Journal of Intelligent Information Systems, 7, 51–73 (1996)

Panorama: A Database System that Annotates Its Answers to Queries with their Properties

AMIHAI MOTRO ami@gmu.edu
Department of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030-4444

Abstract. 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. This paper describes a relational database system that similarly annotates its answers with their properties. The process 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 advantages of the method described 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 can be expressed as a polynomial in the size of the meta-database. (4) It can be implemented *externally* with any commercial relational database system.

Keywords: relational database, cooperative answering, answer characterization, meta-answer

1. Introduction

Given a query, a typical database system is concerned only with answering it correctly and efficiently. In contrast, when responding to similar 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 a simple example, consider an inquiry about bookstores in Washington. After listing several bookstores, the person answering the question might add comments such as:

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 might be incomplete, whereas 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.

We refer to the various statements about the answer as *properties* of the answer. In this paper, we describe a database system that similarly annotates its answers with their properties. The process does not involve "understanding" or "intelligence", as these terms are commonly understood. Basically, it 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.

Database systems that venture beyond the straightforward execution of queries and attempt to provide their users with additional information that is deemed helpful are usually termed *cooperative*. Examples of cooperative database systems include (Kaplan, 1982; Corella, 1984; Motro, 1986; Cuppens and Demolombe, 1988; Gal, 1988; Motro, 1990; Gaasterland, 1992; Gaasterland, Godfrey and Minker, 1992). In volunteering information about the properties of the answers it provides, the database system described here is attempting to be cooperative as well.

Our work is done in the framework of relational databases, and, as we shall demonstrate, it is feasible to extend an existing relational database system to store meta-information and compute meta-answers.

The results reported here are based on earlier theoretical results described mostly in (Motro, 1989b, 1990b). Also, specific applications of this general method to intensional answering and access authorization were described in (Motro 1989a, 1992). The focus of this paper is a general framework for managing *arbitrary* properties, and a language and a system that have been implemented to realize this general goal. This paper is organized as follows. Section 2 establishes a general and formal framework for asserting and manipulating meta-information, and it demonstrates the potential of this framework, by discussing various kinds of properties that may be of interest. The next two sections are devoted to Panorama, a prototype system for asserting and manipulating meta-information. Section 3 reviews the method: it sketches the overall approach, and then describes in detail the representation of meta-information and the process used to infer individual meta-answers (the complexity of this process is discussed in the Appendix). Section 4 focuses on the software: the language extensions and the architecture of the system. Section 5 concludes with an evaluation of the effectiveness of our approach, and a discussion of open research problems, including alternative approaches to the representation of meta-information and the computation of meta-answers.

2. The Framework

As mentioned in the introduction, in the course of human conversation one may provide various kinds of statements about one's answer. Our approach is independent of the specific kinds of statements involved. In this section we establish a general and formal framework for stating database properties, and for inferring the properties that apply to individual answers. We then demonstrate the potential of this framework by discussing various kinds of properties that may be of interest.

2.1. View Inferencing

We assume the following definition of a relational database (Maier, 1983). A *relation scheme* R is a finite set of *attributes* $A_1, \ldots, A_m$. With each attribute A_i a set of values, called the *domain* of A_i, is associated. A *tuple* t of relation scheme R is a function that assigns every attribute A_i a value from its domain. A relation r on the relation scheme R, denoted $r(R)$, is a finite set of tuples of R. A *database scheme* D is a set of relation schemes $R_1, \ldots, R_n$. A database d of the database scheme D, denoted $d(D)$, is a set of relations $r_1(R_1), \ldots, r_n(R_n)$. [1]

A *view* V is an expression in the relation schemes of D that defines a new relation scheme, and for each database d defines a unique relation v on this scheme.[2] The most frequent use of views is to formulate *queries*. Assume a query view Q on a database scheme D. The relation q defined by Q in a database d is called the *answer* to Q in the database d.

A *property* is any label that can be attached to a view. A property p is *inherited*, if all views derived from views with property p, also have property p. Note that inheritance is relative to the particular set of relation operators used in the derivation of new views. In this paper we shall consider only operations and properties that support inheritance.

Formally, we assume a set of properties, a set of view definitions, and a set of pairs (V, p), each asserting that view V has property p. A view with a property will be referred to as a *property view*.

Given that specified parts of the database possess a particular property, we are interested in determining the parts of an arbitrary view that inherit this property. Formally, this question is stated in the *view inference problem* defined as follows:

Assume that views $V_1, \ldots, V_n$ have property p, and consider a view V. Does V have property p? (i.e., can V be derived from $V_1, \ldots, V_n$?) If not, then which views of V have property p?

When this arbitrary view V corresponds to a user query, the view inferencing problem becomes a mechanism for annotating answers with their properties. Next, we discuss various kinds of properties that either have been shown to be useful, or have the potential to be useful.

2.2. Kinds of Properties

2.2.1. Soundness and Completeness

In (Motro, 1989b) we observed that the primary concern of users of any information system is the *integrity* of its answers. This concern may be divided into two parts: (1) Is the answer sound? i.e., is the information accurate in all respects? (2) Is the answer complete? i.e., does it include all the occurrences that actually exist? Hence, answers have integrity, if they contain *the whole truth* (completeness) and *nothing but the truth* (soundness).

We introduced a new model of integrity based on new kinds of integrity properties, called *soundness properties* and *completeness properties*.[3] A soundness property asserts that a particular view of the database is guaranteed to be sound, and a completeness property asserts that a particular view of the database is guaranteed to be complete. More specifically,

we assume a hypothetical database that models the real world perfectly. A database view is sound if it is contained in the corresponding real world view, and it is complete if contains the corresponding real world view.

Soundness and completeness properties are related to several well-known database concepts: *null values*, the *closed, open* and *locally open world assumptions*, and *integrity constraints*.

A null value (Date, 1990) denotes uncertainty of the real world value. Thus, a null value is practically a declaration of the *unsoundness* of a particular data value. The Closed World Assumption states that a database contains all the data that it attempts to model; the complementary Open World Assumption admits the possibly that some data may be missing (Reiter, 1978). Thus, our assumption here is essentially "open world", except for specific views that are declared to be "closed world". In our terminology, the Closed World Assumption corresponds to the assumption that every database relation is complete. The Locally Open World Assumption (Gottlob and Zicari, 1988) allows users to specify views of the database that are open. Thus, it corresponds to declarations of the *incompleteness* of particular views.[4] Finally, standard integrity constraints (Korth and Silberschatz, 1986) were shown in (Motro, 1989b) to be a specific kind of soundness properties.

