Cooperative Database Systems

Amihai Motro
Department of Information and Software Systems Engineering, George Mason University, Fairfax, Virginia 22030-4444

In recent years, there has been considerable interest in techniques that enhance the cooperative behavior of database systems and many different techniques have been developed. These techniques approach the goal of cooperation in diverse ways; the differences may be in the specific task in which they offer cooperation, in the details of the solution, and even in the very interpretation of cooperative behavior. In this article we classify many different techniques into categories of cooperation, and we survey the techniques in some of these categories. Finally, we consider the challenges that remain and offer directions for new research. © 1996 John Wiley & Sons, Inc.

I. INTRODUCTION

In recent years, there has been considerable interest in techniques that enhance the cooperative behavior of database systems and many different techniques have been developed. These techniques approach the goal of cooperation in diverse ways; the differences may be in the specific task in which they offer cooperation, in the details of the solution, and even in the very interpretation of cooperative behavior.

Research in cooperative database systems is strongly indebted to research in artificial intelligence, where of particular importance is the work done by Joshi, Webber, and others at the University of Pennsylvania.[1–5] Database researchers, for the most part, have deemphasized issues related to natural language understanding, and have concentrated on developing techniques for well-known data models, such as those based on relations or on logic.

In this article we classify many different cooperative database techniques into several categories, and we survey the techniques in some of these categories. Thus, the scope of the classification and survey is confined to selected areas. Cooperative database interfaces are also covered in Refs. 6 and 7.

Our classification is described in Section II. Our formalism distinguishes between *retrieval goals*, which are the data requirements that users have when they approach the system, and *database queries*, which are the formal statements in which goals are presented to the system. Our classification establishes two broad categories of cooperation.

The first category includes techniques for assisting users in the formulation

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 11, 717–731 (1996)
 CCC 0884-8173/96/100717-15

of goals, and in the conversion of goals to good queries. Roughly speaking, the first category is concerned with situations in which a user is essentially saying "I don't know what I want" or "I know what I want, but I don't know how to say it (in a language understood by this database system)".

Techniques in the second category assume that the database system has been presented with a good query that implements correctly the goal of a user; these techniques address the *adequacy* of this goal or query. Roughly speaking, the second category is concerned with situations where a user should be told "you shouldn't be asking *this*" or "what you probably want is *that*".

Our survey is presented in Section III, and it focuses on the second category. All the techniques described begin with queries that are apparently correct, but are found to be unnecessarily complex, or infeasible, or incomplete.

A problem that still remains to be addressed are proper queries that *do not* correspond to the goals of the submitting users. This specific trap is a frequent source of frustration to database users, but has not received adequate attention, probably due to its inherent difficulty. Section IV concludes this article with a discussion of this problem and possible cooperative techniques that could alleviate it.

II. CLASSIFICATION

Our definition of a cooperative database system is very broad. A database system is *cooperative* if it imitates some cooperative aspect of human behavior. Cooperation is one aspect of *intelligence*. Other aspects include the ability to reason, the ability to acquire and apply knowledge, or the ability to understand natural language.[8]

Our classification of cooperative database techniques is based on the stage of the retrieval process where cooperation is introduced. A typical database interaction is an iterative process, as follows:

repeat
  (Re)formulate *goal*
  Convert *goal* to *query*
  Submit *query*
  Examine *answer*
until *goal* is satisfied

In this process, a user approaches the database system with a specific data requirement, termed *goal*. The user then expresses this goal in a formal *query*, which is submitted to the database system. Upon examining its result, the user determines whether the goal has been satisfied or not. In the former case, the process terminates; in the latter case, it is repeated: the user may now reformulate the goal, or he may convert the same goal to a different query. To allow users to abandon the querying process, a special goal *null* may be assumed, which is always converted to an empty query and is vacuously satisfied.

This typical interaction suggests a simple classification of the kinds of

cooperation techniques that database systems may offer into two broad categories:

(1) Techniques for assisting users in the formulation of goals, and in the conversion of goals to good queries.
(2) Techniques that analyze apparently good queries for possible deficiencies in the goals that they express.

