Imprecision and incompleteness in relational databases: survey

A Motro

Most database systems are designed under assumptions of precision and completeness of both the data they store and the requests to retrieve data, even though in reality these assumptions are often false. In recent years, considerable attention has been given to issues of imprecision and incompleteness in databases. The article surveys and compares the basic approaches that address these issues in relational databases.

databases, relational databases, fuzzy sets, imprecision, incompleteness, modelling

Most database systems are designed under assumptions of precision and completeness of both the data they store and the requests to retrieve data, even though in reality these assumptions are often false. In recent years, considerable attention has been given to issues of imprecision and incompleteness in databases, and this article surveys and compares the basic approaches that address these issues in relational databases.

Consider the following example, a relation EMPLOYEE with attributes (NAME, DEPARTMENT, AGE, TELEPHONE_NUMBER, SALARY). A typical tuple in this relation may have the value (Sue, accounting, 38, 395–7453, 52,000). Now consider this information:

- Tom's department is either shipping or receiving.
- There is probability 0.6 that the salary of Harry is in the range 60,000–75,000, probability 0.3 that it is in the range 75,000–90,000, and probability 0.1 that it is some other value.
- Betty is young.
- John's telephone number is unknown.
- It is unknown whether Bob has a telephone.

In each case, the value of some attribute cannot be specified with a single element from the assumed domain of values; but, in some of those cases, other information is available. In the first case, the value is one of several possible values; in the second, a probability distribution is available; in the third, the value is described by an imprecise term; in the other two cases, the value is simply unknown (note that the latter case is less informative than the former). In any case, the available information cannot be represented in the standard relational model.

Assuming that the above information has been accommodated in some way, now consider these retrieval requests:

- Employees of the shipping department.
- Employees with high salary.
- The salary of Harry.
- Employees under 25 years of age.
- Employees with a telephone.

In each case, it is either impossible to formulate an appropriate query in a standard relational language or else the response to the query is not obvious. Should Tom be included in or excluded from the list of employees of the shipping department? Either way, the answer may be wrong. How would the database system determine what constitutes 'high salary'? Can a useful estimate be given of Harry's salary? Is Betty under 25? Should Bob be included among those with a telephone? If it is assumed that the database contains only precise data, then all but the second request are quite ordinary; the second request, however, is a sensible request, which simply cannot be expressed in relational database systems.

Consider now this additional information:

- Data on the employees of the accounting department are guaranteed to be valid.
- There may be additional employees in the personnel department.

These statements provide additional information on the quality (validity and completeness) of the information in the database. Such information can qualify the answers given in response to any of the above queries. For example, the answer to the query about persons with high salary should mention that this information is guaranteed to be valid only for the employees of the accounting department, and complete for all employees, except those of the personnel department.

As databases are models of the real world, the issue of their precision and completeness has interesting epistemological implications. In particular, the term 'real

Computer Science Department, University of Southern California, Los Angeles, CA 90089-0782, USA. Currently at Information Systems and Systems Engineering Dept, George Mason University, Fairfax, VA 22030-4444, USA.
Paper submitted: 4 December 1989.
Revised version received: 5 May 1990.

 0950-5849/90/090579-10

world', when used in this paper, does not refer to the actual state of things, but to a subjective observation of this world by an individual or group of individuals. Thus statements about the precision and completeness of a database are actually statements about the relationship between the database and a particular subjective view of the real world. Because the real world is elusive, it has been suggested that 'absolute truth' cannot be the criterion for the correctness of a database, and instead database designers, administrators, and users should concern themselves only with the question of whether they know of some other database that is 'closer' to the truth. A more thorough discussion of these issues has been given by Thompson[1].

In database research it is sometimes assumed that a database is a complete model of the real world. Under this assumption, called the closed world assumption (CWA)[2], a database contains all occurrences of the data it attempts to model. More accurately, the CWA states that if a fact is not included in the database (and cannot be inferred from it), then it is false. In practice, however, this assumption is not realistic, as most databases include at least some information that is possibly incomplete. Indeed, the research described in this paper usually assumes that the CWA does not hold. (Later, research is described that assumes a restricted version of the CWA.)

This article reviews and compares approaches to issues of imprecision and incompleteness in databases. The works discussed are all within the framework of the relational model of databases[3], and it is assumed that the reader is familiar with the model and its terminology[4]. In addition, the following concepts and terms will be used.

It is assumed that each attribute of every relation is associated with a predefined 'domain', which specifies the permissible values of that attribute (e.g., the set of integers in the range 0–120 for the attribute AGE, the set {shipping, receiving, accounting, personnel} for the attribute DEPARTMENT, etc.). A value stored in the database is 'definite' if it is an element of the associated domain and is known with certainty to be the correct value. When a definite value is not available, it is possible that an approximation is available, or else the value is altogether missing. These possibilities shall be referred to as 'imprecise' data and 'incomplete' data, respectively. Databases that store only definite data will be referred to as 'standard' databases. Databases that can also store imprecise or incomplete data will be referred to as 'extended' databases.

A retrieval request specifies a tuple of attributes and a qualification. It is answered by the tuples of values of the corresponding attributes, which are stored in the database and satisfy the qualification. The qualification is constructed from phrases that compare attributes and definite values using standard mathematical comparators (e.g., DEPARTMENT = shipping, AGE < 25, INCOME = SALARY). Such requests will be referred to as 'specific' queries. Requests that are not specific will be referred to as 'vague' queries.

