Responding with Knowledge

Amihai Motro
George Mason University

ABSTRACT: By using their knowledge bases, expert database systems can respond to queries that involve high level, complex concepts. However, the answers provided by these systems are composed entirely of low level facts. Recently, researchers have been developing query answering systems that include knowledge in their answers. While the specific goal and approach of each of these individual efforts may be different, all share one common purpose: to extend the functionality of database systems and improve the usefulness of their answers. This paper reviews the state of the art in knowledge-rich responses, and the open problems that still need to be addressed.

1. INTRODUCTION

In a conventional database system, a *database* is a large collection of facts, a *query* is a predicate over the collection of facts, and an *answer* is the subset of the database that satisfies the query predicate.

Since the early 1980s, considerable research activity has been concerned with issues related to the enrichment of these conventional databases with *knowledge*. Roughly speaking, knowledge consists of statements of relationships that hold over *multitudes* of database facts.[1] Such knowledge may have different forms and may be utilized in several ways. Still, the availability of knowledge has not altered the basic principle of conventional query answering systems: users define sets of facts with suitable predicates, and the database system materializes these sets.

Recently, there have been numerous research efforts that go beyond this conventional approach, allowing database answers to include knowledge statements as well. While the specific goal and approach of each of these individual efforts may be different, all share one common purpose: to extend the functionality of database systems and improve the usefulness of their answers.

These efforts can be classified into two broad categories, according to the query mechanism they define. Works in the first category maintain the definition of a query as a predicate over the database of facts (i.e., a *data* query), but modify the definition of an answer to include, in addition to the facts that satisfy the query predicate, various knowledge statements that *concern* these facts. Works in the second category extend the definition of queries to permit inquiries about the knowledge itself; such *knowledge* queries are answered with suitable knowledge statements.

International Journal of Expert Systems Pp. 121–138, Volume 6, Number 1, ISSN: 0894-9077

Altogether, one observes three types of query answering mechanisms in knowledge-rich databases: (1) the conventional mechanism that receives data queries and answers them with facts, (2) a mechanism that receives data queries and answers them with combinations of facts and knowledge statements, and (3) a mechanism that receives knowledge queries and answers them with knowledge statements.

In this paper we discuss the various query answering mechanisms that include knowledge in the answers they generate. We begin by examining in more detail the role of knowledge in database systems.

1.1 The Role of Knowledge in Database Systems

The term database knowledge is hard to define precisely. In an attempt to provide a definition we observe that the information in databases is of various kinds. The bulk of the database consists of low level *facts* (also called *data*). The *schema* of the database is information describing the structure of the data; for example, the definition of various database classes and their subclass relationships. Additional conditions which the facts must obey may be expressed with *integrity constraints*. Finally, definitions of derived classes may be expressed with *inference rules*. Clearly, the facts, the schema, the constraints and the rules all capture information about the environment that is modeled by the database. However, while facts capture information which pertains to individual real world objects, the schema, the constraints and the rules are capable of capturing information that pertains to multitudes of real world objects. Consequently, we use the term database *knowledge* to refer collectively to all the information stored in a database except the facts. Other terms often used to describe the fact and knowledge components of a database are, respectively, *extension* and *intension*.[2]

This broad definition is partly justified by the observation that, depending on the details of the data model, the same information can often be represented by different forms of knowledge. For example, assume a database on students and their achievements, which is represented in a record-based model by the record type STUDENT and the attributes SNAME and GPA. Now assume that students whose grade-point average is at least 3.7 are defined to be honor students. This particular information may be specified with either (1) a new attribute of STUDENT, called HONOR, that has an associated formula that defines its values; (2) a new subtype of student, called HONOR_STUDENT, that has an associated formula that determines its instances; (3) a new type HONOR_STUDENT and an integrity constraint that guarantees its instances to be instances of STUDENT with the required GPA value; or (4) a new type HONOR_STUDENT and an inference rule that defines it from the actual type STUDENT and the GPA requirement.

Whatever form of knowledge is used, the representation of knowledge in a database may be regarded as the *transfer* of expertise from humans to the database. In the previous example, with a "knowledge-poor" database, to prepare a list of honor students, the human must know the definition of honor student, and issue a query to list all the students whose grade-point average is at least 3.7. With a "knowledge-rich" database, the definition of honor student is already present in the database, and the human can issue a simpler query to list the honor students.

As this example demonstrates, the principal role of knowledge in knowledge-rich databases is to allow users to formulate queries in terms that are closer to their perceptions. This approach is considered superior to the traditional approach, where the

database system delivers "raw" data, which is then post-processed outside the database (applying the same knowledge) to provide the requested answer. We note that in both approaches, the final answer consists entirely of data. Knowledge was applied to *generate* the answer, but itself is not *part* of the answer.

In addition to this role, knowledge is also applied in other important roles, such as monitoring the integrity of the database (Ullman, 1988) or optimizing the processing of queries (Chakravarthy, Fishman, & Minker, 1986). Still, in each case the answers provided by the system are entirely conventional: *data* that satisfy the qualification specified in the query. The foci of this paper are query answering systems that include *knowledge* in their answers.

2. RESPONDING WITH KNOWLEDGE TO DATA QUERIES

As defined earlier, a data query is a predicate over the database of facts, and is answered by the database subset that satisfies this predicate. The process of computing the answer to a query is often referred to as *extending* the definition of the predicate with facts, or computing the *extension* of the predicate. The answer itself is sometimes called an *extensional* answer.