This broad classification is refined and discussed in more detail in the next two subsections.

A. Generating Proper Queries

Cooperative techniques in the first category concentrate on the stage in the retrieval process that ends in a query that is acceptable to the database system. This category may be divided into three classes:

(1) Formulation of goals.
(2) Conversion of goals to queries.
(3) Assurance that queries correctly express goals.

Techniques in the first class address situations in which a database user is essentially saying "I don't know what I want." That is, the user does not have goal, or the goal is not specific enough. The most popular technique for cooperating with such users is *browsing*. Browsers are exploratory interfaces that allow users to search databases without a specific goal.[9-11] A browsing session may provide a user with a specific goal (it could even satisfy the goal), or it may convince the user that the database does not contain any data of interest. Presumably, users without goals could also benefit from a system capable of conducting *clarification dialogues*. But, apparently, this technique, which has been applied in natural language understanding systems, has not become popular in query systems.

Techniques in the second class address situations in which a user is essentially saying "I know what I want, but I don't know how to say it." That is, the user has a specific goal, but is incapable of expressing it in the formal language that is supported by the database system. A technique that cooperates with such users is *interactive query construction*, which constructs the target query incrementally, by guiding the user through a series of menu-driven dialogues. A notable example is RABBIT.[12] This system applies a paradigm of repetitive reformulation of an initial goal. At each iteration the user is presented with the answer to the current query. Having observed the answer, the user can then refine the query in one of several possible ways. Another technique[13] monitors the activity of users in browsing sessions, in an attempt to detect repetitive actions, which could be captured in concise queries that retrieve the same information.

Indeed, any method that reduces the formal burden on its users is cooperative in the translation of goals to queries. These include flexible and tolerant

interfaces, such as Flex,[14] Query-by-example interfaces,[15,16] as well as graphical interfaces and natural language and NL-like interfaces.

The third class considers the possibility that a query submitted to the database system was acceptable to the system both syntactically and semantically, yet did not implement correctly the intent (goal) of the user. In such situations the user would be alerted to the problem only if he is familiar with some aspects of the answer, and recognizes that the answer that was computed does not satisfy all of these aspects; otherwise, such errors go unnoticed. The author is not aware of any system that assists users in detecting such errors. Further discussion of this problem is deferred to Section IV.

B. Detecting Deficient Goals

Now assuming that users know what they want, and how to correctly state it in good queries, the second category includes techniques that analyze submitted queries to detect deficiencies in the goals that they express.* This category may be divided into four classes:

(1) Detection and simplification of goals/queries that are unnecessarily *complex*.
(2) Detection and modification of goals/queries that are *unsatisfiable*.
(3) Detection and modification of goals/queries that are *incomplete*.
(4) Annotation and explanation of answers.

At times, queries are more complex than they need to be. Usually, such queries incorporate information which is *redundant* (i.e., satisfied tautologically). This redundancy is often the result of *misconceptions* on behalf of the user that submitted the query. By detecting and reporting such redundancies, the database system is informing the user of possible misconceptions. For example, assuming a database with a constraint that requires all professors to have a Ph.D., a user who inquires about "professors with a Ph.D." is under the misconception that possibly some professors do not have a Ph.D. By detecting this redundancy and informing the user of a possible misconception, the database system is demonstrating cooperative behavior. Moreover, by ignoring this redundancy, and answering the query as presented, the database system might contribute to the reinforcement of such misconceptions.

There are two possible interpretations for query *unsatisfiability*. First, when the answer to the query is *empty*; i.e., the current instance of the database contains no data that satisfy the specification expressed in the query (the query predicate). Second, when the query specification *conflicts* with the stored knowledge (the integrity constraints); i.e., the answer to this query would be empty in any database instance. It is quite obvious that in the latter case the user is not aware of some important aspect of the environment being modeled by the database. But, as we shall see, in the former case, too, there is a

*At this point a query is assumed to correspond correctly to a goal, so one could talk about either the goal or the query as deficient.

possibility of erroneous presuppositions of the user. Techniques in the second class address either of these interpretations of unsatisfiability.

