Accommodating Imprecision in Database Systems: Issues and Solutions

Amihai Motro

Information Systems and Systems Engineering Department
George Mason University
Fairfax, VA 22030-4444

Abstract

Most database systems are designed under assumptions of precision of both the data stored in their databases, and the requests to retrieve data. In reality, however, these assumptions are often invalid, and in recent years considerable attention has been given to issues of imprecision in database systems. In this paper we review the major solutions for accommodating imprecision, and we describe issues that have yet to addressed, offering possible research directions.

1 Introduction

Information stored in a database is *precise* if it is assured to be identical to the "real world" information which it represents. When precise information is unavailable, it is often the case that some relevant information is nonetheless available. In these cases, it may be advantageous to design methods by which this information, termed *imprecise* information, can be stored, manipulated and retrieved. Imprecision may also be present in requests to retrieve data, when users, either intentionally or by necessity, formulate their queries in imprecise terms.

Depending on the data model used, the information stored in a database may take different forms, and imprecision could affect each and every one of them. For example, in the entity-relationship model, imprecision could occur in both entities and relationships. However, as virtually all recent research work in the area of database imprecision has been in the context of the relational data model, our discussion here is limited to this model. The structures of the relational model admit two different kinds of imprecision. The first kind involves imprecision at the level of data values; for example, the values of SALARY in the relation EARN (EMPLOYEE, SALARY) may be imprecise. The other kind involves imprecision at the level of the tuple; for example, the values of each of the attributes of the relation ASSIGN (EMPLOYEE, PROJECT) may be precise, but there may be uncertainty as to the precise assignment of employees to projects.

The relevant information that is available in the absence of precise data may take different forms. First, consider these examples of imprecision at the level of data values. It may be known that the true data value belongs to a specific set of values. Such imprecise data is often referred

This work was supported in part by the AT&T Affiliates Research Program.

to as *disjunctive* data. If the set to which the value belongs is simply the entire domain, then the imprecise data is referred to as *unavailable* (or *unknown*, or *missing*). If each of the candidate values is accompanied by a number describing the probability that it is indeed the true value (and the sum of these numbers for the entire set is 1), then the imprecise data is *probabilistic*. Note that probabilistic data subsumes all the types of imprecise data described so far, as well as precise data. Occasionally, the information available in the absence of precise data is a *descriptive* term. Such imprecise data is referred to as *fuzzy*. Imprecision is also possible at the level of tuples. For example, it may be known that the true tuple belongs to a set of tuples (disjunctive data), or a particular number may be available that describes the possibility that a given tuple belongs to the relation (fuzzy data).

It is useful to distinguish between the ability to accommodate *imprecise data* and the ability to handle *imprecise queries*, as each capability could be provided separately. It is possible to develop a model that accommodates imprecise data, and yet retain the standard query language, with its semantics extended to define when imprecise data match queries. On the other hand, it is possible to allow only standard databases, but extend the query language and the query processing methods to permit imprecise queries. By accommodating imprecise data, a database system provides for databases (and, therefore, answers) that are better approximations of the real world (the alternative being to ignore altogether information which is "imperfect"). The ability to handle imprecise queries is very helpful for requests which are intrinsically vague. With a database system that can only evaluate standard (specific) queries, the user must emulate such requests with standard queries. Usually, this means that the user is forced to retry a particular query repeatedly with alternative values, until it matches data that are satisfactory (if the user is not aware of any close alternatives, then even this solution is infeasible).

2 Solutions

Many of the approaches to the modeling of imprecision in databases are based on the theory of fuzzy sets [11, 12]. The approach described here is derived from [9, 14, 10].

The basic concept of fuzzy set theory is the *fuzzy set*. A fuzzy set F is a set of elements, where each element has an associated value in the interval [0,1] that denotes the *grade* of its membership in the set. For example, a fuzzy set may include the elements 20, 30, 40, and 50, with grades of membership 1.0, 0.7, 0.5 and 0.2, respectively.

As a relation is a subset of the product of several domains, one approach is to define relations that are fuzzy subsets of the product of fuzzy domains. Since each such relation is a fuzzy set, each of its tuples is associated with a membership grade. This definition admits imprecisions at the tuple level. For example, the tuple (Dick, Pascal) belongs to the relation PROFICIENCY(PROGRAMMER, LANGUAGE) with membership grade 0.9 [1]. To represent such relations, the usual attributes are supplemented with a column that assigns the membership grades to the tuples of the relation.

Consider the fuzzy set defined earlier, and assume it is named YOUNG. It is also possible to interpret this set as the definition of the term "young": it is a term that refers to 20 year olds with possibility 1.0, to 30 year olds with possibility 0.7, etc. Thus, fuzzy sets may be applied to describe imprecise terms.

[1] Alternatively, this tuple may be interpreted as stating that Dick's proficiency in Pascal is 0.9.