View inferencing on the properties of soundness and completeness determines the soundness and completeness of each answer issued by the database. In the introductory example, the first three statements concern the integrity of the answer: the first statement concerns both soundness and completeness, the second statement concerns completeness, and the third statement concerns soundness.

2.2.2. Emptiness

The information stored in a database is of two kinds. *Extensional* information (often called data) is information that applies to individual real world objects. *Intensional* information (often called knowledge) is information that applies to multitudes of real world objects (Tsichritzis and Lochovsky, 1982). In the relational data model extensional information is expressed with *relations* over domains of *data values*, and intensional information is expressed with *integrity constraints*, which are formulas in predicate logic that assert required relationships among the data values.

An *answer* to a query is a set of data values that satisfy the qualification specified in the query. Therefore, answers are derived entirely from the extensional information in the database. Indeed, the only intensional information that characterizes this set of values is the qualification specified in the query that generated it. Still, the intensional information in the database may suggest additional characterizations of the extensional answer. If this intensional information is extracted, database values may gain additional meaning. Thus, a database query may be answered both *extensionally* (the usual answer), and *intensionally* (a set of characterizations). A survey of intensional answering techniques may be found in (Motro, 1994).

In (Motro, 1989a) we described a model in which integrity constraints are used to derive intensional answers of two kinds, called *constraints* and *containments*. A constraint defines a condition that is satisfied by the entire answer; it is therefore a condition *necessary* for

inclusion in the answer. A containment defines a subset of the database that is contained in its entirely in the answer; it is therefore a condition ***sufficient*** for inclusion in the answer. In the introductory example, the last two statements are intensional descriptions of the answer: the fourth statement is a constraint, and the fifth is a containment.

Constraints can be described as views with a property of *emptiness*. This follows from the fact that the integrity constraint $(\forall x_1) \ldots (\forall x_n)\ \alpha(x_1, \ldots, x_n) \Rightarrow \beta(x_1, \ldots, x_n)$, where x_i are domain variables and α and β are safe relational calculus expressions with these free variables, may also be stated as $\{(x_1, \ldots, x_n) \mid \alpha(x_1, \ldots, x_n) \wedge \neg\beta(x_1, \ldots, x_n)\} = \emptyset$. Consequently, view inferencing on the property of emptiness will derive the constraints that apply to individual answers, from the constraints that apply to the entire database. In other words, global intensional information is used to derive individual intensional answers.

2.2.3. Permissibility

The prevailing approach to access control in relational databases is to associate views with users. The database system maintains a set of view definitions, a set of user names, and a set of pairs (V, u). Each such pair gives user u permission to access view V. Given a query, the database system consults the permission pairs to determine whether the query, or any part of it, should be permitted (Stonebraker and Wong, 1974; Griffiths and Wade, 1976).

The permission pair (V, u) may be regarded as a property view; i.e., view V has the property *permitted to u*. Given a query submitted by u, view inferencing on the set of views permitted to u yields the views of the answer that should be permitted to u. The latter views are then applied to the entire answer to extract the data that are permitted to u. A model based on these principles was described in (Motro, 1992).

2.2.4. Other

It is possible to extend some of the properties discussed above to convey additional information; for example, the *time* that a particular view has been certified to be complete, or the *person* who certifies a particular view to be sound. Properties describing access permissions may be refined to include the kind of permission: read, write, etc.

As an example of other properties with potential to be useful, assume a database system that stores views that have been materialized recently; i.e., the answers to the n most recent queries are saved. Such views would then have the property *materialized*. Given a query, it may be useful to determine whether the query can be computed from materialized views, or whether the system must access the base relations. This could be especially useful in a distributed database environment, where some base relations are distributed.

3. The Method

In this section we describe the particular method used in Panorama to implement the framework of property views and view inferencing. Our approach to the view inference problem is essentially *algebraic*, and we term it *meta-processing*.[5]

3.1. Overview

We represent the definitions of the given views in special relations, using the concept of *meta-tuples*. A meta-tuple defines a subview (i.e., a selection and a projection) of a single relation, and several such meta-tuples can be used together to define more general views (i.e., views that involve more than one relation). All meta-tuples that define subviews of the same relation are stored together in one *meta-relation*, whose structure mirrors the actual relation. Standard algebraic operations (product, selection and projection) are extended to these meta-relations.

When a query is presented to the database system, it is performed *both* on the actual relations, resulting in a set of tuples that satisfy the query (the *answer*), and on the meta-relations, resulting in definitions of views of the answer that inherit the particular property of the given views (the *meta-answer*).

This approach is illustrated by the commutative diagram shown in Figure 1. The horizontal lines describe the relationships between meta-relations and relations, and the vertical lines describe query processing and meta-processing. The solid line describes the standard relational model: the relation A is derived from database D to answer the query Q. The dashed lines describe the extended model: the meta-relations D' define property views of the database relations D, and query processing is extended to manipulate also D', to yield the meta-relation A', that defines the property views of the answer A.

3.2. Representation of Meta-information

We consider only views that are defined by *conjunctive relational calculus expressions* (Ullman, 1982). Using domain relational calculus, expressions from this family have the form:

$$\{(x_1, \ldots, x_n) \mid (\exists y_1, \ldots, y_m)\, \psi_1 \wedge \ldots \wedge \psi_k\},$$

where the ψs may be of two kinds:

1. **membership:** $R(z_1, \ldots, z_p)$, where R is a relation scheme (of arity p), and the zs are either xs or ys or constants.

2. **comparative:** $w_1\, \theta\, w_2$, where w_1 is either an x or a y, w_2 is either an x or a y or a constant, and θ is a comparator (e.g., $<$, $\leq$, $>$, $\geq$, $=$, $\neq$).

In particular, each x and each y must appear at least once among the zs.

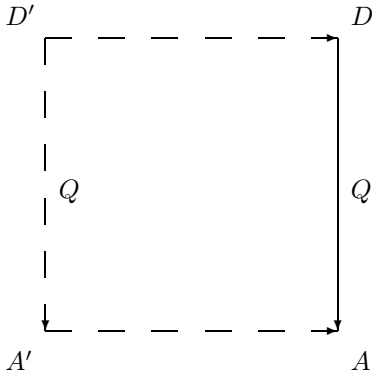

Figure 1. Meta-processing

We refer to such views as *conjunctive views*. While this family is a strict subset of the relational calculus, it is a powerful subset. The family of conjunctive relational calculus expressions has the same expressive power as the family of relational algebra expressions with the operations *product*, *selection* and *projection* (where the selection expressions are conjunctive).