Answers that are sets of facts (especially when the sets are large) have limited capacity of explanation and illumination. In human conversation, simple requests for facts are sometimes better answered with more abstract statements. For example, when the answer to the question "who are the employees who earn over $50,000?" is a long list of names, an attractive alternative might be "the engineers." Of course, such answers can only be provided when relevant knowledge and reasoning capabilities are available.

With these observations in mind, several researchers have attempted to harness the new knowledge bases and their inference engines to provide similar advantages to users of database systems. Specifically, they designed mechanisms that respond to data queries with knowledge statements. The knowledge statements are usually in the form of predicates, of a kind similar to those used in queries. An answer that includes knowledge statements is sometimes called an *intensional* answer.

In this section we review five different methods for intensional answering. We begin by describing a classification system that will assist us in comparing the various methods.

2.1 Classification

There are many ways to include knowledge in answers, and we offer three key criteria by which such answers may be classified: *perfectness, purity,* and *invariance.* These often conflicting criteria are discussed below.

An intensional answer can be extended over the same collection of facts, and can be related to the extensional answer in one of several ways. If the extension of the intensional answer is equal to the extensional answer, then the intensional answer is *perfect.* In the above example, the intensional answer "engineer" is perfect, if the set of engineers is identical to the set of employees who earn over $50,000. If the extension of the intensional answer is contained in the extensional answer, then the intensional answer is a constituent of the extensional answer. In the example, the intensional answer "engineer" is a *constituent* of the extensional answer, if every engineer earns over $50,000. If

the extension of the intensional answer contains the extensional answer, then the intensional answer is a constraint on the extensional answer. In the example, the intensional answer "engineer" is a constraint on the extensional answer, if every employee who earns over $50,000 is an engineer.

The predicates in intensional answers may already be *defined* in the knowledge base (as may be the case with the predicate "engineer"), but could also be new predicates that are *derived* from it. Consider again the query "who are the employees who earn over $50,000?" and the predicate "if female then designer" (or equivalently, "males or designers"). This predicate might be a constraint on the answer, in which case its meaning is that the female employees who earn over $50,000 are all designers. Or it might be a constituent of the answer, in which case its meaning is that males and designers all earn over $50,000.

We have defined intensional answers as answers that *include* knowledge statements. This suggests that intensional answers may include facts as well, a possibility that at first appears to be in contradiction with the very purpose of such answers. The motivation behind such *mixed* answers (intensional answers that do not include facts are called *pure*) is perfectness. Perfect answers are attractive, because they provide precise characterization of the facts sought by the user (indeed, a perfect intensional answer is an alternative specification of the query). However, perfect answers are not always feasible. Mixing knowledge statements with facts is a way of "perfecting" pure intensional answers that are imperfect.

As an example, assume that the (pure) intensional answer "engineer" is a constituent of the employees who earn over $50,000; that is, all engineers earn over $50,000, but there may be some non-engineers who earn as much. If the facts describing these individuals are added to the answer, the answer becomes perfect (but mixed). Similarly, assume that "engineer" is a constraint on the employees who earn over $50,000; that is, all the employees who earn over $50,000 are engineers, but some engineers may not earn that much. If the facts describing these individuals are subtracted from the answer, the answer becomes perfect (but mixed). Facts may be both added to and subtracted from a knowledge statement, in which case a perfect answer may be obtained from a knowledge statement that is neither a constituent nor a constraint. For example, the answer "the engineers, and also Betty, but not John."

Because they include facts, mixed intensional answer may become invalid once the database extension is modified.[3] Thus, "engineer" may be a valid constituent of the set of employees who earn over $50,000, given a particular database extension, but invalid given another extension. On the other hand, assume that the database systems enforces integrity constraints that state that the minimal salary for employees with college degrees is $40,000, and that employees with engineering degrees earn at least $20,000 over that minimum. In that case, "engineer" will be a valid constituent of the set of employees who earn over $50,000, given *any* database extension. An intensional answer that is valid given any database extension is called *invariant*. Invariant answers are computed entirely from the knowledge, without any reference to the facts presently stored in the database.

Evidently, answers that are perfect, pure and invariant are most desirable. Quite often, however, such answers are infeasible, and some of these objectives must be sacrificed (we have already seen how purity may be sacrificed for perfectness). Indeed, none

of the methods surveyed here achieve all three objectives. Yet the answers they generate are highly useful.

2.2 Survey

Several approaches to intensional answering are reviewed below. The works are described according to their data model assumptions: generalization hierarchy (i.e., semantically-rich or object-oriented), relational, and logic-based (i.e., deductive). We focus on the kinds of intensional answers generated, rather than the methods proposed for generating them.

2.2.1 Shum and Muntz

Shum and Muntz (1988a) assume a data model that includes a generalization hierarchy of classes. As this structure is an essential component of the semantically-rich and object-oriented approaches, the results may be applied in data models that adhere to these approaches. Noting that answers that are exhaustive enumerations of individual objects are not always the most efficient or the most effective means of information exchange, the authors are concerned with implicit representation of answers through concise expressions that involve both classes and individuals. An expression may include classes and individuals as either *positive* or *negative* terms (i.e., they are either added to the answer or subtracted from it). For example, an acceptable answer to the query "who earns over $50,000?" is "all engineers except John" or "all engineers and all managers except junior managers." Because they contain extensional information, these perfect answers are mixed and therefore not invariant.

The authors note that a query may be answered with several different intensional answers, and the main issue they consider is how to determine which answer is best. They define an answer as optimal if it has the smallest number of terms. Among answers with the same number of terms, answers with the maximal number of positive terms are preferred.