Consider now standard (nonfuzzy) relations, but assume that the elements of the domains are not values, but fuzzy sets of values. This definition admits imprecisions at the data value level. Having fuzzy sets for values permits specific cases where a value is one of five types: (1) A set; for example, the value of DEPARTMENT can be {shipping, receiving} or the value of SALARY can be 40,000–50,000. Note that the interpretation of such sets is purely *disjunctive*: exactly one of the elements of the set is the correct value. (2) A fuzzy value; for example, the value of AGE can be young. (3) An estimate; for example, the value of SMART can be 0.8. (4) A null value. (5) A simple value.

Finally, by defining a fuzzy relation as a fuzzy subset of the product of fuzzy domains of fuzzy sets, both kinds of imprecision can be accommodated.

To manipulate fuzzy databases, the standard relational algebra operators must be extended to fuzzy relations. The first approach, where relations are fuzzy sets but elements of domains are "crisp", requires relatively simple extensions of these operators. The second approach, where relations are crisp but elements of domains are fuzzy, introduces more complexity because the "softness" of the values in the tuples creates problems of value identification (e.g., in the join, or in the removal of replications after projections). Also, in analogy with standard mathematical comparators such as $=$ or $<$, which are defined via sets of pairs, the second approach introduces fuzzy comparators such as *similar-to* or *much-greater-than*, which are defined via fuzzy sets of pairs. These fuzzy operators offer the capability of expressing fuzzy (imprecise) retrieval requests.

Experimental database systems based on ideas similar to those described here have been implemented by various researchers. One example is the FRDB system [14]. Its overall architecture involves three components: (1) A database for storing extended relations (standard relations over domains of fuzzy sets). (2) An auxiliary database that stores the definitions of fuzzy sets that are used for domain values (e.g., YOUNG) and fuzzy comparators (e.g., *much-greater-than*). (3) Rules that define additional fuzzy concepts via the concepts stored in the auxiliary database (e.g., the definitions of "young age" and "high salary" may be combined to define "successful"). Each statement in the query language of FRDB performs one fuzzy relational operation, and a query is executed with a sequence of such statements. An optional argument of each statement is a *threshold* value between 0 and 1. This value is used to filter the tuples of the result, retaining only tuples whose membership grade exceeds the threshold value. Users of the FRDB system can present queries such as "select person, who is young and very smart", "select person who is much more intelligent than athletic, or who is not very young" and "select city, where summer-temperature is not much greater than winter-temperature, and is not so large".

An alternative approach to imprecise querying is to define *distances* among the elements of the domain, and then use these distances to derive measures of equality and similarity. By dividing each distance by the *diameter* of the domain (the largest distance among its elements), a measure of dissimilarity is obtained. Its complement to 1 is then a measure of similarity. Equality is defined as similarity with value 1. With an appropriate *radius* to serve as a threshold distance, every two values are either similar or dissimilar. A selection condition such as $A \sim a$ would then be satisfied by every element of the domain of A, that is similar to a. VAGUE [8] is an example of an extension to a conventional database system that uses data distances to interpret vague queries (another such example is ARES [4]). VAGUE extends the relational model with the concept of *data metrics*, and its query language with a *similar-to* comparator. A data metric defines the distance between every two elements of a domain; for example, in a personnel database there may be metrics to measure distances between titles, between salaries, between qualifications, as well as a metric to measure

distances between employees. Database designers are provided with several different methods for defining data metrics, and users are offered several features with which they can adapt the available metrics to their own needs. A *similar-to* comparison is satisfied with data values that are within a predefined distance of the specified value; for example, the vague comparison "language *similar-to* Pascal" may be satisfied by Pascal, Algol and Modula. To present queries, users need only to know about the new comparator. In contrast with a standard query, that establishes a rigid qualification and is concerned only with data that match it precisely, a vague query of the kind that can be handled by VAGUE establishes a *target* qualification and is concerned with data that are *close* to this target.

Buckles and Petri [1] define fuzzy databases differently. Relations are extended to allow values that are *sets* of domain elements. Therefore, a fuzzy tuple is a sequence of sets $(d_1, d_2, \ldots, d_n)$; it represents a *set* of specific tuples $(a_1, a_2, \ldots, a_n)$, where $a_i \in d_i$, for all i. Thus, a fuzzy tuple models uncertainty: each of the specific tuples that it represents is equally likely to be the "correct" tuple. Additionally, each database domain has an associated *similarity matrix* that assigns a value between 0 and 1 to each pair of domain elements. The output of standard relational algebra operators, such as join or project, is post-processed to merge tuples (by performing unions of their respective components), if a pre-specified similarity threshold is not violated.

Recently, Garcia-Molina and Porter [3] suggested modeling uncertainty by using traditional probability distributions (rather than the possibility distributions of the fuzzy models). A *probability distribution function* of a variable X over a domain D assigns each value $d \in D$ a value between 0 and 1, as the probability that $X = d$. One important difference between probability and possibility distribution functions is that the sum of the probabilities assigned to the elements of X must be exactly 1. The definition of a probabilistic database is similar to the second definition of fuzzy databases: standard relations, but with domain values that are, in general, probability distribution functions. A feature of this model is that it allows probability distributions that are incompletely specified: each such distribution is complemented with a "missing value" which is assigned the balance of the probability.

