JUNE 1989 VOL. 12 NO. 2

a quarterly bulletin of the
Computer Society of the IEEE
technical committee on

Data Engineering

CONTENTS

SPECIAL ISSUE ON IMPRECISION IN DATABASES

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC
IEEE

IEEE COMPUTER SOCIETY

The LOTUS Corporation has made a generous donation to partially offset the cost of printing and distributing four issues of the Data Engineering bulletin.

Data Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Data Engineering . Its scope of interest includes: data structures and models, access strategies, access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Opinions expressed in contributions are those of the individual author rather than the official position of the TC on Data Engineering, the IEEE Computer Society, or organizations with which the author may be affiliated.

Membership in the Data Engineering Technical Committee is open to individuals who demonstrate willingness to actively participate in the various activities of the TC. A member of the IEEE Computer Society may join the TC as a full member. A non-member of the Computer Society may join as a participating member, with approval from at least one officer of the TC. Both full members and participating members of the TC are entitled to receive the quarterly bulletin of the TC free of charge, until further notice.

Message from the TC Chair.

I would like to take this opportunity to give you a status report on our TC and to let you know of some plans that will affect our TC in the near future. In 1988, the IEEE Computer Society's Technical Activities Board (TAB) initiated an advertising campaign in IEEE Computer with a Special Section describing the TAB activities, especially those of the Technical Committees including our own, Data Engineering. From November 1988 through April, 1989 our membership grew by 55%, from 939 to 1455.

Our TC has been active in sponsoring a number of Conference and Symposia, and our cooperation has been sought by several conferences and workshops. The TCDE is sponsoring or co-sponsoring the following conferences during 1988-1989:

International Conference of Data Engineering
International Conference on Very Large Data Bases
International Symposium on Databases in Parallel and Distributed Systems
International Conference on Data and Knowledge Systems for Manufacturing and Engineering
International Conference on Expert Database Systems.

We are cooperating with the Symposium on the Design and Interpretation of Large Spatial Databases.

As our numbers grow, and our activities increase, so too do our costs, primarily those associated with the publication of the Data Engineering Bulletin. Last year we were fortunate to have the sponsorship of the Lotus Development Corp. which underwrote three issues. This year the costs are being absorbed by our TAB budget allotment and Conference earnings. However, our income does not presently cover all of our expenses. Therefore, we will propose to the IEEE Computer Society Board of Governors the enactment of a Dues Policy for TC Members. These dues will provide a guaranteed minimum amount that will sustain the publication of our Bulletin. We hope that you will support the new policy once it takes effect.

IEEE Computer Society Task Force on Neural Networks

The IEEE Computer Society Task Force on Neural Networks seeks active researchers, scientists, application developers, and project managers to participate in the task force effort. The task force needs technical contributions and welcomes active participation from government, academy, and industry. To volunteer, or for further information, please contact

Dr. Kamal N. Karna
Chairman and Coordinator
IEEE/CS Task Force on Neural Networks
Computer Communications and Graphics
823 Flagler Drive
Gaithersburg, MD 20878-1938.

Larry Kerschberg

Letter from the Editor

Most research in database systems is based on assumptions of precision and specificity of both the data stored in the database, and the requests to retrieve data. In reality, however, both may be imprecise or vague. Considerable amount of research has focused on the issues of imprecision and vagueness in databases, and this issue of Data Engineering is devoted to this topic.

The wide variety of approaches to issues of imprecision is evidenced in part by the proliferation of the terms used to describe data and retrieval specifications which are not entirely "crisp", including vague, uncertain, imprecise, incomplete, fuzzy, approximate, and ambiguous. Still, much of the work in this broad area falls into one or more of these three categories.

The first category includes formal models with constructs for representing imprecise data and for expressing imprecise retrieval requests. Suggested representations for imprecise data include intervals of values, "fuzzy" values (with appropriate definitions), values accompanied by certainty factors, and null values (of one kind or another). Retrieval requests for matching imprecise data often include a threshold value, that provides the degree of specificity of the request. Fuzzy values, similar to those allowed in the representation, are also allowed in queries. Note, that this category includes the considerable body of work on null values.

The second category includes systems that enable specification of imprecise queries in databases that contain only crisp data. To satisfy such queries these systems employ some kind of mechanism for determining proximity among data items of the same domain. The query language then provides constructs for specifying the desirable values that should be retrieved, and a process of "weak matching" is used to satisfy queries. This approach has been demonstrated with several experimental database systems. Note, that this category also includes various information retrieval systems (such as those intended for bibliographic searches).

The models and systems in the first two categories assume that the user has a precise notion of what he or she is looking for. The last category includes systems for dealing with retrieval requests whose imprecision is a result of vagueness of the retrieval goal itself (vagueness is in the user's mind, so to speak). Systems in this category include various kinds of browsers, that allow users to explore the contents of the database even without specific retrieval goals, and interactive query construction aids, that assist in crystalizing vague retrieval goals into specific queries.

Four of the seven papers in this issue fall into the first category. Prade and Testemale review a general approach for handling imprecision, based on possibility theory. Zemankova describes FIIS, a knowledge-based system extended to deal with various aspects of imprecision. Tzvielli discusses various problems associated with the representation and access of uncertain data. And Imielinski discusses incompleteness in logical databases. In the second category, Eastman compares approximate retrieval in information retrieval systems and in database management systems. In the third category, D'Atri and Tarantino discuss three

styles of interaction that are suitable for users who lack sufficient knowledge to express formal queries. Finally, I review three user interfaces in the second and third categories.

As these papers demonstrate, this important area is currently very active, with many of the research projects now addressing recent technologies and the opportunities and problems they suggest. Examples include issues of fuzziness and incompleteness in knowledge-rich databases, and browsing interfaces for object-oriented databases.

Ami Motro
University of Southern California
May 1989

The possibilistic approach to the handling of imprecision in database systems

Henri PRADE - Claudette TESTEMALE

Laboratoire Langages et Systèmes Informatiques
Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier, 118 route de Narbonne
31062 TOULOUSE Cedex (FRANCE)

Abstract

In this paper we are interested in both the handling of flexible requests and the management of data pervaded by imprecision, uncertainty or vagueness. The matching of an item of data against a request is no longer an all-or-nothing process. A degree of matching reflects our lack of certainty that the item of data satisfies the request, and may be due either to the fact that the available information in the data is insufficient or to the fact that the item of data corresponds only approximately to what is requested. All these facets of the treatment of imprecision in databases are dealt with in the framework of possibility theory. This paper discusses previous and recent works with retrospection and points out the main references where the ideas are developed in detail.

1 - Introduction

There have been many attempts to introduce various kinds of flexibility in the handling of database queries (Kunii, 1976 ; Tahani, 1977 ; Chang, 1982 ; Ichikawa and Hirakawa, 1986 ; Motro, 1988).

These research works are motivated by different reasons. We may want to avoid null answers to a query by broadening the scope of the search, thus finding items which are compatible with a relaxed interpretation of the query. In addition, vague predicates are often used in natural languages and it may be desirable to keep their meaning flexible in the treatment of the query. Moreover, the vagueness of the predicates suggests that their satisfying is a matter of degree, which induces an ordering among the items which more or less correspond to the query. Lastly, in the case of a multi-criteria request, it is possible that the different criteria have not the same importance.

At the same time, other research works have focused on the problem of accommodating null values (values that are unknown or do not apply), e.g. (Codd, 1979), partial information on the values of attributes (Lipski, 1979) and uncertain information (Wong, 1982). Indeed, the available information about items may be neither precise nor certain and it is still desirable to take into account all the existing information when answering a query.

In the following we introduce the main ideas and survey the main results of an approach, based on possibility theory and developed by the authors (Prade and Testemale, 1984, 1987a), which enables us to deal with flexible queries as well as imprecise and uncertain data. This

approach which has been also considered by other authors, particularly (Zemankova and Kandel, 1988) and (Bosc et al., 1988), proposes a unique framework for managing the points discussed above.

The next section considers the problem of flexible requests. Section 3 is devoted to the modeling and the management of imprecise and uncertain data. After the presentation of the main principles in sections 2 and 3, section 4 points out various extensions of the model, to handle more complex queries or knowledge.

2 - Handling flexible queries

In this section, it is assumed that each attribute is single-valued and the precise value of each attribute is available for each item in the database.

In case of a request with a non-vague specification, which is not satisfied by any item, we may look for existing items which are close, in some sense, to some ideal item that satisfies this request. For instance, if we are looking for people who are at least 30 years old, and if there are no such people in the database, we may according to the context enlarge the query and accept a 29 years old person as satisfying the query. In such an example, it is clear that the distance between 30 and the age of the considered person is an indication of the relevance of the item. More precisely, from a distance defined on the attribute domain and a threshold, we define a tolerance relation which enables us to express if an attribute value is sufficiently close to a value which is fully compatible with the query.

However, there does not always exist a "natural" distance which can be defined on a domain and the choice of the threshold may be regarded as somewhat arbitrary. The introduction of fuzzy tolerance relations (Cayrol et al., 1982 ; Buckles and Petry, 1982) attached to attribute domains may be a means to overcome these two problems. Indeed, given an attribute domain $\mathcal{D}$, a fuzzy tolerance relation T may be defined through its membership function μ_T from $\mathcal{D} \times \mathcal{D}$ to [0,1], such that the closer (or the more interchangeable) the values d and d', the closer to 1 the degree of membership $\mu_T(d,d')$. The value of μ_T may be given explicitely by an expert for each pair (d,d') for discrete domains, when no distance is available, or μ_T may be built from a distance, especially on continuums (e.g. closed sub-intervals of the real line). For instance, given a distance δ, we may define μ_T as

$$\mu_T(d,d') = \max\left(0, 1 - \frac{\delta(d,d')}{\lambda}\right)$$

where λ is a positive real number. Then, given a request asking for a selection of items having their attribute value in an ordinary subset R of $\mathcal{D}$, the composition $T \circ R$ defined by

$$\mu_{T \circ R}(d') = \sup\{\min(\mu_T(d,d'), \mu_R(d)), d \in \mathcal{D}\}$$

(where μ_R is the characteristic function of R), enlarges the scope of the request in a fuzzy way. Note that $\mu_{T \circ R}(d') = 1$ if and only if d' is considered as perfectly interchangeable with a value which belongs to R (or such that the distance $\delta(d,d')$ equals zero if we are using the expression of μ_T suggested above). More generally, $\mu_{T \circ R}(d')$ will indicate to what extent an item whose attribute value is d' may be considered as admissible as an answer to the request ; this gives a natural way for ranking the items with respect to a request. The fuzzy tolerance relation T acts as a fuzzy threshold since we have now a gradual transition between values that lead us to accept the corresponding items and values that lead us to reject them. Moreover, in case of a parametrized fuzzy relation, it is still possible to relax (or diminish) the tolerance by modifying the value of the parameter λ upon request.

By using a fuzzy tolerance relation T, an ordinary subset R specified in the request is replaced by a fuzzy subset T ∘ R. We may also allow more direct fuzzy specifications of subsets in the request. The advantage is that the flexibility which is introduced applies only to the request without being attached to the attribute domain in general. This applies to any kind of domain (whether discrete or not, whether ordered or not). For more discussions along this line, see (Prade and Testemale, 1987c).

Generally speaking, a fuzzy set P defined on $\mathcal{D}$ through its membership function μ_P can be viewed as a nested collection of ordinary subsets P_α such that

$$P_\alpha = \{d \in \mathcal{D}, \mu_P(d) \geq \alpha\} \text{ for } \alpha \in]0,1]$$

(Indeed if $\alpha > \alpha'$, we have $P_\alpha \subset P_{\alpha'}$). More particularly, the core of P defined by $P_1 = \{d \in \mathcal{D}, \mu_P(d) = 1\}$ and the support of P defined by $Supp(P) = \{d \in \mathcal{D}, \mu_P(d) \geq 0\}$ are worth considering. All values which are at least somewhat admissible are in the support and the core includes only the most preferred values. (This can be viewed as an extension of the idea of preference developed in (Lacroix and Lavency, 1987)). Here, the idea of preference is graded by μ_P for the values which are in the support without being in the core.

So far, we have considered elementary conditions pertaining to one attribute. The case of compound conditions expressed via logical expressions is dealt with using fuzzy set operations. Let $A_i(x)$ be the value of the attribute A_i for the item x and P_i the subset expressing the restriction for $A_i(x)$, in the request. Conjunctive (resp. disjunctive) aggregations of the elementary degrees of matching $\mu_{P_i}(A_i(x))$ are performed applying min (resp. max) operation to the degrees ; $1 - \mu_{P_i}(A_i(x))$ represents the extent to which $A_i(x)$ belongs to the complement of P_i. In some applications, we may like to express that some elementary conditions are less important than others. In this case, conjunctive and disjunctive aggregations are, respectively, generalized by

$$\min_i \max(\mu_{P_i}(A_i(x)), 1 - \omega_i)$$

$$\max_i \min(\mu_{P_i}(A_i(x)), \omega_i)$$

where ω_i is a weight of importance of the condition bearing on the attribute A_i in the request ; see (Dubois et al., 1988b) for justifications and related discussions. The weights are supposed to satisfy the normalization condition $\max_i \omega_i = 1$. Clearly, when all the elementary conditions are equally important, (i.e. $\forall i, \omega_i = 1$), the two operations above reduce, respectively, to min and max. When $\omega_i = 0$, there is no condition on the attribute A_i. We observe, in the case of the conjunctive combination, that even if $A_i(x)$ fails to satisfy the restriction P_i of importance ω_i, the global result of the combination would not be penalized below $1 - \omega_i$.

However, conjunction and disjunction operations, other than min and max, can be used. There exist more drastic conjunction operations (e.g. the product) and less drastic disjunction operations (e.g. the "probabilistic sum" $a + b - a \cdot b$). There also exist many intermediary operations between min and max (e.g. the arithmetic mean) which can model compensatory 'and' for instance (a low degree of satisfaction for one elementary condition can be somewhat balanced by a high degree of satisfaction for another condition). The reader is referred to Chapter 3 of (Dubois and Prade, 1988b) for a complete presentation of the existing operations and of a procedure which enables us to elicitate the right operation in practical cases (e.g. which kind of 'and' the author of the request has in mind : drastic, logical, compensatory, etc.).

3 - Treatment of incomplete and uncertain information

In our approach, the available information about the value of a single-valued attribute A for an item x will be represented by a possibility distribution $\pi_{A(x)}$ on $\mathcal{D} \cup \{e\}$ where e is an

extra element which stands for the case when the attribute does not apply to x. The possibility distribution $\pi_{A(x)}$ may be viewed as a fuzzy restriction of the possible value of A(x) ; $\pi_{A(x)}$ is a mapping from $\mathcal{D} \cup \{e\}$ to [0,1].

For instance, the information 'Paul is young' will be represented by :

$$\begin{cases} \pi_{Age(Paul)}(e) = 0 \\ \pi_{Age(Paul)}(d) = \mu_{young}(d) \qquad \forall\, d \in \mathcal{D} \end{cases}$$

where μ_{young} is a membership function which represents the vague predicate 'young' in a given context.

It is important to notice that the values restricted by a possibility distribution are considered as *mutually exclusive*. The degree $\pi_{A(x)}(d)$ rates the possibility that $d \in \mathcal{D}$ is the right value of the attribute A for x. $\pi_{A(x)}(d) = 1$ only means that d is a completely possible value for A(x), but does *not* mean that it is certain that d is the value of A for x, except if $\forall\, d' \neq d$, $\pi_{A(x)}(d') = 0$. Moreover the possibility distribution $\pi_{A(x)}$ is supposed to be normalized on $D \cup \{e\}$, i.e. $\exists\, d \in \mathcal{D}$ such that $\pi_{A(x)}(d) = 1$ or $\pi_{A(x)}(e) = 1$, since either at least one value of the attribute domain is completely possible, or the attribute does not apply.

This approach proposes a unified framework for representing precise values of attributes, partial (but non-fuzzy) values as well as fuzzy information concerning the value of attributes, and the following null value situations :

i) the value of A for x is completely unknown : $\forall d \in \mathcal{D}, \pi_{A(x)}(d) = 1, \pi_{A(x)}(e) = 0$

ii) the attribute A does not apply to x : $\forall d \in \mathcal{D}, \pi_{A(x)}(d) = 0, \pi_{A(x)}(e) = 1$

iii) we don't know whether the situation is i or ii : $\forall\, d \in \mathcal{D}, \pi_{A(x)}(d) = 1, \pi_{A(x)}(e) = 1$.

From the possibility distributions $\pi_{A(x)}$ and a subset P (non fuzzy or fuzzy), we can compute the fuzzy set $\prod P$ (resp. NP) of the items whose A-value possibly (resp. necessarily) satisfies the condition P.

The membership degree of an item x to $\prod P$ and NP are respectively given by (Dubois and Prade, 1988b) :

$$\mu_{\prod P}(x) = \Pi(P\,;\, A(x)) = \sup_{d \in \mathcal{D}} \min(\mu_P(d), \pi_{A(x)}(d))$$

$$\mu_{NP}(x) = N(P\,;\, A(x)) = \inf_{d \in \mathcal{D} \cup \{e\}} \max(\mu_P(d), 1 - \pi_{A(x)}(d))$$

Note that $\prod P$ and NP always satisfy the inclusion relation $NP \subseteq \prod P$, provided that $\pi_{A(x)}$ is normalized, i.e.

$$\forall x, \mu_{NP}(x) \leq \mu_{\prod P}(x).$$

Thus, in case of incomplete information, we are able to compute the fuzzy set of items which (more or less) *possibly* satisfy an elementary condition and to distinguish among them the items for which we are more or less *certain* that they satisfy this condition. Note that here possibility and necessity are matters of degree. In case of non-fuzzy requests (i.e. P is a non fuzzy subset of $\mathcal{D}$), a stronger inclusion holds since then NP is included in the core of $\prod P$. When the information is precise, i.e. $\pi_{A(x)}$ is equal to 1 for one element d and is 0 elsewhere in $\mathcal{D} \cup \{e\}$, it can be checked that $\mu_{\prod P}(x) = \mu_P(A(x)) = \mu_{NP}(x)$.

Selections involving disjunction, conjunction or negation of elementary conditions can be

handled using the following basic relations of possibility theory :

$$N(P ; A(x)) = 1 - \Pi(\bar{P} ; A(x))$$
$$N(P_1 \times P_2 ; A_1(x) \times A_2(x)) = \min(N(P_1 ; A_1(x)), N(P_2 ; A_2(x)))$$
$$\Pi(P_1 + P_2 ; A_1(x) \times A_2(x)) = \max(\Pi(P_1 ; A_1(x)), \Pi(P_2 ; A_2(x)))$$
$$N(P_1 + P_2 ; A_1(x) \times A_2(x)) = \max(N(P_1 ; A_1(x)), N(P_2 ; A_2(x)))$$
$$\Pi(P_1 \times P_2 ; A_1(x) \times A_2(x)) = \min(\Pi(P_1 ; A_1(x)), \Pi(P_2 ; A_2(x)))$$

where

- the attribute A_i and the subset P_i ($i = 1,2$) refer to the same domain,
- the overbar denotes set complementation (defined by $\mu_{\bar{P}}(d) = 1 - \mu_P(d)$),
- $A_1(x) \times A_2(x)$ denotes extended Cartesian product defined by

$$\pi_{A_1(x) \times A_2(x)}(d_1,d_2) = \min(\pi_{A_1(x)}(d_1), \pi_{A_2(x)}(d_2))$$

  $P_1 \times P_2$ is similarly defined,
- $P_1 + P_2 = \overline{\bar{P}_1 \times \bar{P}_2}$ expresses a disjunctive condition, namely

$$\mu_{P_1+P_2}(d_1,d_2) = \max(\mu_{P_1}(d_1), \mu_{P_2}(d_2)).$$

Note that the above expressions of $N(P_1 + P_2 ; A_1(x) \times A_2(x))$ and $\Pi(P_1 \times P_2 ; A_1(x) \times A_2(x))$ require the logical independence of the attribute values respectively restricted by $\pi_{A_1}(x)$ and $\pi_{A_2(x)}$, to be valid.