Finally, a database system that can only store databases with definite data and evaluate specific queries will be referred to as a 'conventional' system.

While there is a wide variety of approaches to imprecision and incompleteness in relational databases, most models and systems that have been developed can be classified as extensions to the relational model for expressing:

- imprecision
- incompleteness
- validity and completeness

Models in the first category extend the relational model to represent imprecision, with incompleteness treated as a specific case of imprecision. In addition, appropriate retrieval language and query evaluation algorithms are defined to allow the specification and evaluation of vague queries. Models in the second category extend the relational model to represent incomplete information and evaluate queries correctly in the presence of incompleteness. Models in the last category store information about the quality of the data (its validity and completeness) and use this information to qualify their answers accordingly.

The next three sections are devoted to these categories. The last section provides a comparative discussion of the various approaches.

MODELLING IMPRECISION

Most approaches to the modelling of imprecision in databases are based on the theory of fuzzy sets, and first a few basic concepts from this theory are established. Then a model for fuzzy databases is defined and several variations discussed.

Fuzzy-set theory was first introduced by Zadeh[5,6] as a generalization of classical-set theory, and since then has been adapted and applied to various applications, including databases and artificial intelligence. Its basic concept is the fuzzy set, which is defined as follows.

Assume a universe of elements U. A fuzzy set F is a set of elements from U, where each element has an associated value in the interval [0,1] which denotes the grade of its membership in the set.

In classical-set theory a set S is characterized by a membership function χ_s, which maps the elements of U to either 1 or 0, to denote their membership in S:

$$\chi_s : U \rightarrow \{0,1\}$$
$$\chi_s(u) = \begin{cases} 1 \text{ if } u \in S \\ 0 \text{ if } u \notin S \end{cases}$$

In analogy, in fuzzy-set theory a fuzzy set F is characterized by a membership function μ_F, which maps each element of the universe U to a value between 0 and 1, to denote its grade of membership in F:

$$\mu_F : U \rightarrow [0,1]$$

A fuzzy set is often denoted by a list of the elements whose membership grades are greater than 0, along with these grades. For example, a fuzzy set YOUNG with

Table 1. Fuzzy relation with precise domain values

PROFICIENCY		
PROGRAMMER	LANGUAGE	μ
Dick	Pascal	0.9
Dan	Fortran	1.0
Mary	Cobol	0.3
Mary	Lisp	0.8

elements 20, 30, 40, and 50, and grades of membership 1.0, 0.7, 0.5, and 0.2, respectively, is denoted:

YOUNG = {1.0/20,0.7/30,0.5/40,0.2/50}

The usual set operations of classical-set theory are extended to fuzzy sets by defining the membership function of the result of each operation. For example, if A and B are two fuzzy sets with membership functions μ_A and μ_B, respectively, then their intersection, union, complement, and (Cartesian) product are defined by the following membership functions:

$$\begin{aligned} \mu_{A\cup B}(u) &= \max\{\mu_A(u),\mu_B(u)\} \\ \mu_{A\cap B}(u) &= \min\{\mu_A(u),\mu_B(u)\} \\ \mu_{\bar{A}}(u) &= 1-\mu_A(u) \\ \mu_{A\times B}(u_1,u_2) &= \min\{\mu_A(u_1),\mu_B(u_2)\} \end{aligned}$$

The empty fuzzy set $\emptyset$ is a fuzzy set with membership function $\mu_\emptyset(u)=0$ for all $u \in U$. Given two fuzzy sets A and B, A is a fuzzy subset of B, if $\mu_A(u) \leqslant \mu_B(u)$, for all $u \in U$. Most of the properties that hold for classical-set operations, such as distributivity of union and intersection or DeMorgan's laws, can be shown to hold for their fuzzy counterparts. The only exception is the law of excluded middle, where in fuzzy-set theory:

$A\cap\bar{A} \neq \emptyset$ and $A\cup\bar{A} \neq U$.

Fuzzy-set theory has formed the basis for several extensions of the relational model aimed at accommodating various forms of imprecision. The model described is derived from other sources[7–9].

In the relational model two slightly different kinds of imprecision are observed. The first involves imprecision at the level of data values; for example, the salary of John in the relation EARN = (EMPLOYEE, SALARY) may be imprecise. The second kind involves imprecision at the level of tuples; for example, there may be uncertainty about the membership of the tuple (Dick, Pascal) in the relation PROFICIENCY = (PROGRAMMER, LANGUAGE).

There are several ways to define fuzzy databases. 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. As each such relation is a fuzzy set, each of its tuples is associated with a membership grade. This definition admits imprecisions of the second kind. As an example, consider the fuzzy relation in Table 1. The usual attributes are supplemented with a column that assigns the membership grades to the tuples of the relation. Thus the tuple (Dick, Pascal) belongs to the relation PROFICIENCY with membership grade 0.9. (Alternatively, this tuple may be interpreted as stating that Dick's proficiency in Pascal is 0.9.)

Table 2. Relation with imprecise domain values

EMPLOYEE				
NAME	DEPARTMENT	AGE	SMART	SALARY
Sue	accounting	38	0.8	52,000
Tom	{shipping, receiving}	27	0.6	low
Betty	⊥	young	0.9	40,000–50,000
Jim	personnel	42	1.0	high

Consider again the fuzzy set YOUNG. It can also be interpreted as a description 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.