A query is considered *incomplete* when it is highly possible that its "literal" answer would not completely satisfy the user who submitted it, and would therefore trigger follow-up queries. An example of an incomplete query is a query that generates an empty answer; in many cases such a query would be followed by other queries that will "weaken" the specification, with the intention of matching some data. Another example is a query that requests only part of a "cluster" of information that consists a meaningful whole; quite often, such a query will be followed by additional queries that will ask for the "rest." Techniques in the third class attempt to detect such queries and enlarge them automatically.

Like the techniques in the third class, the techniques in the fourth class also provide additional information to users. But this additional information is not data, but information *about* the data. For example, in response to a query to retrieve the employees who earn more than \$50 000, the database system could also inform the user that these employees are the senior managers and engineers, except for Betty (a junior manager who earns more than this amount) and Tom (an engineer who earns less than this amount). Such answers have become to be known as *intensional answers*. Similarly, the database system could annotate the answer with respect to other properties; for example, to qualify portions of the data with respect to their established accuracy.

It is obvious that in each and every case, the objective of the database system is to imitate the cooperative behavior of humans. The next section expands on techniques in these four classes (i.e., the second category): techniques that operate on queries that are apparently correct (by standard database systems), yet are unnecessarily complex, unsatisfiable, or incomplete.

III. SURVEY

The survey in this section is divided into four parts, in correspondence with the four classes of techniques that were defined in Section II-B.

A. Complex Goals

The initial motivation for simplifying unnecessarily complex queries was *optimization*: the opportunity to reduce the work of processing the query in the database. As an example, consider a personnel database, and assume that it enforces the constraint that the starting salary of engineers is \$50 000. Consider now a query on the female engineers who earn more than \$45 000. Naively, this query requires that every employee record be checked for three qualifications: *Sex* = "female," *Profession* = "engineer," and *Salary* > \$45 000. However, when the constraint is considered, the last qualification becomes redundant, suggesting an improvement to the naive query processing strategy.

Most research efforts here have been in the framework of the logic data

model, and the typical simplification detects redundant conjuncts: a constraint $\alpha \rightarrow \beta$ transforms the query $\alpha \wedge \beta$ to the simpler query α.

This very process can also be motivated as a cooperative procedure.[17-19] A query with redundancies indicates that the inquirer may have misconceptions regarding the semantics of the environment modeled by the database. A cooperative answer to the previous query would then be a list of female engineers, accompanied by a statement to the effect that *all* engineers earn more than \$50 000.

B. Unsatisfiable Goals

Strictly speaking, a query is *unsatisfiable* when it would return an empty answer from *every* instance of the database. Obviously, such queries reflect deficient retrieval goals. When a query evaluates to an empty answer, it is *suspect* of being unsatisfiable. We discuss two techniques for determining whether such a query is indeed unsatisfiable.

1. Empty Answers

From the various situations in which cooperative behavior is appropriate, probably the one situation that has received the most attention is when users are provided with empty answers in response to their queries.[5,17,20-23]

Frequently, queries that have empty answers indicate *erroneous presuppositions* of the inquirers. In human conversation, they are often answered with an explanation that points out the error. Standard database systems do not detect these problems, and process these queries in the usual manner. Consider this query about the students who failed the course CS-345 in the summer semester of 1995. Most probably, the inquirer assumes that CS-345 was offered in the summer of 1995, and would interpret an empty answer, as if all the students passed this course successfully. In reality, it could be that CS-345 was never offered in the summer of 1995. Moreover, that the system did not complain about the query might be interpreted by the inquirer as an *affirmation* that indeed this particular course was offered in that particular period. A more formal presentation of these observations, based on Ref. 24 follows.

Queries often reveal the *presuppositions* of the inquirers. Specifically, a person who requests data believes that such data may exist (otherwise, why ask?). This belief becomes even *stronger* when the query is *broadened*; i.e., its predicate is replaced by a subsuming predicate. Conseqeuntly, it is reasonable to assume that users expect that answers to their queries could possibly be empty, but they do not expect answers to even broader queries to be empty.