These combination formulas are consistent with the fuzzy set operations (based on min and max) considered in section 2, when the available information becomes precise, since then the measures of possibility and necessity become equal to a membership degree, as pointed out above.

The case of other combination operations in compound requests (e.g. product, arithmetic mean), for which no decomposition formula exists for the possibility and necessity measures in presence of incomplete information, can be dealt with by using a fuzzy-real-valued compatibility degree for estimating the agreement between the information and what is required. Then an extended version of the considered combination operation is performed on these fuzzy real values and finally a possibility and a necessity degree can be extracted in a standard way from the global compatibility measure which has been thus computed. The reader is referred to Chapter 3, pp. 98-99 and Chapter 4, pp. 125-126 of (Dubois and Prade, 1988b), for detailed definitions and justifications.

For simplicity, in sections 2 and 3, we have focused on the selection operation. Queries demanding an extended join operation on relational tables containing fuzzy information are discussed in (Prade and Testemale, 1984, 1988 ; Dubois and Prade, 1988b, Chapter 6).

4 - Various extensions of the approach

In our approach, multiple-valued attributes can be treated formally in the same manner as single-valued ones, using possibility distributions defined on the power set of the attribute domains rather than on the attribute domains themselves. Indeed, in the case of multiple-valued attributes, the mutually exclusive possibilities are represented by subsets of values. However, it is possible to approximate a possibility distribution on the power set of a domain in terms of two fuzzy subsets of the domain which represent the values which are more or less certainly (resp. possibly) part of the multiple-value of the attribute (Dubois and Prade, 1988a). An approach based

on these ideas has been proposed in information retrieval, where a document is described in terms of key words which are more or less certainly relevant for it, and in terms of key words which are only more or less only possibly relevant for it (Prade and Testemale, 1987b).

The notion of cardinality has been extended to fuzzy sets and ill-known sets (sets for which are only known more or less possible elements and more or less certain elements), see e.g. (Dubois and Prade, 1985). This enables us to handle queries involving cardinalities of sets of items (Prade, 1984). Besides, queries asking for the satisfaction of *most* of the *important* elementary conditions of a compound pattern are discussed by (Kacprzyk and Ziolkowski, 1986) and (Dubois et al., 1988b).

The framework of possibility theory allows the representation of more complex knowledge, like dependency relations pervaded with imprecision and uncertainty (Raju and Majumdar, 1988 ; Prade and Testemale, 1987a). If-then rules expressing dependencies can be used to produce plausible estimate for missing values, in an analogical reasoning procedure (Arrazola et al., 1988).

5 - Concluding remarks

Possibility theory offers a powerful tool for the representation and the treatment of flexible queries as well as partial information.

In spite of the apparent complexity of the expressions of the possibility and necessity degrees, the approach is computationally tractable at least when we restrict ourselves to possibility distributions which are defined on small-sized discrete domains or whose shape is trapezoidal when the domain is a continuum. The approach is robust due to the use of the operations max and min, which are not very sensitive to small variations. In practice, it is sufficient to elicitate possibility distributions in a rough way ; i.e. identify what values are completely impossible, what are the values which are the most possible ones, and then remember that it is mainly the ordering of possibility degrees with is meaningful in possibility theory. Moreover, extensions of indexation techniques have been proposed (Bosc and Galibourg, 1987).

Lastly, let us mention the close relationship between possibility theory-based approaches to incomplete information systems and Lipski's approach (Lipski, 1979) ; see (Dubois et al., 1988a) for a preliminary investigation of this relationship, bridging the gap between possibility theory and modal logic.

References

Arrazola I., Plainfossé A., Prade H., Testemale C. (1988) Extrapolation of fuzzy values from incomplete data bases. In : Tech. Rep. n° 298, Laboratoire L.S.I., Univ. P. Sabatier, Toulouse. To appear in Information Systems.

Bosc P., Galibourg M. (1987) Indexing principles for a fuzzy data base. Proc. 2nd Inter. Fuzzy Systems Assoc. Congress (IFSA), Tokyo, Japan, 653-656.

Bosc P., Galibourg M., Hamon G. (1988) Fuzzy querying with SQL : extensions and implementation aspects. Fuzzy Sets and Systems, 28(3), 333-349.

Buckles B.P., Petry F.E. (1982) A fuzzy representation of data for relational databases. Fuzzy Sets and Systems, 5, 213-226.

Cayrol M., Farreny H., Prade H. (1982) Fuzzy pattern matching. Kybernetes, 11, 103-116.

Chang C.L. (1982) Decision support in an imperfect world. In : Tech. Rep. RJ3421 (40687), IBM Research Lab., Computer Science, San Jose.

Codd E.F. (1979) Extending the database relational model to capture more meaning. ACM Trans. Database Systems, 4(4), 397-434.

Dubois D., Prade H. (1985) Fuzzy cardinality and the modelling of imprecise quantifications.

Fuzzy Sets and Systems, 16, 199-230.
Dubois D., Prade H. (1988a) On incomplete conjunctive information. Computers and Mathematics with Applications, 15(10), 797-810.
Dubois D., Prade H. (1988b) Possibility Theory : an Approach to Computerized Processing of Uncertainty. Plenum Publ. Comp., 1988, (French version : 1st ed. 1985, 2nd ed. 1987).
Dubois D., Prade H., Testemale C. (1988a) In search of a modal system for possibility theory. Proc. 8th Europ. Conf. on Artificial Intelligence, Münich, 501-506.
Dubois D., Prade H., Testemale C. (1988b) Weighted fuzzy pattern matching. Fuzzy Sets and Systems, 28(3), 313-331.
Ichikawa T., Hirakawa M. (1986) ARES : a relational database with the capability of performing flexible interpretation of queries. IEEE Trans. on Sofware Engineering, 12(5), 624-634.
Kacprzyk J., Ziolkowski A. (1986) Database queries with fuzzy linguistic quantifiers. IEEE Trans. on Systems, Man and Cybernetics, 16, 474-479.
Kunii T.L. (1976) Dataplan : an interface generator for database semantics. Information Sciences, 10, 279-298.
Lacroix M., Lavency P. (1987) Preferences : putting more knowledge into queries. Proc. 13th Very Large Data Bases Conf., Brighton, 217-225.
Lipski W., Jr. (1979) On semantic issues connected with incomplete information databases. ACM Trans. on Database Systems, 4(3), 262-296.
Motro A. (1988) VAGUE : a user interface to relational databases that permits vague queries. ACM Trans. on Office Information Systems, 6(3), 187-214.
Prade H. (1984) Lipski's approach to incomplete information databases restated and generalized in the setting of Zadeh's possibility theory. Information Systems, 9(1), 27-42.
Prade H., Testemale C. (1984) Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries. Information Sciences, 34, 115-143.
Prade H., Testemale C. (1987a) Representation of soft constraints and fuzzy attribute values by means of possibility distributions in databases. In : Analysis of Fuzzy Information - Vol. 2 : Artificial Intelligence and Decision Systems (J.C. Bezdek, ed.), CRC Press, Boca Raton, Fl., 213-229.
Prade H., Testemale C. (1987b) Application of possibility and necessity measures to documentary information retrieval. In : Uncertainty in Knowledge-Based Systems (B. Bouchon, R.R. Yager, eds.), Springer Verlag, 265-274.
Prade H., Testemale C. (1987c) Fuzzy relational databases : representational issues and reduction using similarity measures. J. of Amer. Soc. for Information Systems, 38(2), 118-126.
Prade H., Testemale C. (1988) Approche possibiliste des informations incomplètes et des requêtes flexibles. Modèles et Bases de Données (AFCET, Paris), n° 8, 3-17.
Raju K.V.S., Majumdar A.K. (1988) Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. ACM Trans. on Database Systems, 13(2), 129-166.
Tahani V. (1977) A conceptual framework for fuzzy query processing - A step toward very intelligent database systems. Information Processing and Management, 13, 289-303.
Wong E.A. (1982) Statistical approach to incomplete information in database systems. ACM Trans. on Database Systems, 7(3), 470-488.
Zadeh L.A. (1978) Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems, 1(1), 3-28.
Zemankova M., Kandel A. (1984) Fuzzy Relational Data Bases - A Key to Expert Systems. Verlag TÜV Rheinland, Köln.

FIIS: A FUZZY INTELLIGENT INFORMATION SYSTEM

Maria Zemankova

Department of Computer Science
University of Tennessee
Knoxville, TN 37996-1301, U.S.A.*

Abstract

FIIS (Fuzzy Intelligent Information System) is designed with the goal of modeling human information processing. The system is composed of a relational database and a knowledge base that contains descriptions of fuzzy sets, domain similarity relations and rules. These concepts are employed in intelligent query processing, based on flexible inference that supports approximate matches between the data in the database and the query. The design issues also include customization of the system to suit the needs of individual users and explanatory capabilities.

1. INTRODUCTION

One of the distinguishing features of human intelligence is the ability to reason with uncertain data or knowledge. Hence, any intelligent information system has to incorporate management of uncertainty in its design. Systems such as ARES [10] and VAGUE [18] concentrate on the application of domain similarity in flexible interpretation of queries in order to avoid loss of close matches that would not satisfy crisp queries. Query generalization [8, 17], reformulation [5, 8, 25], or value extrapolation [1] are other approaches to intelligent query processing that are capable of providing meaningful answers where classical approaches would not yield any answers or a very restricted subset of the possible answers.

It has been demonstrated that fuzzy sets theory is a suitable framework for representation and manipulation of uncertainty in data bases [2, 4, 11, 12, 19, 21, 29, 30], information retrieval systems [3, 4, 12], expert systems [7, 9, 20, 28], and other applications, mainly developed in Japan [23, 24]. Some of the criticisms of fuzzy set theory concentrate on the subjectivity of assigning membership functions to concepts. This concern, although justified, is not unique to the fuzzy sets approach to modeling of uncertainty: knowledge acquisition in general is a recognized bottle neck in building knowledge-based systems. The goal of machine learning is to overcome this difficulty [13]. In particular, learning of imprecise concepts is receiving more attention. In this connection, it is interesting to note that the "flexible" interpretations of logical connectives [14] are very similar to those proposed by Zadeh in 1965 [26]. Responding to the criticisms and to the current trends in AI, a simple form of "learning from examples" is used in FIIS to acquire descriptions of fuzzy sets [31].

The model of an intelligent information system described here is based on a deductive relational database model [15], concepts of similarity [4, 21, 29], fuzzy sets [26, 27], and approximate reasoning [7, 28]. Missikoff and Wiederhold [16] propose a

---

* Currently: National Science Foundation, Database and Exp. Systems,Washington, D.C. 20550

unified approach for expert and database systems and specify criteria that a system needs to satisfy in order to meet this goal. The criteria satisfied by FIIS are: application independence, internal knowledge management, knowledge expandability, deductive power, manipulation of degrees of confidence, transparent reasoning, and multi-user environment. A very important issue of alternative search strategies is not addressed in the current system. However, a planned merge with the APPLAUSE system [6] that is specifically designed to conduct alternative reasoning strategies would produce a comprehensive intelligent information system.

2. INTELLIGENT INFORMATION SYSTEM

2.1 FIIS Structure and Functions

The system is divided into the Data Base (DB), the Knowledge Base (KB), and the Inference Engine (IE). The DB is a relational database. The KB and IE are based on the fuzzy set theory in order to model human knowledge and reasoning that are in most cases approximate (or fuzzy) in nature. It is possible to customize the knowledge representations stored in the KB to reflect a particular user's perception of data in the DB, as well as to adjust the flexibility of query evaluation. FIIS's explanatory capabilities show the reasoning involved in query evaluations in a step-by-step fashion. A prototype model has been implemented in Prolog [22].

Currently, the KB is built either by the user providing *explicit definitions* of concepts, or *examples* of concepts, which the system completes to concept definitions (learning). Concepts previously defined (*i.e.*, learned by the system) can be used in specifying (*i.e.*, teaching) new, more complex concepts, hence the system possesses the ability of incremental learning.

Since the new concepts defined in the KB enhance the expressive power of the query language, it can be said that FIIS is a system with a growing language. The query language is a logic-based language [15] where high-level aggregate constructs allow natural language-like query specification built from names of attributes, attribute values, fuzzy comparison operators (*e.g.*, MUCH GREATER), relations, fuzzy sets, rules, logical connectives, and fuzzy modifiers (*e.g.*, VERY, MORE-OR-LESS) [30]. Query evaluation is performed by the IE by application of approximate reasoning. A match measure between facts and the query is computed, and facts that satisfy (match) the query with a degree greater than or equal to a pre-specified threshold of acceptance are retrieved.

2.2 Data Base

The Data Base (DB) stores the actual data values, or facts as *relations*. The user is prompted for a name of the new relation and names of the *attributes* which form the relation. If an attribute has not been used in a previously defined relation, the user is asked to define its *domain*. There are three *domain types:*

scalar - attribute values are usually character strings

numeric - attribute values are integer or real numbers

unit - attribute values are the unit interval [0,1], indicating a fuzzy set membership value.

Domains may contain atomic values only. A scalar domain may be either unrestricted, or an enumerated set of scalar values. Numeric domains can be open (i.e. all legal numeric values are allowed), or a range can be specified.

As an example, consider relations PERSON and P_SKILL. The relation PERSON has attributes (NAME, AGE, INTELLIGENT). The attribute NAME has an unrestricted textual (scalar) domain. The domain of the attribute AGE is a numeric range $\{0 \leq x \leq 100\}$, and the domain of the attribute INTELLIGENT is the unit interval [0,1] where a value represents the degree of membership in a fuzzy set INTELLIGENT. The relation P_SKILL has attributes (NAME, EXPERTISE), where the attribute NAME is the same as in the relation PERSON, and the attribute EXPERTISE has an enumerated scalar domain {DBMS, EXPSYS, AI, UNCERT(ainty)}. An attribute used in more than one relation (*e.g.*, NAME) is assumed to have the same domain and the same semantics in all relations. This permits a natural join to be performed automatically when a query requests data from more than one relation.

2.3 Knowledge Base

The KB is a collection of *similarity* and *proximity relations*, *fuzzy set definitions*, and *rules*. Concepts defined in the Knowledge Base provide the interpretation of terms used in queries, hence they serve as a link between the facts in the Data Base, the concepts from the Knowledge Base, and the Inference Engine that drives the intelligent query processing.

2.3.1 Similarity and Proximity Relations

Similarity and proximity relations are used in the implementation of approximate matching, which forms the basis for approximate reasoning.

Similarity Relation

Since similarity is a symmetric and reflexive relation (however, in general it is not transitive for unordered domains), users are only prompted for pair-wise similarities corresponding to the upper triangular elements in a similarity matrix. For example, for the above attribute EXPERTISE, the similarity definition process is as follows:

Value1	Value2	Similarity
DBMS	EXPSYS	0.6
DBMS	AI	0.4
DBMS	UNCERT	0.2
EXPSYS	AI	0.9
EXPSYS	UNCERT	0.5
AI	UNCERT	0.7

Similarity is used in answering queries using fuzzy comparison operators. For example, if a query specifies to retrieve a PERSON whose EXPERTISE IS AI (rather than EXPERTISE = AI) with the threshold of acceptance set to 0.7, a PERSON whose EXPERTISE is AI or EXPSYS or UNCERT would be retrieved, based on the similarity relation shown above.

The tabular form of the similarity relation definition can be also used for ordered domains (e.g. numeric domains, or a domain of letter grades A, B, C, D, F). In this case, similarities of direct successors are requested, and similarities of the remaining pairs are computed by the application of min-max similarity transitivity relation [4, 21].

Absolute and relative proximity relations

Absolute proximity relation, *absprox*, assigns the same proximity value to equally distant points in a domain. On the other hand, relative proximity relations, *Irelprox* and *Drelprox*, assign increasing/decreasing values of proximity to equally distant points further from the lower (L)/upper (U) boundary of the domain. To illustrate a case where an increasing relative proximity is desirable, consider the age of human beings. In many contexts, persons who are 31 and 32 years old are more similar to each other than persons who are 1 and 2 years old.

Absolute proximity and increasing relative proximity relations are specified by the parametric functions (1) and (2) , respectively [29, 30]:

$$absprox(x, y) = \exp(-p * |x - y|) \qquad (1)$$

$$Irelprox(x, y) = \exp\left(-p * \left|\frac{x - L}{y - L} - \frac{y - L}{x - L}\right|\right) \qquad (2)$$

where $p > 0$, x, y are elements of a domain [L, U], and $x, y > L$ or $x, y < U$.

As the behavior of these functions is not straightforward they should not be manipulated by casual users of the system. It is difficult to specify the value of the parameter p, in order to obtain the desired degree of proximity values in the numeric domain for which proximity is being defined. Instead, the user is asked to provide an example of a proximity value between two domain values, and the value of the parameter p is computed. For example, to define an increasing relative proximity for the attribute AGE with the domain $\{0 \leq x \leq 100\}$, the user may respond to the prompts as follows:

```
Increasing [I] or Decreasing [D] relative proximity: I
Sample AGE value x:          60
Sample AGE value y:          65
Value of Irelprox (x, y):    0.8
```

The resulting values may be examined at specified domain values, and accepted, if satisfactory.

```
Irelprox ( 0, 5) =      0.00
Irelprox (20, 25) =     0.53
Irelprox (95,100) =     0.87
```

2.3.2 Concepts Represented as Fuzzy Sets

Fuzzy set definitions enable the user to build a vocabulary of terms that describe sets with vague boundaries. These concepts can be used in queries, or in specification of other fuzzy sets or rules.

Fuzzy sets can be defined in terms of an S-curve, a Bell-curve, a Table, or a combination of other previously defined fuzzy sets [22, 30]. In this case, the user provides a "copy" of his knowledge representation to the KB. However, a low-level form of learning occurs when new, more complex fuzzy sets are defined in terms of previously defined fuzzy sets. The system is capable of finding the corresponding representation of the new fuzzy set in terms of attribute values, thus allowing the user to use high-level concepts in the query language rather than attribute-name, attribute-value specification employed in the majority of databases.

For example, assume that attributes HEIGHT and WEIGHT with domains {HEIGHT $\geq$ 5'0"} and {WEIGHT $\geq$ 80 lb.}, respectively, are defined in the DB. Fuzzy sets TALL and SHORT can be defined as increasing and decreasing S-curves, respectively, NORMAL_WEIGHT can be expressed as a Bell-curve, and HEALTHY can be defined as a fuzzy set combination:

```
HEALTHY = {NOT VERY TALL OR NOT VERY SHORT} AND NORMAL_WEIGHT.
```

The effect of logical connectives and modifiers is implemented using the standard fuzzy logic interpretations, *e.g.* membership value of an intersection of two fuzzy set is a minimum of the constituent membership values [24].