The representation of conjunctive views in meta-relations recalls the representation of QBE queries in skeleton tables (Zloof, 1977).

For each relation $r(R)$, a *meta-relation* $r'(R')$ is defined. R' is identical to R, except for one additional attribute called $Property$. Also, an auxiliary relation $comparison$ is defined with scheme $Comparison = (X, Compare, Y)$. The meta-relations will be used to store membership formulas of views. Their tuples will be referred to as *meta-tuples*. Comparative formulas will be stored in the relation $comparison$.

Consider a view

$$V = \{(x_1, \ldots, x_n) \mid (\exists y_1, \ldots, y_m)\ \psi_1 \wedge \ldots \wedge \psi_k\}.$$

A formula ψ of the kind $R(z_1, \ldots, z_p)$ is first modified so that the zs that are xs are suffixed with $*$, and the zs that are variables (i.e., xs or ys) that appear only once in the whole expression are replaced with $\sqcup$ (blank). Hence, each component of the modified formula is either a constant, a variable, or a blank, and each may be suffixed by $*$. This tuple is extended with the property of the view V, and is stored in r'. A formula ψ of the kind $w_1\, \theta\, w_2$, where θ is not $=$, is transformed to the tuple (w_1, θ, w_2) and stored in the auxiliary relation $comparison$. If θ is $=$, then all occurrences of w_1 in the other formulas are substituted with w_2. Finally, we assume that variable names are not shared among views.

For our examples, we assume a simple consumer information database, whose scheme is shown in Figure 2. The domains of *Product.Name* and *Availability.Product* are identical, and the domains of *Store.Name* and *Availability.Store* are identical. In the following exam-

ples, the values S, C, and E are used, respectively, to designate the properties of soundness, completeness and emptiness, and views that have these properties will be called, respectively, *sound*, *complete*, and *empty*.

Product	=	(*Name*, *Model*, *Manufacturer*)
Store	=	(*Name*, *Location*, *Telephone*)
Availability	=	(*Product*, *Store*, *Price*)

Figure 2. Scheme of a consumer information database

Let V_1 be a complete view describing the manufacturers whose products are carried by the MarkUp Company:

$$\{(a) \mid (\exists b_1)(\exists b_2)(\exists b_3)\ \mathit{Product}(b_1, b_2, a)\ \wedge\ \mathit{Availability}(b_1, \mathrm{MarkUp}, b_3)\}$$

V_1 is represented with two meta-tuples:

$$(\mathrm{C}, x_1, \sqcup, *) \in \mathit{product}'$$
$$(\mathrm{C}, x_1, \mathrm{MarkUp}, \sqcup) \in \mathit{availability}'$$

Let V_2 be a sound view describing the names and locations of stores:

$$\{(a_1, a_2) \mid (\exists b)\ \mathit{Store}(a_1, a_2, b)\}$$

V_2 is represented with one meta-tuple:

$$(\mathrm{S}, *, *, \sqcup) \in \mathit{store}'$$

Let V_3 be a sound view describing the names and prices of products for which the MarkUp Company charges over 750:

$$\{(a_1, a_2) \mid \mathit{Availability}(a_1, \mathrm{MarkUp}, a_2)\ \wedge\ a_2 > 750\}$$

V_3 is represented with two meta-tuples:

$$(\mathrm{S}, *, \mathrm{MarkUp}, x_2 *) \in \mathit{availability}'$$
$$(x_2, >, 750) \in \mathit{comparison}$$

Let V_4 be an empty view describing the Virginia stores selling radar detectors (i.e., radar detectors are not available in Virginia):

$$\{(a) \mid (\exists b_1)(\exists b_2)\ \mathit{Store}(a, \mathrm{Virginia}, b_1)\ \wedge\ \mathit{Availability}(\mathrm{radar\_detector}, a, b_2)\}$$

V_4 is represented with two meta-tuples:

$$(\mathrm{E}, x_3 *, \mathrm{Virginia}, \sqcup) \in \mathit{store}'$$
$$(\mathrm{E}, \mathrm{radar\_detector}, x_3 *, \sqcup) \in \mathit{availability}'$$

3.3. Manipulation of Meta-information

Meta-relations are manipulated with their own product, selection and projection operations. We define these operations and review their properties, and then describe how queries are processed in the meta-database.

The product of two meta-relations, called *meta-product*, matches meta-tuples with the same property.

DEFINITION 1 *Assume that $r'(R')$ and $s'(S')$ are meta-relations that define views of R and S. The product of r' and s', denoted $r' \times s'$, is defined as follows. For every pair u and v of meta-tuples (having the same property p) from r' and s', respectively,*

$$u = (p, u_1, \ldots, u_m)$$
$$v = (p, v_1, \ldots, v_n)$$

$R' \times S'$ includes the meta-tuple:

$$w = (p, u_1, \ldots, u_m, v_1, \ldots, v_n)$$

For example, assume that the views with the same property "Virginia stores" and "televisions costing over 750" hold, respectively, over the relations *Store* and *Availability*, and assume a query that forms the product of these two relations. The property view "Virginia stores and televisions costing over 750" would hold over the answer.[6]

The selection from a meta-relation, called *meta-selection*, includes a simple condition that compares an attribute to a constant or one attribute to another. It selects a meta-tuple if the attributes used to specify the selection are projection attributes (i.e., are suffixed by $*$). This restriction guarantees that the information used for defining a property view is itself covered by the property.[7] In general, every such property view will continue to hold over the answer to the selection query. However, in two special cases, the meta-answer can be simplified. First, if the query selection condition is *implied* by the corresponding meta-tuple selection condition, then the meta-tuple condition may be cleared (i.e., $\sqcup$ is substituted). Intuitively, all the data retrieved by the query satisfy the meta-tuple restriction, so the restriction is no longer relevant. Second, if the query selection condition *contradicts* the meta-tuple selection condition, then the meta-tuple can be discarded. Intuitively, none of the data retrieved by the query satisfy the meta-tuple restriction, so the property view is no longer relevant.

For brevity, we define here only meta-selections with conditions that compare an attribute to a constant. The definition for conditions that compare two attributes is quite similar.