In another study Shum and Muntz (1988b) are concerned with a different kind of intensional answer, based on aggregate expressions. An aggregate expression is a sequence of terms of the kind $r/t\ C$, where C is a class, t is its total number of individuals, and r is the number of these individuals who belong to the answer. For example, an acceptable answer to the query "who earns over $50,000?" is "90/120 engineers + 20/30 managers." An intensional answer must "cover" the extensional answer, but an individual may be covered by more than one term. This requirement provides some kind of "perfectness" of characterization, but it is important to note that this is not quite the same as the perfectness defined in Section 2.1, which compares the extension of the intensional characterization with the actual extensional answer. It is not obvious how to evaluate the extension of these aggregate expressions, as each expression corresponds to many different extensional answers.[4] If we relax the definition of perfectness to include situations, where *one* of the possible extensions of the intensional answer is equal to the extensional answer, then these aggregate expressions are perfect. While the computation of these answers depends on the extension, the statements themselves (sums of fractions of classes) may be considered purely intensional.

Again, a query may be answered with several different intensional answers, and the main issue considered is how to determine which answer is best. Two criteria for op-

timality are recognized: conciseness and preciseness. Conciseness is simply the number of terms in the answer. Preciseness measures the amount of information encapsulated in the expression, and is based on the concept of entropy, known from information theory. For example, since each extensional answer is always covered by the root class, a possible answer to the previous query is "110/480 employees." While this answer is more concise than the former answer, it is less precise (conveys less information). To handle these often conflicting criteria, the authors attempt to find the most precise answer for a given expression length.

2.2.2 Motro

In the framework of relational databases, this author shows how to compute intensional answers from known integrity constraints (Motro, 1989). The generation of intensional answers is treated as an application of the following more general problem, called the *view inference problem: Given a query and a set of database views that possess a particular property, what views of the answer possess this property?* Consider the property of being *empty*. The problem then becomes: given a query and a set of empty views, what views of the answer are empty? Since empty views are statements of constraints,[5] the problem becomes: given a query and a set of constraints, what are the constraints that apply to the answer? The author describes a method for handling the general problem, and a simple extension of this process infers views of the database that are contained entirely in the answer. Altogether, the intensional answers characterize the extensional answers in two ways: (1) with *constraints*—views of the extensional answer that are empty; and (2) with *constituents*—views that are contained entirely in the extensional answer. These imperfect answers are pure and invariant.

As an example, consider a database with relation EMPLOYEE(NAME, TITLE, SALARY, DEPARTMENT), and two constraints: one states that all employees of the design department earn over $50,000, and the other states that all employees in researcher positions are in the design department. The query "who are the employees of the design department?" will be answered extensionally (by the usual query processing mechanism) with a list of individuals, and intensionally (by the view inferencing mechanism) with two characterizations: "(1) all employees retrieved earn over $50,000, and (2) all employees in researcher positions retrieved."

2.2.3 Chu and Lee

Chu and Lee describe a method by which intensional information is gathered from the extension of a plain relational database (Chu & Lee, 1990). Among the works surveyed here, this work is unique in that the knowledge base used for generating intensional answers is dependent on the extension. This new information allows the system to interpret its data as if it is structured in accordance with a model which is an extension of the entity-relationship model, with entity-sets, one-to-many relationships, generalization relationships and rules. For example, assume a relation EMPLOYEE with attributes NAME and POSITION, and a relation POSITION with attributes RANK and LEVEL. Using knowledge of the key attributes, it is possible to infer one-to-many relationships between entity-sets (relations); for example, between POSITION and EMPLOYEE. Also, by selecting specific values for non-key attributes, it is possible to define new entity-sets that would be related through generalization relationships to existing entity-sets; for example, the entity-set SENIOR_EMPLOYEE of the employees for whom LEVEL = senior. Final-

ly, by observing the behavior of the data, it is possible to infer rules that express relationships between the values of attributes; for example, RANK > 6 → LEVEL = senior.

A typical query specifies a set of output attributes and a condition on related attributes; for example, "list the names of the employees with rank 8," or "list the names of the employees who are senior." In the first query it can be concluded from the example rule that each employee in the answer is senior. In the second query it can be concluded from the same rule that the employees with rank greater than 6 are all included in the answer. Thus, rules can be applied in both forward direction (deduction) and backward direction (abduction) to infer intensional statements, such as "the answer is *contained in* the set of senior employees," or "the answer *contains* the set of employees with rank greater than 6."[6] Note that a particular query may require the "chained" application of numerous rules, some deductively and some abductively. These imperfect characterizations are purely intensional, but are not invariant. The authors then consider the addition of "rules" that apply only to individuals, to complete the definitions of derived entity-sets; for example, "John is also senior" (although John's rank may be lower than 6). Clearly, if such "rules" are available to complete all such definitions, it is possible to generate perfect intensional answers; however, these answers would then be mixed.

2.2.4 Cholvy and Demolombe, Pirotte and Roelants

Cholvy and Demolombe (1986) assume a logic-based model consisting of a set of first-order formulas over a given set of base predicates. The formulas are regarded as axioms, and the set must be consistent. Axioms express information considered invariant. For example, an axiom may declare that "all engineers earn over $50,000," but not that "John is an engineer." Hence, all extensional information is excluded.

An intensional answer to a query $Q(X)$ is a set of formulas $A(X)$ such that $(\forall X)A(X) \rightarrow Q(X)$ is a theorem. For example, the query "who earns over $50,000?" is answered intensionally "all engineers." Because an intensional answer must derive the query, but not vice versa, it is an imperfect characterization. Obviously, it is also purely intensional and invariant.

