# Answering Queries Using Templates With Binding Patterns (Extended Abstract)

Anand Rajaraman
Yehoshua Sagiv
Jeffrey D. Ullman
*Department of Computer Science*
*Stanford University*

*ABSTRACT*

When integrating heterogeneous information resources, it is often the case that the source is rather limited in the kinds of queries it can answer. If a query is asked of the entire system, we have a new kind of optimization problem, in which we must try to express the given query in terms of the limited query templates that this source can answer. For the case of conjunctive queries, we show how to decide with a nondeterministic polynomial-time algorithm whether the given query can be answered. We then extend our results to allow arithmetic comparisons in the given query and in the templates.

## I. Motivation

A data-integration system such as Tsimmis (Papakonstantinou, Garcia, and Widom [1994], Chawathe et al. [1994]) translates information sources of arbitrary type into a common data model and language. If a source is an SQL database, then its interface with the Tsimmis system is fairly clear: one can ask it SQL queries over its database schema and nothing else. However, the source may be radically different from an SQL database; it could be a collection of ASCII documents, a spreadsheet file, and so on.

In these situations, it is necessary to interface the source with the common language and model. We not only need to relate terms of the global model to terms that appear at the various sources (which we must do even if the sources are SQL databases). We must also describe how queries in the global query language are carried out at each source.

The approach taken by Tsimmis is to use a *translator generator*. The input to the generator is a list of rules each consisting of:

*i*) A *query template*, which is a parametrized query in the global language. Assigning values to the parameters of the template results in a query; for example, the parametrized template `Salary = $1` might be instantiated to `Salary = 10000`. Thus, a query template can describe an infinite number of queries.

*ii*) A sequence of operations that must be carried out to answer any query obtained by instantiating the template.

**Example 1:** Suppose the information source is a genealogy, and the only two query forms it can answer are:

1. Given any individual $C$, find $C$'s parents.
2. Find all the individuals who have parents specified by the information source.

There are various reasons why the source might be able to respond to only a limited set of query forms. For example, it could be that the source is a conventional relation with an index on children, so it is easy to ask question (1) but hard to ask the inverse question "find the children of a given individual." The same index supports question (2), although not cheaply. As another example, some bibliographic search sources require that at least

Y. Sagiv's permanent address: Dept. of CS, Hebrew Univ., Jerusalem, Israel. Work supported by NSF grant IRI–92–23405, BSF grant 92–00360, and USAF contract F33615–93–1–1339.

one argument (e.g., name of author or title of book) be bound.

Now, let us see how we could answer some queries that are not of the form (1) or (2). First consider query "Find the grandparents of individual $a$." This query is answered by:

*i*) Find the set $P$ of parents of $a$, using query (1).

*ii*) For each individual $p$ in set $P$, use query (1) to find the parents of $p$.

*iii*) The answer is the union of all the individuals found in step (*ii*).

Figure 1 suggests the approach to answering this query.

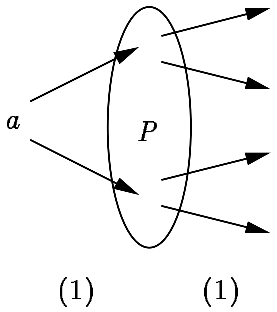


**Fig. 1.** Composing *parent* with itself.

Next, consider the query "Find the grandchildren of individual $g$." This query cannot be answered at all if we only have query (1) to work with. However, with the help of query (2) (find all individuals), we can answer this query as follows:

*i*) Use (2) to find all individuals.

*ii*) Use (1) to find the parents of the individuals found in (*i*).

*iii*) Use (1) to find the parents of the individuals found in (*ii*).

*iv*) Find those individuals from (*iii*) such that $g$ is one of their grandparents.