In other words, a query that has an empty answer, but all its broader queries have nonempty answers, is within the expectations of the user; we call it a *genuine* empty answer. However, a broader query that has an empty answer indicates an *erroneous presupposition* of the user. The presupposition being, simply, that data that satisfy the broader query must exist.

In the above example, three broader queries should be tried:

(1) The students who received a grade (not necessarily "F") in CS-345 in summer 1995.
(2) The students who failed CS-345 (not necessarily in summer 1995).
(3) The students who failed a course (not necessarily CS-345) in summer 1995.

If all three answers are nonempty, we conclude that the original empty answer is genuine (i.e., the original query is, in general, satisfiable). Any empty answer, however, indicates an erroneous presupposition. The corresponding erroneous presuppositions are:

(1) CS-345 was not offered in summer 1995.
(2) The grade "F" was never used in CS-345.
(3) The grade "F" was never used in summer 1995.

Assume now that the first of these answers is found to be empty. The same procedure could now be applied to this answer, as well. Its broader queries are: (1) Students who received a grade in CS-345, and (2) Students who received a grade in summer 1995. The corresponding erroneous presuppositions are: (1) CS-345 has never been offered (possibly there is no such course designation), and (2) No courses were held in summer 1995 (possibly there is no such semester designation).

Obviously, each of these new erroneous presuppositions is more *significant* than the earlier ones. Which leads us to define a *maximal erroneous presupposition (MEP)*, as one that corresponds to a query that has an empty answer but all its broader queries have nonempty answers, and to conclude that only MEP are significant. The following procedure summarizes this technique:

(1) When a query returns an empty answer, search for all MEP.
(2) If none is found,† then the empty answer is genuine.
(3) Otherwise, report every MEP to the user.

The specific techniques for gradual broadening of queries depends on the data model in use. An important aspect of every implementation is how to manage the additional queries efficiently.

2. *Unsatisfiable Queries*

Our interpretation of an empty answer was that a presupposition (that there are data that match the query) was refuted. In effect, the database system is saying that *there are no data* to support this presupposition. In the presence of additional database knowledge (e.g., integrity constraints), it may be possible to conclude that *there can be no data* to support a given presupposition. Such rejection is much stronger and more informative.[24]

As an example, assume a personnel database that enforces two constraints: (1) The starting salary of engineers is at least $50 000, and (2) senior managers

†The query itself is the only MEP.

earn more than engineers. Consider now a query on the senior managers that earn less than $40 000. A strict answer would be that there are none. A cooperative answer would be that *there can be none*, because all senior managers must earn at least $50 000.

The discovery of unsatisfiable queries is often done together with the discovery of redundancies in queries, a subject that was discussed in Section III-A. As mentioned there, these techniques are usually within the framework of the logic database model[18,19,25] and are also motivated by query optimization: to avoid costly processing of a query in the actual database, if it can be established a priori that the query is unsatisfiable.

C. Incomplete Goals

We consider a query *incomplete* when it is highly possible that its "literal" answer would not completely satisfy the user who submitted it, and would therefore trigger follow-up queries. In human interaction, "literal" answers could be annoying at times, and a cooperative human is expected to anticipate follow-up queries voluntarily. We discuss two techniques for anticipating follow-up queries.

1. Partial Answers

In Section III-B.1 we discussed queries that are unsatisfiable because their answers are empty, and we distinguished between empty answers that are *genuine* and empty answers that reflect *erroneous presuppositions*. In both cases broader queries had to be evaluated. The empty answer was declared genuine, only if all broader queries had nonempty answers, whereas a broader query whose answer was also empty indicated an erroneous presupposition.

While a genuine empty answer confirms that the query was "reasonable," very frequently, the user who submitted the query would follow up with one or more queries that relax some aspect of the original query specification, with the purpose of retrieving data that fulfill "most" of the original specification.

The broader queries that were already computed in response to the original query provide precisely this information. Thus, the techniques discussed in Section III-B1 have a dual purpose: when an empty answer is declared genuine because all broader queries evaluated to nonempty answers, these nonempty answers are offered in partial fulfillment of the original query. The cooperative trait here can be summarized as "never say none."