Pirotte and Roelants (1989) adopt the same definition of intensional answers, but assume the ordinary logic-based model, with two notable exceptions. (See Section 3.2.1, below.) First, rules are assumed to be nonrecursive. Second, a derived predicate may also have an additional rule (of a different form) associated with it, that guarantees that the definition of this predicate is complete (i.e., facts not generated by its defining rules are inconsistent with the database).

The main thrust of this work is the use of integrity constraints for improving intensional answers. Specifically, intensional answers may be identified as inconsistent (and discarded), or they may be simplified considerably. For example, consider the constraint that "all employees earn under $80,000." It can help identify an answer such as "all employees who earn over $90,000" as inconsistent (i.e., always empty), and it can help transform an answer such as "all employees who earn under $90,000" into the simpler answer "all employees."

2.2.5 Imielinski

The model adopted by Imielinski (1987) differs from the ordinary logic-based model in three respects. First, rules are not used to define new predicates, but to *augment* base

predicates. Second, while rules are allowed to be recursive, mutual recursion is disallowed. Third, queries are expressed in the relational algebra. The author argues that rules should be allowed to occur in answers, and defines an answer as a set of *facts* that satisfy the query, and a set of *rules* that may be applied to these facts to generate additional facts that satisfy the query. Hence, the structure of an answer is identical to the structure of the database itself, with an extensional part and an intensional part. As an example, assume a rule that states that employees of the same department must have the same skills, and consider the query "who are the employees with programming skills?" This query would be answered by a set of persons (facts) and a rule specifying all those in their departments. The facts in the answer were either present in the database or were derived by the application of other rules. Exhaustive enumeration of this answer may be performed upon request.

Obviously, these answers are perfect (i.e., their extensions are identical to the extensional answers). Because of their extensional component, these answers are mixed and therefore not invariant. Note, however, that only the extensional part of the answer depends on the extension; the intensional part is computed only from the database intension. In other words, when the database extension changes, only the extensional part of answers needs to be evaluated anew.[7]

2.3 Effectiveness

Perfectness, purity, and invariance were offered in Section 2.1 as criteria for classifying the various *kinds* of intensional answers, and our survey in Section 2.2 refers to these criteria. While the survey does not address the various *methods* proposed for generating these answers, we offer five additional criteria for assessing the effectiveness of such methods: *completeness*, *non-redundancy*, *optimality*, *relevance*, and *efficiency*. These criteria are discussed briefly below. For more detailed assessment of the effectiveness of the various intensional answering methods with respect to these criteria, see Motro (forthcoming).

A method is *complete* if it discovers all the intensional answers that exist. Note the difference between completeness (a complete method generates all intensional answers) and perfectness (a perfect intensional answer is equivalent to the query).

Non-redundancy is concerned with avoiding intensional statements that provide no additional information over the query itself, or its extensional answer, or other intensional statements. For example, an intensional statement which is a rephrasing of the query is redundant. Similarly, an intensional statement which is a disjunction of terms of the kind $X = a$, where a is a value of the extensional answer, is redundant. To avoid a conflict between completeness and non-redundancy, completeness may be interpreted as the computation of all non-redundant answers.

When multiple intensional answers exist, it is sometimes sensible to define a measure that describes the "goodness" of each answer. A method is *optimal* if it generates the best answer according to the measure. The multiple intensional answers may be simply syntactic variants (i.e., a situation involving redundancies), in which case an optimal method selects the most desirable (canonical) variant.

Relevance is concerned with avoiding intensional statements that have little or no value to the user. For example, a user who inquires about the programmers proficient in

Prolog may be uninterested in finding out that they all have medical insurance from Prudential. Non-redundancy and relevance are both concerned with undesirable answers. However, while redundant answers can be identified accurately and unambiguously, relevance is often a matter of opinion or degree.

Efficiency is concerned with the cost of deriving intensional answers. While the volume of intensional information is usually much smaller than the volume of extensional information, the processing of intensional information involves more complex algorithms. Efficiency often conflicts with the other four criteria, as attempts to satisfy these criteria may contribute to the complexity of the method.

Note that completeness and optimality are often alternative criteria. Completeness is important for imperfect intensional answers, where each intensional statement may contribute a different characterization. On the other hand, a single perfect answer is often sufficient, and finding the most desirable answer may be important.

2.4 Directions for Further Research

The issue of *relevance* presents the most difficult challenge to the usability and effectiveness of intensional answers. While criteria such as completeness, non-redundancy, optimality and efficiency are usually well-defined and quantifiable, relevance is a more elusive criterion.

The general approach is to identify somehow the set of *concepts* that are relevant to the user, and then admit only valid knowledge statements that incorporate these concepts. Several strategies have been offered for identifying the relevant concepts. One approach is to assume that the set of relevant concepts includes every concept of the database intension (thus every intensional answer is relevant). Another approach is to assume that the relevant concepts are those mentioned in the query. A third approach is to assume that the relevant concepts are supplied by the user (either in a predefined "user profile," or by means of a dialogue). The advantage of the first approach is its ultimate simplicity, but in many respects it evades the central problem. The third approach provides more accuracy, but also demands user involvement.

Clearly, there is advantage in judging the relevance of answers "automatically" (i.e., without user involvement). But a problem with the second approach is that it is often too restrictive; while it is reasonable to assume that a concept mentioned in the query is relevant, other concepts may be relevant as well. One possibility for addressing this problem, while still avoiding the need to consult the user, is to define a priori *relevance dependencies* among the intensional concepts. The set of concepts relevant to a query would then be the *closure* of the set of concepts mentioned in the query, according to the predefined relevance dependencies. Formally, a relevance dependency $X \rightarrow Y$ implies that whenever the concepts X are relevant, then the concepts Y are also relevant. For example, assume the intensional concepts NAME, TITLE, SALARY, ADDRESS, and PHONE, and the relevance dependencies NAME,TITLE → SALARY and NAME,ADDRESS → PHONE. Thus, salaries are relevant to queries that mention employees and their titles, while telephone numbers are relevant to queries that mention employees and their addresses.