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 of the first kind. An example of a relation with values that are fuzzy sets is shown in Table 2. It is assumed that the fuzzy sets high, low and young are defined separately. Having fuzzy sets for values permits specific cases where a value is either:

- A standard set, such as {shipping, receiving} or 40,000–50,000 (a fuzzy set whose elements all have membership grade 1.0; these values are omitted from the notation). Note that the interpretation of such sets is purely disjunctive: exactly one of the elements of the set is the correct value.
- A simple value, such as Tom or personnel (a singleton fuzzy set, whose only element has membership grade 1.0; both the set notation and membership grade are omitted).
- A null value denoted with ⊥ (an empty fuzzy set).
- A singleton fuzzy set, such as {0.8/smart} (the set notation is omitted, and if the name of the element coincides with the name of the attribute it is omitted as well).

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 (to answer queries), the standard relational algebra operations must be extended to fuzzy relations. These operations include ordinary set operations, such as intersection, union, difference, and product, as well as special operations, such as selection, projection, and (natural) join.

Consider the first definition of a fuzzy database, where relations are fuzzy subsets of the product of fuzzy domains. The definitions of the relational set operations are then similar to the definitions of fuzzy-set operations. The treatment of selection, projection, and join is shown below.

Let R be a fuzzy relation with attributes α. For the selection by condition ψ from R define a function that assigns all the tuples over the attributes α membership grades in the selection. The membership grade of a tuple that satisfies ψ is identical to its membership grade in R; the membership grade of a tuple that does not satisfy ψ is 0.

Let R be a fuzzy relation with attributes α, and assume $\beta \subseteq \alpha$. For the projection of R on β, define a function that assigns all the tuples over the attributes β grades of membership in the projection. Each tuple over the attributes β corresponds to several tuples over the attributes α (with different membership grades in R). The membership grade of a tuple in the projection is defined as the maximum of the membership grades of the corresponding tuples in R. For example, if 0.3/(Mary, databases, Cobol) and 0.8/(Mary, databases, Lisp) are two tuples in the fuzzy relation (PROGRAMMER, EXPERTISE, LANGUAGE), then 0.8/(Mary, databases) is a tuple in the projection onto (PROGRAMMER, EXPERTISE).

Let R_1 and R_2 be two fuzzy relations with attributes α_1 and α_2, respectively. For the (natural) join of R_1 and R_2, define a function that assigns all tuples over the attributes $\alpha_1 \cup \alpha_2$ grades of membership in the join. Each tuple over the attributes $\alpha_1 \cup \alpha_2$ corresponds to one tuple over the attributes α_1 (with a membership grade in R_1) and one tuple over the attributes α_2 (with a membership grade in R_2). The membership of a tuple in the join is defined as the minimum of the membership grades of the corresponding tuples in R_1 and R_2. For example, if 0.8/(Mary, databases) is a tuple in the fuzzy relation (PROGRAMMER, EXPERTISE), and 0.3/(Mary, Cobol) is a tuple in the fuzzy relation (PROGRAMMER, LANGUAGE), then 0.3/(Mary, databases, Cobol) is a tuple in the join (PROGRAMMER, LANGUAGE, EXPERTISE).

Consider now the second definition of a fuzzy database, where relations are standard, but the elements of the domains are fuzzy sets of values. The 'softness' of the values in the tuples introduces problems of identification. For example, consider a join of R_1 and R_2 on a common attribute AGE, and assume the tuple in R_1 has the value young, while the tuple in R_2 has the value 21. The join should be able to identify these two values of AGE, and connect the two tuples.

To define the relational operations in this case, a few more concepts must be established. A possibility distribution function of a variable X assigns each $u \in U$ the possibility that $X = u$:

$$\pi_X(u) = \text{poss}(X = u) \text{ for all } u \in U$$

A fuzzy set F induces a possibility distribution function of a variable X, whereby: $\pi_X(u) = \mu_F(u)$. Thus possibility distribution functions and fuzzy sets are interchangeable. The possibility measure of a variable X assigns every set $A \subseteq U$ the highest possibility of any of its elements:

$$\prod_X(A) = \max_{u \in A}\{\pi_X(u)\} \text{ for all } A \subseteq U$$

In some respects, possibility measures are analogous to the more common probability measures. (For simplicity it is assumed that U is finite; if U is infinite then supremum should be substituted for maximum in the definition of a possibility measure.) Assume now that A is a fuzzy subset of U. The possibility measure of A is corrected to account for the fact that the membership of an element u in a set A is only a possibility:

$$\prod_X(A) = \max_{u \in U} \min\{\pi_X(u), \mu_A(u)\} \text{ for all fuzzy } A \subseteq U$$

Standard mathematical comparators, such as = or <, are defined via sets of pairs of values; each set includes the pairs for which the comparison is true. Similarly, fuzzy comparators are defined via fuzzy sets of pairs of values. For example, the operator similar-to on the domain of persons may be defined by providing $\mu_{\text{similar-to}}(a,b)$ for every two persons a and b. Other examples of fuzzy operators are much-greater-than for integers, distant for geographical locations, and so on.