A fuzzy set ATTRACTIVE can be defined as a table with attributes EYES and HAIR with domains {BLUE, BROWN, GREEN} and {BLOND, RED, BROWN, BLACK}, respectively, as follows:

EYES	HAIR	ATTRACTIVE
BLUE	BLOND	1.0
BLUE	BLACK	0.8
BROWN	BROWN	0.4
BROWN	BLACK	0.6
BROWN	RED	0.0
GREEN	RED	0.7
GREEN	BLACK	0.5

The absence of a possible pair EYES-HAIR values (e.g. BLUE, RED) is interpreted as having an unknown membership value in the fuzzy ATTRACTIVE. The membership value can be interpolated (learned) based on the similarities of attribute values and the corresponding known membership values [31].

2.3.3 Concepts Represented as Rules

Rules are used to express relationships between facts in the Data Base and other rules or fuzzy sets in the Knowledge Base. The general form of a rule is:

```
IF   <antecedent>      [M]
THEN <consequent>      [C]
```

Here, the `<antecedent>` is a condition involving attribute names and values, comparison operators, attribute variables, fuzzy sets or other previously defined rules, including fuzzy modifiers and connectives. `[M]` specifies an optional match qualifier. It is the required degree of the computed match, m, between the antecedent and the data in the database that has to be reached during the

antecedent evaluation. In FIIS, the value of the match qualifier defaults to the current acceptance threshold, T, if not specified during the rule definition process.

The <consequent> defines the new concept, *i.e.* it defines the outcome of the rule. [C] specifies an optional level of confidence in the rule consequent when the antecedent is evaluated to be true (*i.e.*, m = 1). In FIIS, confidence defaults to 1, indicating absolute confidence. Lower levels can be specified, reflecting weaker confidence in the rule outcome.

A rule satisfaction, S, is derived by first computing the degree of the condition satisfaction, m. This measure must exceed the required degree of match, M, and the current acceptance threshold, T, in order to consider the rule's consequent as plausible. The value of S is further influenced by the confidence level, C. Hence,

$$S = \begin{cases} m * C & \text{if } m \geq M \text{ and } m \geq T \\ \text{Unacceptable} & \text{otherwise.} \end{cases}$$

Suppose a rule called PEERS is to be defined, which says that two people are peers if they are approximately of the same age. In FIIS this rule has the form:

```
Consequent: PEERS   { NAME(X); PNAME(Y) }  [C = 1.0]
Antecedent: approx  { AGE(X) is AGE(Y) }   [M = 0.6]
```

Here, X and Y are variables which are declared to refer to the attribute NAME in the relation PERSON stored in the DB.

As another example, let us consider a rule defining friends as peers who have at least one common hobby. Assuming that the attribute HOBBY exists in the DB, and using the previously defined rule PEERS, the rule FRIENDS may be defined as:

```
Consequent:  FRIENDS     { NAME(X); PNAME(Y) }       [C = 0.9]
Antecedent:  PEERS and   { HOBBY(X) is HOBBY(Y) }    [M = T]
```

Here, the confidence level of 0.9 indicates that even if the antecedent is fully satisfied, we are not fully confident that X and Y are friends. Note that the match qualifier has the default value of the threshold of acceptance, T. This allows the user to modify how strictly the hobbies of X and Y have to match during the querying process, by changing the value of T.

3. QUERYING

3.1 Query Specification

A query specification is composed of two parts: (i) attributes to be listed, and (ii) a condition to be satisfied [22]. The attributes to be listed do not have to belong to the same relation, as natural join is performed automatically, based on common attribute names. Only ambiguous joins require user intervention by specifying which, if any, of the system-proposed joins are to be used. The condition part of a query specifies criteria which have to be satisfied by the tuples in order to be displayed.

Queries in FIIS approach the expressiveness of a natural language. To demonstrate

this, let us consider a request "Find name, age, salary and area of expertise of persons who are fairly young, or who are middle-aged and rich". Assume that all relations, attributes and fuzzy sets used in the query are defined. The corresponding query would be specified as follows:

```
Specify attributes to be displayed

:- NAME AGE SALARY EXPERTISE

Specify the query condition

:- FAIRLY YOUNG OR { MIDDLEAGED AND RICH }
```

When a query involves a rule, the rule name with any instantiations is specified as the query condition. Suppose we would like to list all peers of a specific person Eric, using the above defined rule PEERS. The query condition is as follows:

```
:- PEERS { NAME(ERIC) }.
```

3.2 Query Evaluation

The query condition is broken down into the simplest forms directly involving attribute values. Tuples composed of attributes involved in the query condition are evaluated to yield a degree of query satisfaction, S, with a value in the unit interval [0,1].

The degree of satisfaction for a simple condition is combined with the degrees of satisfaction for other simple conditions by the application of the standard interpretations of the fuzzy binary connectives and modifiers [27]. Finally, all those tuples for which $S \geq T$, (*i.e.* the degree of query satisfaction is equal to or exceeds the acceptance threshold value) have their respective attribute values displayed, along with the degree of satisfaction for each tuple. The tuples are presented in the descending order of the degrees of query satisfaction.

For example, the result for the above query asking for NAME, AGE, SALARY and area of EXPERTISE of PERSONs who are FAIRLY YOUNG OR {MIDDLE-AGED AND RICH} satisfying the default acceptance threshold value of 0.50 is presented in the following way:

NAME	AGE	SALARY	EXPERTISE	S
AL	22	45000	EXPSYS	1.00
AL	22	45000	AI	1.00
JIM	24	43000	DBMS	0.99
ERIC	38	50000	UNCERT	0.94
TOM	42	48000	AI	0.87
TOM	42	48000	UNCERT	0.87
...				
JOHN	30	25000	DBMS	0.50

The user is given an opportunity to see an explanation of the reasoning process involved in the query evaluation. An explanation for Eric has the following form:

```
In relations PERSON and P_SKILL:

NAME        AGE   SALARY      EXPERTISE          S
-------------------------------------------------
ERIC        38    50000       UNCERT           0.94

The fuzzy set  "MIDDLEAGED"  is defined as a Bell-curve
and membership evaluates to  0.96 for AGE 38

The fuzzy set  "RICH"  is defined as an S-curve
and membership evaluates to  1.00 for SALARY 50000

The expression "middle-aged AND rich"  evaluates to 0.96
because AND  ==> min :       min(0.96, 1.00) = 0.96

The fuzzy set  "YOUNG"  is defined as an S-curve
and membership evaluates to  0.60 for AGE 38

The expression "FAIRLY young"  evaluates to 0.77
because FAIRLY ==> sqrt :    sqrt(0.60) = 0.77

The expression "fairly young OR (middle-aged and rich)" evaluates to 0.96
because OR   ==> max :       max(0.77, 0.96) = 0.96

The query satisfaction degree            S = 0.96
exceeds the default acceptance threshold T = 0.50,
hence the tuple is retrieved.
```

The default acceptance threshold of 0.50 can be changed to any value in the unit interval. By changing the threshold of acceptance level, the user can control the flexibility of the approximate reasoning. Setting the threshold to 1 would result in crisp, or Boolean query evaluation.

4. CONCLUSION

The prototype model of FIIS has shown that a system based on relational database techniques and logic programming (adapted to support approximate reasoning) provides a suitable framework for information systems. In future versions, it is desirable to incorporate representations of other forms of knowledge, such as propositions involving fuzzy quantifiers, functions, trends, and temporal information. The system would also benefit from learning methods capable of finding rules involving relationships among variables, detecting dependencies or patterns between attributes or concepts, and expressing the learned concepts in an easy to understand form. With these enhancements, the variety of questions that can be answered by the system would increase dramatically, as the reasoning patterns could derive not only facts or rules that partially match, but those that are more general or more specific. However, even in its present form FIIS exhibits features of intelligent information processing.

ACKNOWLEDGMENT
The implementation of FIIS was carried out by Suresh Rajgopal and Richard Roland.

REFERENCES

[1] Arrazola, I., Plainfossé, A., Prade, H., Testamale, C. (1988) "Extrapolation of fuzzy values from incomplete data bases", Tech. Rep. LSI, Univ. Paul Sabatier, Toulouse, No. 298

[2] Baldwin, J.B., Zhou, S.Q. (1984) "A fuzzy relational interface language," *Fuzzy Sets and Systems*, **14**(2), 155-174.

[3] Bookstein, A. (1980) "Fuzzy requests: an approach to weighted Boolean searches," *Journal of the American Society for Information Science*, **31**(3), 240-247.

[4] Buckles, B.P., Petry, F.E. (1987) "Generalized database and information systems," in: *Analysis of Fuzzy Information*, J.C. Bezdek, ed., Vol II, Artificial Intelligence and Decision Systems, CRC Press, Inc., Boca Raton, FL, 177-201.

[5] Cuppens, F., Demolombe, R. (1988) "Cooperative answering: a methodology to provide intelligent access to databases", in L. Kerschberg (ed.), *Expert Database Systems, Proc. from the Second International Workshop*, Tysons Corner, VA, Apr. 25-27, 1988, 333-353.

[6] Dontas. K, Zemankova, M. (1988) "APPLAUSE: an experimental plausible reasoning system", in Z.W. Ras, L. Saitta (eds.), *Methodologies for Intelligent Systems, 3*, North Holland, 29-39. (Extended version will appear in *Information Sciences)*

[7] Dubois, D. (1984) "The management of uncertainty in expert systems: the possibilistic approach," *Proc. 10th Triennial IFORS Conf.*, Washington, D.C., North Holland Publ., Amsterdam, 949-964.

[8] Fertig, S., Gelernter, D. (1988) "Musing in an Expert Database", in L. Kerschberg (ed.), *Expert Database Systems, Proc. from the Second International Workshop*, Tysons Corner, VA, Apr. 25-27, 1988, 383-399.

[9] Hall, L.O., Kandel, A. (1986) *Designing Fuzzy Expert Systems*, Verlag TUV Rheinland, Koln.

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

[11] Kacpryzyk, J., Ziolkowski, A. (1986) "Database queries with fuzzy linguistic qualifiers," *IEEE Trans. on Systems, Man, and Cybernetics*, **16**(3), 474-479.

[12] Kerre, E.E., Zenner, R.B.R.C., De Caluwe, R.M.M. (1986) "The use of fuzzy set theory in information retrieval and databases," *Journal of the American Society for Information Science*, **37**(5), 341-345.

[13] Michalski, R.S., J.G. Carbonell, T.M. Mitchell (eds.) (1983, 1986) *Machine Learning - An Artificial Intelligence Approach I, II*, Morgan Kaufmann Publ. Inc.

[14] Michalski, R.S. (1987) "How to Learn Imprecise Concepts: a method for employing a two-tiered knowledge representation in learning", in *Proc. of the 4th Int. Workshop on Machine Learning* , P. Langley (ed.), Morgan Kaufmann Publ. Inc., 50-58

[15] Minker, J. (1987) "Deductive databases: an Overview of Some Alternative Theories," in Ras, Z.W., Zemankova, M. (eds.), *Methodologies for Intelligent Systems*, Elsevier Science Publ. Co., 148-158.

[16] Missikoff, M., Wiederhold, G. (1986) "Towards a unified approach for expert and database systems" in L. Kerschberg (ed.), *Expert Database Systems, Proc. from the First International Workshop*, Benjamin-Cummings Publ. Co., Menlo Park, CA, 383-399.

[17] Motro, A. (1986) "SEAVE: A mechanism for verifying user presuppositions in query systems," *ACM Trans. on Office Information Systems*, **4**(4), 312-330.

[18] Motro, A. (1988) "VAGUE: a user interface to relational databases that permits vague queries", ACM Trans. on Office Information Systems, **6**(3), 187-214.

[19] Prade, H., Testamale, C. (1984) "Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries," *Information Sciences*, **34**(2), 115-143.

[20] Prade, H. (1985) "A computational approach to approximate and plausible reasoning with applications to expert systems", *IEEE Trans. on Pattern Analysis and Machine* Intelligence, **PAMI-7**(3), 260-283.

[21] Prade H., Testemale, C. (1987) "Fuzzy relational databases: representational issues and reduction using similarity measures," *Journal of the American Society for Information Science*, **38**(2), 118-126.

[22] Rajgopal, S., Roland, R., Zemankova, M. (1987) "A user's guide for the fuzzy intelligent information system", Tech. Rep. Computer Science Dept., University of Tennessee, CS-88-72, August 1987.

[23] Smith, E.T. (ed.) (1989) "Developments to Watch: Why the Japanese are going in 'fuzzy logic'", *Business Week*, Feb. 20, 1989, 148.

[24] Stix, G. (1988) "Fuzzy Wuzzy (in Japan) is a subway motorman", *The Institute, News Supplement to IEEE Spectrum*, **12**(5), May 1988, 1 & 4.

[25] Williams, M.D. (1984) "What makes RABBIT run?", *Int. J. Man-Machine Studies*, **21**, 333-352

[26] Zadeh, L.A. (1965) "Fuzzy sets", *Information and Control*, **3**, 177-200.

[27] Zadeh, L.A. (1975-76) "The concept of a linguistic variable and its application to approximate reasoning," *Information Sci.*, Part I., 1975, **8**, 199-249, Part II., 1975, **8**, 301-357, Part III., 1976, **9**, 43-80.

[28] Zadeh. L.A. (1983) "The role of fuzzy logic in the management of uncertainty in expert systems," Fuzzy Sets and Systems, **11**(2), 199-227.

[29] Zemankova, M., Kandel, A. (1984) *Fuzzy Relational Databases - A Key to Expert Systems*, Verlag TUV Rheinland, GmbH, Cologne.

[30] Zemankova, M., Kandel, A. (1985) "Implementing imprecision in information systems", *Information Sciences*, **37**(1,2,3), 458-463.

[31] Zemankova, M. (1988) "FILIP: a fuzzy intelligent information system with learning capabilities", Tech. Rep. Computer Science Dept., Univ. of Tennessee, May 1988.

REPRESENTATION AND ACCESS OF UNCERTAIN RELATIONAL DATA

Arie Tzvieli

Bell Communication Research
RRC 4E707
444 Hoes Lane
Piscataway, NJ 08854

Abstract: In this paper we discuss several problems associated with the representation and access of uncertain data within the relational database model. Among the problems considered are the semantics of uncertainty representation, levels of data which may be uncertain, and the quantitative and qualitative representation of certainty values related to data.

The traditional database access using query languages is quite limited when uncertain data is involved, and special treatment is required in this case. We discuss the representation and access of uncertain data. We propose a method for computing the certainty of combined expressions when queries are underspecified. We advocate the use of special certainty-related operations on relations, the *relational transformers*, operators which may be considered a generalization of aggregate functions, to perform sophisticated customized database access.

1. INTRODUCTION

In many real life situations, it is impossible or impractical to achieve complete knowledge of the relevant facts (or they may have not occurred yet), while nevertheless some action must be taken. As an example, setting a meeting date might be dependent on the weather, and yet, when many people are to participate, this decision has to be made well in advance. Uncertainty management is concerned with behavior guidelines for these types of situations, so that the actions chosen could be justified on a rational basis. In other words, models are developed and employed to choose the best action, based on the available data, and to assess the certainty that the chosen action is the right one.

Indeed, the term *uncertainty* itself is vague; quite often, the uncertainty is specified by giving a *certainty value*, and it could also mean *possibility, fuzziness, probability*, etc. The following discussion could be useful, regardless of our choice of interpretation of *uncertainty*.

The most important observation regarding uncertainty is, that usually additional information exists, which could be employed to achieve higher reliability in the use of the uncertain data. However, to accommodate the inclusion of this information in the relational model, we have to generalize it and to introduce several new concepts.

Main issues in the area of uncertainty management which are relevant to relational database systems:

- *Representation of uncertainty:* how should uncertain data be represented in a relational database. This is a composite issue that could be further broken down into semantics and representation.

First let us consider the issue of semantics. What is the meaning of a statement such as: "the certainty that [supplier S1 is located at city C1 and has a status St1] is 70%". Does 70% refer to the supplier, to the city, to the status, or to the relationship between them all?
The second issue is the representation domain and form of certainty knowledge: should we use percents, real numbers, the interval [0, 1], natural language terms, intervals, expectancy and standard deviation, etc. While in some cases conversion between alternative representations is possible, in other cases it is not. The choice of representation seems to be tied strongly to the chooser's interpretation of the term *uncertainty*.

- *Simple retrieval of uncertain data.*
  At issue is the reference to the certainty value in query expressions. Is the certainty data part of the model (such as: every relation must have a certainty attribute named "Certainty") and therefore part of the query language, or is the representation dependent on the application, and therefore certainty management can not be built into the query language.

- *Combination of uncertain data qualifications in queries.*
  Given the certainty values of some simpler expressions, what is the certainty which should be assigned to a combination of those expressions.

- *Complex and special operations on uncertain data.*
  As an example, we may consider using thresholds to filter out data whose certainty is below a given value. Special operations can be developed that will increase the query language expressiveness and which will enable the sophisticated use of certainty values.

- *Analysis of the certainty of simple components, given the certainty of composite facts.*
  This use of uncertainty data is somewhat more complex than the previous ones. Consider the following example: suppose we know that the certainty of a move up in the stock and bond market is α. What is the chance of a move up in the stock market? This problem is the complementary problem of combining uncertainties. Intuitively, it seems that we can expect only crude estimates to result from this kind of analysis.

- *The development of a suitable logical system for reasoning under uncertainty.*
  Assume that we are given a set of facts and their associated certainties, and we need to find the certainty of a logical conclusion of those facts. Since in uncertain situations both a fact and its complement are possible to some degree, first order consistency (where either a formula or its complement are satisfied in a model) is not a reasonable assumption (the interested reader is referred to [Tz88], which discusses some challenging problems).

- *Vague queries, applied to certain and / or uncertain data.*
  In this case, the query itself is ambiguous (e.g., find a beautiful woman in a matchmaker's database). The query's ambiguity creates uncertainty, which may be combined in a complex way with the uncertainty of the data.

Our goal in this paper is to explore some of these problems. The author's personal opinion is that research in the management of uncertainty is at an early stage, that a complete solution is very difficult, and that much additional research is needed.

A Brief Description of the Work.

A simplistic approach to uncertainty representation, which leaves us within the classical relational model, is to add to each tuple a certainty attribute. We consider in Section 2 the possible semantics of this approach. Since more than one possible interpretation of the uncertainty in *certainty augmented* tuples exists, this issue must be resolved. Next, we consider some methods for representing uncertainty within the relational model, and we propose a generalization of the relational model to accommodate practical needs, by including non-relational data in the data dictionary.

To enable complex queries on uncertain data, we suggest the use of *relational transformers*, which transform a relation into a modified relation. We prefer the use of the term *relational transformers* to *relational modifiers* since we envision operations that are more complex than modification of tuples. As an example of a complex operation we may consider the computation of the transitive closure of a relation, in which the edge information is uncertain. The specific transformers implemented for an application could help customize a system to the user's need, and give a system extra expressive power. The idea is further elaborated in Section 3.

The combination of uncertain expressions into composite ones can follow different intuitions (see for example Dubois and Prade [DP84] for a discussion of implication). We observe that the prevailing approach toward combinations is *or-biased,* in the sense that the certainty of a conjunction is not higher than the certainty of any of its components, and the certainty of a disjunction is not smaller than that of any of its components. Consequently, in most cases there is a difference between the certainty of a conjunction and that of a disjunction. However, when the nature of the combination itself is vague, an approach which is not biased toward any of the specific possible connectives is preferable, since the potential error is smaller. We propose to use in this case a *fair* approach, which is based on the averaging of the certainty values over the potential types of combination. Specifically, we consider two cases of potential combinations: implication and conjunction, and disjunction and conjunction.