If a model such as a fuzzy data model is not used, then all imprecise information must be ignored, and the information is then considered *unavailable*. When the value of a particular attribute for a particular tuple is unavailable, a special value, called *null*, is stored instead. Hence, unavailability is the least informative form of imprecision. Recall that in many of the models mentioned earlier, unavailability, also called *incompleteness*, is handled as a special case. Due to the prevalence of this type of imprecision, and the relative simplicity of the "information", unavailability has been the subject of intensive research (see [6, Chapter 12]). Much of this research focuses on the precise semantics of null values, and on the appropriate extension of the relational algebra to databases with null values (e.g., [2]).

The models described so far incorporate modeling extensions that permit modeling imprecise information. An alternative approach is discussed in [7]. Instead of modeling the "imperfections" of the available data, it suggests declaring the portions of database that are assured to be prefect models of the real world (and thereby the portions that are possibly imperfect). With this information included in the database, the database system can *qualify* the answers it issues in response to queries with regard to their precision: each answer is accompanied by statements that define the portions that are guaranteed to be perfect.

3 Discussion

Commercial database systems have been relatively slow to incorporate imprecision capabilities: the only capabilities widely available are for handling null values, and for specifying imprecise queries through the use of regular expressions. Examining the possible reasons for this slow acceptance may suggest directions for further research. First, database practitioners are concerned primarily with the *performance* of a database system. However, many of the query evaluation algorithms for matching imprecise data or for processing imprecise queries are fairly complex and inefficient. Practitioners are also concerned with *compatibility*. This dictates that capabilities for accommodating imprecision should be offered as strict extensions of existing standards. Third, database practitioners have often been dissatisfied with various *idiosyncratic implementations* of imprecision capabilities (minor examples are the inability to specify an incomplete date value, or the inability to sort answers with null values flexibly). This may have had a chilling impact on further implementations.

Another hindrance for database systems with imprecision capabilities may lie in the expectations of users. A fundamental principle of database systems has been that queries and answers are never open to "subjective" interpretations, and users of database systems have come to expect their queries to be interpreted unambiguously and answered with complete accuracy. In contrast, users of information retrieval systems would be pleased with a system that delivers high rate of recall (proportion of relevant material retrieved) and precision (proportion of retrieved material that is relevant). A database system that must adhere to the principles of unambiguity and complete accuracy cannot accommodate the full range of imprecise information; for example, it can accommodate null values or disjunctive data, but not (due to its subjective nature) probabilistic or fuzzy data. Further research is required on systems that would integrate the capabilities of information retrieval systems and database systems attractively.

As mentioned earlier, most of the research effort on imprecision has been in the context of the relational data model. At present, much research and development efforts in the area of databases is focused on "next generation" database systems [2]. It appears advantageous to incorporate imprecision capabilities into early versions of these systems, as such extensions are more difficult to effect once "standards" have been agreed upon. Preliminary work in this area includes investigation of incompleteness in logical databases [5] and the design of a knowledge-rich system that can deal with various aspects of imprecision [13].

The ability to respond to imprecise queries may be considered as one aspect of *generalized* (non-standard) query answering. Other aspects include *intensional* answering (where answers are described with predicates or constraints), *cooperative* answering (where information considered relevant to the query is delivered along with, or in place of, the standard answer), and *approximative* answering (where an approximation is provided in lieu of the standard answer, possibly because of limited computational resources). Various models and techniques have been developed to handle these individual aspects of generalized query answering, and further research is required to integrate them into a single model of retrieval.

[2]The term is used loosely to refer to object-oriented or knowledge-rich systems.

References

[1] B. P. Buckles and F. E. Petry. A fuzzy representation of data for relational databases. *Fuzzy Sets and Systems*, 7(3):213–226, May 1982.

[2] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, December 1979.

[3] H. Garcia-Molina and D. Porter. *Supporting Probabilistic Data in a Relational System.* Technical Report TR-147, Department of Computer Science, Princeton University, February 1988.

[4] T. Ichikawa and M. Hirakawa. ARES: a relational database with the capability of performing flexible interpretation of queries. *IEEE Transactions on Software Engineering*, SE-12(5):624–634, May 1986.

[5] T. Imielinski. Incomplete information in logical databases. *Data Engineering*, 12(2):29–40, June 1989.

[6] D. Maier. *The Theory of Relational Databases.* Computer Science Press, Rockville, Maryland, 1983.

[7] A. Motro. Integrity = validity + completeness. *ACM Transactions on Database Systems*, 14(4):480–502, December 1989.

[8] A. Motro. VAGUE: a user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems*, 6(3):187–214, July 1988.

[9] H. Prade and C. Testemale. Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries. *Information Sciences*, 34(2):115–143, November 1984.

[10] K. V. S. V. N. Raju and A. Majumdar. Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. *ACM Transactions on Database Systems*, 13(2):129–166, June 1988.

[11] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, June 1965.

[12] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1(1):3–28, 1978.

[13] M. Zemankova. FIIS: a fuzzy intelligent information system. *Data Engineering*, 12(2):11–20, June 1989.

[14] M. Zemankova and A. Kandel. Implementing imprecision in information systems. *Information Sciences*, 37(1,2,3):107–141, December 1985.