Consider a simple selection condition of the kind A θ a, where A is an attribute of a fuzzy relation R (the domain of A will be denoted D), θ a fuzzy comparator, and a the name of a fuzzy set (a possibility distribution). Examples of such selection conditions are 'age is young', salary is much-greater-than 50,000', and 'performance is much-higher-than mediocre'.

Let the variable x. A denote the value of attribute A in tuple x, and let E be the (fuzzy) set of elements that are in relation θ with an element of a. The possibility measure of E is:

$$\prod_{x.A}(E) = \max_{d \in D} \min\{\pi_{x.A}(d), \mu_E(d)\}$$

$\pi_{x.A}(d)$ is the possibility that the value of attribute A in tuple x is d, and $\mu_E(d)$ is the possibility that d is in relation θ with an element of a. The latter is given by:

$$\mu_E(d) = \max_{d' \in D} \min\{\mu_\theta(d,d'), \mu_a(d')\}$$

$\mu_\theta(d,d')$ is the possibility that d is in relation θ with d', and $\mu_a(d')$ is the possibility that d' is in a.

This measure computes for each tuple x in R a membership grade in the selection. Thus it derives all the tuples that possibly satisfy the condition (and their membership grades). It is also possible to define a dual measure (called the necessity measure) that expresses the necessity of an event as the impossibility of the opposite event. The necessity measure will derive the more restricted set of tuples that necessarily satisfy the condition (and their membership grades).

The above computation may be generalized to selection conditions that compare two attributes and to compound selection conditions. This approach also defines the join of two relations, where values of the same attribute in two different relations must be compared. The notion of similarity is also used in the definition of projection, to eliminate redundancies (tuples that are considered 'similar' to other tuples). Elimination of redundancies is also performed during other set operations, such as union and intersection. Clearly, the computational expense for query processing is much higher for the second kind of fuzzy databases.

Finally, the two approaches to the definition of relational operators on fuzzy databases may be combined to define operations on relations that are fuzzy subsets of the product of fuzzy domains of fuzzy sets.

The model developed by Prade and Testemale[7] also includes a special domain value that denotes inapplicability, i.e., a particular attribute is not applicable to a particular tuple. This model can therefore deal with two kinds of incompleteness: unavailability and inapplicability.

Experimental database systems based on ideas similar to those described here have been implemented by various researchers. One example is the FRDB system[8]. Its overall architecture involves three components:

- A database for storing extended relations (standard relations over domains of fuzzy sets).
- 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).
- 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 langauge 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 filters the tuples of the result, retaining only tuples whose membership grade exceeds the threshold value. Users 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'.

Several alternative approaches to imprecision are now reviewed that are not based so closely on fuzzy-set theory.

Recall that comparators are defined via sets: a standard comparator is defined by a set of pairs, a fuzzy comparator by a fuzzy set of pairs. While any set may serve to define a comparator, comparators may have individual semantics that should be captured in the defining set. For example, the set that defines the comparator = (equality) is expected to include every pair (a,a), and if (a,b) is included, then (b,a) should be included as well. This models the reflexivity and symmetry of equality. By analogy, the fuzzy set that defines the comparator $\sim$ (similarity) is expected to include every pair (a,a) with possibility 1, and (a,b) and (b,a) should always have equal possibilities.

Additionally, equality is defined to have transitivity: if (a,b) and (b,c) are included, then (a,c) is included as well. Defining the transitivity of similarity is slightly more complicated. If (a,b) and (b,c) have possibilities $\text{sim}(a,b)$ and $\text{sim}(b,c)$, respectively, then (a,c) should have at least the smallest of these possibilities:

$$\text{sim}(a,c) \geq \min\{\text{sim}(a,b), \text{sim}(b,c)\} \text{ for all } a,b,c \in D$$

where D denotes the domain. Therefore, it follows that

$$\text{sim}(a,c) \geq \max_{b \in D} \min\{\text{sim}(a,b),\text{sim}(b,c)\} \text{ for all } a,c \in D$$

This form of transitivity is usually referred to as max-min transitivity. Note that $\text{sim}(a,b)$ is the possibility that a and b are similar, but may also be interpreted as the level of their similarity.

An alternative approach 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[10] is an example of an extension to a conventional database system that uses data distances to interpret vague queries (another example is Ares[11]). 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. 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. 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. Recall that a specific query 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. For example, a query to retrieve the employees who are engineers and who can program in Pascal can be relaxed by the users of Vague into a query that retrieves the employees who have a title similar-to engineer, and who are proficient in a language similar-to Pascal (and possibly match an applied mathematician who can program in Algol). The advantage over conventional database systems is evident. If the database system can only evaluate specific queries, the user must emulate a request that is intrinsically vague with specific 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.

Buckles and Petry[12] 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. Addi-

tionally, 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[13] suggested modelling uncertainty by using traditional probability distributions (rather than possibility distributions). 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.

MODELLING INCOMPLETENESS

If a model such as the fuzzy data model is not used, then all imprecise information must be ignored, and the information is then considered to be unavailable. When the value of a particular attribute for a particular tuple is unavailable, a special value, called null and denoted $\perp$, is stored instead. Although the same symbol is used for all occurrences of unavailable data, it must be assumed that these nulls represent different values.

When an entity is modelled by a set of attributes, it is assumed that every occurrence of this entity would possess each of these attributes. In practice, however, there would often be exceptions; for example, while most employees would probably have names, departments, and salaries, some employees would possibly not be assigned to departments. This information is then considered inapplicable. Again, when the value of a particular attribute for a particular tuple is inapplicable, a null value is stored instead.

