

DML

SET OUTPUT TO n

SELECT select-clause

FROM rel-name {, rel-name}*

[WHERE comparison]

[(EXACT)]

select-clause := * | item-list

item-list := item [(Status = Ind)] {, item [(Status = Ind)]}*

ind := ind-val {, ind-val}*

ind-val := C | U | S | R

comparison := comparison relop comparison | (comparison) |
attribute-name operator value | notopr comparison

operator := <> | < | <= | > | >= | ≈

value := constant | attribute-name

relop := AND | OR

notopr := NOT

BIBLIOGRAPHY

[BUCK 84] Buckles, B. P. and F. E. Petry, "Extending the Fuzzy Database with Fuzzy Numbers," **Information Sciences,** 34, 1984, pp 145-155.

[CHAW 89a] Chawla, D. , "A Data Model for Approximate Retrieval in Databases," MS Thesis, Dept. of Computer Science, University of South Carolina, Columbia, SC 29208, June 1989.

[CHAW 89b] Chawla, D. and Eastman, C. M., "An Approach to Approximate Retrieval in Databases," **Proceedings of the 4th International Symposium in Methodologies for Intelligent Systems,** October 1989, Charlotte, North Carolina, pp 69-74.

[CODD 86] Codd, E. F. , "Missing Information (Applicable and Inapplicable) in Relational Databases," **SIGMOD Record,** Vol. 15, No. 4, December 1986, pp 55-78.

[CODD 87] Codd, E. F. , "More Commentary on Missing Information in Relational Databases (Applicable and Inapplicable Information)," **SIGMOD Record,** Vol. 16, No. 1, March 1987, pp 42-50.

[EAST 87] Eastman, C. M. , "Approaches to Approximate Retrieval in Database Management Systems," **Proceedings of the North American Fuzzy Information Processing Society Workshop,** May 5-7, 1987, Purdue University, pp 140-147.

[GRAN 79] Grant, J. , "Partial Values in a Tabular Database Model," **Information Processing Letters,** Vol. 9, No. 2, 17 August 1979, pp 97-99.

[GRAN 77] Grant, J. , "Null Values in a Relational Database," **Information Processing Letters,** Vol. 6, No. 5, 1977, pp 156-157.

[HIRA 84] Hirakawa, M. , T. Shimuzu and T. Ichikawa, "Functional Augmentation of Relational Operations in ARES," **Management and Information Systems,** S. K. Chang, Ed. , New York: Plenum, 1984, pp 237-251.

[ICHI 80] Ichikawa, T. , T. Kikuno, N. Kamibayashi and M. Hirakawa, "ARES in a Relational Database Environment," **Proceedings of COMPSAC '80**, October 1980, pp 557-561.

[ICHI 86] Ichikawa, T. and M. Hirakawa, "ARES : A Relational Database with Capability of Performing Flexible Interpretation of Queries," **IEEE Transactions on Software Engineering,** Vol. 12, No. 5, May 1986, pp 624-634.

[JAES 82] Jaeschke, G. ,and H. J. Schek, "Remarks on the Algebra of Non First Normal Form Relations," **Proceedings of the ACM Symposium on Principles of Database Systems,** Los Angles, California, March 1982, pp 124-138.

[KACP 86] Kacprzyk, J. and A. Ziolkowski, "Database Queries with Fuzzy Linguistic Quantifiers", **IEEE Transactions on Systems, Man, and Cybernetics,** Vol. 16, No. 3, May/June1986, pp 474-478.

[LIPS 79] Lipski, W. Jr. , "On Semantic Issues Connected With Incomplete Information Databases," **ACM Transactions on Database Systems,** Vol. 4, No. 3, September 1979, pp 262-296.

[McCL 88] McClelland, E. B. , R. P. Trueblood and C. M. Eastman, "Two Approximate Operators for a Database Query Language: Sounds_like and Close_to," **IEEE Transactions on Systems, Man, and Cybernetics,** Vol. 18, No. 6, November/December 1988, pp 873-884.

[MOTR 88] Motro, A. , "VAGUE: A User Interface to Relational Databases that Permits Vague Queries," **ACM Transactions on Office Information Systems",** Vol. 6, No. 3, July 1988, pp 187-214.

[MOTR 86] Motro, A. , "BAROQUE: A Browser for Relational Databases," **ACM Transactions on Office Information Systems",** Vol. 4, No. 2, April 1986, pp 164-181.

[ROWE 84] Rowe, N. C. ,"Modelling Degrees of Item Interest for a General Database Query System," **International Journal of Man-Machine Studies,** Vol. 20, No. 5, May 1984, pp 421-443.

[SHEN 88] Shen, S. , "Database Relaxation: An Approach to Query Processing in Incomplete Databases," **Information Processing & Management,** Vol. 24, No. 2, 1988, pp 151-159.

[TAHA 77] Tahani, V. ,"A Conceptual Framework for Fuzzy Query Processing--A Step Toward Very Intelligent Database Systems," **Information Processing and Management,** Vol. 13, No. 5, 1977, pp 289-303.

[WILL 84] Willams, M. D. ,"What Makes RABBIT Run," **International Journal of Man-Machine Studies,** Vol. 21, No. 4, October 1984, pp 333-352.

[ZANI 82] Zaniolo, C. , "Database Relations with Null Values," **Proceedings of the ACM Symposium on Principles of Database Systems,** Los Angels, California, 1982, pp 27-33.

[ZEMA 85] Zemankova, M. and A. Kandel, "Implementing Imprecision in Information Systems," **Information Sciences,** Vol. 37, No. 1-3, 1985, pp 107-141.