DEFINITION 2 *Assume that $r'(R')$ is a meta-relation that defines views of R. Let A_i denote an attribute of R' and let λ denote a primitive selection predicate of the kind $A_i\,\theta\,c$. The selection from r' by predicate λ, denoted $\sigma_\lambda(r')$, is defined as follows. Consider a meta-tuple u from r',*

$$u = (p, u_1, \ldots, u_i, \ldots, u_m)$$

and denote by μ the selection predicate expressed by u_i.[8]
(1) If u_i is suffixed by $*$, and $\lambda \Rightarrow \mu$, then $\sigma_\lambda(r')$ includes the meta-tuple:

$$w = (p, u_1, \ldots, *, \ldots, u_m)$$

(2) Otherwise, if u_i is suffixed by $*$, and λ and μ are not contradictory, then $\sigma_\lambda(r')$ includes the meta-tuple:

$$w = (p, u_1, \ldots, u_i, \ldots, u_m)$$

For example, assume that the property view "televisions costing over 750" holds over the relation *Availability*, and consider four selection queries: (1) "products costing over 1,000", (2) "products costing under 500", (3)"products costing over 500", and (4) "products costing under 1,000". In every case the original view continues to hold over the answer, but in two cases this meta-answer can be simplified. In the first case, because the answer includes *only* products over 750, the property view can be simplified to "televisions". In the second case, because the answer includes *no* products costing over 750, the property view can be discarded. In the latter two cases, because the answer includes *some* products costing over 750, the property view must remain unchanged.

The projection of a meta-relation, called *meta-projection*, removes a single attribute. It retains a meta-tuple only if the attribute to be removed is not a selection attribute (i.e., it is $\sqcup$). The meta-tuple is modified to remove the projection attribute. Intuitively, this guarantees that a property view is not unduly "broadened" by discarding a restriction.

DEFINITION 3 *Assume that $r'(R')$ is a meta-relation that defines views of R. Let A_i denote an attribute of R'. The projection of r' that removes the attribute A_i, denoted $\pi_{R'-A_i}(r')$, is defined as follows. For every meta-tuple u from r',*

$$u = (p, u_1, \ldots, u_m)$$

If u_i is $\sqcup$ (possibly suffixed by $*$), then $\pi_{R'-A_i}(r')$ includes the meta-tuple:

$$w = (p, u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_m)$$

For example, assume that the property views "all refrigerators" and "all RCA televisions" hold over the relation *Product*, and consider a query that eliminates the attribute *Manufacturer*. After the meta-projection — because it does not restrict the manufacturer — the first property view will still be "all refrigerators" and will hold over the answer. On the other hand, the second property view restricts the manufacturer, and will have to be discarded or else the meta-projection would broaden it to "all televisions".

3.4. Meta-Processing

These definitions were shown to be correct (Motro, 1989b), in the sense that the meta-product defines views that would be obtained by applying the product to the views defined in the original meta-relations; the meta-selection defines views that would be obtained by

applying the selection to the views defined in the original meta-relation; and the meta-projection defines views that would be obtained by applying the projection to the views defined in the original meta-relation. This correctness result is formally stated as follows.

Let d be an instance of this database, let u, v, and w be meta-tuples as in the preceding definitions, and let $u(d)$, $v(d)$, and $w(d)$ denote, respectively, the relations defined by the meta-tuples u, v and w. Then

$$\begin{array}{ll} \text{product}: & w(d) = u(d) \times v(d). \\ \text{selection}: & w(d) = \sigma_{\lambda} v(d). \\ \text{projection}: & w(d) = \pi_{R'-A_i}(v(d)). \end{array}$$

Also, each of the properties mentioned in this paper (soundness, completeness, emptiness and permissibility) can be shown to be inherited by these meta-operations. This inheritance result is stated as follows.

Let d be an instance of this database, let u, v, and w be meta-tuples as in the preceding definitions, and let $u(d)$, $v(d)$, and $w(d)$ denote, respectively, the relations defined by the meta-tuples u, v and w. Then

product: if $u(d)$ and $v(d)$ have the same property, then $w(d)$ has this property.
selection: if $u(d)$ has a property, then $w(d)$ has this property.
projection: if $u(d)$ has a property, then $w(d)$ has this property.

These two results may be summarized as follows. Assume a database $d(D)$ and a corresponding meta-database $d'(D')$ that concerns one specific property p. Let Q be a conjunctive query against $d(D)$, and let F be the relational algebra expression that implements Q. Let F' be the relational algebra expression obtained from F by substituting every reference to a relation scheme R with a reference to R'. F operates on the actual database relations to yield the answer $a(A)$, and F' operates on the meta-relations to yield the meta-relation $a'(A')$ whose tuples define views of A.[9] Then, the views in a' inherit the property p.

This result guarantees that meta-tuples in a' are views of A that have the property p. However, some meta-tuples may still share variables with meta-tuples outside a', and thus define views that are not expressible entirely within the scheme A'. Such views are avoided if F' is modified so that all products are performed first, and their result is pruned to retain only those meta-tuples that do not share variables with other meta-tuples. Also, to minimize "losses" of property views, it is advantageous to perform selections before projections (Motro, 1989b). Altogether, F' is transformed to a sequence of products, followed by selections, and ending with projections. This simple strategy for implementing conjunctive queries is not necessarily the most efficient. However, we note that efficiency is not so essential for meta-processing, because meta-relations are relatively small. For the "regular" processing, where efficiency is essential, a different strategy may be implemented. The Appendix shows that the complexity of this naive method of meta-processing is $O(n^{m+3})$, where n indicates the size of the meta-database and the constant m indicates the size of the given query.

The result guarantees that the method for generating integrity constraints is *sound*, but it does not guarantee that it is *complete*. That is, this method generates views of the result that indeed have the property, but does not necessarily generate all such views. A method that

would guarantee completeness would undoubtedly be of a different complexity altogether. However, for our purpose here, to provide knowledge about database answers, completeness is not an absolute requirement. Yet, several simple refinements were developed, that generate additional desirable views (Motro, 1989b). Thus, the meta-operations implemented in Panorama are actually more involved than those defined above.