Thus there are two distinct kinds of nulls, each with its individual semantics*. It is usually agreed that nulls must not be admitted in the key attributes of a relation, for example, the attribute NAME of relation EMPLOYEE. Intuitively, the key attributes are the subject of the relation, and the non-key attributes describe this subject. Although such occurrences of incompleteness are possible (e.g., there may be an employee whose department and salary are known, but not his name), incompleteness in the subject may lead to serious problems. For example, if their names are missing, the tuples of two different employees that happen to have the same attributes would be identified by the system as a replication.

*It has been suggested that when it is unknown whether the incompleteness is due to unavailability or to inapplicability, a null of a third kind should be stored, for example, when it is not known whether an employee is assigned to a department or not.

Table 3. Using functional dependencies to complete relation

SALARY_HISTORY			
NAME	AGE	YEAR	SALARY
Sue	38	1989	48,000
Sue	$\perp$	1990	52,000
Jim	42	1989	46,000
Jim	42	1990	50,000

Table 4. Using inferred functional dependencies to complete relation

ASSIGNMENT			
NAME	TITLE	SALARY	PROJECT
Dick	programmer	40,000	optimization
Dan	$\perp$	48,000	interfaces
Dan	$\perp$	$\perp$	networks

The remainder of this section considers only nulls that denote unavailability and refers to relations with nulls as incomplete and to relations without nulls as complete. An incomplete relation may be considered as representing a set of complete relations, each describing an admissible way of filling in the nulls with nonnull values.

It is possible to fill in some null values automatically, using available semantic information. For example, consider the relation in Table 3, which describes the salary history of employees. Clearly, it can be inferred that the AGE attribute in the second tuple is also 38. In filling in this null, the fact that the attribute AGE is functionally dependent on the attribute NAME is used, and therefore if two tuples agree on their NAME attribute, they must also agree on their AGE attribute.

The concept of functional dependency[3,4] is fundamental to relational databases and is defined formally as follows. Let R be a set of attributes, and let $X, Y \subseteq R$. If every two tuples that agree on the attributes X must also agree on the attributes of Y, then Y is functionally dependent on X, denoted $X \rightarrow Y$. Functional dependencies describe the semantics of a set of attributes and play an important role in the theory of database design.

Given a set R of attributes and a set F of functional dependencies among these attributes, additional functional dependencies can be inferred. The set of all functional dependencies that can be inferred from the functional dependencies in F is called the closure of F and is denoted F^+. One method to compute F^+ is to initialize it to the set F and augment it through repeated application of the following three rules, known as Armstrongs' axioms:

- Reflexivity: $X \rightarrow Y$ is in F^+, for all $X \subseteq R$ and $Y \subseteq X$.
- Augmentation: If $X \rightarrow Y$ is in F^+, then $W \cup X \rightarrow W \cup Y$ is in F^+, for all $W \subseteq R$.
- Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$ are in F^+, then $X \rightarrow Z$ is in F^+.

Consider now the relation in Table 4, which describes the

Table 5. Marked nulls

EMPLOYEE (a)		
NAME	TITLE	SALARY
Dick	⊥	40,000
Mary	⊥	⊥
Dan	scientist	48,000

EMPLOYEE (b)		
NAME	TITLE	SALARY
Dick	$\perp_1$	40,000
Mary	$\perp_2$	$\perp_3$
Dan	scientist	48,000

EMPLOYEE (c)		
NAME	TITLE	SALARY
Dick	$\perp_1$	40,000
Mary	$\perp_1$	$\perp_3$
Dan	scientist	48.000

EMPLOYEE (d)		
NAME	TITLE	SALARY
Dick	$\perp_1$	40,000
Mary	$\perp_1$	40,000
Dan	scientist	48.000

assignment of employees to projects. The assumed semantics are that each employee has a unique title, but may be assigned to several projects, and that the title determines the salary. In terms of functional dependencies: NAME → TITLE and TITLE → SALARY. Neither dependency can fill in any nulls. The former dependency cannot be used, because although the last two tuples agree on NAME, both values of TITLE are unavailable. The latter dependency cannot be used either, because although one of the values of SALARY is available, under the earlier assumption the two null values of TITLE cannot be assumed identical. However, the functional dependency NAME → SALARY, which can be inferred by transitivity from the given functional dependencies, can be used to determine that the value of SALARY in the third tuple should be 48,000.

Consider now the relation described in Table 5(a), and assume that here also NAME → TITLE and TITLE → SALARY. Assume that it is known that Dick and Mary have the same title (though that title is unknown). It is obvious that Mary's salary is then also 40,000. To infer this value properly, the concept of marked nulls is introduced. Initially, as all the null values in the database are assumed to represent different nonnull values, different null symbols are used. Table 5(b) shows the initial assignment of marked nulls for the same example. Whenever two unavailable values are known to be identical, the corresponding null values are identified. Table 5(c) reflects the knowledge that Dick and Mary have the same title. Now the second dependency can be used to conclude that $\perp_3$ is 40,000. The result is shown in Table 5(d).

So far the representation of incompleteness has been discussed. A more complex issue is how to manipulate databases with incomplete information. Here attention is limited to the problem of querying such databases (the updating problem has been discussed[14,15]). The main issue in querying incomplete databases is how to extend the standard relational algebra operations to incomplete relations, so that the results of queries make sense. Various approaches have been proposed[16]. Here the focus is on that advocated by Codd[17].