This solution is suggested in Fig. 2. Evidently, the solution is much more expensive than we would like, since it examines the entire genealogy instead of proceeding from the given individual. However, with queries (1) and (2) as our only information-gathering options, there is no more efficient way to answer the query. □

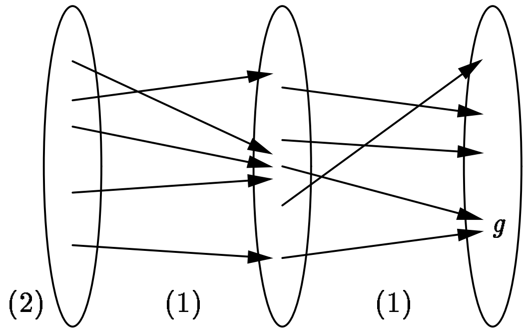


**Fig. 2.** Solving the grandchild query.

## II. A Formal Model

Example 1 suggests the following way to model a set of given query forms and queries that may or may not be answerable from these forms.

1. We assume that there are certain predicates about which queries are posed. These predicates correspond to concepts that are in the shared or global model for the information to be integrated. In Example 1, $parent(C, P)$ would be such a predicate.

2. We assume that there are certain given query templates, which we call *views* for consistency with certain other works to be discussed in Section III. A view consists of a "head" and a "body." The *head* consists of:

   a) A predicate denoting the view,

   b) Arguments for the predicate, and

   c) A binding pattern (or adornment, as in Ullman [1989]) indicating which arguments of the predicate are expected to be bound (provided as parameters of the query represented by the view) and which are free (produced as answers to the query).

   The *body* of the view is a program in some notation that produces a result to the query.

**Example 2:** Consider Example 1. The two queries we are allowed to perform on the genealogy become two views. The parameters of the queries are the bound arguments, and the results are the free arguments. We assume *parent* is the predicate representing child-parent information. Thus, we can write query (1) as:

$v_1^{bf}$(C,P) :- parent(C,P)

That is, $v_1$ expects a binding for $C$, which becomes the first argument of *parent*, and it produces values for $P$, the second argument of *parent*.

Similarly, query (2) is written:

$v_2^f$(C) :- parent(C,P)

That is, with no argument, all the values of $C$ that are a first argument of some *parent* fact are produced. □

### Queries

Now, we must define what it means for a program to be a solution to a given query. A *query* is denoted exactly as a view. That is, it has a head with a binding pattern and it has a body that is a program over the predicates that represent the information of the source.

**Example 3:** The query of Example 1 in which we are asked to find the grandparents of individual $C$ is expressed

$answer_1^{bf}$(C,G) :-
    parent(C,P) & parent(P,G)

That is, we are given a binding for $C$ and are asked to find the related values of $G$ according to the program in the body that composes *parent* with itself.

The query

$answer_2^{fb}$(C,G) :-
    parent(C,P) & parent(P,G)

looks similar, but expresses the more difficult (for the views of Example 1) query in which we are given a grandparent and asked to find the grandchildren. □

### Valid Solutions

A query is solved by a program that uses views to obtain answers from the external world. In general, the program can create its own data, corresponding to IDB predicates in datalog programs (see Ullman [1988]), and it can use arithmetic comparisons or other predicates that have a meaning independent of any information source.

In most of what follows, we shall assume that solutions are conjunctive queries in the form of Chandra and Merlin [1977], although we shall also consider bodies that are datalog programs over the views. For each notation used to describe programs that are solutions, we need to define precisely what it means for the views to be used properly, i.e., for the solution to be *valid*.

### Conjunctive Queries as Solutions

The simplest case is when the views and solutions are conjunctive queries. All subgoals have view names as predicates, and we assume they are evaluated in left-to-right order. There are two conditions a solution must satisfy to be valid.

1. The binding patterns must be appropriate. That is, if the $i$th subgoal has predicate $v_j$, and the binding pattern for the head of view $v_j$ requires a certain argument be bound, then any variable appearing in that argument of the $i$th subgoal must either appear in one of the first $i-1$ subgoals, or it must appear in the query head in a bound argument (see Ullman [1989] for details of how bindings pass from arguments to variables). It is not required that other arguments of the $i$th subgoal be "free."
2. The *expansion* of the solution, in which each subgoal is replaced by the body of the appropriate view, must be equivalent to the given query.