3.5. Example

With the database described in Figure 2 and the property views V_1, V_2, V_3, and V_4, consider this query about products costing over 1,000 and the stores selling them:

$$\{(a_1, a_2, a_3, a_4) \mid (\exists b)\ Store(a_3, a_4, b)\ \wedge\ Availability(a_1, a_3, a_2)\ \wedge\ (a_2 > 1,000)\}$$

The meta-processing of this query is accomplished with a meta-product of *Store'* and *Availability'*, then a meta-selection of *Name=Store*, and finally a meta-projection on *Product, Price, Store, Location*. The meta-answer includes two meta-tuples:

$$(\mathrm{S}, *, x*, \mathrm{MarkUp}, \sqcup)$$
$$(\mathrm{E}, \mathrm{radar\_detector}, \sqcup, \sqcup, \mathrm{Virginia})$$

These tuples describe, respectively, a sound view of the names and prices of products that are sold by the MarkUp Company, and an empty view of the Virginia stores that sell radar detectors.

4. The System

Panorama[10] is an experimental system that implements the concepts discussed in this paper. Ideally, meta-processing should be integrated into the database system. Instead, our implementation is a front-end to INGRES (Sun Microsystems, 1987), a commercially available relational database management system. Meta-relations are implemented as standard relations. Thus, they are similar to other auxiliary system tables that store indices, permissions, etc.

4.1. Language Extensions

Panorama recognizes a restricted form of the query statement

retrieve (*attributes*)
where *qualification*

where the qualification is a conjunction of simple comparisons. A simple comparison has the form $A\ \theta\ B$, where A and B are attributes or constants, and θ is from the set $\{=, \neq, >, \geq, <, \leq\}$. This statement corresponds to the conjunctive queries defined in Section 3.2.

In addition, Panorama extends the query language with three statements to manipulate and query the meta-database. To add a new property view to the meta-database, the following statement is provided:

append property p(*attributes*)
where *qualification*

Note that users are allowed to invent new properties. To delete a property view from the meta-database, the following statement is provided:

delete property p(*attributes*)
where *qualification*

These statements may require **range** declarations. To list views with a particular property, the following statement is provided:

print property p

4.2. System Architecture

As Panorama is an interface to INGRES, users are assumed familiar with its query language, QUEL, and with its standard user interface, Terminal Monitor. Upon starting Panorama the user is placed in Panorama's user interface, which emulates Terminal Monitor. All user input is first parsed by Panorama, and then processed as follows:

- Input recognized as a request to query or manipulate the meta-database (i.e., **append property**, **delete property** and **print property**) is executed by Panorama. In executing these requests, Panorama issues QUEL commands to access and manipulate the meta-database as necessary. The answer computed by Panorama is displayed to the user.
- Input recognized as a conjunctive query is passed to INGRES, but is also executed by Panorama. INGRES processes the query in the usual way, returning an answer that is then displayed to the user by the Panorama user interface. Panorama processes the query in the meta-database (again, using QUEL), deriving a meta-answer that is displayed to the user alongside the usual answer.
- All other input is passed unchanged to INGRES. All output (e.g., answers, error messages, etc.) is displayed to the user by the Panorama user interface.

The meta-answer that accompanies each answer is translated back to view definitions:

property p(*attributes*)
where *qualification*

As each view in the meta-answer ranges over a single relation (the answer), **range** declarations are not necessary.

This architecture provides the impression of a simple extension of the underlying database system. Also, all user-system interaction is in QUEL-like structures, and the internal representation of property views is transparent to users.

4.3. Example

Consider again the database from Figure 2. The property views V_1, V_2, V_3, and V_4 would be declared to Panorama as follows:

1. **range of** p **is** product
   range of a **is** availability
   append property complete (p.manufacturer)
   where p.name = a.product
   and a.store = "MarkUp"

2. **range of** s **is** store
   append property sound (s.name, s.location)

3. **range of** a **is** availability
   append property sound (a.product, a.price)
   where a.store = "MarkUp"
   and a.price $>$ 750

4. **range of** s **is** store
   range of a **is** availability
   append property empty (s.name)
   where s.name = a.store
   and s.location = "Virginia"
   and a.product = "radar_detector"

Consider now the previous query regarding stores selling products costing over 1,000:

range of s **is** store
range of a **is** availability
retrieve (a.product, a.price, a.store, s.location)
where a.store = s.name
and a.price $>$ 1,000

The answer to this query is a four attribute relation *(Product, Price, Store, Location)*. The accompanying meta-answer consists of two property views:

1. **property** sound (product, price)
   where store = "MarkUp"

2. **property** empty (*)
   where product = "radar_detector"
   and location = "Virginia"

The first statement guarantees to the user that the prices listed for products carried by the MarkUp Company are all sound. The second statement reminds the user that radar detectors are not available in Virginia.

5. Discussion

In this final section we examine our work with regard to several important effectiveness criteria, we point out various issues that require further attention, and we suggest alternative research directions.

5.1. Effectiveness

The effectiveness of Panorama, a query system that volunteers information that qualifies and explains the data it retrieves, may be evaluated by considering four key criteria: the *relevance* and *completeness* of the volunteered information, the *expressivity* of the language for specifying it, and the *efficiency* of computing it. The success of Panorama in meeting these criteria is discussed below.

5.1.1. Relevance

As with other kinds of voluntary information, the *relevance* of the information to the querying user is a crucial issue. Whereas users specify clearly the data they need, it is not as clear which characterizations of this data would be of interest to them.

For example, a user asking about the televisions carried by the MarkUp Company may be interested in the fact that the prices quoted for RCA products are sound, but probably not in the fact that the store does not carry radar detectors.

One possible solution to the relevance issue is to establish the concepts that are relevant to users, by means of user-system dialogues. Alternatively, "user profiles" could be used to specify the concepts in which users are interested. These profiles would be provided by the users, or possibly inferred by the system over a period of time. Such solutions have the potential of accuracy and effectiveness, but would also be involved and demanding.

Panorama's approach is considerably simpler: a property is judged to be *relevant*, if it is entirely expressible in the attributes sought by the user. For example, the information that radar detectors are not sold in Virginia is considered relevant to queries that inquire about both product names and store locations. This strategy is implemented by the very meta-processing algorithm described in Section 3.3, which at every phase discards views with outside references.

A possible research direction is to *rank* properties with respect to their relevance to individual queries. This will enable the system to control the meta-answers it provides. Initially, the system would present only the properties ranked as most relevant. If the user is still interested in additional properties, then, gradually, less relevant properties will be presented. This assures that the system's judgement of relevance, an inherently imperfect task, will not prevent the user from receiving certain properties.