Another important issue is how to communicate intensional answers to users adequately. Relatively little effort is required for adequate presentation of extensional answers (tabulation, sorting, grouping, etc.). This is because extensional information is

relatively simple, and all users may be assumed familiar with its form and meaning. Intensional information, however, is more complex (e.g., rules, constraints, hierarchies, views), and users may not always be assumed familiar with its form and meaning. Hence, the presentation of intensional answers may require more effort.

It is reasonable to assume that the user is familiar with the query language that he is using. Therefore, the syntax and semantics of the query language could be adopted for the presentation of intensional answers. However, this by itself does not guarantee comprehensibility.

As discussed in Section 2.2.1, Shum and Muntz are concerned with compact representations of the extensional answers through the available hierarchy of classes. Considering only *predefined* classes is somewhat limiting, as *ad-hoc* classes, created through any of the attributes, could be just as effective for intensional answers. For example, the query "who earns more than $50,000?" could be answered intensionally by "all the employees assigned to project 3382 and Betty." Here, the assignment of employees to projects is assumed to be information which is not represented in the class hierarchy. And, if imperfect characterizations are used, then an intensional answer to the same query could include the observation "all employees assigned to project 3382."

Thus, their approach could be generalized to discover ad-hoc classes that are related (through containment or equality) to the result. In other words, the intensional answers computed by Shum and Muntz are expressions that contain only the unary predicates that define classes. These could be generalized to expressions that involve *any* of the database predicates. In either approach the answers depend on the extension, but in the more generalized approach the search is much less restricted. Clearly, the intensional answers should create only those ad-hoc classes that appear to be relevant to the query.

This possibility of inferring intensional answers from purely extensional information recalls the work of Chu and Lee, discussed in Section 2.2.3. The fundamental difference is that Chu and Lee discover intensional characterizations of the *entire* database extension, and then proceed to conclude the characterizations that apply to particular queries (the latter process is similar to most other methods). The possibility discussed here is to discover characterizations of *specific* extensional answers.

Obviously, the discovered ad-hoc classes reflect the behavior of the data. This leads us to further generalize intensional answers to discover and report any behavior of the data in the answer, which is markedly different from their behavior in the entire domain. For example, if the proportion of engineers is in general 40 percent, but only 10 percent among the employees who earn over $50,000, then an intensional answer to the same query would include the observation "only 10 percent are engineers."

3. RESPONDING WITH KNOWLEDGE TO KNOWLEDGE QUERIES

Our definition of database knowledge included all information that pertains to multitudes of database facts. As such it admits the entire definition of the database. Our discussion of knowledge queries is divided into two. In Section 3.1 we briefly discuss language features of conventional database systems for querying the database definition. In Section 3.2 we discuss in considerable detail a more ambitious effort to develop a language for querying the rule base of a deductive database.

3.1 Querying the Database Schema

The knowledge stored in a database may include statements defining the database schema (e.g., generalization hierarchies), integrity constraints, inference rules, and views (often used for controlling database access). Consequently, every database system must provide some means for maintaining this information. This may be done with a special database administration language, with extensions to the standard query language,[8] or with specialized interfaces. In most cases, however, the query facility is either a browsing tool or a retrieval mechanism limited to simple searches.

Examples of requests that can be handled are (1) scroll through the definition of the database schema, (2) display the access permissions of a particular user, (3) display the integrity constraints associated with specific attributes, (4) display the inference rules that define a specific predicate, and so on. Clearly, while these are indeed queries against the database knowledge, their capacity is quite limited.

3.2 KDL: A Knowledge and Data Language

Initial results of research to develop a single language for expressing queries against both the data and the knowledge of a deductive database were reported in Motro and Yuan (1990). We begin with several motivating examples.

Assume a knowledge-rich database with a relation STUDENT(SNAME,GPA) that stores the grade-point averages of students, and an inference rule that defines HONOR-STUDENT as a student whose grade-point average is at least 3.7. Consider now these two English language queries: "who are the honor students?" and "what does it take to be an honor student?" Both are natural queries, and the answers to both are included in the database. However, conventional database systems can handle the first (which applies the knowledge to access the data), but not the second (which addresses the knowledge directly).

For a second example, assume that STUDENT includes two more attributes NATIONALITY and MARITAL-STATUS, and consider these two English language queries: "are all foreign students married?" and "must all foreign students be married?" Obviously, these are two different queries; but the user may not be aware that to a database system "are they?" and "must they?" are of two different *kinds*. The first may be processed by searching for students who are foreign but not married, returning "yes" if none were found and "no" otherwise. Thus, it is a simple "data query." To answer the second query the system should look for a possible item of knowledge that *requires* each foreign student to be married. Again, while the information is available, the query cannot be formulated and the answer cannot be retrieved.

Similarly, consider the query "could an honor student be foreign?". To answer a query of the kind "could it?" the system must check whether a hypothetical item of knowledge (e.g., a foreign honor student) would *contradict* the stored knowledge.

As a fourth example, assume that in addition to honor students, there is another category of excellence called the Dean's List, and consider this query: "what is the difference between an honor student and a Dean's List student?" To answer this query the system should *compare* the definitions of these two concepts, identify their maximal shared concept, and elucidate the difference. Other possible answers to such queries are that one concept is subsumed by the other, or that the concepts are unrelated.