The first question is the value of a comparison $x = y$ if x or y or both are null. The appropriate result is always the unknown truth value, rather than true or false.

Table 6. Example of extended join

EMPLOYEE		
NAME	SALARY	DEPARTMENT
Sue	52,000	accounting
Jim	⊥	personnel
Betty	48,000	⊥

DEPARTMENT	
D_NAME	MANAGER
accounting	Bob
personnel	Mike
shipping	⊥

EMPLOYEE⋈DEPARTMENT			
NAME	SALARY	DEPARTMENT	MANAGER
Sue	52,000	accounting	Bob
Jim	⊥	personnel	Mike
Betty	48,000	accounting	Bob
Betty	48,000	personnel	Mike
Betty	48,000	shipping	⊥

Accordingly, a three-valued logic is defined, where the usual values true and false are joined with a third value, called maybe. The logic operations and, or, and not are defined with the following truth tables:

∧	*false*	*maybe*	*true*
false	*false*	*false*	*false*
maybe	*false*	*maybe*	*maybe*
true	*false*	*maybe*	*true*

∨	*false*	*maybe*	*true*
false	*false*	*maybe*	*true*
maybe	*maybe*	*maybe*	*true*
true	*true*	*true*	*true*

¬	
false	*true*
maybe	*maybe*
true	*false*

The existential (∃) and universal (∀) quantifiers behave like iterated ∨ and ∧, respectively. In addition, each comparator must be extended to handle nulls. The extension of equality has already been discussed: $x = y$ is defined to be maybe if either x or y or both are nulls. Other comparators are extended in a similar way. Finally, a logical expression has the value maybe if there is at least one substitution of the null values by nonnull values that yields the value true, and at least one substitution that yields the value false.

The extension of the set theoretical relational operations (e.g., union, intersection, difference, product) is straightforward, with the provision that all null values are considered identical. Thus two tuples that agree on their nonnull values are identified. The definition of projection is also unchanged, but note that after projection a relation may include a tuple with null values only. The (natural) join operation tests whether tuples in two relations agree on a particular subset of their attributes. As this evaluation may yield either true, false, or maybe, the extended join is defined as two different relations: the true-join includes the tuples for which the test yields true; the maybe-join includes the tuples for which the test yields maybe. An example of join is shown in Table 6 (the

join attributes are DEPARTMENT and D_NAME); the true-join is shown at the top, and the maybe-join at the bottom. Similarly, the extended selection is defined to have two parts: a true-selection and a maybe-selection.

There are several difficulties with the three-valued logic scheme, requiring careful reconsideration of various operations. For example, if x is a variable that ranges over an attribute with null values, then the comparison $x = x$ is no longer guaranteed to evaluate to true, and if R is a relation with null values, then the natural join of R with itself is no longer guaranteed to evaluate to R. The three-valued logic approach has been discussed and criticized[18].

MODELLING VALIDITY AND COMPLETENESS

A database is a model of the real world, and its designers and administrators invest continuous efforts to ensure that this model approximates the real world as accurately as possible. Still, if the database is large enough, it is almost certain to be an imperfect model. Consequently, the answers issued by the database system in response to queries are imperfect as well.

The question of the integrity of the information that is received from a database (or, for that matter, from any other source) is usually the primary concern of the user. Usually, this question has two parts:

- Is the information valid?
- Is the information complete?

For example, when the personnel database of a large corporation is requested to list all the employees in the accounting department, the concern is with the validity of the information received (is each person named indeed an employee of the accounting department?), as well as with its completeness (are there any other employees in the accounting department that were not listed?).

In other words, answers have integrity if they contain the whole truth (completeness) and nothing but the truth (validity). The issue of completeness is usually not addressed in database systems and the issue of validity only in part, through the mechanism of integrity constraints.

In general, integrity constraints are formulas in predicate calculus that express relationships that must be satisfied by the database[4]. Assuming that initially a database satisfies the constraints, thereafter update requests are accepted only if they do not violate any of the constraints. Such constraints enhance the validity of a database (and hence the validity of its answers), but they cannot ensure it. For example, it is not possible to express constraints that ensure that the information about each employee in accounting is valid, or that all the employees in accounting are included.

The concept of information completeness is related to the CWA mentioned earlier. Recall that the CWA was criticized on the grounds that usually it is unrealistic to assert it on the database as a whole. In practice, the CWA can only be made on some subsets of the database.

Recently, a new model for integrity was developed[19] that addresses these problems. New kinds of integrity constraints, called validity constraints and completeness constraints, were introduced. A validity constraint asserts that a particular subset of the database is guaranteed to be valid, and a completeness constraint asserts that a particular subset of the database is guaranteed to be complete. Together, validity and completeness constraints are assertions on the integrity of the database. (The relationship between validity and completeness and the previous terms imprecision and incompleteness will be discussed in the final section.)

In addition, the new integrity model is designed to certify the integrity of answers. For each answer issued in response to a query, it determines:

- whether the answer is valid (the information is ensured to be correct) or only partially valid (specified portions are ensured to be correct)
- whether the answer is complete (the information is ensured to include all the real-world occurrences) or only partially complete (specified portions are ensured to include all the real-world occurrences)

In effect, the database is qualifying its answers by the scope of its knowledge (as defined by the constraints).

