Annotating Answers with Their Properties

Amihai Motro

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 provide some *abstract* characterization of the answer. This paper describes a user interface to relational databases 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.

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 provide some *abstract* characterization of the answer.

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:

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

The first statement grants assurance that the answer is both sound (all information provided is accurate) and complete (there are no other bookstores in Washington). The second statement states that the answer may be incomplete, while the third statement limits its soundness to 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 article, we describe a user interface to relational databases that similarly annotates its answers with their properties. The process does not involve any "understanding" or "intelligence". 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 the user along with each answer issued.

This work was supported in part by NSF Grant No. IRI-9007106.

The system that we describe is based on earlier theoretical results described mostly in [2] and [3]. We review some of those results, and then briefly describe a prototype system that has been completed recently.

2 Deriving Meta-Answers

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 kinds of statements involved.

A *property* is any label that can be attached to a database view. Several examples of properties are discussed below. A view known to include *all* pertinent information would be tagged with the property *complete*. Similarly, a view known to include only information that has been verified would be tagged with the property *sound*. A view that a particular user, say Smith, is allowed to access would be tagged with the property *access/Smith*. Consider the integrity constraint $\alpha(x_1,\ldots,x_n) \Rightarrow \beta(x_1,\ldots,x_n)$. It may be stated as $\{(x_1,\ldots,x_n) \mid \alpha(x_1,\ldots,x_n) \wedge \neg\beta(x_1,\ldots,x_n)\} = \emptyset$. Thus, integrity constraints may be stated as views that are tagged with the property *empty*. Properties such as soundness and completeness may be refined to include the name of the party providing the guarantee, or timestamps indicating the date on which the guarantee was provided, or when it might expire. Properties describing access permissions may be refined to include the kind of access: read, write, etc.

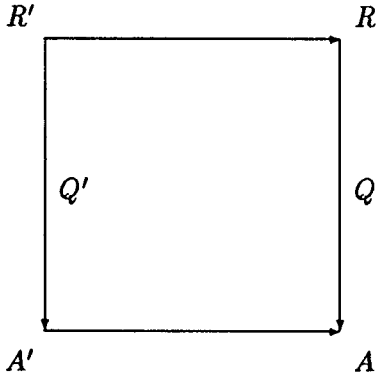
Formally, we assume a set of properties $\mathcal{P}$, a set of view definitions $\mathcal{V}$, and a set of pairs (v,p), each asserting that view v has property p. A view with a property will be referred to as a *property view*.

Given a query Q, we would like to discover the properties that are applicable to its answer; e.g., is the answer sound? is it complete? does it satisfy any constraints? Moreover, we would like to discover properties that are applicable to *views* of the answer; e.g., which part of the answer is sound? which part is complete?

This inference problem was defined in [2, 3], where an algebraic approach was advocated. All views and queries are assumed to be expressible as conjunctive relational calculus expressions (or, alternatively, as relational algebra expressions with product, projection, and selection with conjunctive predicates). We review this approach below.

For each database relation R, a *meta-relation* R' is defined, whose scheme is identical to the scheme of R. The definitions of property views are then stored as tuples in these meta-relations, in a representation that recalls the representation of QBE [4] queries in skeleton tables. Hence, every database has a structurally identical counterpart, called the meta-database, which stores properties of the database.

The relational algebra operations of selection, projection and product are then extended to operate on the meta-database. When a query is presented to the database system it is performed *both* on the actual database, resulting in an answer (a relation), and on the meta-database, resulting in a meta-answer (a meta-relation). The meta-answer defines views of the answer that possess particular properties. This approach is illustrated by the commutative diagram shown below. The horizontal lines describe the relationships between meta-relations and relations, and the vertical lines describe query processing and meta-processing.



To illustrate the representation, assume a simple bibliographic database with the following scheme:

ARTICLE = (TITLE, AUTHOR, JOURNAL, YEAR, PAGES)
JOURNAL = (NAME, PUBLISHER)

Let ARTICLE' and JOURNAL' denote the corresponding meta-relations. Consider this view on the title and author of articles published in ACM journals:

$$\{a_1, a_2 \mid (\exists b_1)(\exists b_2)(\exists b_3)$$
$$(a_1, a_2, b_1, b_2, b_3) \in \text{ARTICLE}$$
$$\wedge\ (b_1, \text{ACM}) \in \text{JOURNAL}\}$$

This view is represented by two meta-tuples:

$$(*, *, x_1, \sqcup, \sqcup) \in \text{ARTICLE}'$$
$$(x_1, \text{ACM}) \in \text{JOURNAL}'$$

Note that this brief example does not address other fine points of the representation, such as how specific properties are associated with views, or how view formulas such as (YEAR < 1980) are stored.

To illustrate the processing, consider a request to list the title and journal of every article by Charla Tan, that appeared in 1991 in ACM journals. This query could be processed in the database by a product of ARTICLE and JOURNAL, followed by a selection of AUTHOR="Charla Tan", PUBLISHER="ACM" and YEAR="1991", followed by a projection on TITLE and JOURNAL. Its answer A will have two columns with titles and journal names. A dual query will be processed in the meta-database, using the extended algebraic operations. Its answer A' will also have two columns. The tuples of A' will describe views of the answer A that have specific properties. For example, $(*, \sqcup)$ would indicate that the projection of the answer on TITLE possesses a property. Similarly, $(*, \text{TODS})$ would indicate that the selection of JOURNAL="TODS" from the answer possesses a property. (Note that the actual representation used indicates the particular property possessed by each view.)

The precise definitions of the extended operations and a discussion of their correctness, along with more detailed examples may be found in [2, 3].

3 Panorama

Panorama is an experimental system that implements the concepts discussed in this article. Ideally, meta-processing should be integrated into the database system. Instead, our implementation is a front-end to a commercially available relational database management system (INGRES). Meta-relations are implemented as standard relations. Thus, they are similar to other auxiliary system tables that store indices, permissions, etc. 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 $\{=, \neq, >, \geq, <, \leq\}$. In addition, Panorama extends the query language with three statements to retrieve and manipulate 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

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 Conclusion

Consider, for example, a large library database, with several views (subsets of the collection) guaranteed to be sound, other views guaranteed to be complete, and yet other views guaranteed to be empty (constraints). A user submitting a query to retrieve the articles (title, journal, year and page numbers) authored by Charla Tan after 1968, will receive a table of articles, annotated by information such as:

1. **property** complete (*)
   where journal="TSE"
2. **property** sound (title, journal, year)
   where year > 1970
3. **property** empty (*)
   where journal="TKDE" **and** year < 1989

The first statement guarantees the user that the information delivered is complete with respect to articles that appeared in TSE. The second statement guarantees that for articles printed after 1970 all the information except the page numbers is sound. The third statement notifies the user that no articles could have appeared in TKDE prior to 1989. Clearly, such information may prove valuable to the user in assessing the answer received.

Implementation of the prototype system that we described was completed recently. Research is continuing in several directions. One direction being explored is the possibility of using logic, rather than algebra, as the framework for representing and manipulating property views. In this framework, views are represented as logic clauses, and the problem of discovering property views of answers is related to the problem of finding query residues [1]. Another important issue is the filtering of meta-answers to control the communication of property views to users, presenting first the information that is judged most relevant to the query.

References

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

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

[3] A. Motro. Relational meta-answers. In Z. Ras and M. Zemankova, editors, *Intelligent Systems: State of the Art and Future Directions*, chapter 15, pages 371–386, Ellis Horwood, Chichester, England, 1990.

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