The subject of the management of uncertainty has been a very popular academic research topic, and we apologize for the omission of reference to many important works. Many valued logics were the pioneers of uncertainty representation and management. Ackermann [Ac67] may serve as an introduction to the subject. Representatives of current research are Halpern & Rabin [HR83], Nilsson [Ni86], Keisler [Ke77] and Haack [Ha74]. The problem of "unknown values" in databases has received considerable attention; as a representative consider Lipski [Li79]. The theory of fuzzy sets attracted much interest. The reader is referred to Zadeh [Za65, Za83]. Shafer's theory of evidence [Sh76] is very elegant and interesting. Dubois and Prade contributed books and many good papers to the management of uncertainty. Their book [DP80] is a comprehensive survey on fuzzy logic up to 1980, and their second book [DP88] contains a description of their interval approach to uncertainty representation. They have also considered, with Testmale, uncertainty in the context of relational databases [PT84]. Zemankova and Kandel considered fuzzy databases [ZK84]. Buckles and Petry [BP82] have also considered issues in fuzzy databases. Relevant work can also be found in artificial intelligence books and articles, such as Kanal and Lemmer [KL86].

The paper is organized as follows: Section 2 discusses semantics and representations of uncertain data. Section 3 discusses vague queries and advocates the use of the relational transformers. Section 4 summarizes the paper.

2. The Representation of Uncertainty.

2.1 Uncertainty of What?

The simplest approach to the representation of uncertain data is to use the standard relational model. Given a relation

SUPPLIER(Sname, S#, City, Status),

we can add an attribute **Certainty** which will express the certainty associated with each tuple in the relation. This approach works for some cases, and it does not require any modification of the relational model. Therefore, it is easy to implement both for data storage and data access. It has, however, limited expressive power and ambiguous semantics. Let us consider the issue of semantics first. Suppose that the tuple

<AT&T, S1, Piscataway, 75, .66>

is a member of the SUPPLIER relation (augmented with a Certainty attribute). What is the meaning of the Certainty value .66?
Evidently, it could be interpreted in a number of ways:

a. The certainty that (AT&T's supplier number is S1) is .66.

b. The certainty that (the status of S1 is 75) is .66.

c. The certainty that (AT&T's S# is S1 **and** S1's city is Piscataway **and** the status of S1 is 75) is .66.

Other interpretations are also possible.

How can this ambiguity be resolved? Consider the following approaches:

1. Agree to a single interpretation convention, such as: 'the certainty value relates to the complete tuple' (the third option above);
2. Break every relation into binary components (using an artificial tuple-id TID attribute, if required), so that the certainty is uniquely associated with a single attribute;
3. Add additional attribute(s) to specify the association between the certainty attribute and other attributes.

Note that the first option requires the least amount of space, while the second is often the most space consuming, since some information is certain and need not be specified in cases one and three. On the other hand, the first option requires knowledge of metadata which is usually not recorded in the database itself, and is therefore less accessible. We can also adopt a combined approach; binary relations be used to express the certainty of single attributes, and n-ary relations be used to express the certainty of a relationship between multiple attributes.

In the past, certainty management was associated with facts and boolean combinations of facts. We would like to point to other contexts in which certainty management could play important role: certainty of sets of facts (such as sets of tuples, or sets of formulas), and aggregate functions. As an example, consider the following situation:
"For employee number in the range 1 to 1000, the years-of-service attribute is correct in 80% of the cases, since until 5 years ago starting-date-of-work was kept only as a year (no month or day)".
The description of the above situation is achieved using a formula:
if employee number is in the range 1-1000, then certainty of years-in-service is .8.
This description is more concise than a specification of certainty .8 for each of the employee records, where the employee number is in the range 1-1000. Furthermore, in many cases a formula description carries semantic information which is not captured by a certainty-per-tuple description. However, using formulas in a relational database requires the solution of two problems:

1. Storage of formulas in a database.
2. The use of formulas in update / query operations.

For the storage of formulas, we could use relations in which the formulas will be stored, either as text or structured according to their parse tree. The use of formulas in query operations requires the integration of reasoning capabilities within the database systems, as in deductive databases. Even if the issues of representation of formulas and reasoning within a database system are solved satisfactorily, it still could prove more efficient to associate the certainty (in the above example) with each relevant tuple (similar to materialized views).

Another useful context for uncertainty management is aggregate functions. Given a relation, we may have information regarding the certainty of the value of some aggregate functions applied to it. For example, consider the case where the salaries of employees are confidential, while their average salary is public knowledge. In this context, a statement like: "The certainty that (the sum of salaries in the TOY department > 50k) is 85%" can make perfect sense.

2.2 Representation of Uncertainty Values.

In our previous discussion, we implicitly assumed that a suitable representation of the uncertainty values exists. In this subsection, we consider the problem of suitable representation of the uncertainty information.

Between 1925 and 1940, several logicians (e.g., Lukasiewitz, Tarski) investigated logics which had three truth values: true, false, and a third value, which could mean *unknown, neither true nor false, inapplicable,* paradoxical. This ambiguity is somewhat parallel to the use of the terms *null, unknown, non-applicable* in relational database theory. In the 1940s, the many-valued logics were generalized to include an infinity of truth values, both enumerable and non-enumerable. This approach amounts to representing uncertainty by a single (rational or real) number. The use of percents to represent certainty is a special case of this method.

This approach has the potential benefits of accuracy, easy incorporation into the relational model, and the conciseness of added certainty information. Its main drawback lies in its inability to describe more complex certainty data such as: certainty associated with predicates and aggregate functions, probabilistic distribution of values, uncertainty of certainty values (i.e., second order uncertainty), etc.

Other approaches use more than a single number to express uncertainty. For example, Cheeseman [Ch84] proposed the use of expectancy together with standard deviations to capture uncertainty. Some probability distributions require two or more numbers to characterize the possible values (such as a uniform distribution over a given interval). A two-value approach could also be used to represent the certainty of facts and the certainty of the certainty value itself. Another interesting use of the two-valued approach is in representing uncertainty using an interval; Dubois and Prade [DP88] elaborate on this method. In a natural extension, the two-value approach could be generalized to an n-value approach. Usually, the complexity and overhead of uncertainty representations using n-value approaches become prohibitively expensive for large values of n.

An approach which is qualitative rather than quantitative is one which uses a hierarchy of natural language terms to represent certainty. An example of such a hierarchy is: absolutely certain, very certain, certain, somewhat certain, uncertain. Some research seems to suggest (see Bonissone and Decker [BD85]) that the differences between natural language hierarchies having more than nine terms and nine-term hierarchies are mostly negligible.

A benefit of the use of natural language hierarchies is that it is easy for human beings to understand, but their use introduces another problem. Natural language hierarchies have different meaning to different persons; their use may amplify the uncertainty of the data, degrading the system's reliability.

There are several representations of uncertainty that do not fit within the relational model, and require its generalization. One such example is the use of predicates, potentially including aggregate functions. Another example is the use of mathematical formulas to specify probability distribution of uncertainty values. The incorporation of such unusual objects in a database system may enhance the expressive power, but prototypes having this capability are still several years away.

3. The Access of Uncertain Data.

In this section, the vague combination of uncertain data is considered, and the relational transformers are discussed.

3.1 The Vague Combination of Uncertain Data.

In the following discussion, assume that the certainty is represented by a number in the interval [0, 1]. Many formulas were suggested to compute the certainty value of a combined expression, based on the certainty values of the components and the logical connective involved (and / or / not / implication). These formulas can be used whenever boolean queries address a database which contains uncertain data. The reader is referred to Weber [We83] and to Dubois and Prade [DP84] for very interesting discussions concerning the appropriate choice of computational formulas.

The choice of a computational formula for the certainty of conjunctions can be the starting point for a coherent computation of the certainty of combinations involving other connectives. For a logical formula ϕ with certainty value α, the certainty of $\neg\phi$, the complement of ϕ, is usually taken as $1-\alpha$. The certainty expressions for the other connectives are usually based on their definitions in first order logic using conjunction and negation. For example:

$$\phi \vee \psi := \neg(\neg \phi \wedge \neg \psi), \quad \text{and} \quad \phi \rightarrow \psi := (\neg \phi) \vee \psi.$$

The following are examples of proposed formulas to compute the certainty of a conjunction:

1. min(a, b)
2. ab
3. max(0, a + b -1).

All of these formulas are examples of an *or-biased approach:* the certainty of a conjunction is not greater than the certainty of each of its components. As a result the certainty of a disjunction is not lower than those of its components. Thus, for most of the possible certainty values of the components, the certainty of a disjunction is

significantly different from the certainty of a conjunction. This kind of difference also occurs between other pairs of connectives.

Consider the query *What is the certainty that a person who smokes is ill?*. This could be interpreted either as: "What is the certainty that a person smokes **and** is ill", or as: "What is the certainty that smoking **implies** illness". The connective is not specified in the query. Ambiguity involving other connectives is possible as well.

To minimize the potential error, we propose to assign to ambiguous combinations a certainty value which is the average of the values that would have resulted from the choice of each potential connective, using existing computing formulas. Since the proposed approach is impartial to the type of combination, we shall refer to it as the *fair approach*. Let us compute the conjunction-implication fair approach corresponding to the use of min(a, b) for conjunction, max(a, b) for disjunction, and max(1-a, b) for implication. In the following table, Λ denotes conjunction and $\rightarrow$ implication.

a, b values	Λ	$\rightarrow$	fair
a=b; a<.5	a	1-a	.5
a=b; a>.5	a	a	a
a<b; 1-a<b	a	b	(a+b)/2
a<b; b<1-a	a	1-a	.5
b<a; 1-a<b	b	b	b
b<a; b<1-a	b	1-a	(1+b-a)/2

In the case of ambiguity between conjunction and disjunction, the fair approach yields the *arithmetical average* of the two certainty values involved in the combination, for all three examples of conjunction computational formulas given above.

3.2 The Relational Transformers.

We shall present the concept of **relational transformers** through illustrative examples. Although query languages allow the expression of ad-hoc queries, they are still limited in their expressive power (see [Zv86] for a theoretical discussion). The addition of special operations, which we call relational transformers, could enhance the expressive power of the language. Usually, a descriptive style of queries is preferable, and the relational transformers could embed in their implementation complex procedurality, from which the user is shielded. Another benefit of adding relational transformers to query languages is the added flexibility to customize a system to the special needs of its applications.

Several examples of relational transformers follow.

A threshold relational transformer.

In many cases, we are interested only in data whose certainty is above some given threshold, while the other data is simply ignored. The threshold transformer

threshold(α, R)

creates a copy of R in which all the tuples whose certainty is below α have been erased.

Normalization of data.

Suppose we constrain certainty values to be in the interval (l-val, r-val). To be able to use data with certainty values in the interval (a, b), we may want to transform the given certainty values using some mapping, such as:

new_val := l-val + (old_val - a) * (r-val - l-val) / (b - a)

An exception to a certainty distribution.

Management by exceptions is a popular management technique, that could be applied to the management of

certainty values as well. Given an expected certainty distribution over a relation, and a criterion which defines exceptions, this relational transformer computes the sub-relation containing only the exceptions.

Categorization.

The categorization transformer accepts a list of categories (such as: 0 to .2 - very uncertain, .2 to .5 - uncertain, and substitutes the relevant category for the certainty value of each tuple. Notice that the inverse transformation does not exist.

Transitive connectivity.

The previous transformers are likely to be useful in every system where certainty values are expressible. The transitive connectivity transformer is of a more specialized nature. This transformer may be used to compute the transitive closure of relations, whenever some uncertainty is associated with the existence of edges, in a tabular description of a graph.

While these examples of relational transformer are all in the context of uncertainty management, the same (or similar) transformers may also be useful in conventional database systems. A tool chest of relational transformers may be a positive step toward customization of database system for individual application.

In order to implement relational transformers, it is required that the query language be able to recognize the relational transformers. The specification of the relational transformers can be done either locally by the DBA or by the vendor.

4. Summary.

We have discussed several issues in the management of uncertainty within database systems, and problems related to the representation of uncertainty. We outlined two promising ideas: the *fair approach* to vague combinations of uncertain facts, and the *relational transformers* as a tool for customization and enrichment of query languages.

Acknowledgement: Thanks to Ami Motro for many suggestions which improved this presentation.

REFERENCES

[Ac67] Ackermann R.: "An Introduction to Many-Valued Logics". Dover Publications Inc., New York, 1967.

[BD85] Bonissone P.P. and Decker S.K.: "Selecting Uncertainty Calculi and Granularity: An Experiment in Trading-off Precision and Complexity". In: Proceedings of the Workshop on Uncertainty and Probability in Artificial Intelligence. UCLA, L.A., CA, Aug. 14-16, 1985.

[BP82] Buckles B.P. and Petry F.E.: "A Fuzzy Representation of Data for Relational Databases". Fuzzy Sets and Systems 7, 1982, pp. 213-226.

[Ch85] Cheeseman P.: "In Defense of Probability". Proceedings of the 9th IJCAI, 1985, pp. 1002-1009.

[DP80] Dubois D. and Prade H.: "Fuzzy Sets and Systems: Theory and Applications". Academic Press, New York, 1980.

[DP84] Dubois D. and Prade H.: "Fuzzy Logics and the Generalized Modus Ponens Revisited". Cybernetics and Systems 15, 1984, pp. 293-331.

[DP88] Dubois D. and Prade H.: "Possibility Theory". Plenum Press, New York, 1988.

[Ha74] Haack S.: "Deviant Logic". Cambridge University Press, 1974.

[HR83] Halpern J.Y. and Rabin M.O.: "A Logic to Reason about Likelihood". Proceedings of 15th STOC, 1983, pp. 310-319.

[KL86] Kanal L.N. and Lemmer J.F.: "Uncertainty in Artificial Intelligence". Elsevier Science Publishers, 1986.

[Ka75] Kaufmann A.: "Introduction to the Theory of Fuzzy Subsets", Vol. I, Academic Press, 1975.

[Ke77] Keisler H.: "Hyperfinite Model Theory". In: Logic Colloqium 76, Gandy R.O. and Hyland J.M.E. editors, North-Holland, 1977, pp. 5-110.

[Li79] Lipski W.: "On Semantic Issues Connected With Incomplete Information Databases". ACM TODS 4, 1979, pp. 262-296.

[Ni86] Nilsson N.J.: "Probabilistic Logic". Artificial Intelligence 28, 1986, pp. 71-87.

[PT84] Prade H. and Testmale C.: "Generalizing Database Relational Algebra for the Treatment of Incomplete or Uncertain Information and Vague Queries". Information Sciences 34, 1984, pp.115-143.

[RM86] Raju K.V.S.V.N. and Majumdar A.K.: "Fuzzy Functional Dependencies in Fuzzy Relations". In the Proceedings of the 2nd Data Engineering Conference, IEEE Computer Society, 1986, pp. 312-319.

[Sh76] Shafer G.: "A Mathematical Theory of Evidence". Princeton University Press, Princeton, N.J., 1976.

[Tz88] Tzvieli A.: "PL - A Probabilistic Logic". Proceedings of the 4th Data Engineering Conference, 1988, IEEE Computer Society, pp. 462-469.

[Tz88a] Tzvieli A.: "On Implementations of Production Systems Using DBMS". Proceedings of the 3rd International Conference on Data and Knowledge Based Systems, Israel, Beeri C. and Schmidt J.W. eds., Morgan Kaufmann, 1988.

[Wc83] Weber S.: "A General Concept of Fuzzy Connectives, Negations and Implications Based on t-Norms and t-Conorms". Fuzzy Sets and Systems 11, 1983, pp. 115-134.

[Za65] Zadeh L.A.: "Fuzzy Sets". Information and Control 8, 1965, pp. 338-353.

[Za83] Zadeh L.A.: "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems". Fuzzy Sets and Systems 11, 1983, pp. 199-227.

[ZK84] Zemankova-Leech M. and Kandel A.: "Fuzzy Relational Databases - a Key to Expert Systems". Verlag TUV Rheinland, Koln, 1984.

[ZC86] Tzvieli A. and Chen P.P.: "Entity-Relationship Modeling and Fuzzy Databases". In the Proceedings of the 2nd Data Engineering Conference, IEEE Computer Society, 1986, pp. 320-327.

[Zv86] Tzvieli A.: "On Complete Fuzzy Relational Query Languages". Proceedings of NAFIPS 86, pp. 704-726.

Incomplete Information in Logical Databases

Tomasz Imielinski[1]
Department of Computer Science
Rutgers University

1. Introduction

Quite often information represented in the database is only an approximation of the real, partially unknown, external world. The simplest example is missing information about some fields of records in the database, e.g. the age of John or the salary of Mary. A standard solution, as old as the very concept of the database, is the so called *null value* [Codd 75]. Null values have been used extensively and freely in databases as placeholders to denote missing information [Vassiliou 79], [Grant 86]. While it was easy to place a null value in the database as a placeholder it was much more difficult to process queries in the presence of nulls without clear and well defined *semantics.*

Example

Consider the following relation R=

Person	Wife
Robin	@

A null value "@" in the attribute "Wife" of the relation R above could mean that (i) Robin has a wife, but we do not know who she is (ii) Robin does not have a wife (iii) Robin is a woman and the attribute wife is not applicable to him. Depending on which interpretation is taken, we will answer queries about R differently. In any case these answers should be *consistent* with the underlying interpretation of the null value. ■

In this paper we deal exclusively with type (i) interpretations; i.e. only logical incompleteness. The other possible interpretations of nulls are known as "nonexistent" and "nonapplicable" [Vassiliou 79] and are not treated here.

But null value is only the simplest manifestation of incompleteness of information. Frequently we have some partial knowledge about "missing values". For instance, we may know that John's age is between 30 and 35 or that Steve is either an associate or a full professor. This leads to a concept of partial nulls or OR-objects [Imielinski 89]. Here we know that the missing value exists and we even know the set of values to which it belongs. Finally, we may deal with general disjunctive information of the form "either John is a manager or Steve is a supervisor". [Grant 86].

Admitting and representing such information in the database is easy; it is considerably more difficult to specify what constitutes the answer to a query and how to compute it.

We address this and the following questions in the paper:

1) What is the meaning of incomplete information in the database and how to describe it

2) How to correctly process queries in the databases with incomplete information.

3) How complex is query answering in the presence of incompleteness.

---

[1]This research was supported by NSF under contract DCR 85-04140.

We concentrate on the case when data is known to exist but we do not know the exact values. In other words, we do not discuss here the case when data does not exists or is not applicable. On the other hand we talk about much wider class of incomplete information databases than simply databases with nulls. In particular we discuss:

1) Databases with marked nulls.

2) Databases with OR-objects.

3) Deductive databases (i.e. databases with rules) with incomplete information.

Note that incompleteness of information considered here has a *logical* nature, i.e. there is more than one *possible* world modeled by the database, but within this set of possible worlds we have no "preferences".

Generally we distinguish between two approaches - model theoretic (or algebraic) and proof theoretic. We will start from the null values and then apply the same principles to the more sophisticated forms of incompleteness.