As a simple example, assume a database with relations EMPLOYEE = (NAME, DEPARTMENT, SALARY) and DEPARTMENT = (D_NAME, MANAGER), and assume it satisfies two integrity constraints that guarantee valid information about employees in the accounting department, and complete information about employees in the departments managed by Mike. The answer to a query to retrieve all employees who earn over 35,000 is certified to be valid with regard to the accounting department, and complete with regard to the employees who work for Mike. Clearly, answers that are accompanied by such 'certification of quality' are more meaningful.

In discussing the new kinds of constraints, the intuitive terms database 'subsets' or 'portions' have been used. Formally, validity and completeness constraints are defined as views. A view is an expression in the relational algebra (or calculus) that defines a derived relation in terms of the database relations. Assume a database schema (i.e., a set of relation definitions) S, let e be a view of S, and let D be a database with scheme S (i.e., a set of relations that extend the definitions S). The derived relation obtained by applying the view e to the database D is denoted by e(D).

Assume now that the portion of the real world modelled by the database is represented in a hypothetical database W, and denote the actual database D (W and D have identical relation schema S, but their extensions are different). Let v and c be two views of S. The view v is a validity constraint satisfied by D if $v(D) \subseteq v(W)$. The view c is a completeness constraint satisfied by D if $c(D) \supseteq c(W)$. Hence validity and completeness are treated with perfect duality.

If both the database and the real-world environment it

models are static, then a database with integrity will always retain its integrity. In general, however, real-world environments are subject to changes, and databases must change accordingly. Such changes might violate the integrity of databases.

This integrity may be violated in two ways.

- First, through changes to the database that do not reflect actual changes in the real world. For example, an employee who works in a department managed by Mike is deleted from the database, thus violating the aforementioned completeness constraint, or the salary of an employee in the accounting department is changed in the database, thus violating the aforementioned validity constraint.
- Second, through changes in the real world that have not been incorporated into the database. For example, a new employee starts to work in a department managed by Mike, or the salary of an employee in the accounting department is raised.

Both kinds of violations can only be avoided by human supervision: the database administrator must monitor closely the views of the database that are governed by validity and completeness constraints, updating them to reflect all the changes in the real world, and nothing else.

Assume now validity and completeness constraints with the following restrictions:

- Views of validity constraints do not include any true tuples.
- Views of completeness constraints do not include any false tuples.

For example, assume that it is mandatory that employees of the design department earn a salary of at least 40,000. Then the view of the employees of the design department that earn less than 40,000 is guaranteed not to include any true tuples. Similarly, assume that it is mandatory that every programmer knows Pascal. Then the view of the programmers who know Pascal is guaranteed not to include any false tuples. (It is assumed that only values from the domain of programmers are used.) The integrity of such views can be monitored without human supervision: any update that inserts a tuple into a validity view of this kind (clearly, it is impossible to delete from such views), or any update that deletes an existing tuple from a completeness view of this kind (clearly, it is impossible to insert a new tuple into such a view) can be rejected automatically. Clearly, validity constraints of this kind correspond to 'traditional' integrity constraints. Thus the 'traditional' integrity model is subsumed in the validity component of the new integrity model.

A method for representing the new integrity constraints and inferring the validity and completeness of answers has also been described[19]. The definitions of the integrity constraints (i.e., view definitions) are stored in special relations, called meta-relations, whose structure mirrors the structure of the actual relations. Standard relational algebra operations, such as product, selection, and projection are extended to the meta-relations. When a query is submitted to the database, it is performed in parallel both on the standard relations, yielding the standard answer, and on the meta-relations, yielding a meta-answer. The meta-answer contains definitions of views of the answer which are guaranteed to be valid or complete. These definitions are then translated to statements of the validity and completeness of the answer and are presented alongside the answer.

DISCUSSION

The model described in the previous section is fundamentally different from those described in the second and third sections. First, the former models define incompleteness as unavailability of data, and imprecision as availability of approximative values (incompleteness, therefore, is a strong form of imprecision). The latter model defines validity as being correct. Thus, to a large extent, validity corresponds to precision and completeness. The latter model then defines completeness as having all the correct data. Second, while the former models accommodate imprecision and incompleteness by storing approximative or null values, the approach of the latter model is to describe the stored data that are valid or complete. Finally, while the former models extend query evaluation to match properly the stored representations of imprecise and incomplete data, the latter model extends query evaluation to define the data in the answer that are valid or complete.

Among the models presented in the second and third sections, the fuzzy database model is the most general. It defines a framework that incorporates data imprecision, as well as data similarity and null values. Of course, a fuzzy database requires more investment. Depending on the particular version of the model, membership grades must be provided for each tuple of each fuzzy relation, definitions for every fuzzy domain value and for every fuzzy comparator, and so on. The other approaches reviewed also require investment (though less extensive) in the definitions of similarities or probability distribution functions. In contradistinction, a database that incorporates only null values requires no auxiliary definitions.

A similar pattern is observed in the simplicity and efficiency of the required extensions to the query evaluation algorithms. Evaluating a vague query in a fuzzy database (and to a lesser degree, evaluating a vague query in any of the alternative approaches described in the second section) is much more laborious than evaluating a query in a database with null values.