Often, "knowledge queries" are follow-ups to "data queries." The answer to a query may be unexpected, raising questions about the meaning of a certain term (the first ex-

ample). Or the answer to a query may seem to indicate some regularity, and the user is intrigued as to whether this regularity is indeed obligatory or only coincidental (the second example).

As these examples demonstrate, often there is a genuine need to query the database knowledge. Also, users who are not familiar with the internal representation of data and knowledge may not be able to distinguish easily between data queries and knowledge queries. This distinction is also undesirable; not only should access to knowledge be provided, but access to data and knowledge should be unified within a single instrument.

Presently, the query language, called KDL, includes two "twin" statements, one for expressing data queries ("retrieve the data elements that satisfy ψ") and one for expressing knowledge queries ("describe the data elements that satisfy ψ"). The data query corresponds to the basic retrieval statement of various knowledge-rich databases. Essentially, the knowledge query statement allows users to inquire about a single concept. Following are some English language equivalents of queries that are possible with this statement (the answers are based on the example knowledge base described below): (1) "what is an honor student?" (answer: "a student with GPA above 3.7"), (2) "when are math students whose GPA are above 3.7 eligible for teaching assistantship in the database course?" (answer: "when they have completed this course under the professor currently teaching it with final grade above 3.3, or under some other professor with final grade 4.0"), and (3) "when are students who have completed a course with 4.0 eligible for teaching assistantship in this course?" (answer: "when they are honor students").

3.2.1 The Model

The data model is based on first-order logic, with definitions similar to those commonly used (Ullman, 1988). A knowledge-rich database D consists of three mutually disjoint sets of predicates[9]:

- A set P of predicates, and, for each predicate, an associated set of stored facts. Each predicate is defined to be *true* over its set of associated facts, and *false* otherwise.
- A set R of built-in predicates (the facts which make these predicates true are assumed to be known, and are treated as if they are stored).
- A set S of predicates, and, for each predicate, an associated set of rules. Each predicate is the head of each of its associated rules. Rules have the form $q \leftarrow p_1 \wedge \ldots \wedge p_n$, where q and p_i each are atomic formulas.

The first two sets are referred to as the *extensional database* (EDB), and the third set is referred to as the *intensional database* (IDB). The entire database (i.e., the EDB facts and the IDB rules) is understood as a collection of *axioms*, and the resolution principle is established as the *rule of inference*. A fact or a rule Φ, which is a logical consequence of the database (i.e., follows from the axioms by the rule of inference) is a *theorem*, denoted $\vdash\Phi$. Φ is also said to be *logically derived* from the database. A theorem may also involve an *hypothesis*. If $(p \leftarrow \varphi) \leftarrow \psi$ is a theorem,[10] then $p \leftarrow \varphi$ is a theorem under

the hypothesis ψ, denoted $\psi \vdash (p \leftarrow \varphi)$. $(p \leftarrow \varphi)$ is also said to be *logically derived* from the database and the hypothesis ψ.

As an example, consider the following knowledge-rich database. The EDB has six predicates, as follows:

student(Sname,Major,Gpa)
enroll(Sname,Course)
teach(Prof,Course)
prereq(Course,Pcourse)
taught(Prof,Course,Sem,Eval)
complete(Sname,Course,Sem,Grade)

The predicates *student*, *enroll*, and *teach* have the obvious meanings (the last two pertain to the current semester). *prereq* means that course *Pcourse* is a prerequisite to course *Course*. *taught* means that professor *Prof* taught course *Course* in semester *Sem* and received evaluation *Eval*. *complete* means that student *Sname* completed course *Course* in semester *Sem* and received grade *Grade*. In addition, the EDB includes the following built-in predicates: $=, \neq, >, \geq, <, \leq$. The IDB has three predicates, defined as follows:

$$
\begin{aligned}
honor(X) &\leftarrow student(X, Y, Z) \wedge (Z > 3.7) \\
prior(X, Y) &\leftarrow prereq(X, Y) \\
prior(X, Y) &\leftarrow prereq(X, Z) \wedge prior(Z, Y) \\
can\_ta(X, Y) &\leftarrow honor(X) \wedge complete(X, Y, Z, U) \wedge (U > 3.3) \wedge \\
&\quad\ taught(V, X, Y, Z, W) \wedge teach(V, Y) \\
can\_ta(X, Y) &\leftarrow honor(X) \wedge complete(X, Y, Z, U) \wedge (U = 4.0)
\end{aligned}
$$

The predicate *honor* is defined as students with grade-point average over 3.7. The predicate *prior* is defined recursively as the transitive closure of *prereq*. The predicate *can_ta* defines the students who can be teaching assistants for a course, as honor students that had taken the course previously from the professor currently teaching this course with final grade over 3.3, as well as honor students who had completed the course with the final grade 4.0.

3.2.2 Data Queries

In most knowledge-rich database systems the language used to express knowledge statements is essentially the query language as well, and a query is a set of rules that defines the derived predicate that should be computed and printed. A query may also be a predicate symbol with a list of arguments, in which case the definition of this predicate should be available to the system.

Consider this query statement:

retrieve p
where ψ

where p is an atomic formula (the subject of the query), and ψ is a conjunction of atomic formulas (the qualifier of the query). The variables that appear in p are assumed to be *free*; all other variables are quantified existentially.

This statement finds the database values whose substitution for the variables of p and ψ satisfies $p \wedge \psi$, retrieving the values of the free variables. The **where** clause may be omitted, in which case the formula ψ is assumed to be *true*.