5.1.2. Completeness

The second criterion of effectiveness considers whether the system finds *all* the properties that apply to its answers. In other words, whether Panorama's meta-answers are complete. Clearly, completeness interacts with relevance: meta-answers are complete if they include all the information judged to be relevant.[11]

It is interesting to contrast answer completeness with meta-answer completeness. An answer (or, more generally, a view extension) is complete, if it includes all the occurrences that exist in reality. A meta-answer is complete, if it includes all the properties that hold over the answer.

Our treatment of view completeness assumed that this property must be *declared*. That is, a system is not capable of concluding that a view contains all the occurrences that exist, and therefore must rely on declarations of completeness. Our treatment of meta-answer completeness is similar: a system is not capable of discovering all the properties that hold over its answers, and therefore must limit itself to those properties that can be concluded from information that was provided to it.

Thus, the completeness criterion is whether the meta-answers of Panorama include all the relevant properties that can be inferred from the available meta-information; i.e., it refers to the completeness of the *inference process*.

In Section 3.3 we observed that our inference process is sound, but not necessarily complete: some relevant views may be missing from the meta-answer. Also, we mentioned that the meta-operations implemented in Panorama incorporate various refinements, intended to increase completeness. However, such refinements also increase complexity.

Such incompleteness need not be considered a critical flaw, because a sense of incompleteness about such answers would persist, even if the system did an exhaustive job of inference. The reason is that users are usually well aware that meta-answers reflect only the *available* knowledge, and that this knowledge is often incomplete. In other words, the incompleteness of the inference process (perhaps not assumed by users) is "absorbed" in the incompleteness of the available knowledge (probably assumed by users).

For example, when Panorama lists the products carried by the MarkUp Company and notes that the set of manufacturers is guaranteed to be complete, users might infer that no other relevant information about the answer is available to the system, but they would probably not infer that every other statement about the answer is necessarily invalid. Therefore, users already regard the information in meta-answers as incomplete.

An exception to this observation is that upon receiving meta-information, users might conclude that this information is complete *with respect to that specific subject*. For example, if the system notes that radar detectors are not available in Virginia, users might conclude that all other products are available in Virginia. Here, the incompleteness of the available knowledge or of the inference process is more critical.

A possible direction for research is to consider how to declare *properties of meta-information*, and then infer *properties of meta-answers*. For example, if particular views are stated as the *only* views with a certain property (i.e., meta-completeness), then, given a complete inference process, the system could guarantee that a meta-answer is complete with respect to that property.

Indeed, this approach is already practiced by access control mechanisms, where all meta-information describing permissibility is assumed complete: the set of views describing the data permitted to users is assumed to be exhaustive, and information that cannot be derived from these views is not permitted. With a complete inference process, the system could guarantee that data not inferred as permissible are indeed impermissible.

5.1.3. Expressivity

Expressivity is concerned with the power of the language for specifying database properties. In Panorama, meta-information is specified with conjunctive views (i.e., product-selection-projection expressions). This language also describes the kind of queries for which meta-answers are computed.[12] As mentioned earlier, while this language is a strict subset of the relational calculus (or algebra), it is a powerful subset.

Probably the most valuable extension to this language would be *aggregate expressions*, especially in queries. The main issue here is whether a property is inherited over aggregations. For some properties, for example permissibility, the answer is simple: an aggregate view of permissible views is permissible. However, consider the property of soundness, whereby a view is sound if its evaluation in the database is contained in its evaluation in the real world. An aggregate view of sound views may yield a value which is not sound (i.e., different from the value in the real world). As an example, consider the view "prices of RCA products" and the query "average price of RCA products". Even if the view is sound (i.e., the database stores real world prices for RCA products), the answer to the query may be unsound (i.e., the average database price for RCA products is different from the average real world price of RCA products). Clearly, this is due to the possibility of the view being incomplete. This and other issues relating to properties of aggregate views require additional research.

5.1.4. Efficiency

Efficiency is concerned with the cost of deriving meta-answers. While the volume of meta-information is usually much smaller than the volume of extensional information, its processing may involve more complex algorithms. Efficiency often conflicts with completeness, as attempts to satisfy completeness often add to the complexity of the method.

The duality of meta-processing and regular query processing implies that the cost of generating meta-answers is essentially the cost of processing the query on the meta-relations. As mentioned earlier (and shown in detail in the Appendix), the cost of meta-processing can be expressed as a polynomial in the size of the meta-database. Thus, Panorama performs quite well in this respect.

5.2. Alternative Approach to View Inferencing

Our meta-processing solution to view inferencing is essentially algebraic. The view inference problem can also be treated in the framework of logic-based (deductive) databases. Within this framework relations and views are modeled, respectively, with *extensional* predicates (predicates defined by stored facts) and *intensional* predicates (predicates defined by rules) (Ullman, 1988).

The approach we sketch here is adapted from the work of Chakravarthy, Grant and Minker (1988, 1990), where the subject is the optimization of query processing by considering integrity constraints. The authors show how integrity constraints, which are assertions on the database, can be *compiled* into equivalent assertions, called *residues*, that are attached to individual extensional predicates. When a query is submitted, these residues are *reduced* to assertions that apply to the query. The query optimizer then uses these query residues to transform the original query into equivalent queries that can be processed faster in the database.

We observe that property views are also assertions on the database, and hence a similar method can be applied to these assertions as well. The residues attached to each extensional predicate correspond to views of the extensional predicate that have the property, and the residues subsequently derived for each query correspond to views of the answer that have the property. The use of logic-based methods to store database properties and the properties of answers is a subject of our ongoing research.

5.3. Discovering Properties of Answers in the Data

A basic premise of our work has been that all meta-information is asserted by humans; i.e., that properties of the database are knowledge which is *declared*. Such knowledge may be regarded as part of the database *intension*: "permanent" information to which every *extension* of the database must conform.[13]

Recently, there has been much interest in the *discovery* of knowledge in databases; i.e., searching the extension of databases for patterns and regularities of behavior (Piatetsky-Shapiro and Frawley, 1989; Piatetsky-Shapiro, 1991). Discovered knowledge may not have the "permanence" of declared knowledge (i.e., it may not hold over other extensions of this database). Nevertheless, it has the potential to become valuable source of information regarding the database. Research in knowledge discovery is closely related to issues of machine learning (Michalski, 1983).