As an example, consider a query on the female employees with 7 years experience in management, and assume that the strict answer is "none". A cooperative response would offer the user options such as these:

(1) Female employees with 7 years experience.
(2) Female employees with experience in management.
(3) Employees with 7 years experience in management.
(4) Female employees with 6 years experience in management.
(5) Female employees with 7 years experience in human resources.

D. Augmented Answers

The information stored in databases can often be "clustered" into topics, where each topic constitutes a meaningful whole. Quite frequently, a query that requests only part of such a topic would be followed by additional queries that will ask for the "rest." As an example, consider a query about morning flights from London to Paris. A strict answer would simply list these flights, giving information such as airlines, flight numbers, and departure times. A cooperative answer would also provide *related* information, such as arrival times, prices, and availability. It could also include information on transportation from the airport to the destination city and nearby hotels. It could list early afternoon flights as well, and even suggest comparable alternatives, such as train departures.

Such a technique is described in Ref. 26, where the data model is a logic formalization of an Entity–Relationship model with aggregation and generalization relationships. User queries are "completed" in three ways: (1) By relaxing the query specification—This is similar to the techniques described in Section III-B.1. (2) By "closuring" the requested attributes with additional attributes—The additional attributes originate from entities that participate with the query subject in an aggregation structure; e.g., the hotel attributes. (3) By adding tuples that represent other "objects" altogether—The additional objects originate from entities that participate with the query subject in a generalization structure; e.g., the train tuples. Roughly speaking, this technique expands strict answers by adding both new rows and new columns to the answer tables.

E. Explanation and Annotation

Our survey focuses on cooperative techniques that operate on queries that are acceptable to standard database systems. But whereas the preceding three subsections dealt with queries that are deficient in some way (unnecessarily complex, unsatisfiable, or incomplete), the techniques we are about to discuss here are intended to provide users with additional insights into the answers that are computed for them by the database system. Our discussion is divided into two topics: intensional answers and meta-answers.

1. Intensional Answers

An intensional answer is a complement of the conventional answer, comprising either a terse description of the answer, or various useful statements that concern the answer.

The term *intensional answer* comes from a distinction often made between the *intension* and the *extension* of a database. The intension of a database is the set of definitions of the data structures for the particular database (also called *schema*). The extension of the database is the set of database values that populate these data structures. The term *intension* also refers to various other elements of the definition of the database, such as statements that express

relationships that must be satisfied by every extension (integrity constraints), or statements that define new data structures and their extensions in terms of the basic structures (views or inference rules). Specifically, the intension of a relational database includes the definitions of the base relations, the definitions of views, and the integrity constraints. The intension of a logic-based database includes the definitions of the base predicates, the inference rules and the integrity constraints. In semantically rich or object-oriented models the intension includes the definition of the various classes and their associated hierarchies.

A database query is an intensional statement (for example, in the relational model it is a view). Its answer is the extension of this intensional statement. While the intensional information in the database is utilized in the processing of each query, answers are composed entirely of extensional information. Consequently, the query itself is often the only intensional characterization of the retrieved set of values, of which the user is aware. Still, the intensional information in the database may include additional characterizations of the extensional answer. If this intensional information is derived and retrieved, database answers would gain additional meaning.

Several researchers have recently addressed themselves to the issue of intensional answers, including.[22,27–33] While all share a common goal, to respond to queries more abstractly by using the intension of the database, the individual approaches are often very dissimilar: they adopt different frameworks (i.e., the data model and its intensional information), they define their intensional answers differently, and they develop their own specific methods for computing them. Space does not permit us to describe here individual methods. A comparative survey of intensional answering may be found in Ref. 34.

As a broad example, consider a query on the employees who earn more than \$50 000. Using different data models and different techniques, possible intensional answers include:‡

(1) The senior managers and engineers, except for Betty (a junior manager who earns more than this amount) and Tom (an engineer who earns less than this amount).
(2) The senior managers and their supervisors.
(3) The answer given includes all the engineers.
(4) All the employees that are included in the answer given are senior.