2. Null Values - two approaches

A database can be viewed either from a model theoretic perspective, as a model of the real world, or from proof theoretic perspective, as a collection of logical formulas.

2.1. Model Theoretic Approach

In the model theoretic approach we extend the concept of relation to represent incomplete information. Let us first see how complete data is represented in the relational model. For instance, the following relation STUDENT with three attributes: Name, Age, Address is viewed as a table

Name	Age	Address
John	27	NY
Steve	26	LA

Formally, in the relational model only *domain constants* can appear as entries in tuples of relations. The *relational algebra* or *relational calculus* [Codd 70] are tools to operate on these tables. We assume that the reader is familiar with basic relational operations, such as join, projection, selection, union and difference.

Representing incomplete data in the relational model requires extension of the concept of relation to admit nulls, as special types of constants. In this way we obtain *tables with null values* or V-tables as they have been called in the literature [Imielinski 84]. Formally a table with (marked) null values or a V-table is a relation with variables as well as constants allowed as entries. These variables are called *marked null* values. For example, in the table STUDENT we could allow two marked nulls x and y.

Name	Age	Address
John	x	NY
Steve	26	y

The new table STUDENT specifies that there is a number of *possible* states of reality represented by this table. Each of them is a *relation*, resulting from STUDENT by substituting all possible domain constants for variables x and y (actually we can be more specific and say that we substitute all possible values of Age for x and all possible values of Address for y). In consequence we obtain a set of possible relations representing the state of the real world; we know that the real state of the world belongs to this set, but we do not know which one it is.

More formally, with each table T we associate a value of a *representation function* rep(T), which is a set of "possible worlds" represented by T. Let v be a substitution of domain constants for variables. Such a substitution is called a *valuation*. By v(T) we denote the relation resulting from T by substitution of v(x) for every variable x occurring in T. For a V-table T the representation function rep(T) is defined as follows:

$$\{S: \text{there exists v such that } v(T) \subseteq S\}$$

Now, having represented our data in such a tabular form, we want to process queries expressed in relational algebra (or relational calculus). To this end we have to *extend* relational algebra operations on these tables. Obviously we cannot do this extension in the arbitrary way - we have to "be faithful" to the underlying semantics. Let f be an arbitrary relational expression, and T an arbitrary table. The best way to define the extension f(T) of f over T would be to satisfy

$$rep(f(T)) = f(rep(T)) = \{f(S): S \in rep(T)\}$$

Indeed, in such case f(T) would preserve all information about possible values of f(S)[2]. Unfortunately, as explained in [Imielinski 84] and in [Imielinski 83] this will be too difficult to achieve. Fortunately, it is also unnecessary; we can require less as explained in [Imielinski 84] and later reformulated more elegantly in [Lipski 84a] . By a *True set* of the set of worlds X we mean $\cap X$, i.e the set of all tuples which belong to all worlds in X. Here are our modified requirements for a *faithful* extension of relational algebra operations on tables T:

(1) Preservation of True Sets: For any relational algebra expression f built from operations from Ω: $\cap rep(f(T)) = \cap f(rep(T))$: i.e. the table f(T) should preserve information about all tuples which for sure belong to $f(S^*)$ according to the table rep(T).

(2) Recursiveness: $f(g(T)) = (fg)(T)$

This states that we should preserve the important feature of relational algebra - the intermediate results of subexpressions of a given expressions can be stored as "views" which can be later used to the computation of the full expressions without having to start from scratch.

The postulates (1) and (2) are truly minimal acceptable requirements for any reasonable "extension" of relational algebra. They are still pretty restrictive - i.e. extensions of some operations over some types of tables will be impossible. Intuitively: *the expressive power of a given type of a table must be sufficient to represent the results of all relational algebra expressions from the set of expressions under consideration.* In many cases, in order to correctly extend a given set of operations on a given set of tables it is necessary to "enrich" the table itself; i.e. add to its expressive power.

For example it is demonstrated in [Imielinski 84] that if we admit only one null value (unmarked null) into the table then it may become impossible to extend even a simple class of operations consisting of projection and join over the tables. This is illustrated by the following example:

---

[2]according to the original table the real state of the world - S^* is one of possible worlds in rep(T), therefore the "real" state of the answer $f(S^*)$ is one of the possible worlds in f(rep(T))

Example

Let T =

A	B	C
a	@	c

and let f= $\pi_{AB}(T)$ X $\pi_{BC}(T)$

Evaluating this expression in the standard algebraic way, by first evaluating projections and then joining the resulting tables, initially yields:

U = $\pi_{AB}(T)$ =

B	C
@	c

W = $\pi_{BC}(T)$ =

A	B
a	@

There is no way now, however, to recognize that the two occurrences of a null value "@" (in U and W respectively) represent the same value (in general two occurrences of nulls will not match since it is certainly possible that they represent different values). Therefore the join of these two tables will be the empty set, not the original table T, as expected.

This problem can be remedied by introducing marked nulls, as in V-tables. However, V-tables cannot support relational operations built from projection, join and selection admitting negative selection conditions. An example similar to the one showed above is given in [Imielinski 84]. If the selection conditions do not use negation, then the following straightforward extension of relational operations satisfies (1) and (2):

All operations treat nulls as pairwise different constants.

For example, assume f(T) = $\pi_{AC}(\pi_{AB}(T)$ X $\sigma_{(B=C)\vee(C=c)}(\pi_{BC}(T))$

is evaluated over the table T

A	B	C
x	y	c
a	b	c
a'	b'	c'
a	y	z
x	d	d

We get

$\pi_{AB}(T)$ = {xy, ab, a'b', ay, xd}

$\pi_{BC}(T)$ = {yc, bc, b'c', yz, dd}

$\sigma_{(B=C)\vee(C=c)}(\pi_{BC}(T))$ = {yc, bc, dd}

$\pi_{AB}(T)$ X $\sigma_{(B=C)\vee(C=c)}(\pi_{BC}(T))$={xyc, abc, xdd}

f(T) = {xc, ac, xd}

This naive extension will not support correctly (in the sense of (1) and (2)) the relational operations with arbitrary selection conditions (inequalities are not handled properly, since all null values are always assumed different from any domain constants).

It has been shown in [Imielinski 84] that to fully support all major relational algebra operations (i.e project, join, select, union) one has to extend the syntax of the tables further; one such extension, in the form of so called conditional tables has been defined there. Conditional tables can represent arbitrary disjunctive information. They are V-tables with an additional column called "COND" which specifies conditions built as logical combinations of equality descriptors of the form (x=a) and (x=y). For any tuple t its condition is denoted by t[COND]. Intuitively , a tuple v(t) belongs to v(T) iff the condition of this tuple is true after substituting v(x) for x in t[COND]. Conditional tables can represent arbitrary disjunctions. The correct extension of relational algebra to conditional tables is provided in [Imielinski 84]. All relational operations are correctly supported. They are defined on a tuple by tuple basis: Selection, for any tuple t and any selection condition E generates a condition E(t) by substituting attribute names by variables in t. Then the new tuple s is formed such that s[COND]=t[COND]$\wedge$E(t) and s[A]=t[A] for all other attributes. Cartesian product of two tuples t and s is defined as a new tuple w such that w[COND]=t[COND]$\wedge$s[COND] and other fields of w are simply a concatenation of fields of t and s. Join is defined through cartesian product, finally selection and projection and union are defined as on relations without null values.

Defining new types of tables to represent incomplete information and extending relational algebra to them has been a pretty active research area in lat couple of years. Unfortunately, many authors extend relational operations in an arbitrary way totally ignoring the above semantical considerations (i.e postulates like (1) and (2)). The following example illustrates both the common mistake and the application of the criterion:

Example

Let us consider the set of tables admitting, in addition to domain constants, also finite sets as entries. The occurrence of a set A={a_1,...a_n} is interpreted disjunctively - at least one of the values in A is the "real" value[3]. Such sets are can be called OR-sets and naturally correspond to restricted nulls; for instance a tuple <John, {Manager, Supervisor, Director}> in a table R[Employee, Position] means that John is a manager or a supervisor or a director. Given a table T with OR-sets the representation function rep(T) is defined in a straightforward way following described above semantics of OR-sets. Each possible world in rep(T) is defined by replacing each OR-set with its member values.

To process queries we must extend relational algebra operations to these tables. Let us concentrate on just two operations - projection and join.

First, we are going to show that the straightforward way of extending join operation fails: Let us define join by extending "equality" over OR-sets:

An OR-set is not "equal" to any other OR-set nor to any domain constant

The rationale for this definition is that given an OR-set X={a_1,...a_n} and an individual constant, say b (possibly an element of X), it is always possible that "the real" unknown value for which X stands will be

[3]This is contrary to the sets interpreted conjunctively, as in nested relations

different from "b". Similar rationale justifies the case of inequality two OR-sets. The following example illustrates that such an extension will not preserve "True sets" (the first part of (1) and (2) and therefore will loose information.

Let

R =

A	B
a	$\{b_1, b_2, b_3\}$

and let

S =

B	C
b_1	c
b_2	c
b_3	c

Let f = π_{AC}(R x S), where x stands for natural join. If we extend join according to the above rule then f(R,S) will result in an empty table. However <a,c> $\in \cap$f(rep(R), rep(S)). Therefore, this extension of join will not be complete with respect to the underlying semantics and requirement (1) fails. We can also show that the proposed extension does not satisfy the above requirements. Indeed, let R

A	B	C
a	$\{b_1, b_2\}$	c

and let f= $\pi_{AC}(\pi_{AB}(R)$ x $\pi_{BC}(R))$. If projection is extended naturally over tables with OR-sets (essentially the same as projection on relations) and join is extended the way we described before then the f(R) = 0, while we have to have f(R) = {<a,c>} (this is a true set of f(rep(R)). Indeed, after computing two projections join will not unify two OR-sets[4]. Notice that the proposed operation of join would be correct if our set of operations contained *only* join and no other operations. It is the interaction of join with projection which causes problems. The common mistake in extending relational algebra is to consider correctness of each operation separately ignoring their interaction with other operations. These interactions are captured by the recursiveness requirement.

Clearly join cannot be defined "locally" (at the level of individual tuples); in this example four tuples contributed to <a,c> in the result, this number could be arbitrary large. What is therefore a correct and complete extension of projection and join which would preserve true sets and be recursive, i.e. satisfy both our requirements? Suprisingly, the answer is negative - there is *no* such extension; the expressive power of tables with OR-sets is not sufficient to support both projection and join. In order to demonstrate this informally we construct two tables R and S with OR-sets and a relational algebra expression f such that f(T) would have to represent "arbitrary" disjunctions. Let R, S and U be the following relations:

[4]In order to fix this we would have to introduce marked nulls. This was done in [Imielinski 84] in the concept of V-tables

R =

A	B
a	$\{b_1, b_2\}$

S =

B	C
b_1	c_1
b_2	c_2

U =

B	C	D
b_1	c_1	d
b_2	c_2	d

In order to represent rep(R) x rep(S) by some table T we would have to represent a disjunction of the form "<a,b_1,c_1> or <a,b_2,c_2>" in T. Indeed, otherwise we could not correctly perform a project-join expression of hypothetical table T with U (i.e π_{AD}(T x U)) = {<a,d>}. Unfortunately the disjunction <a, b_1,c_1> or <a, b_2, c_2> cannot be represented by any table with OR-sets. In order to represent such disjunction we would have to extend the notion of a table with OR-sets into a table with "OR-tuples" of the form <a, {<b_1,c_1>, <b_2, c_2>}>

■

The above example illustrates the usefulness of our criterion in checking whether it is possible at all to extend certain operations to certain types of tables. One has to be very careful in dealing with any complex objects in tables (a null value can be treated as the special case of a complex objects) since even such simple extension as the one shown here leads to problems with such simple operations as projection and join. Additionally, our criterion gives an indication how a given type of a table has to be extended in order to support a given set of operations.

As we have seen, the model theoretic approach has its limitations. We now present the proof theoretic approach.

2.2. Proof Theoretic Approach

In the proof theoretic approach, the database is treated as a set of first order formulas. For example relational database corresponds to a conjunction atomic formulas built from predicates corresponding to relations in the database. For example a table

Student	Course
Joe	313
Steve	211

Will be represented as a conjunction of two formulas

R(Joe, 313) and R(Steve, 211).

However, this simple transformation is not sufficient to correctly represent negative information. Indeed,

from the table R we could conclude that Joe is *not* taking 211, while the conjunction of two formulas above does not imply it. For the correct representation of negation one needs additional formulas expressing the so called *Closed World Assumption* [Reiter 78]. It essentially says that if some atomic formula cannot be derived from the database then it is false. In the case of relational databases CWA means that if a tuple is not present in the database then its relationship does not hold in a "real" state of the world. In order to express it formally in logic, Reiter introduced a notion of extended relational theory, where new axioms logically "reconstruct" closed world assumption for relational databases. The interested reader is referred to [Reiter 78]. In [Reiter 85] Reiter extends CWA to incorporate null values. He also presents the algorithm to process queries in the presence of nulls. Interestingly, his algorithm and naive evaluation from [Imielinski 84] are equivalent; the former is top-down though, while the latter is bottom-up. Both methods cannot handle arbitrary selection conditions - this can be incorporated but at additional computational cost. The advantage of proof theoretic approach is that it is more general than a model theoretic approach. In particular, it handles deductive rules in a natural way, as we show in the next section.

3. Incomplete Deductive Databases

A deductive database is a database which in addition to the tuples (or records) in relations also contains rules. The rules are typically referred to as the "database intention". One of the main advantages of the proof theoretic approach is that it treats uniformly both rules and tuples of relations, while the model theoretic approach needs to treat them separately. Typical rules considered in DATALOG have only one consequent are called Horn clauses. Rules occurring in deductive databases with incomplete information are more complex - they have either disjunctive consequences or have existentially quantified variables in consequents:

Example

Let us consider a university database and the following rule: "Every student before taking a course has to take all its prerequisities " This rule can be expressed in the following way:

$$Take(x,y,t) \wedge Prerequisite(u,y) \rightarrow \exists_v Take(x,u,v) \wedge Before(v,t)$$

where Take(x,y,t) stands for "x" taking a course "y" at a semester "t".

Note, that any temporal statement of the form "next week", "last month" etc results in a rule which introduces indefinite data (since it is not known exactly about which point of time in the "next week" or "last month" we are talking).

■

Example

Many diagnostic rules can be viewed as deductive rules either with disjunctive conclusions or with existential quantification. For instance,

$$Have(x,Headache) \rightarrow Have(x, influenza) \vee Have(x, brain\ tumor) \vee ...$$

Recursive rules with disjunctions in the consequent occur in a natural way too. A rule stating that a child inherits a blood group from either his father or his mother is a good example.

■

Formally rules in deductive databases have a form of *function-free* clauses:

(1) $P_1 \wedge ... P_k \rightarrow R_1 \vee ... R_q$[5]

[5]Individual variables and quantifiers are not marked here.

where k≥0 and q>0. A *body* of a rule is the set of all literals occurring on the left hand side of the implication in (1). Any literal occurring on the right hand side of (1) is called a *consequent* literal. Depending on the quantification, we classify these rules as follows:

Definite rules The rule (1) is called *definite* if q=1 and all quantifiers in (1) are universal. These are DATALOG (PROLOG without function symbols) [Bancihlon 86] rules.

Indefinite rules The rule (1) is called *indefinite* if q>1 and all quantifiers in (1) are universal.

Skolem rules These rules have not been considered in [Gallaire 84]. They are Horn rules (q=1) and allow existential quantification with the additional restriction that all body variables in (1) (i.e. variables which appear in the body of (1)) must be universally quantified. Such rules are called skolem rules because after skolemization *skolem functions* [Loveland 79] appear in the consequent literals.

The Indefinite and Skolem rules appear in deductive databases with incomplete information.

4. Complexity

There is a severe price to be paid for allowing incomplete information in the database. This price is the complexity of query processing. It is well known that all relational queries have polynomial data complexity (i.e. each query can be evaluated in time which is polynomial in terms of the database size). It is no longer the case, when the incomplete data is allowed:

Example

The graph 3-colorability problem is, given a graph, to determine whether its vertices can be colored in 3 different colors in such a way that no two vertices connected by an edge are colored by the same color. This problem is computationally very hard - it belongs to the class of NP-complete problems [Garey 79], which are most likely exponential. We show here that 3-colorability problem can be encoded as the following query answering problem in the disjunctive database:

Let V represent a set of nodes of a graph G and let R be a binary relation (a subset of V X V) representing edges of G. Let H be a table with OR-objects constructed as follows:

H ={{<N, {c1,c2,c3}>: N in V} where c1, c2, c3 stand for colors.

Here {c1,c2,c3} stand for an OR-object, that is the tuple <N,{c1, c2, c3}> is interpreted as a disjunction of tuples <N,c1>v<N,c2>v<N,c3>.

Let r and h be predicates corresponding to relations R and H respectively. Let Q be the following query:

$Q = \exists_{x,y,z}\, r(x,y) \wedge h(x,z) \wedge h(y,z)$

It is easy to see that this query is true in the database D=<R,H> (i.e is true in each model M in rep(D)) iff G is NOT 3 Colorable. Therefore evaluating Q in D is CoNP-complete[6].

As a consequence, every correct algebraic evaluation of some simple "project-join" queries will be prohibitively expensive (unless CoNP=P). In particular, there is no way to evaluate Q on the database in the "tuple by tuple" basis (i.e the way the standard relational operations are defined on relations, and the way selection is defined in this paper). Indeed, performing an "independent" test for each tuple would lead to a PTIME query processing procedure.

This has very negative consequences for any "bottom-up" proof procedure for deductive rules in indefinite databases. Each bottom-up application (which is essentially a project-join mapping) could

[6]*Hence, most likely, exponential*

potentially require solving a CoNP-complete problem! Effectively, the only solution is to generate, in each step of a bottom-up procedure all possible disjunctions implied by OR-tables. This solution however is totally impractical. Generally, bottom-up techniques for deductive databases with disjunctive information seem not to present a reasonable alternative to top-down approaches (as it is the case for DATALOG rules where two approaches are quite comparable).

The situations gets much worse when we allow existential quantification into the set of rules as well as recursive rules. As demonstrated in [Imielinski 87] there is no effective algorithm with guaranteed termination to process queries in such databases (i.e. the problem is *undecidable*). In other words the problem is as difficult as evaluation of arbitrary PROLOG programs. However, if existential quantification is "out of" recursion (i.e. there are no recursive or mutually recursive rules with existential quantifiers) then query processing can still be done in PTIME. Indefinite deductive rules, even without recursion, cause complexity of query processing to increase to CoNP-complete. The paper [Imielinski 87] contains a complete set of results about complexity of query processing in the presence of incomplete data. It turns out [Imielinski 89] that for simple databases with disjunctive information called databases with OR-objects (when disjunctions occur only on one column of a relation) we can characterize "difficult" queries completely (i.e those which lead to CoNP-complete query evaluation). A simple algorithm for doing this is provided there - it allows to detect expensive queries at compile time.

5. Conclusions

We gave a short overview of problems related to the representation and processing of incomplete information in relational and deductive databases. It was demonstrated that allowing more incompleteness in the database very quickly leads to a sharp increase in terms of complexity of query processing. We also showed the inherent limits of the bottom-up algebraic approach to query processing. These limits are induced by a correctness criterion, which requires extensions of relational operations to preserve the semantics of tables.