Note that when p is an EDB predicate, testing its satisfiability is done by checking a stored relation. However, when p is an IDB predicate, the processing of this statement may require recursive evaluation of rules. It is also possible for p to be a new predicate altogether, which is then assumed to be *defined* through ψ. In this case the **retrieve** statement finds the database values whose substitution for the variables of ψ satisfies ψ, retrieving the values of the free variables. The user can thus avoid defining a new predicate.

Example 1: Retrieve the honor students enrolled in the databases course. This data query is phrased as follows:

retrieve *honor*(*X*)
where *enroll*(*X*, *databases*)

Example 2: Retrieve the math students whose GPA are above 3.7 and who are eligible for teaching assistantship in the databases course. This data query is phrased as follows:

retrieve *answer*(*X*)
where *can_ta*(*X*, *databases*) **and** *student*(*X*, *math*, *V*) **and** (*V* > 3.7)

Note that *answer* is not a known predicate.

3.2.3 Knowledge Queries

As pointed out earlier, a data query applies the knowledge (i.e., the rules) to generate the answer, but knowledge itself is not part of the answer. Consider this query statement:

describe p
where ψ

where the subject p is an atomic formula with an IDB predicate and the qualifier ψ is as before.

This statement finds theorems $p \leftarrow \varphi$, where φ is a conjunction of atomic formulas that are logically derived from the database under the hypothesis ψ. The **where** clause may be omitted, in which case, since there is no hypothesis, this statement retrieves the theorems $p \leftarrow \varphi$ that are logically derived from the database—in particular, every rule whose head predicate is p (after proper identification of the head of the rule with p).

Example 3: When is a math student whose GPA is above 3.7 eligible for teaching assistantship in the databases course? This knowledge query is phrased as follows:

describe *can_ta(X, databases)*
where *student(X, math, V)* **and** $(V > 3.7)$

The answer includes the following two theorems that can be logically derived from the database under the given hypothesis:

$$\begin{aligned} \textit{can\_ta}(X, \textit{databases}) &\leftarrow \textit{complete}(X, \textit{databases}, Z, U) \wedge (U > 3.3) \wedge \\ &\quad \textit{taught}(V, X, Y, Z, W) \wedge \textit{teach}(V, Y) \\ \textit{can\_ta}(X, \textit{databases}) &\leftarrow \textit{complete}(X, \textit{databases}, Z, U) \wedge (U = 4.0) \end{aligned}$$

The student must have either completed the course under the professor currently teaching it with a final grade over 3.3, or the student must have completed the course with a final grade of 4.0.

Example 4: What does it take to be an honor student? This knowledge query requires no **where** clause:

describe *honor(X)*

Clearly, the definition of *honor* is a theorem that can be logically derived from the database:

$$\textit{honor}(X) \leftarrow \textit{student}(X, Y, Z) \wedge (X > 3.7)$$

The student must have GPA over 3.7.

Note that the **retrieve** statement for querying data and the **describe** statement for querying knowledge have similar syntax, differing only in their initial keyword. Thus, pairs of questions such as "retrieve the honor students" and "describe the honor students" are expressed identically, except for the initial keyword.

3.2.4 Directions for Further Research

Specific algorithms for processing knowledge queries of the kind described above are given in Motro and Yuan (1990). Some remaining issues and future directions are discussed below.

An answer to a knowledge query is *free of redundancies* if none of its formulas is a logical consequence of any of its other formulas. An answer to a knowledge query is *complete* if no other sound and nonredundant formula exists. Issues of redundancy and completeness of knowledge answers require further investigation. Also, the qualifier ψ was assumed to be a conjunction of atomic formulas. Additional work is necessary to generalize this formula to allow disjunctions and negations.

Work is also underway to extend the present **describe** statement to allow new kinds of queries. We sketch here four such extensions. Recall that the semantics of the **describe** statement admitted answers whose body φ and the hypothesis ψ are together sufficient to derive the subject p. Consequently, hypothesis formulas that are entirely

unnecessary for the derivation are simply ignored. For example, a query to describe the honor students, and a query to describe the honor students that have taken the database course, are answered identically. At times, it may be beneficial to provide only those answers where ψ proved necessary. For example, the answer to the query:

describe *honor*(*X*)
where necessary ***complete***(*X*, *Y*, *Z*, *U*) **and** (*V* > 3.3)

would generate only those answers were both formulas of the qualifier were necessary. A second extension is demonstrated by this query that would inquire whether honor status is *required* for a teaching assistantship:

describe *can_ta*(*X*, *Y*)
where not *honor*(*X*)

The answer *false* would indicate that honor status is required for a teaching assistantship. A third extension is demonstrated by this subjectless query that would inquire whether students with GPA under 3.5 are *allowed* to be teaching assistants:

describe
where *student*(*X*, *Y*, *Z*) **and** (*Z* < 3.5) **and** *can_ta*(*X*, *U*)

The answers *true* and *false* would indicate whether this hypothetical situation is possible. A fourth extension is demonstrated by this query that would inquire about the advantages of honor status:

describe *
where *honor*(*X*)

The "wildcard" subject would express all the subjects that are derivable from this qualifier.

Finally, the language is being extended to allow knowledge queries that *compare* two concepts. A possible syntax would combine two **describe** statements:

compare
(**describe** p_1
where ψ_1)
with
(**describe** p_2
where ψ_2)

The answer should elucidate the maximal shared concept. (If it is empty, then the two concepts are *unrelated*; if it is equal to one of the given concepts, then one concept *subsumes* the other.)