2. *Meta-Answers*

When responding to queries, humans often volunteer additional information *about* their answers. Among other things, they may *qualify* the answer as to its reliability, and they may *characterize* it abstractly. As an example, consider this inquiry about bookstores in Washington.[35] After listing several bookstores, the person answering the question might add coments such as:

‡These answers are intended to show the *kinds* of statements that different methods will extract; obviously, they are not mutually consistent.

(1) "This list is perfect, trust me."
(2) "There might be some other bookstores of which I am not aware."
(3) "I am confident about all these bookstores, except the last one, which might have been converted into a video boutique."
(4) "All these bookstores are located south of M Street."
(5) "These bookstores include all those that are located in Georgetown."

The first statement grants assurance that the answer is both sound (all information provided is accurate) and complete (there are no other bookstores in Washington). The second statement states that the answer may be incomplete, while the third statement asserts the soundness of all but the last item. The fourth and the fifth statements provide useful characterizations of the answer. Clearly, such characterizations and "quality assurances" are often very valuable to the recipient of the information.§

The various statements about the answer are *properties* of the answer. A relational database system that similarly annotates its answers with their properties is Panorama.[35] The system assumes that various assertions about properties of the data have been stored in the database (meta-information). These assertions are then used to infer properties of each answer provided by the system (meta-answers). Meta-answers are offered to users along with each answer issued, and help them to assess the value and meaning of the information that they receive.

The claimed advantages of Panorama include: (1) It is *extensible* in that it allows users to determine the kinds of properties that the system will maintain and manipulate. (2) It has a built-in mechanism for determining the *relevance* of computed meta-information. (3) It is *efficient*: the number of operations required for meta-processing a given query requires can be expressed as a polynomial in the size of the meta-database.

IV. CONCLUSION

In this article we attempted to provide a systematic overview of the various cooperative database techniques that exist. We conclude our article with discussions of problems that remain to be solved. Section IV-A discusses a specific problem already mentioned in preceding discussions. Section IV-B provides a more general assessment of the future of this area.

A. Assuring That Queries Correctly Correspond to Goals

Successful interaction with a database system requires users to have adequate retrieval goals and the ability to express these goals in a formal query language. The techniques described in this article assist users in meeting these requirements. Yet none of these techniques can help a user who has an adequate

§The last two statements are *intensional statements* similar to statements 3 and 4 in the previous example. Indeed, the scope of such cooperative answers includes intensional answers.

goal, but implements it *incorrectly* in an adequate query; i.e., the query corresponds to an adequate but *different* goal. Database users would be alerted to such mistakes only if they had a priori knowledge of the form and contents of the answer to their query.

This problem is an instance of the general problem of the so-called "logical" programming errors, where the programmer writes the wrong program. Detecting such errors is a difficult challenge, and the prevailing approaches are based on testing and proving program correctness. In the case of database queries, however, the problem should be considerably simpler, because database queries operate in a restricted context of specific databases. We offer here two possible techniques for alleviating this problem.

One way to address this problem is to add various *assertions* to the query about *necessary conditions* that must be satisfied by its answer. These assertions are a formal expression of the a priori knowledge mentioned above. Given these assertions, the system can determine automatically when an answer does not meet the assertions and inform the user that either the query or the assertions were mistated. It must be emphasized that because it relies on necessary conditions, this process cannot guarantee that a query that generates an answer that satisfies the assertions is indeed correct. However, with extensive assertions, many query errors can be detected. Examples of assertions include: (1) specification of the range of answer cardinality (e.g., if the answer is empty then query must be wrong); (2) specification of data that must show up in the answer; and (3) specification of data that must not show up in the answer. As a simple example, a query on the employees that earn more than $50 000 may be accompanied by assertions that "the answer should include at least 50 such employees, and one of them should be Betty." If the answer does not satisfy these assertions then the query (or the assertions) was mistated.

Another indication that a query might be wrong is that it has never been asked before. An altogether new query—one which is different in substance from all previously asked queries (not merely in constants or abstraction level)—is suspect of errors; especially if the "history" of previously asked queries has accumulated over a considerable amount of time.

B. Future Directions

The area of cooperative database systems has attracted much attention, and many clever techniques have been proposed. Yet challenges still remain, and we describe here four challenges that we consider to be of primary importance.