An interesting research direction is to extend systems such as Panorama with a mechanism that annotates answers with *discovered* properties. So far, research in knowledge discovery has focused on discovering patterns and regularities in the data, information which may be stated as integrity constraints (i.e., the property of emptiness). The possibility of discovering in the data other kinds of properties, namely soundness and completeness, is sketched below.

Essentially, the approach is to take advantage of *repetition* of information. That is, when a particular fact is repeated (independently), it will be taken as evidence that this fact is sound; and when a particular set of facts is repeated (independently), it will be taken as evidence that this set is complete. As an example, when a *functional dependency* $A \rightarrow B$ is

discovered in a database and A is known to be sound, then the view AB will be considered *evidently sound*. Similarly, when a multivalued dependency $A \twoheadrightarrow B$ is discovered in a database and A is known to be complete, then the view AB will be considered *evidently complete*.

5.4. Summary

In this paper we proposed that query systems be extended with the capability of annotating their answers with properties of the answers. Such answers would be computed from properties that have been stated on the entire database. We defined the view inference problem as the general framework for representing database properties and for computing properties of individual answers. We described a particular method, called meta-processing, for solving the view inference problem and analyzed its complexity, and we described a prototype software system, called Panorama, that implements this method. Finally, we evaluated the effectiveness of our approach, and we proposed additional research directions. The advantages of our method 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. (4) It can be implemented *externally* with any commercial relational database system.

Acknowledgments

This work was supported in part by NSF Grants No. IRI-8609912 and IRI-9007106. The implementation of Panorama was done by Chua Leng and Ockkeun Lee. Thanks are also due to Alex Brodsky and Sean X. Wang for their valuable comments.

Appendix

Complexity of Meta-Processing

To assess the complexity of meta-processing, we shall assume a conjunctive query that involves

1. A meta-product of m_1 meta-relations.
2. A meta-selection condition that conjoins m_2 comparisons, each comparing an attribute with a constant or one attribute with another.
3. A meta-projection that removes m_3 attributes.

m_1, m_2 and m_3 are *constants* for the given query. Let m denote the largest of these constants. The data-complexity of meta-processing is only a function of the size of the meta-database. Let n denote the cardinality of the largest meta-relation (including $comparison$).

We shall consider the meta-operations as defined in Section 3.3, and the naive processing strategy that performs a sequence of meta-products, followed by a sequence of meta-selections (each involving a single comparison), and followed by a sequence of meta-projections (each removing a single attribute).[14] A meta-product matches every meta-tuple of one meta-relation with every meta-tuple of another meta-relation. Hence, the meta-products will require at most $O(n^m)$ operations. The input of the meta-selections has cardinality not exceeding n^m. A meta-selection compares the selection predicate of the query with every meta-tuple, so there are $m \cdot n^m$ such comparisons. As we shall see, each comparison requires at most $O(n^3)$ operations. Hence, the meta-selections will require at most $O(n^{m+3})$ operations. Finally, assuming that the meta-selections remove no meta-tuples, the input of the meta-projections has cardinality not exceeding n^m. Each meta-projection performs a simple check on every meta-tuple. Hence, the meta-projections will require at most $O(n^m)$ operations. Altogether, the complexity of meta-processing is $O(n^{m+3})$.

It remains to show that each comparison in the meta-selection requires at most $O(n^3)$ operations. A meta-selection compares the selection predicate of the query, λ, with the corresponding selection predicate in every meta-tuple, μ, to determine whether

1. λ and μ contradict (the meta-tuple is discarded),
2. $\lambda \Rightarrow \mu$ (the meta-tuple condition is cleared and the meta-tuple is retained).
3. Neither (1) nor (2) holds (the meta-tuple is retained unchanged).

The discussion of the complexity of the meta-selection is simplified if we assume that meta-tuples are tuples that are composed entirely of variables that are all different. A restriction previously expressed with a constant is now expressed with an entry in $comparison$ that equates the corresponding variable to the constant. A restriction previously expressed with multiple occurrences of the same variable is now expressed with entries in $comparison$ that equate the corresponding variables. (A non-restriction previously expressed with a blank is now expressed with a variable that does not appear anywhere else.)

The predicate λ has the form $A\ \theta\ c$ or $A\ \theta\ B$ and is translated to an expression in which A and B are replaced by the corresponding meta-tuple variables. The predicate μ is the conjunction of the triplets in the relation $comparison$.

First, to determine whether λ contradicts μ, the following algorithm can be used.

1. For $\lambda\ \wedge\ \mu$ construct the following graph. Every variable or constant is represented by a node. Every $\geq$ or $\leq$ comparison is translated to a directed edge between the corresponding nodes. Every $=$ comparison is translated to a pair of opposite directed edges between the corresponding nodes.
2. Detect all cycles in this graph. The nodes of each cycle must be related through equality. Hence, if the nodes of some $\neq$ comparison appear in the same cycle, then λ and μ contradict.
3. Otherwise, construct a new graph as follows. The nodes are as before, except that the nodes of every maximal cycle are merged into a single node. Every $\geq$, $>$, $\leq$, or $<$

comparison is translated to a directed edge between the corresponding nodes. Nodes of constants are interconnected with directed edges that express greater-than relationships.

4. If there is a cycle in this graph, then λ and μ contradict, because a cycle now indicates an element that is strictly greater than itself.

It can be shown that this algorithm is sound and complete: it detects all the existing contradictions in the given expression.

Second, $\lambda \Rightarrow \mu$ if and only if every comparison in μ is implied by λ. Let $x\ \theta\ y$ denote a comparison in μ, where x is a variable and y is a variable or a constant. There are six different cases, each requiring a fixed number of operations.

1. $x = y$. The only λ formula that implies this comparison is $x = y$.

2. $x \neq y$. The λ formulas that imply this comparison are $x > y$, $x < y$ or $x \neq y$. If y is a constant, then also $x > y'$, where y' is any constant such that $y' \geq y$, and $x < y'$, where y' is any constant such that $y' \leq y$.

3. $x \geq y$. The λ formulas that imply this comparison are $x \geq y$, $x > y$ or $x = y$. If y is a constant, then also $x \geq y'$ and $x > y'$, where y' is any constant $y' \geq y$.

4. $x > y$. The λ formula that implies this condition is $x > y$. If y is a constant, then also $x > y'$, where y' is any constant $y' \geq y$.

5. $x \leq y$. The λ formulas that imply this comparison are $x \leq y$, $x < y$ or $x = y$. If y is a constant, then also $x \leq y'$ and $x < y'$, where y' is any constant such that $y' \leq y$.