4. CONCLUSION

In this paper we discussed database answers of a new type—answers that include knowledge statements. Two different types of query answering mechanisms were sur-

veyed: mechanisms that use database knowledge to characterize and illuminate conventional answers, and mechanisms that handle queries about the knowledge available to the database. These mechanisms are complementary to the conventional query answering mechanisms that respond to conventional queries with facts.

Responding to data queries with data and to knowledge queries with knowledge appears to be intuitive. Additionally, we argued that data queries are sometimes better answered with knowledge. This brings to mind a fourth possibility: answering queries about the knowledge with facts. Such an exchange may prove useful as a means of *illustrating* answers to queries about the knowledge *with examples.*

Undoubtedly, the ultimate goal should be to design a single query answering model which will integrate all the various mechanisms. A single language would allow users to express both data and knowledge queries, and answers would combine facts and knowledge in the "most desirable" way.

For the most part, the knowledge, from which the answers described in Sections 2 and 3 were inferred, had been identified and stored earlier by "knowledge experts." It is knowledge that had been acquired from other sources and is known to be invariant to the extension.

As demonstrated by the work of Chu and Lee (Section 2.2.3) and others (Piatetsky-Shapiro & Frawly, 1989; Tsur, 1990), it is also possible to acquire useful knowledge from the extension itself. Knowledge thus "mined" from the extension may not be as permanent and universal as asserted knowledge (i.e., it is not guaranteed to hold over all database extensions), and must therefore be offered with confidence estimations. However, mined knowledge may prove to be extremely useful in many applications.

A related issue is the derivation of intensional characterizations of extensional answers from the database extension (Section 2.4). Here, the extensional answer and the entire database extension are two "populations," which should be mined for knowledge statements that will characterize the latter within the universe of the former.

Acknowledgment: This work was supported in part by NSF Grant IRI-9007106 and by the AT&T Affiliates Research Program.

NOTES

1. A more detailed discussion of database knowledge is discussed in Section 1.1.
2. This definition of knowledge implies that even conventional databases (e.g., relational) contain knowledge.
3. However, pure intensional answers are not necessarily extension-invariant!
4. As is the case with other kinds of statistical summaries.
5. This follows from the fact that every constraint of the form $(\forall x_1) \ldots (\forall x_n)\ (\alpha(x_1, \ldots, x_n) \rightarrow \beta(x_1, \ldots, x_n))$, where x_i are domain variables and α and β are safe relational calculus expressions with these free variables, may be rewritten as an empty view: $\{x_1, \ldots, x_n \mid \alpha(x_1, \ldots, x_n) \wedge \neg\beta(x_1, \ldots, x_n)\} = \emptyset$.
6. These characterizations are essentially the same as the characterizations by constraints and constituents.
7. This is in contrast with the mixed answers of Shum and Muntz, where the entire answer depends on the extension.
8. In certain architectures the standard query language need not even be extended.

9. A *predicate* is a Boolean-valued function, and an *argument* is either a variable or a constant. (To distinguish between variables and constants, the first character of a variable name is always a capital letter.) An *atomic formula* is a predicate symbol with a list of arguments.
10. Note that this is another notation for the rule $p \leftarrow \varphi \wedge \psi$.

REFERENCES

Chakravarthy, U.S., Fishman, D.H., & Minker, J. (1986). Semantic query optimization in expert systems and database systems. In L. Kerschberg, (Ed.), *Expert Database Systems Proceedings from the First International Workshop*, (pp. 659–674). Menlo Park, CA: Benjamin/ Cummings.

Cholvy, L., & Demolombe, R. (1987). Querying a rule base. In L. Kerschberg, (Ed..), *Expert Database Systems: Proceedings from the First International Conference*, (pp. 477–485). Charleston, SC, April 1–4.

Chu, W.W., & Lee, R.-C. (1990). *Using type inference and induced rules to provide intensional answers.* Technical Report CSD-900006, Computer Science Department, University of California at Los Angeles, March.

Imielinski, T. (1987). Intelligent query answering in rule based systems. *Journal of Logic Programming, 4*(3):229–257.

Motro, A. (forthcoming). Intensional Answers to Database Queries. *IEEE Transactions on Knowledge and Data Engineering* (to appear).

Motro, A. (1989). Using integrity constraints to provide intensional responses to relational queries. In *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, (pp. 237–246). Amsterdam, The Netherlands, August 22–25.

Motro, A., & Yuan, Q. (1990). Querying database knowledge. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, (pp. 173–183). Atlantic City, NJ, May 23–25.

Piatetsky-Shapiro, G., & Frawly, W. (Eds.), (1989). *Proceedings of IJCAI-89 Workshop on Knowledge Discovery in Databases*, Detroit, MI, August 20.

Pirotte, A., & Roelants, D. (1989). 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*, (pp. 652–659). Los Angeles, CA, February 6–10.

Shum, C.D., & Muntz, R. (1988a). Implicit representation for extensional answers. In *Proceedings of the Second International Conference on Expert Database Systems*, (pp. 257–273). Tysons Corner, VA, April 25–27.

Shum, C.D., & Muntz, R. (1988b). An information-theoretic study on aggregate responses. In *Proceedings of the Fourteenth International Conference on Very Large Data Bases*, (pp. 479–490). Los Angeles, CA, August 25–28.

Tsur, S. (1990). Data dredging. *Data Engineering, 13*(4):58–63.

Ullman, J.D. (1988). *Database and Knowledge-Base Systems*, Volume I. Rockville, MD: Computer Science Press.

Address all correspondence to Amihai Motro, Department of Information and Software Systems Engineering, George Mason University Fairfax, VA 22030-4444.