Cooperative techniques are often isolated strategies for performing specific tasks. They may assume different data models and use different formalisms. The foremost challenge is that of *integration*: to incorporate as many techniques as possible into a single, elegant paradigm of cooperation.

The proof of any cooperative techniques is in its successful implementation and application. Many of the cooperative techniques we surveyed have not progressed beyond modest proof-of-concept demonstrations. A critical chal-

lenge is to construct *viable prototypes* that will convince both users and software developers of their usefulness and practicality.

Cooperation involves voluntary extension of help. As in other cases of voluntary assistance, it is important to assure that the assistance is *valuable and relevant* to the recipient (or else it will be considered a nuisance, and ignored altogether). Many cooperative techniques lack built-in controls for filtering their output to assure that it benefits the specific user with which they communicate. Quite often, the need for such controls would not surface in small examples, but only in actual field tests.

The information stored in databases varies with respect to factors such as soundness, completeness, certainty and currency. Also, the evolution of systems for integrating multiple databases[36] raises the inherent problem of inconsistency among answers obtained from multiple sources.[37] The automatic determination of answer *goodness* and the automatic *resolution of inconsistencies* are examples of new areas where database systems can extend assistance to their users.

This work was supported in part by ARPA grant, administered by the Office of Naval Research under Grant No. N0014-92-J-4038.

References

1. A.K. Joshi, "Mutual beliefs in question answering systems," in *Mutual Belief*, N. Smith, Ed., Academic Press, New York, 1982.
2. E. Mays, S. Lanka, A.K. Joshi, and B.L. Webber, "Natural language interaction with dynamic knowledge bases: Monitoring as response," Vancouver, Canada, August 24–28, 1981, in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 61–63.
3. B.L. Webber, "Questions, answers and responses: Interacting with knowledge-base systems," in *On Knowledge Base Management Systems*, Springer-Verlag, Berlin, Germany, 1986, pp. 353–402.
4. B.L. Webber and E. Mays, "Varieties of user misconceptions: Detection and correction," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, August 8–12, 1983, pp. 650–652.
5. S.J. Kaplan, "Cooperative responses from a portable natural language query system," *Artif. Intell.* **19**(2), 165–187 (October 1982).
6. L. Bolc and M. Jarke, Eds, *Cooperative Interfaces to Information Systems*, Topics in Information Systems, Springer-Verlag, Berlin, Germany, 1986.
7. T. Gaasterland, P. Godfrey, and J. Minker, "An overview of cooperative answering," *J. Intell. Inform. Syst.* **1**(2), 123–158 (October 1992).
8. P.H. Winston, *Artificial Intelligence*, Addison-Wesley, Reading, MA, 1977.
9. A. Motro, "BAROQUE: A browser for relational databases," *ACM Trans. Office Inform. Syst.* **4**(2), 164–181 (April 1986).
10. T.R. Rogers and R.G.G. Catell, *Object-Oriented Database User Interfaces*, Technical report, Information Management Group, Sun Microsystems, 1987.
11. A. D'Atri, A. Motro, and L. Tarantino, *ViewFinder: An Object Browser*. Technical Report ISSE-TR-95-103, Department of Information and Software Systems Engineering, George Mason University, March 1995.
12. M.D. Williams, "What makes RABBIT run?" *Int. J. Man-Mach. Studies*, **21**(4), 333–352 (October 1984).