6. $x < y$. The λ formula that implies this condition is $x < y$. If y is a constant, then also $x < y'$, where y' is any constant such that $y' \leq y$.

With respect to complexity, the dominant steps in these two algorithms is the detection of cycles. This process can be done by applying a transitive closure algorithm to the given graph. Such an algorithm requires at most $O(k^3)$ operations, where k is the number of nodes in the given graph (Aho, Hopcroft and Ullman, 1974). Note that k is at most twice the cardinality of $comparison$.

Notes

1. While these definitions avoid any ordering among attributes of schemes or among values of tuples, our notation for schemes and tuples will indicate an ordering.
2. In particular, a view can be simply a relation. A view may also be an expression that involves, recursively, the names of other views.
3. Our terminology here is slightly different from that used in (Motro, 1989b).
4. The kind of views that may be declared open (incomplete) (Gottlob and Zicari, 1988) is significantly less general than the kind of views considered in this paper.
5. In Section 5.2, we shall sketch briefly an alternative approach for implementing this framework, which is based on logic.
6. As is often the case, prior to selection this product has unintuitive semantics.

7. For example, a sound view is defined using sound information only.
8. If u_i is a constant c, then μ is $u_i = c$; if u_i is a variable, then μ is the condition that binds this variable to other variables or constants; if u_i is blank, then μ is true.
9. F' may be considered a representation of a meta-query Q' that was derived from the user query Q.
10. *Pan* indicates completeness, inclusiveness; *rama* stands for Relational Answers and Meta Answers.
11. Therefore, the approach of ranking properties with respect to their relevance will prove safer with regard to completeness.
12. Similar expressivity is provided by the alternative view inferencing solution discussed in Section 5.2.
13. Indeed, Panorama does not address the issue of whether a declared property holds. For some properties, e.g., emptiness, "automatic" verification is possible; for other properties, e.g., soundness or completeness, verification requires human involvement.
14. Clearly, there are many opportunities for optimizing this strategy; e.g., perform all selections "simultaneously", perform all projections "simultaneously", perform selections and projections in the same "pass", etc.

References

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*. Reading, Massachusetts: Addison-Wesley.

Chakravarthy, U. S., Grant, J., and Minker, J. (1988). Foundations of semantic query optimization for deductive databases. In J. Minker (editor), *Foundations of Deductive Databases and Logic Programming*, pages 243–273. Los Altos, California: Morgan Kaufmann.

Chakravarthy, U. S., Grant, J., and Minker, J. (1990). Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207.

Corella, F., Kaplan, S. J., Wiederhold, G., and Yesil, L. (1984). Cooperative responses to Boolean queries. In *Proceedings of the IEEE Computer Society First International Conference on Data Engineering* (Los Angeles, California, April 24–27), pages 77–85.

Cuppens, F., and Demolombe, R. (1988). Cooperative answering: A methodology to provide intelligent access to databases. In *Proceedings of the Second International Conference on Expert Database Systems* (Tysons Corner, Virginia, April 25–27), pages 333–353.

Date, C. J. (1990). *An Introduction to Database Systems, Volume I (Fifth Edition)*. Reading, Massachusetts: Addison Wesley.

Gaasterland, T. (1992). *Generating Cooperative Answers in Deductive Databases*. PhD thesis, Department of Computer Science, University of Maryland.

Gaasterland, T., Godfrey, P., and Minker, J. (1992). Relaxation as a platform for cooperative answering. *Journal of Intelligent Information Systems*, 1(3/4):293–321.

Gal, A. (1988). *Cooperative Responses in Deductive Databases*. PhD thesis, Department of Computer Science, University of Maryland.

Gottlob, G., and Zicari, R. (1988). Closed world assumption opened through null values. In *Proceedings of the Fourteenth International Conference on Very Large Data Bases* (Los Angeles, California, August 25–28), pages 50–61.

Griffiths, P. P., and Wade, B. W. (1976). An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1(3):242–255.

Kaplan, S. J. (1982). Cooperative responses from a portable natural language query system. *Artificial Intelligence*, 19(2):165–187.

Korth, H. F., and Silberschatz, A. (1986). *Database System Concepts*. New York, New York: McGraw-Hill.

Maier, D. (1983). *The Theory of Relational Databases*. Rockville, Maryland: Computer Science Press.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In R. S. Michalski, T. M. Mitchell, and J. Carbonell (editors), *Machine Learning: An Artificial Intelligence Approach*. Los Altos, California: Morgan Kaufmann.

Motro, A. (1986). SEAVE: A mechanism for verifying user presuppositions in query systems. *ACM Transactions on Office Information Systems*, 4(4):312–330.

Motro, A. (1989a). 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), pages 237–246.

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

Motro, A. (1990a). FLEX: A tolerant and cooperative user interface to databases. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):231–246.

Motro, A. (1990b). Relational meta-answers. In Z. Ras and M. Zemankova (editors), *Intelligent Systems: State of the Art and Future Directions*, pages 371–386. Chichester, England: Ellis Horwood.

Motro, A. (1992). A unified model for security and integrity in relational databases. *Journal of Computer Security*, 1(2):189–213.

Motro, A. (1994). Intensional answers to database queries. *IEEE Transactions on Knowledge and Data Engineering*, 6(3):444–454.

Piatetsky-Shapiro, G. (editor) (1991). *Proceedings of AAAI-91 Workshop on Knowledge Discovery in Databases* (Anaheim, California, July 14–15).

Piatetsky-Shapiro, G., and Frawley, W. (editors) (1989). *Proceedings of IJCAI-89 Workshop on Knowledge Discovery in Databases* (Detroit, Michigan, August 20).

Reiter, R. (1978). On closed world data bases. In *Logic and Databases*, pages 55–76. New York, New York: Plenum Press.

Stonebraker, M., and Wong, E. (1974). Access control in a relational database management system by query modification. In *Proceedings of ACM Annual Conference* (San Diego, California, November), pages 180–186.

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

Tsichritzis, D. C., and Lochovsky, F. H. (1982). *Data Models*. Englewood Cllifs, New Jersey: Prentice Hall.

Ullman, J. D. (1982). *Principles of Database Systems*. Rockville, Maryland: Computer Science Press.

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

Zloof, M. (1977). Query-by-Example: A database language. *IBM Systems Journal*, 16(4):324–343.