More work has to be done along the lines of [Imielinski 89] where the concept of *complexity tailored design* is proposed. In this approach we only allow incomplete data into the database if it does not increase complexity of a selected set of important queries (views). It is clear, that considering the complexities involved when incomplete data is present, we need more *control* over the type of incomplete information which is allowed in the database. Indeed, the value of information has to be strictly related to the cost of its processing. [Imielinski 89] is a first step in this direction. Another interesting direction is the connection of logical incompleteness of data discussed here with fuzzy and probabilistic databases. Finally, the work in artificial intelligence such as [Levesque 85] where epistemic notions of knowledge and belief are discussed has still to find its way into the database community (Lipski's early paper [Lipski 84b], in fact preceding [Levesque 85] is a step in this direction).

References

[Bancihlon 86] Bancihlon,F, Ramakrishnan.
An Amatour Introduction to Recursive Queries.
In *Proceedings of ACM SIGMOD.* 1986.

[Codd 70] Codd, E.F.
A Relational Model of Data for Large Shared Data Banks.
CACM 13(6):377-387, June, 1970.

[Codd 75] Codd, E. F.
Understanding Relations (installment #7).
FDT Bulletin of ACM SIGMOD 7(3-4):23-28, 1975.

[Gallaire 84] Gallaire H., Minker,J.,Nicolas,J.
Logic and Databases - A Deductive Approach.
ACM Computing Surveys :153-185, 1984.

[Garey 79] Garey M.R. and Johnson, D.S.
Computers and Intractability: A Guide to the theory of NP-completness.
W.H.Freeman and Co, 1979.

[Grant 86] Grant,J.,Minker,J.
Answering Queries in Indefinite Databases and the Null Value Problem.
Advances in Computing Research , 1986.

[Imielinski 83] Imielinski, T.
On Algebraic Query Processing in Logical Databases.
In Gallaire H., Minker, J., Nicolas, J. M. (editor), *Advances in Database Theory II.*
Academic Press, 1983.

[Imielinski 84] Imielinski, T. , Lipski, W.
Incomplete Information in Relational Databases.
JACM 31(4):761-791, October, 1984.

[Imielinski 87] T.Imielinski.
Incomplete Deductive Databases.
November, 1987.

[Imielinski 89] Imielinski,T. and K. Vadaparty.
Complexity of Query Processing in Databases with OR-objects.
In *Proceedings of the ACM Symposium on Principles of Database Systems.* 1989.

[Levesque 85] Levesque,H.
Computer and Thought lecture at IJCAI 85.
1985.

[Lipski 84a] Lipski, W.
On Relational Algebra with Marked Nulls.
In *ACM PODS* . Waterloo, Canada, 1984.

[Lipski 84b] Lipski,W.
Algebra on Tables with Null Values.
In *ACM PODS Symposium.* 1984.

[Loveland 79] Loveland,D.
Automated Theorem Proving.
Springer Verlag, 1979.

[Reiter 78] Reiter, R.
On closed world databases.
Logic and databases.
Plenum Press, 1978.

[Reiter 85] Reiter,R.
A sound and sometimes complete query evaluation algorithm for relational databases with null values.
JACM , 1985.

[Vassiliou 79] Vassiliou, Y.
Null values in database management - a denotational semantics approach.
In *ACM SIGMOD.* 1979.

Approximate Retrieval: A Comparison of Information Retrieval and Database Management Systems

C. M. Eastman

Department of Computer Science
University of South Carolina
Columbia, South Carolina 29208

Abstract

An information retrieval system supports the storage and retrieval of documents and other textual data. Retrieving documents by content usually involves some form of approximate retrieval. It is reasonable to expect that some of the work in information retrieval might be useful in designing database systems to support approximate retrieval. Some of the similarities and differences in approximate retrieval in these two kinds of system are discussed; we consider issues of evaluation, matching, interaction, and clustering.

Information Retrieval Systems

The primary goal of an information retrieval system is to manage information which is in the form of textual data, often documents. Indexing is a problem of central concern in information retrieval; it involves the description of the content of the document through choice of a set of index terms. Indexing can be done manually or automatically. It does not appear to be possible to devise a perfect indexing scheme. Requests which are based on content involve the specification of a possibly complex topic in either natural language or a formal query language. The system then selects documents which match the request.

Documents may have formatted data associated with them as well, but the major problems arise from the use of textual data. An information retrieval system is generally used to handle one set of documents. The document descriptions may be complex, but all documents are described in the same way. Thus there is only one record type being managed by the system. This homogeneity is an important difference from database management systems, which must manage a variety of record types and support combinations of these record types.

There is growing interest these days in systems which can manage both formatted and textual data (and possibly other kinds of data as well). The capabilities of both information retrieval systems and database management systems are being extended. And the line between formatted and unformatted data is fuzzy at best (Eastman, 1988).

There are many books which provide an introduction to information retrieval systems. Three which are appropriate for computer scientists are Salton and McGill (1983), Salton (1989), and van Rijsbergen (1979). Two journals in which more recent work can be found are *Journal of the American Society for Information Science* and *Information Processing and Management*. In addition, the *Annual Review of Information Science and Technology* contains review articles.

Evaluation in Information Retrieval Systems

It is generally accepted that retrieval based on content is imperfect. Systems are unable to retrieve all appropriate items, and some of the items they do retrieve turn out not to be of interest. It is thus necessary to evaluate the *effectiveness* of systems as well as their *efficiency*. The basic concept underlying the evaluation of information retrieval systems is that of *relevance*. A document is relevant to a request if it is on the appropriate topic. Even though relevance is hard to define formally, it has been a useful concept in evaluation. Saracevic (1975) provides a broad discussion of relevance.

Two measures are commonly used in evaluating the effectiveness of information retrieval systems. *Recall* is the fraction of relevant documents which are retrieved. *Precision* is the fraction of retrieved documents which are relevant. A number of other measures have been developed, but recall and precision continue to be most widely used. An excellent book dealing with experimentation and evaluation in information retrieval is Sparck Jones (1981).

Evaluation in Approximate Database Retrieval

Evaluation efforts in the database field have focussed on questions of efficiency rather than questions of effectiveness. Since the items to be retrieved can be clearly identified, effectiveness is not a problem. However, the use of approximate retrieval in databases may present such problems. Identifying attribute values on a numeric scale which are close to a specified value is straightforward. However, identifying tuples which are close to other tuples involves multiple attributes. The selections and the ranking depend of the distance metric and weights used. Some of the tuples selected may meet the user's requirements better than others. It might thus be appropriate to evaluate the effectiveness of the system using measures similar to those used for information retrieval systems.

Matching in Information Retrieval

The approach used to match documents to queries depends on the underlying model of information retrieval used. In the boolean model, queries are expressed as boolean expressions; documents which match the boolean expression are retrieved. In the vector model, both queries and documents are described by vectors of terms, which may be binary or weighted. If weights are used, they are usually determined automatically rather than being supplied by the users. A distance or similarity measure is used to select documents which closely match a given query. Documents can be easily ranked in the vector model, and some techniques have been developed for ranking documents in boolean systems.

A large number of dissimilarity and similarity measures have been examined for possible use in information retrieval systems. These are often, but not always, distance metrics. The ones most commonly used in experimental work have been the Jaccard and the cosine measures. Several fuzzy measures have been proposed, but they have not been implemented in experimental systems. The overall conclusion from studies comparing different similarity measures in information retrieval is that, while there are some differences in performance, any reasonable measure will work. (A reasonable measure may be regarded as one in which the similarity increases monotonically as the number of matching concepts increases.)

Matching in Approximate Database Retrieval

Approaches comparable to both the boolean and the vector models have been used in approximate database retrieval. However, the attributes used in a database schema are generally chosen from several different domains. In addition, it is necessary to consider not only the similarity between two attribute values but also the similarity between two tuples The situation is thus more complex than in information retrieval systems.

A number of similarity measures have been proposed for use in approximate database retrieval. These include fuzzy measures, conventional distance metrics, and tabular measures. A system may support more than one measure. The effectiveness of such measures can be evaluated using approaches similar to those used in information retrieval. However, I would expect that the specific choice of measure would make little difference. It might be more productive to compare the efficiency of possible measures.

Interactions in Information Retrieval Systems

It is very difficult to construct a satisfactory query to an information retrieval system the first time. Therefore, most information retrieval systems provide support for iterative specification of queries. The query modifications may be done by the user, by the system, or by both.

Boolean systems generally provide support for query modification by automatically numbering sets of retrieved documents. In addition, the sizes of the sets are given. These set numbers can then be combined in later statements in order to find unions or intersections of the sets. This ability means that users do not need to construct complex queries in one step. They can revise the query in order to get a better description of the topic and to adjust the number of retrieved items.

Relevance feedback can be used for automatic query modification. The system presents the current responses to the user, who then indicates which of them are relevant. The system uses these responses to modify the query, usually by increasing the weight for terms found in relevant documents. Additional documents can then be retrieved. It has been found that approximately three cycles of feedback are sufficient. Evaluating the effectiveness of relevance feedback presents serious difficulties. One impact of relevance feedback is to improve the similarities of relevant documents which have already been retrieved. It is generally accepted that evaluation should be based on the retrieval of newly retrieved relevant documents rather than improved ranking of already retrieved items; several approaches have been developed for this purpose.

Interactions in Approximate Database Retrieval

The forms of interaction described for information retrieval systems can be readily supported because it is assumed that all documents stored in the system are represented in the same way. It is thus easy to modify sets of retrieved documents, because all sets are compatible. In a database management system, the database will usually contain multiple incompatible relation types. A set of tuples selected from one relation can not be merged with another set of tuples selected from an incompatible relation.

In many cases approximate database retrieval may involve only one relation, which might be a base relation or a view. In this situation, the support of automatic set definition and the ability to perform set operations might be useful. However, it would be hard to extend the kinds of approaches used for set manipulation in information retrieval systems to a more general database context. The markings defined by van de Riet *et al.* (1981) allow the management of sets of tuples; however, their use would be much more complex than set management in information retrieval.

In relevance feedback the modifications to the query are made by the system rather than by the user. If it can be successfully implemented in a database context, relevance feedback would be a desirable feature in a database supporting approximate retrieval on multiple attributes. The retrieval by reformulation supported in RABBIT is an example of a form of feedback in a database (Williams, 1984). However, in RABBIT the user, rather than the system, makes the changes in the query. It appears that further work in this area would be desirable. The effectiveness of such retrieval would need to be evaluated.

Clustering in Information Retrieval Systems

Techniques for clustering documents and index terms have been extensively investigated. The goals include improvements in both efficiency and effectiveness. It is hypothesized that documents which are similar to relevant documents are likely also to be relevant. Many different clustering algorithms have been investigated. Experiments have shown improvements in efficiency and effectiveness using clustering; however, the improvements are not dramatic.

Clustering in Database Management Systems

In both information retrieval and database management systems, clustering is used to group together similar items. However, the usage in the database area is narrower; the term clustering is generally used to refer to the physical grouping of records. It is assumed that records which are likely to be retrieved together should be stored together if possible. The clustering techniques investigated in information retrieval (and other areas) usually involve measures of similarity and dissimilarity similar to those used in matching documents and queries in the vector model. Such clustering techniques might well be useful in approximate database retrieval. There has been little work in the database area using such clustering; one approach is described by Hartzband, Holly and Maryanski (1987) and Hartzband and Holly (1988).

Summary

Research results in information retrieval suggest some potentially productive research areas in approximate database retrieval. The evaluation of effectiveness of approximate retrieval could make it easier to compare alternative approaches. Investigation of interactions could lead to improved interfaces. Another area of investigation involves systems combining both textual and formatted data, in which the techniques of approximate retrieval for information retrieval and database management systems could be compared.

Another potential area of investigation is the study of different similarity measures and different clustering algorithms. These questions have been extensively investigated in the area of information retrieval. Although some approaches are better than others, the results are not generally conclusive. This is true in a number of other fields in which multidimensional data analysis is used. Results in the approximate database area would probably be similarly nonconclusive.

Of course, there are problems in approximate database retrieval for which work in information retrieval systems provides no guidance. For example, it is necessary to handle multiple domains and multiple relation types in a database system. This problem does not arise in an information retrieval system.

References

C. M. Eastman, "Formatted and unformatted character data types," Technical Report TR 89002, Department of Computer Science, University of South Carolina, June 1988.

D. Hartzband, L. Holly, and F. Maryanski, "The provision of induction in data model systems. I. Analogy," *International Journal of Approximate Reasoning,* Vol. 1, No. 1, January 1987, pp 5-22.

D. Hartzband and L. Holly, "The provision of induction in data model systems. II. Symmetric Comparison," *International Journal of Approximate Reasoning,* Vol. 2, No. 1, January 1988, pp 1-14.

G. Salton, *Automatic Text Processing: The Transformation and Analysis, and Retrieval of Information by Computer,* Addison- Wesley, Reading, Massachusetts, 1989.

G. Salton and M. McGill, *Introduction to Modern Information Retrieval,* McGraw-Hill, New York, 1983.

T. Saracevic, "Relevance: A review of a framework for the thinking on the notion in information science," *Journal of the American Society for Information Science,* Vol. 26, 1975, pp 321-343.

Karen Sparck Jones, ed., *Information Retrieval Experiment,* Butterworths, London, 1981.

Reind P. van de Riet, Anthony I. Wasserman, Martin L. Kersten, and Wiebren de Jonge, "High-level programming features for improving the efficiency of a relational database system," *ACM Transactions on Database Systems,* Vol. 6, No. 3, September 1981, pp 464-485.

C. J. van Rijsbergen, *Information Retrieval,* Second Edition, Butterworths, London, 1979.

M. D. Williams, "What makes RABBIT run?," *International Journal of Man-Machine Studies,* Vol. 21. No. 4, October 1984, pp 333-352.

From Browsing To Querying *

Alessandro D'Atri †
Dip. Ingegneria Elettrica, Università dell'Aquila
Laura Tarantino
Dip. Informatica e Sistemistica, Università di Roma "La Sapienza"

Abstract

Three styles of interaction with databases: browsing, connection under logical independence, and generalization are discussed by giving their requirements, characteristics and limitations. These styles are suitable for users that have vague knowledge about their goals, the data model, the database, and the database system. A combination of these techniques can be used in an integrated environment wherein a naive user, starting from browsing, can gradually specify his goals more and more precisely, possibly arriving at complex formal query.

1 Introduction

Two distinct models of the real world exist in a database environment: the one in the mind of the user and the one stored in the system. Thus, when talking about imprecision in databases, we must distinguish between *user's imprecise information* (i.e., the user has only vague knowledge on the way the system operates and/or the information it handles) and *system's imprecise information* (i.e., the system contains only vague description of the world). Even if in the most general case these models may be disjoint and may correspond to distinct (and possibly conflicting) representations of reality, for the sake of simplicity we assume here that the user's model is a subset of the system's model, which is an approximation of reality. Hence, in both cases of imprecision there is a problem of information exchange necessary to make these model as closer as possible; the first discrepancy is covered by database querying, the second one by database evolution. Even if some formalisms and techniques may be used in both situations, the user-system interaction problem,which is considered in this paper, is more complex, due to the human-computer interaction requirements.

Imprecision in the user's model may be classified as follows [12]:

- incomplete knowledge of the data model,
- imprecise information on the database schema and/or its instance,

*Work partially supported by the Commission of the European Communities under the ESPRIT project P2424 "KIWIS"

†Partially supported by the MPI under the project on "Formal Methodologies and Tools for Advanced Databases".

- vagueness of user goals,
- incomplete knowledge about the interaction tools (query language and/or interaction primitives).

Traditional relational database systems deal mainly with the second kind of imprecision. In fact, schema information is stored in a separate set of system relations (data dictionary) that can be manipulated like other relations. This leads to distinct interaction environments for schemes and instances and to an interaction process that consists of two sequential phases: schema understanding and database querying. On the other hand, little effort has been given to improve the schema understanding phase in traditional systems, since the size of the schema is very small relative to the size of the instances (strongly typed data models) and instances are supposed to be the major unknown portion of the database. This approach is adequate as long as the user is regular and the schema is small enough to assume the user can easily learn the database structure (possibly by simply browsing in the data dictionary). The approach is weak and inflexible when moving from traditional databases to semantically richer systems (e.g., knowledge bases) without strong data typing and where the schemes are much larger.

The following general classification can be adopted for interaction techniques that take care of the above kinds of imprecision:

- *browsing*, when an observation point is moved in the database;
- *connection under logical independence*, that maintains distinct user's and system's models, taking care of the goal interpretation in terms of queries by a connective approach;
- *generalization*, based on abstractions, adjustments and weakening of previous answers or examples.

This paper discusses requirements, objectives and limitations of these techniques, all of which address distinct aspects of the imprecision in the mind of the user.

2 Browsing

Browsing is essentially a *viewing* technique, aimed at gaining knowledge about the database, rather than a querying technique, and it can handle in a homogeneous way both schemes and instances, without making any distinction between them. The main hypothesis is that the user has no predefined goal and little knowledge about the database and the interaction techniques. The user begins by examining a concept and its neighborhood (adjacent concepts can be considered as a first level of explanation of the examined concept); then a new element is selected from this neighborhood by the user, to be the current one, and its neighborhood is shown; this process continues iteratively. Thus an incremental (step-by-step) enhancement of user knowledge is obtained, by exploring concepts that are logically adjacent.

A browsing environment requires the definition and the management of:

- An *adjacency structure*: if the data model is network-like, adjacency is derived from the network itself; if it is table-like, tuple and attribute adjacency may be considered (notice that

in this case it may be necessary to define an ordering on tuples and/or attributes).

- One or more visible portions of the database (*points of view*), which are presented to the user at any point in the browsing session by a (usually graphical) presentation formalism.
- *Navigation primitives*, that allow the user to establish a starting point in his interaction and to change the point of view in the adjacency structure.

Major limits which are an immediate consequence of the browsing paradigm are:

- The exploration process is a "short-sighted" navigation, since only concepts adjacent to the current one can be reached; hence with large and complex structures, relevant portions of the structure may remain unexplored.
- The user may get confused because of the extremely detailed information shown: no summary is possible (due to the lack of distinction between schema and instances), and no shortcuts are possible to arrive at a more distantly related concept without navigating along all intermediate concepts.
- Most relational database browsers (e.g., [6, 17]) have only limited exploration capabilities, as users are confined to a single relation at a time, and crossing relation boundaries is very hard.
- After a while the step-by-step process becomes tedious, unless the system provides some tools for "automating" the process.
- The browsing environment and the querying environment are distinct, thus separating the learning and the querying activities.

Some browsers described in the literature try to overcome these limitations by various enhancements. For example:

- Graphical interaction primitives (e.g., direct manipulation techniques) and graphical presentation techniques may be used to speed up and to facilitate the formulation of navigation commands [5, 7, 18].
- Neighborhoods could be organized in suitable homogeneous subsets and distinct ordering criteria within each subset could be provided to ease their exploration [16].
- Simple ad-hoc data models have been developed, suitable for navigational browsing [11]; other proposals use network views of relational databases, which are more immediate to users and facilitate inter-relation browsing [12].
- The system may allow the user to select the starting point (e.g., by providing *access by value* [12]); if browsing history is maintained, then users are able to retract their steps.
- Tools should be provided to store browsing results [16].