Obviously, the fuzzy data model (and to a lesser degree the aforementioned alternatives) is more ambitious, and also more risky. The definitions of concepts and similarities that are provided by the database designers and administrators may not reflect the interpretations of these concepts and similarities as recognised by many database users. This may result in responses that do not match the intentions of the queries. In this respect, the

model of incompleteness in the third section is much 'safer'.

Data incompleteness is encountered in virtually every nontrivial application, and database designers and administrators who must decide how to cope with it have three basic options:

- discard every tuple not entirely definite
- complete unavailable data with default values or estimates
- represent incomplete data

The first two options are suitable mostly for non-critical applications. For example, a mailing list would include only complete postal addresses, a telephone list would assume a default area code wherever it is absent, and so on. However, the reality of most applications is that partial data are valuable and should not be discarded, while estimates are often unacceptable. Consequently, the conservative approach for critical applications is to represent incomplete data. As commercial database systems of the present generation offer little support for incompleteness, this representation is often through some *ad hoc* mechanism. It is expected, though, that database systems of the next generation will substantially increase their support in this area.

The situation with data imprecision is somewhat different. Essentially, there are four options for coping with imprecise data:

- discard every tuple not entirely definite
- ignore the uncertainty and assume the data are precise
- disregard the available approximation and treat the data as incomplete
- represent imprecise data

Again, the first two options are suitable mostly for non-critical applications (they correspond to the first two options for incompleteness). However, because of the aforementioned risks inherent in retrieving from databases that accommodate imprecision, quite often the conservative approach for critical applications is to treat imprecision as incompleteness. Systems that accommodate imprecision appear to be more suitable for specific applications: fuzzy databases can prove to be useful in intelligent systems, such as expert systems, and systems such as Ares and Vague can support intelligent user interfaces in applications where uncertainty of answers may be tolerated (e.g., where answers are considered suggestions, whose suitability is later verified by other means).

Research in these areas is continuing, with many of the research projects now addressing recent technologies and the opportunities and problems they suggest. In particular, investigators are looking at issues of imprecision and incompleteness in the context of the logic data model, a model that may be viewed as the extension of the relational data model with rules of inference[20].

ACKNOWLEDGEMENT

This work was supported in part by NSF Grant No IRI-8609912 and by an Amoco Foundation Engineering Faculty Grant.

REFERENCES

1 **Thompson, J P** *Data with semantics* Van Nostrand Reinhold (1989)
2 **Reiter, R** 'On closed world data bases' in *Logic and databases* Plenum Press (1978) pp 55–76
3 **Codd, E F** 'A relational model for large shared data banks' *Comm. ACM* Vol 13 No 6 (June 1970) pp 337–387
4 **Date, C J** *An introduction to database systems, Vol I* (fifth ed) Addison-Wesley (1990)
5 **Zadeh, L A** 'Fuzzy sets' *Inf. Control* Vol 8 No 3 (June 1965) pp 338–353
6 **Zadeh, L A** 'Fuzzy sets as a basic for a theory of possibility' *Fuzzy Sets Syst.* Vol 1 No 1 (1978) pp 3–28
7 **Prade, H and Testemale, C** 'Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries' *Inf. Sci.* Vol 34 No 2 (November 1984) pp 115–143
8 **Zemankova, M and Kandel, A** 'Implementing imprecision in information systems' *Inf. Sci.* Vol 37 Nos 1–3 (December 1985) pp 107–141
9 **Raju, K V S V N and Majumdar, A** 'Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems' *ACM Trans. Database Syst.* Vol 13 No 2 (June 1988) pp 129–166
10 **Motro, A** 'VAGUE: a user interface to relational databases that permits vague queries' *ACM Trans. Office Inf. Syst.* Vol 6 No 3 (July 1988) pp 187–214
11 **Ichikawa, T and Hirakawa, M** 'ARES: a relational database with the capability of performing flexible interpretation of queries' *IEEE Trans. Soft. Eng.* Vol 12 No 5 (May 1986) pp 624–634
12 **Buckles, B P and Petry, F E** 'A fuzzy representation of data for relational databases' *Fuzzy Sets and Syst.* Vol 7 No 3 (May 1982) pp 213–226
13 **Garcia-Molina, H and Porter, D** 'Supporting probabilistic data in a relational system' *Technical report TR-147* Princeton University, Princeton, NJ, USA (February 1988)
14 **Abiteboul, S and Grahne, G** 'Update semantics for incomplete databases' in *Proc. 11th Int. Conf. Very Large Data Bases* Stockholm, Sweden (21–23 August 1985) Morgan-Kaufmann (1985) pp 1–12
15 **Golshani, F** 'Growing certainty with null values' *Inf. Syst.* Vol 10 No 3 (1985) pp 289–297
16 **Maier, D** *The theory of relational databases* Computer Science Press (1983)
17 **Codd, E F** 'Extending the database relational model to capture more meaning' *ACM Trans. Database Syst.* Vol 4 No 4 (December 1979) pp 397–434
18 **Date, C J** 'NOT is not "not"!' in *Relational database writings 1985–1989* Addison-Wesley (1990)
19 **Motro, A** 'Integrity = validity + completeness' *ACM Trans. Database Syst.* Vol 14 No 4 (December 1989) pp 480–502
20 **Ullman, J D** *Database and Knowledge-Base Systems, Vol I* Computer Science Press (1988)