13. A.D'Atri and L. Tarantino, "From browsing to querying." *Data Eng.*, **12**(2), 46–53 (June 1989).
14. A. Motro, "FLEX: A tolerant and cooperative user interface to databases," *IEEE Trans. Knowl. Data Eng.*, **2**(2), 231–246 (June 1990).
15. M. Zloof, "Query-by-Example: A database language," *IBM Syst. J.*, **16**(4), 324–343 (December 1977).
16. M. Zlof, "QBE/OBE: A language for office and business automation," *Computer*, **14**(5), 13–22 (May 1981).
17. J.M. Janas, "Towards more informative user interfaces," in *Proceedings of the Fifth International Conference on Very Large Data Bases*, Rio de Janeiro, Brazil, October 3–5, 1979, pp. 17–23.
18. U.S. Chakravarthy, J. Grant, and J. Minker, "Logic-based approach to semantic query optimization," *ACM Trans. Database Syst.*, **15**(2), 162–207 (June 1990).
19. A. Pirotte, D. Roelants, and E. Zimanyi, "Controlled generation of intensional answers," *IEEE Trans. Knowl. Data Eng.*, **3**(2), 221–236 (June 1991).
20. F. Corella, S.J. Kaplan, G. Wiederhold, and L. Yesil, "Cooperative responses to boolean queries," in *Proceedings of the IEEE Computer Society First International Conference on Data Engineering*, Los Angeles, CA, April 24–27, 1984, pp. 77–85.
21. A. Motro, "Query generalization: A technique for handling query failure," in *Proceedings of the First International Workshop on Expert Database Systems*, Kiawah Island, SC, October 24–27, 1984, pp. 314–325.
22. W.W. Chu, R.-C. Lee, and Q. Chen, "Using type inference and induced rules to provide intensional answers," in *Proceedings of the IEEE Computer Society Seventh International Conference on Data Engineering*, Kobe, Japan, April 8–12, 1991, pp. 396–403.
23. T. Gaasterland, P. Godfrey, and J. Minker, "Relaxation as a platform for cooperative answering," *J. Intell. Inform. Syst.*, **1**(3/4), 293–321 (December 1992).
24. A. Motro, "SEAVE: A mechanism for verifying user presuppositions in query systems," *ACM Trans. Office Inform. Syst.*, **4**(4), 312–330 (October 1986).
25. A. Gal, "*Cooperative responses in deductive databases*," Ph.D. Thesis, Department of Computer Science, University of Maryland, 1988.
26. F. Cuppens and R. Demolombe, "Cooperative answering: A methodology to provide intelligent access to databases," in *Proceedings of the Second International Conference on Expert Database Systems*, Tysons Corner, VA, April 25–27, 1988, pp. 333–353.
27. C.D. Shum and R. Muntz, "Implicit representation for extensional answers," in *Proceedings of the Second International Conference on Expert Database Systems*, Tysons Corner, VA, April 25–27, 1988, pp. 257–273.
28. C.D. Shum and R. Muntz, "An information-theoretic study on aggregate responses," in *Proceedings of the Fourteenth International Conference on Very Large Data Bases*, Los Angeles, CA, August 25–28, 1988, pp. 479–490.
29. A. Motro, "Using integrity constraints to provide intensional responses to relational queries," in *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, Amsterdam, The Netherlands, August 22–25, 1989, pp. 237–246.
30. L. Cholvy and R. Demolombe, "Querying a rule base," in *Proceedings of the First International Conference on Expert Database Systems*, Charleston, SC, April 1–4, 1986, pp. 365–371.
31. A. Pirotte and D. Roelants, "Constraints for improving the generation of intensional answers in a deductive database," in *Proceedings of the IEEE Computer Society Fifth International Conference on Data Engineering*, Los Angeles, CA, February 6–10, 1989, pp. 652–659.
32. T. Andreasen, "Semantic query answering," in *Proceedings of COMAD 90*, New Delhi, India, 1990.
33. T. Imielinski, "Intelligent query answering in rule based systems," *J. Logic Programming*, **4**(3), 229–257 (September 1987).

34. A. Motro, "Intensional answers to database queries," *IEEE Trans. Knowl. Data Eng.*, **6**(3), 444–454 (June 1994).
35. A. Motro, "Panorama: A database system that annotates its answers to queries with their properties," *J. Intell. Inform. Syst.*, **7**(1), 1996.
36. A.R. Hurson, M.W. Bright, and S.H. Pakzad, Eds, *Multidatabase Systems: An Advanced Solution for Global Information Sharing*, IEEE Computer Society Press, Los Alamitos, CA, 1994.
37. A. Motro, *A Formal Framework for Integrating Inconsistent Answers From Multiple Information Sources*. Technical Report ISSE-TR-93-106, Department of Information and Software Systems Engineering, George Mason University, October 1993.