3 Connection Under Logical Independence

This approach is oriented to users with predefined *goals*, but with only partial knowledge about the database structure and its content; in particular they can express a goal as a set of independent

references to elementary information, without knowing how these references are related in the database. Hence, goals are translated by the system into *queries*, first by providing an *interpretation* of the goal (e.g., in the case of a declarative query language, an assertion expressing the query) and then by presenting the *answer* (i.e., the database portion that agrees with this assertion). If the *logical independence* approach is followed by the system, both the interpretation and the answering strategies are encapsulated in the system which shows the user only the information strictly related to the disambiguation and answer presentation processes.

For example, a universal relation interface (e.g., [8, 9, 10]) allows the user to formulate a goal by giving a set of attribute names and data values, without being aware of the logical aggregation of attributes into relation schemes. The interpretation process translates this partial query in terms of the underlying database structure by deciding which relations are to be joined to obtain the answer.

In semantically richer data models (e.g., semantic networks) logical independence may be provided by allowing the user to formulate goals in terms of a set of (complex or elementary) object names, without knowing how such objects are composed, to what conceptual level they belong and how they are related [2]. The interpretation process is devoted to identifying a set of object interrelationships that provides the query intention, and the database portion matching it.

Hence, by formulating goals in terms of abstract concepts, logical independence frees the user from the "logical navigation" through the database, in the same way the relational model free users from the "physical navigation"; the system takes care of the navigation through the schema in order to solve the goal interpretation problem (i.e., the translation of the partially specified query into a complete one by identifying the user intention) in a first phase, and of the navigation through instances during the subsequent answering phase.

Main assumptions and implications in this approach are:

- A simplified user model of the world is needed, that is used for specifying goals and for providing logical independence from the system's model of the world. We may in general suppose that such a simplified model is a destructurization or a summary of the system model (e.g., users of universal relation interfaces see the entire database as a single fictitious relation). Notice that in this model more importance may be given to the fact that a connection exists between two objects, rather than to the exact description of this connection.

- The system handles the goal on the basis of its (richer) data model, according to a *connectivity* approach. In fact, a possible query interpretation (and its corresponding answer) is obtained by connecting together the concepts involved in the goal within the database schema (and the correspondent database instance). More than one connection in the schema may correspond to the set of concepts specifying the goal, and hence more than one answer may be obtained. This problem of ambiguity must be addressed.

Major limits of this approach are:

- Goals can only be specified in a "context-free" way. This follows from the hypothesis that the model of the world the user has in mind is completely distinct from the system model, in terms of the abstraction primitives that are reflected in the connection structure. Hence, the system is not able to handle a partially structured goal, which may contain some structural

information which could ease the disambiguation process.

- A too rigid and static separation between the user and the system models, which does not take care of the user's learning process that reconciles the two models during the interaction.
- The approach is strictly decomposed in two sequential phases, not allowing the user to influence the disambiguation process.
- The further apart the user and the system models, the more difficult it is to interpret the goal. This may give the user the feeling that the answer is not the requested one, and the need arises for techniques that explain the interpretation and ask for confirmation.

Most proposals in the literature deal with ambiguity only in the relational model, and can be classified as follows:

- A *static* approach, (*system-selected interpretation*); some methods present the user the union and/or the intersection of all answers corresponding to all possible query interpretation (thus defining a *maximal-minimal* range among interpretations) [9]. Conversely, other proposals are based on minimization criteria: the system tries to find an interpretation (i.e., a connection in the schema) involving the minimum number of auxiliary objects (e.g., to minimize the number of relation schemes needed, or the number of edges in the semantic network) [1, 2, 19].
- A *dynamic* approach, (*user-selected interpretation*); it is based on the assumption that several meaningful interpretations of the user query are possible and some of them are distinguishable within the user's model. Hence a man-machine dialogue is triggered, devoted to meet the user intentions, assuming that the user can recognize new concepts as "related" or "not related" to his goal [4, 14].

4 Generalization

The main peculiarity of this approach is its *bottom-up* nature, "from answers to goals", in contrast with the conventional *top-down* querying procedure, where the user formulates the goal and the system retrieves the answer. A first kind of generalization occurs when the user provides an "example of the answer" and the system identifies the goal by *generalizing* such an example. Alternatively, starting from an answer that is not considered satisfactory, the user may ask to modify the original query slightly (i.e., for a generalization of it) in order to obtain a new answer "near" the previous one.

The most known application of the first kind of generalization, in the relational model, is Query By Example [21]. In this querying environment the user specifies a set of tables with several tuples. Since these tuples can contain both constants and domain variables, an example of the requested answer is stated. The system, starting from this example, identifies the query, and answer it by filling the tables with tuples that match the example. Form-based interfaces (e.g., [17]) are another example of this generalization approach in relational environments.

This approach may also be applied to semantic networks and other graph-based data models by considering the example generalization as a pattern-matching problem. In particular, starting from the graph given as example, the system identifies the set of subgraphs of the database "similar"

to it. A first step in this direction has been done with the *synchronized browsing* [16], where the user can fix (in a tree-like structure obtained by means of browsing primitives) an example of the information he is interested in, which identifies a family of trees in the network; under the user control, the system moves the tree over the network, and shows other trees matching the example, by replacing displayed objects.

The second kind of generalization concerns (iterative) *reformulations* of a starting query, necessary when the obtained answer is recognized not to be the expected one. The reformulation process can be carried on either with the user cooperation (as in [20]) or automatically by the system which relaxes some conditions to find information in the neighborhood of the previous answer [3, 13, 15].

The main requirements of this interaction style are:

- A *definition environment* (i.e., a simple data model and suitable tools to assist the user) for building examples and rehandling previously generated answers.
- A *generalization mechanism* for capturing the abstractions needed for constructing queries from examples.
- A *failure detection mechanism*, activated either by the user or automatically by the system, for identifying those situations in which the answer is unsatisfactory.
- A concept of *distance between answers* to determine the adequate approximation degree.

Limits of this approach are:

- It is oriented towards users that have mostly precise goals, and are familiar with the data model, and with at least those portions of the database necessary for building the example.
- The above requirements become difficult to be fully addressed in a real system, particularly for large and complex databases; the generalization mechanism and the introduction of distance information may be difficult to manage.
- Since the system must behave as an expert assistant, this implies to gather and maintain knowledge about the user, in order to tailor the cooperation (proposals in this direction can be found in [3]).

5 Conclusions

The approaches discussed so far have been proposed for naive users without the expertise necessary to formulate standard formal queries. They provide adequate solutions to different problems in human-computer interaction with database systems, and the choice of a particular interaction paradigm and its inclusion in a user interface are to be motivated by its friendliness, effectiveness and feasibility.

On the other side, we notice that these interaction styles are oriented to different classes of users, characterized by increasing skills and levels of experience with the system and/or the application domain. Thus, due to complementary characteristics and limits they have, we believe that an integration could lead to a powerful environment aimed at assisting the user in all the phases of

the interaction with the system covering in a continuum the gap between browsing (for very naive users) to querying (for sophisticated users). Suitable combinations of these approaches allow to define new interaction styles that overcome the limits of their separated use. For instance, merging a browsing technique with a generalization technique eases the examples formulation (see [16]) and merging the browsing with connection aspects helps the user during the exploration leading to a "goal-driven" browsing, and so on.

Since these techniques have been defined and introduced in distinct environments, based on distinct data models, a major problem to be solved for their integration is redefining them in a single and homogeneous way, possibly reasoning in abstract terms, independently from the data model. In particular, we consider a graph formalism general and effective enough to capture the most relevant aspects of these techniques in terms of graph problems.

We are working in this direction within the framework of the KIWIS project (ESPRIT project 2424) to build up an integrated knowledge-based and highly interactive environment for large knowledge and data bases, which combines these three approaches in order to assist the user in goal definition, query construction and interpretation, and answer presentation.

References

[1] G. Ausiello, A. D'Atri, M. Moscarini. Minimal Coverings of Acyclic Database Schemata. *Advances in Data Base Theory*, vol.2, (H.Gallaire, J.Minker and J.M.Nicolas, eds.), Plenum Press, New York and London: 27–52, 1984.

[2] G. Ausiello, A. D'Atri, M. Moscarini. Chordality Properties on Graphs and Minimal Conceptual Connections in Semantic Data Models. *Journal of Computer and System Science*, 33(2):179–202, October 1986.

[3] F. Cuppens, R Demolombe. Cooperative answering: a methodology to provide intelligent access to databases. *Proceedings of 2nd International Conference on Expert Database Systems*, pages 333–352, Vienna, VA., 1988.

[4] A. D'Atri, P. Di Felice, M. Moscarini. Dynamic query interpretation in relational databases. *Information Systems*, 14(3), 1989.

[5] A. D'Atri, E. Laenens, A. Paoluzzi, J. J. Snijders, L. Tarantino. A graphical browser to object-oriented knowledge bases. *Journal on Database Technology*, to appear, 1989.

[6] *DBASE-III Reference Manual.* Ashton-Tate, Culver City, California, 1984.

[7] C. Herot. Spatial management of data. *ACM Transactions on Database Systems*, 5(4):493–513, December1980.

[8] H. F. Korth, G. M. Kuper, J. Feigenbaum, A. van Gelder, J. D. Ullman. System/U: A database system based on the universal relation assumption. *ACM Transactions on Database Systems*, 9:331–347, 1984.

[9] D. Maier, D. Rozenshtein, D. S. Warren. Window functions. *Advances in Computing Research*, Vol. 3, (P. Kanellakis, F. P. Preparata, eds.), pages 213–246, 1986.

[10] D. Maier, J. D. Ullman, M. Y. Vardi. On the foundations of the universal relation model. *ACM Transactions on Database Systems*, 9(2):283–308, 1984.

[11] A. Motro. Browsing in a loosely structured database. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 197–207, ACM, New York, New York, 1984.

[12] A. Motro. BAROQUE: an exploratory interface to relational databases. *ACM Transactions on Office Information Systems*, 4(2):164–181, April 1986.

[13] A. Motro. SEAVE: a mechanism for verifying user presuppositions in query systems. *ACM Transactions on Office Information Systems*, 4(4):312–130, October 1986.

[14] A. Motro. Constructing queries from tokens. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 120–131, ACM, Washington D.C., 1986.

[15] 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.

[16] A. Motro, A. D'Atri, L. Tarantino. KIVIEW: The design of an object-oriented browser. *Proceedings of 2nd International Conference on Expert Database Systems*, pages 17–31, Vienna, VA., 1988.

[17] M. Stonebraker et al. The design and implementation of INGRES. *ACM Transactions on Database Systems*, 1(3):189–222, 1976.

[18] M. Stonebraker and J. Kalash. Timber: a sophisticated database browser. In *Proceedings of the Eighth International Conference on Very Large Data Bases*, pages 1–10, VLDB Endowment (available from Morgan-Kaufmann, Los Altos, California), 1982.

[19] J. A. Wald, P. G. Sorenson. Resolving the query inference problem using Steiner trees. *ACM Transactions on Database Systems*, 9(3):348–368, 1984.

[20] M. D. Williams. What makes RUBBIT run? *International Journal on Man-Machine Studies*, 21(4):333–352, October 1984.

[21] M. M. Zloof. Query by example. *IBM Systems Journal*, 16(4): 324–343, 1977.

A Trio of Database User Interfaces for Handling Vague Retrieval Requests

Amihai Motro
Computer Science Department
University of Southern California
University Park, Los Angeles, CA 90089-0782

Abstract

We discuss three user interfaces for situations that involve vague retrieval requests: (1) BAROQUE, a browser for relational databases; (2) VAGUE, a query interpreter for relational databases that can handle neighborhood queries (formal queries with *similar-to* comparators); and (3) FLEX, an adaptive interface to relational databases that can service satisfactorily users with different levels of expertise (users who submit queries of different levels of correctness and well-formedness).

1 Introduction

Most database systems and their associated retrieval tools are designed under the assumption that requests for retrieval would be precise and specific. In this paper we discuss three user interfaces, called BAROQUE, VAGUE and FLEX, that were developed for situations that involve *vague* retrieval requests. The term "vague" refers here to two different situations:

- The user is *naive* and cannot express a formal query. This may be either because he is not familiar with the data model used by the system, or with its formal query language, or with the particular database he wishes to access. It may also be because he does not have a specific retrieval goal, or cannot express his goal in the terms required by the system.
- The user is *sophisticated* and possesses all the knowledge required to use the system expertly. However, his request requires a condition which is vague or imprecise.

BAROQUE is a browser for relational databases; VAGUE is a query interpreter for relational databases that can handle neighborhood queries (formal queries with *similar-to* comparators); and FLEX is an adaptive interface to relational databases that can service satisfactorily users with different levels of expertise (users who submit queries of different levels of correctness and well-formedness). Thus, BAROQUE and FLEX are capable of handling vague requests of the first kind, while VAGUE handles vague requests of the second kind.

All three interfaces access standard relational databases that are managed by the INGRES database system [10].

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

2 BARQOUE

To improve their usability and responsiveness most database systems offer their users a wide variety of interfaces, suitable for different levels of expertise and different types of applications. A particular kind of interface which is now commonly available are *browsers*. Browsers are intended for performing *exploratory searches*, often by *naive users*. Thus, they usually employ simple conceptual models and offer simple, intuitive commands. Ideally, browsing should not require familiarity with the particular database being accessed, or even preconceived retrieval targets. While browsing, users gain insight into the contents and organization of the searched environment. Eventually, the search either terminates successfully or is abandoned.

Often, the conceptual model is a *network* of some kind, and browsing is done by *navigation*: the user begins at an arbitrary point on the network (perhaps a standard initial position), examines the data in that "neighborhood", and then issues a new command to proceed in a new direction. An example of this approach is the interface designed and implemented by Cattell [1]. The interface is to an entity-relationship database, and it features a set of directives for scanning a network of entities and relationships, and presenting each entity, together with its context in a display called *frame*.

Browsers have also been developed for relational systems, for example, SDMS [8], TIMBER [18] and DBASE-III [4]. These are actually tools for scanning relations (including relations that are results of formal queries), and therefore have only limited exploration capabilities. Browsing is confined to a single relation at a time, and it is not possible to browse across relation boundaries. If a user encounters a value while browsing, and wants to know more about it, he must determine first in what other relations this value may appear (quite difficult), then formulate a formal query, and resume browsing in the new relation. Satisfying questions such as "Is x related in any way to y?" is impossible without an extensive scan of the database.

BAROQUE [13] is a relational user interface, designed to address these deficiencies. With the help of an additional relation, but without affecting the existing relational database otherwise, BAROQUE effectively replaces the record-oriented view which is inherent in the relational model, with a network view, making its actual tabular representation transparent. Such networks can support browsing functions of greater utility. More specifically, BAROQUE employs an "entity directory" relation, which associates every database value with the attributes under which it appears. By thus "inverting" the database, BAROQUE can effect "entity behavior": all occurrences of a particular data value are considered collectively as one entity; this entity is related to other entities through the functional dependencies in which the individual occurrences participate. When the user specifies a data value, BAROQUE can construct the appropriate entity and its relationships. The effect resembles a semantic network , in which users can browse with functions like "Describe x" or "List others like x" or "Explain the connection between x and y". Such *access by value* is especially important for users with no knowledge of the organization of the database. For example, the request "Describe CHAPLIN" would construct a frame named CHAPLIN which tabulates all its relationships to other entities; e.g., NAME of PERSON having: FIRST_NAME CHARLES, COUNTRY BRITAIN, YEAR_OF_BIRTH 1889,... DIRECTOR of FILM having TITLE: MODERN TIMES, THE GOLD RUSH, THE CIRCUS,...

Among its other features, the system incorporates the database schema into the same representation; it allows users to switch rapidly back and forth between formal querying (in QUEL [19])

and browsing; and it relies on menus to facilitate communication with users. Upon entry to browse mode, the name of the database is established as the default topic. Thus, this most general entity is provided to the user as the end of a thread. Following it the user may survey the database, and ultimately reach every other entity.

The cost entailed by the browser, in terms of the additional space to store the entity directory and the additional computation for its initialization and its continuous update, is comparable to the cost of a secondary index on every database attribute. If sufficient storage is unavailable, it is possible to implement only part of the entity network, by inverting on selected attributes only. All other values will be listed while browsing in their neighborhoods, but they may not become topics of browsing requests. This has the interesting effect of distinguishing between actual *entities* that participate in relationships, and simple *properties* that describe entities. In fact, the resulting model resembles the Entity-Relationship approach to data modeling. One possible strategy for selective inversion is to invert only on attributes that are keys or foreign keys. Under this strategy, every entity that is assembled occurs at least once as the value of the key in some relation.

Our method for assembling entities is based only on identities of data values. Consequently, values that possess different meaning altogether, but are expressed with the same string of characters, are assembled into one entity. This weakness can be attributed to the limited semantic capabilities of the basic relational model, where the only information available on the meanings of the different attributes are their names and their primitive types (e.g., integer, character). A well-known enhancement to the relational model uses a stronger concept of *domains* to classify the attributes. This enhancement can be readily incorporated into BAROQUE to assemble separate entities for values that belong to multiple domains. For example, assume a database with attribute SALARY defined over the domain DOLLARS, and attribute YEAR_OF_BIRTH defined over the domain YEARS. If the value 1889 appears under both PRICE and YEAR_OF_BIRTH, BAROQUE will create two separate entities: 1889 DOLLARS and 1889 YEARS. Notice, however, that even with the current approach, the names of relationships in which 1889 participates provide different *interpretations* for this entity. For example, BAROQUE will describe the topic 1889 with both PRICE of ITEM having ITEM_NO 6710 and YEAR_OF_BIRTH of PERSON having NAME CHAPLIN. Thus, while the information included in these answers combines different semantics of the entity 1889, it is interpreted clearly, and the user can disregard the portion of the answer that is irrelevant.

3 VAGUE

Requests for data can be classified roughly into two kinds: *specific* queries and *vague* queries. A specific query establishes a rigid qualification, and is concerned only with data that match it precisely. Some examples of specific queries are "How much does Jones earn?" or "When does flight 909 depart?" If the database does not contain salary information on Jones or departure time for flight 909, null answers should be returned; the user is not interested in the earnings of somebody else or in the departure time of a different flight. A vague query, on the other hand, establishes a target qualification and is concerned with data that are *close* to this target. As an example, consider "List the inexpensive French restaurants in Westwood". If there are none, a moderately priced Continental restaurant in Santa Monica may have to do. Similarly, when a project calls for experienced C programmers with background in applied mathematics, we may want the personnel database to mention also that there is an engineer with some knowledge of Pascal.

While many retrieval requests are intrinsically vague, most conventional database systems cannot handle vague queries directly. Consequently, they must be emulated 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. A system that allows users to express vague queries directly is more cooperative, and possibly more efficient. While issues of vague retrieval have been addressed in related disciplines, particularly information retrieval (see [17] or [20]) and fuzzy systems (e.g., [16, 22]), little has been done to extend current relational database technology to provide adequate tools for performing vague retrieval (one exception is [9]). VAGUE [15] is a system that extends the relational data model to provide it with vague retrieval capabilities.

To determine similarity between data values we introduce the notion of distance. Each database domain is provided with a definition of distance between its values, called *data metric*. For example, in a database on restaurants there may be metrics to measure distances between cuisines, between locations, between price ranges, as well as a metric to measure distances between restaurants.

To express vague queries we introduce a vague selection comparator, called *similar-to*. A *similar-to* comparison is satisfied with data values that are within a predefined distance of the specified value. For example, the vague comparison "location *similar-to* Westwood" may be satisfied by Westwood, Santa Monica and Beverley Hills.

Thus, the previous specific query "List the restaurants whose cuisine is French, whose price range is inexpensive, and whose location is Downtown" may be relaxed into a vague query such as: "List the restaurants whose cuisine is *similar-to* French, whose price range is *similar-to* inexpensive, and whose location is *similar-to* Downtown".

This model is quite straightforward, and its satisfactory operation relies almost entirely on the quality of the metrics that are provided for the individual domains. Here, VAGUE allows the database designer four choices. He could use one of several *built-in* metrics; he could provide a procedure that *computes* the distance between every two elements of the domain; he could provide a relation that *stores* the distance between every two elements of the domain; or he could use a *reference* relation (an existing database relation that is keyed on this domain). In the latter case, distances between elements of the domain would be defined as distances between their tuples in the reference relation, where tuple distance is defined as a combination of the individual distances between their corresponding components.

Each tuple in the answer to a query that includes several vague qualifications involves several deviations from the specific values mentioned in these qualifications. By combining these individual deviations into a single value, VAGUE can present the answer to the user in order of optimality. Thus, there are two occasions where VAGUE combines several component distances into a single distance: in one of its metric types, and in the presentation of vague answers.

The design of VAGUE reflects two fundamental requirements: conceptual simplicity within a relational framework and adaptability.

The purpose of VAGUE is to enhance a relational database system with vague retrieval capabilities. An important design guideline is to realize this goal with only minimal deviation from this popular model. The relational data model is extended with a single concept: *data metrics*, and the query language is extended with a single feature: a *similar-to* comparator. (Indeed, the relational data model is *generalized*, since a non-metricized database is a particular type of a metricized

database.) To present queries, users need only to know about the new comparator.

To be useful, a system that implements vague queries, must be able to adapt itself to the views and priorities of its individual users. VAGUE incorporates three adaptability features. (1) Often, distances between values of a given domain may be measured according to various metrics. For example, distances between values of domain CITY may be defined in miles "as the crow flies", or as shortest driving distances, or even as differences between the names of the cities. VAGUE permits *multiple metrics for the same domain.* When a query makes use of a *similar-to* comparator, the user is presented with the various possible semantics of this comparator in its present context, and is asked to select. (2) With referential metrics, one of the metric types available in VAGUE, individual users are allowed to *influence the definition of the metric* according to their own views. For example, the distance between two cities may be defined as a combination of the distances between some of their available attributes, such as size of population, climate, and employment rate. If such a metric is selected, the user is allowed to judge the relative importance of the various attributes in the overall distance. (3) When a query involves several vague qualifications, users are allowed to *express their relative importance* in the overall query. For example, consider the previous vague query about restaurants whose cuisine is similar to French, whose price range is similar to inexpensive, and whose location is similar to Downtown. Each tuple in its answer involves three deviations from the specified values, which are then combined so that the answer may be presented in order of optimality. However, it may be that the user has different willingness to compromise on the various qualifications; for example, the user may be willing to compromise more on the type of the restaurant than on its price range or location. VAGUE allows users to express their relative willingness to compromise, and uses this input in the definition of the corresponding metric.

The design of VAGUE represents a compromise between the sometimes conflicting requirements for simplicity, flexibility and efficiency. Some examples of design compromises are described below. Users of VAGUE cannot provide their own similarity thresholds for each vague qualification. It was observed that this will require that users become familiar with particular data metrics. Instead, VAGUE allows its users to double the threshold and repeat the query. Similarly, except for the ability to enter weights for referential metrics, users of VAGUE are limited to interpretations of similarity (i.e., metrics and thresholds) that have been provided by others. While it is possible to design an interface that will permit users to introduce their own interpretations of similarity, it was determined that the complexity of this task usually would exceed the expertise of many users, especially casual users. Instead, this task is reserved for database designers or administrators, and users are invited to select from menus of metrics that are currently supported. To prevent the querying process from becoming too tedious to the user, VAGUE tries to be economical in its dialogue with the user. At several places it may be possible to gain flexibility by additional interaction; for example, when a vague query does not match any data, it is possible to ask the user which *similar-to* comparator should be weakened (currently, all are weakened simultaneously). Finally, since each tuple in an answer to a vague query must satisfy *all* the vague qualifications, it is possible that a tuple would not be retrieved, even if its total compromise is smaller than that of tuples that were retrieved. This approach was adopted primarily for reasons of efficiency. In addition, because the combination of individual distances into a single distance is sometimes risky, VAGUE prefers not to rely on it for *determining* its answers, only for *ranking* them.

The issue of appropriate similarity measures for retrieval has been researched and debated extensively. Our purpose in designing and implementing VAGUE is not to resolve this issue by adopting any one particular approach, but to provide relational databases with a flexible mechanism, with which different kinds of data metrics may be implemented and tested.

A legitimate concern is that vague queries will be satisfied by meaningless values. Careful selection of the metrics and the parameters during the design of the database is extremely important. Also, by extending the answers to include the values with which the vague qualifications were satisfied (a feature available in VAGUE), users can monitor the judgements made by the system. Finally, it can be assumed that users who consciously present vague queries (to systems or to humans) are well aware of the fact that subjective judgement is involved, and would probably examine answers to vague queries more carefully than answers to specific queries.

4 FLEX

A common method for accessing databases is via query language interfaces (e.g., QUEL [19], SQL[2]). A query language interface defines a formal language, in which all retrieval requests must be expressed. The main advantages of query language interfaces are their *generality* (the ability to express arbitrary requests) and their *unambiguity* (each statement has clear semantics). However, using query language interfaces requires considerable proficiency: Users must understand the principles of the underlying data model, they must have good knowledge of the query language, and they must be familiar with the contents and organization of the particular database being accessed. In the absence of even some of this prerequisite knowledge, using such interfaces can become very inefficient and frustrating. Hence, most query language interfaces do not accommodate naive users very well.

For such users, several alternative types of interfaces have been developed, including form and menu-based interfaces (e.g., QBF [10]), graphical interfaces (e.g., CUPID [12], GUIDE [21], LID [5], SKI [11]), natural and pseudo natural language interfaces (e.g., RENDEZVOUS [3], LADDER [7], INTELLECT [6]), and browsers (e.g., TIMBER [18], SDMS [8], BAROQUE [13]). These interfaces are oriented towards non-programmers, and therefore require only limited computer sophistication. Expressing requests may be as simple as selecting from a menu or filling a form, and familiarity with the contents or organization of the database is usually not required. However, naive user interfaces usually achieve simplicity and convenience at the price of expressivity. Also, as users acquire more expertise, these interfaces tend to become more tedious to use.

Thus, it appears that no *single* user-database interface exists that can service satisfactorily both experts and naive users. Perhaps the only exception are natural language interfaces. Ideally, such interfaces should be able to service satisfactorily all types of users. Unfortunately, existing natural language interfaces have two major problems: they require enormous investment to capture the knowledge that is necessary to understand user requests, and even the best systems are prone to errors.

FLEX [14] is an experimental user interface designed to be used satisfactorily by users with different levels of expertise. It is based on a formal query language, but is *tolerant* of incorrect input. It never rejects queries; instead, it adapts flexibly and transparently to their level of correctness, providing an interpretation at that level. FLEX is also *cooperative*. It never delivers null answers without explanation or assistance. This tolerant and cooperative behavior is modeled after human behavior, and is thus reminiscent of natural language interfaces.

The most prominent design feature of FLEX is the smooth concatenation of several independent mechanisms, each capable of handling input of decreasing level of correctness and well-formedness.

Each user input is *cascaded* through this series of mechanisms, until an interpretation is found.

Initially, the input is processed by a query *parser* to determine whether it constitutes a proper formal query. If parsing is successful, the query is executed. Otherwise, the input is processed by a query *corrector*, that attempts to salvage the query by applying various transformations. The transformations involve both syntactic and semantic corrections, as well as synonym substitution. The corrector is usually successful whenever the input exhibits recognizable structures, and its interpretations are mostly safe. If the corrector fails to produce an interpretation, the input is processed by a query *synthesizer*, that attempts to conclude proper queries from tokens that are recognized in the input. The basic approach of the synthesizer is to model the entire database as a graph, mark the nodes that correspond to tokens that are recognized in the input, span these nodes with a minimal tree, and then translate the tree into a formal query. As these interpretations are not entirely safe, they are offered as suggestions, and are subject to refinements by the user. Finally, if the synthesizer fails to produce an interpretation, a *browser* is engaged to display frames of information extracted from the database on the recognized input tokens. The basic approach of this mechanism is essentially similar to that of BAROQUE.

Hence, FLEX never rejects queries, and the accuracy and specificity of its interpretations correspond to the correctness and well-formedness of the input.

Because it is engaged only *when* needed and only *as much as* needed, FLEX can be used satisfactorily by users with different levels of expertise, and thus appeal to a more universal community of users. For example, a perfect formal query submitted by an expert will be executed immediately without any modification; while a single word submitted by a novice will flow through the entire sequence of mechanisms until finally it will result in a frame of information about this word. FLEX may be viewed as an interface that *adapts* to the level of correctness and well-formedness of its input (providing interpretations of corresponding accuracy and specificity).

This ability to adapt is complemented with features of *cooperative* behavior, whereby null answers are never delivered without explanation or assistance. If the final answer is null, the original query is passed to a query *generalizer*, which issues a set of more general queries to determine whether the null answer is *genuine* (it then suggests related queries that have non-null answers), or whether it reflects *erroneous presuppositions* on behalf of the user (it then explains them). The basic heuristic applied here is that a null answer is genuine, if and only if all the queries that are more general have non-null answers.

Tolerance and cooperation are achieved with only minimal interaction, avoiding excessively long dialogues, which tend to be tedious and discouraging. FLEX approaches its users mainly to determine the domain of an ambiguous token, or to select from a list of possible browsing topics. Both tasks are relatively short and simple.

By providing interpretations of ill-formed queries, FLEX also instructs its users in the proper application of the formal language. By providing alternative interpretations, and allowing them to be refined, FLEX reduces the risk of misinterpretations.

FLEX can also be perceived as an interface that supports multiple languages, each with its own level of expressivity: a formal language, a language whose queries are sets of database tokens, and a language whose queries are individual topics. The mechanisms of FLEX would then be viewed, not as procedures for coping with incorrect formal queries, but as interpreters of these languages. Users may then deliberately submit queries in an "inferior" language; their input will flow through the

interpreters of the "superior" languages, until it arrives at the intended interpreter, and generates the expected database request.

The "knowledge base" used by FLEX consists of three auxiliary relations, that are stored along with the database itself: a DICTIONARY that stores the definition of the database, a LEXICON that maps database values to the attributes under which they appear, and a THESAURUS, which stores synonyms of recognized database tokens. The dictionary is used by every FLEX mechanism, the lexicon is used by the synthesizer and the browser, and the thesaurus is used by the corrector. The DICTIONARY relation is relatively small, the information it contains is fairly standard, and it needs to be updated only when the definition of the database is changed. The LEXICON relation is more demanding in terms of size and maintenance (a similar relation was used in BAROQUE). This relation should not be modified by users; the system should update it automatically, to reflect user updates to other relations (this is similar to the way that secondary indexes are handled in some relational systems). The cost of this relation, in terms of the additional space to store this relation and the additional computation for its initialization and its continuous update, is comparable to the cost of a secondary index on every database attribute. If the required storage is prohibitive, it is possible to implement the lexicon only in part, by inverting on selected domains only; tokens of other domains will not be recognized. The THESAURUS relation is different, in that its information cannot be extracted automatically from the database. It may be constructed gradually by the database owner, using a log of unrecognized words maintained by the system. While the thesaurus enhances the operation of FLEX, it is not as essential as the other two relations.

Work on FLEX is continuing. Current goals include extension of the retrieval language to include a fuller set of operators, improved performance of the generalizer, and improved presentation.

5 Conclusion

FLEX represents our current approach regarding user interfaces to databases, that prefers a single interface with universal appeal over a cluster of specific tools. Thus, we would prefer to incorporate tools like BAROQUE and VAGUE into FLEX. Indeed, some of BAROQUE's capabilities are already available in the last mechanism of FLEX. As VAGUE is a formal extension of the relational database model, it is, indeed, "orthogonal" to FLEX, and both could be combined without conceptual difficulties. In other words, FLEX should become an interface to VAGUE.

References

[1] R. G. G. Cattell. An entity-based database interface. In *Proceedings of ACM-SIGMOD International Conference on Management of Data* (Santa Monica, California, May 14–16), pages 144–150, ACM, New York, New York, 1980.

[2] D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade. Sequel 2: a unified approach to data definition, manipulation, and control. *IBM Journal of Research and Development*, 20(6):560–575, November 1976.

[3] E. F. Codd, R. S. Arnold, J-M. Cadiou, C. L. Chang, and N. Roussopoulos. *Rendezvous version 1: an Experimental English Language Query Language System for Casual Users of*

Relational Databases. Technical Report RJ2144, IBM, San Jose, California, February 1978.

[4] *DBASE-III Reference Manual.* Ashton-Tate, Culver City, California, 1984.

[5] D. Fogg. Lessons from a 'living in a database' graphical query interface. In *Proceedings of ACM-SIGMOD International Conference on Management of Data* (Boston, Massachusetts, June 18–21), pages 100–106, ACM, New York, New York, 1984.

[6] L. R. Harris. Natural language front ends. In *The AI Business*, pages 149–161, The MIT Press, Cambridge, Massachusetts, 1984.

[7] G. G. Hendrix, E.D. Sacerdoti, D. Segalowicz, and J. Slocum. Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, 3(2):105–147, June 1978.

[8] C. Herot. Spatial management of data. *ACM Transactions on Database Systems*, 5(4):493–513, December 1980.

[9] 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.

[10] *SunINGRES Manual Set.* Sun Microsystems, Mountain View, California, Release 5.0 (Part Number 800-1644-01), 1987.

[11] R. King. Sembase: a semantic dbms. In *Proceedings of the First International Workshop on Expert Database Systems* (Kiawah Island, South Carolina, October 24–27), pages 151–171, Institute of Information Management, Technology and Policy, University of South Carolina, Columbia, South Carolina, 1984.

[12] N. McDonald and M. Stonebraker. Cupid: a user friendly graphics query language. In *Proceedings of the ACM-Pacific Conference* (San Francisco, California), pages 127–131, ACM, New York, New York, 1975.

[13] A. Motro. BAROQUE: a browser for relational databases. *ACM Transactions on Office Information Systems*, 4(2):164–181, April 1986.

[14] A. Motro. FLEX: a tolerant and cooperative user interface to databases. *IEEE Transactions on Knowledge and Data Engineering*, to appear.

[15] 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.

[16] 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.

[17] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, New York, 1983.

[18] M. Stonebraker and J. Kalash. TIMBER: a sophisticated database browser. In *Proceedings of the Eighth International Conference on Very Large Data Bases* (Mexico City, Mexico, September 8–10), pages 1–10, VLDB Endowment (available from Morgan-Kaufmann, Los Altos, California), 1982.

[19] M. Stonebraker, E. Wong, P. Kreps, and G. Held. The design and implementation of INGRES. *ACM Transactions on Database Systems*, 1(3):189–222, September 1976.

[20] C. J. van Rijsbergen. *Information Retrieval (Second Edition).* Butterworths, London, 1979.

[21] H. K. T. Wong and I. Kuo. GUIDE: a graphical user interface for database exploration. In *Proceedings of the Eighth International Conference on Very Large Data Bases* (Mexico City, Mexico, September 8–10), pages 22–32, VLDB Endowment (available from Morgan-Kaufmann, Los Altos, California, 1982.

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

Computer Society of the IEEE
Technical Committee Membership Application

Complete this form and return to:
Computer Society of the IEEE
1730 Massachusetts Ave. NW
Washington, DC 20036-1903

Instructions:

Please print in ink or type, one character per box. INFORMATION OUTSIDE BOXES WILL NOT BE RECORDED. Street addresses are preferable to P.O. Boxes for mail delivery. International members are requested to make best use of available space for long addresses.

LAST NAME | FIRST NAME | INITIAL | DR/MR/MRS/MS/MISS/PROF ETC

COMPANY UNIVERSITY AGENCY NAME | DEPARTMENT/MAIL STOP/BUILDING/P O BOX/APARTMENT ETC

STREET ADDRESS (OR POST OFFICE BOX) | DATE: MONTH / DAY / YEAR

CHECK ONE
☐ NEW APPLICATION
☐ INFORMATION UPDATE

CITY | STATE | ZIP CODE

COUNTRY | OFFICE PHONE | HOME PHONE (Optional)

ELECTRONIC MAIL NETWORK | ELECTRONIC MAIL ADDRESS (Mailbox)

TELEX NUMBER (Optional) | IEEE MEMBER AFFILIATE NO | I am a Computer Society Member ☐ YES ☐ NO

The Computer Society shares its mailing lists with other organizations which have information of interest to computer professionals. If you do not wish to be included on those lists, please check here: ☐

TC MEMBERS actively participate in TC activities, i.e. review papers, help organize conferences and workshops, participate in standards development, and help with TC operations.

TC CORRESPONDENTS do not participate in TC activities, but they do receive newsletters and other TC communications.

Place an "F" in each box corresponding to a TC of which you would like to be a FULL member. To be a CORRESPONDENT, place a "C" in the corresponding box. To the right of each TC in which you express interest, please rate your interest in each category indicated from 1 to 5, with 5 being high interest and 1 being no interest.

		Conference Organizing Committee	Workshop Organizing Committee	Conference/Workshop Session Chairman	Help with Newsletter	Write Survey/Technical Papers	Review Papers	Standards Development	Educational Activities	Help with TC Operations	Other *(Specify)*:
☐	Computational Medicine (01) 01										
☐	Computer Architecture (02) 02										
☐	Computer Communications (03) 03										
☐	Computer Elements (04) 04										
☐	Computer Graphics (05) 05										
☐	Computer Languages (06) 06										
☐	Computer Packaging (07) 07										
☐	Computers in Education (08) 08										
☐	Computing and the Handicapped (09) 09										
☐	Data Base Engineering (10) 10										
☐	Design Automation (11) 11										
☐	Distributed Processing (12) 12										
☐	Fault-Tolerant Computing (13) 13										
☐	Mass Storage Systems & Technology (14) 14										
☐	Mathematical Foundations of Computing (15) 15										
☐	Microprocessors & Microcomputers (16) 16										
☐	Microprogramming (17) 17										
☐	Multiple-Valued Logic (18) 18										
☐	Oceanic Engineering & Technology (19) 19										
☐	Office Automation (20) 20										
☐	Operating Systems (21) 21										
☐	Optical Processing (22) 22										
☐	Pattern Analysis & Machine Intelligence (23) 23										
☐	Personal Computing (24) 24										
☐	Real Time Systems (25) 25										
☐	Robotics (26) 26										
☐	Security and Privacy (27) 27										
☐	Simulation (28) 28										
☐	Software Engineering (29) 29										
☐	Test Technology (30) 30										
☐	VLSI (31) 31										
☐	Computer and Display Ergonomics (32) 32										
☐	Supercomputing Applications (33) 33										

Thank you for your interest in Computer Society Technical Activities. We look forward to having you with us!

IEEE Computer Society
1730 Massachusetts Avenue, N W
Washington, DC 20036-1903

Henry F. Korth
University of Texas
Taylor 2124 Dept of CS
Austin, TX 78712
USA