**Example 4:** Consider our running example, with views $v_1^{bf}$ and $v_2^f$ from Example 2 and the query $answer_1^{bf}$ from Example 3 (find the grandparents of $C$). An appropriate solution is

$answer_1^{bf}$(C,G) :-
    $v_1^{bf}$(C,P) & $v_1^{bf}$(P,G)    (1)

This solution satisfies the binding-pattern condition because in the first subgoal, the first argument $C$ is bound by the head, and in the second subgoal, the first argument $P$ is bound by the first subgoal.

We must also expand solution (1) to check that it is equivalent to the given query. In this case, we replace $v_1^{bf}(C,P)$ by the definition according to Example 2, giving us $parent(C,P)$ in place of the first subgoal. We must replace the second subgoal similarly by $parent(P,G)$. The expansion of (1) is thus

$answer_1^{bf}$(C,G) :-
    parent(C,P) & parent(P,G)

This conjunctive query is identical to the query from Example 3, so surely it is equivalent to that query. We conclude that the proposed solution (1) is valid. □

**Example 5:** Now consider the second query from Example 3, where we are asked to find grandchildren. The following is *not* a valid solution:

$$\begin{array}{l} answer_2^{fb}(\mathtt{C,G})\ \texttt{:-} \\ \quad v_1^{bf}(\mathtt{C,P})\ \texttt{\&}\ v_1^{bf}(\mathtt{P,G}) \end{array}$$

The reason is that in the first subgoal, the first argument $C$ does not receive a binding from the head, which only binds the second argument $G$. Moreover, reversing the order of the subgoals does not help; the first argument $P$ is then free.

However, using view $v_2$ we can produce a valid solution:

$$\begin{array}{l} answer_2^{fb}(\mathtt{C,G})\ \texttt{:-} \\ \quad\quad v_2^{f}(\mathtt{C})\ \texttt{\&} \\ \quad\quad v_1^{bf}(\mathtt{C,P})\ \texttt{\&} \\ \quad\quad v_1^{bf}(\mathtt{P,G}) \end{array} \tag{2}$$

Note that the first subgoal, with predicate $v_2$, requires no bound argument but provides a binding for $C$. That binding allows the second subgoal to have its first argument bound, as $v_1^{bf}$ requires. The third subgoal also has its required binding for the first argument. the fact that its second argument is also bound, because $G$ appears in a bound argument of the head, is irrelevant. Thus, the condition of proper binding patterns is met.

When we expand (2) we get

$$\begin{array}{l} answer_2^{fb}(\mathtt{C,G})\ \texttt{:-} \\ \quad \mathtt{parent(C,X)}\ \texttt{\&} \\ \quad \mathtt{parent(C,P)}\ \texttt{\&} \\ \quad \mathtt{parent(P,G)} \end{array}$$

Note that local variables in view definitions, such as $P$ in the definition of $v_2$ (see Example 2) must be replaced, during expansion, by new local variables to avoid accidently using $P$ in two different ways. This expansion is not identical to the original query of Example 3, but it is easy to see that these two conjunctive queries are equivalent. Intuitively, although the first subgoal, $parent(C, X)$, is needed because of the binding-pattern requirement, it is redundant when we consider only the functions defined by the bodies in the original query and the expanded solution. □

### Solutions and Views Involving Arithmetic Comparisons

We can extend both view definitions and solutions to conjunctive queries that have subgoals that compare the arithmetic values of two variables, as studied in Klug [1988], Zhang and Ozsoyoglu [1993], Levy and Sagiv [1993], or Gupta et al. [1994]. The validity conditions do not change, except that we must introduce a "safety" condition to assure that if there is a subgoal such as $X < Y$ in a body, then both variables $X$ and $Y$ have previously been bound, either in the body or the head. We expect this safety condition to apply to both view definitions and solutions.

### Solutions and Views Involving Datalog Programs

We can allow view bodies and solution bodies to be datalog programs, even recursive programs. We require that the rules of the program involve only views in the role of EDB programs, although they may involve arbitrary IDB predicates. In these programs, we assume that in the left-to-right evaluation order for each rule there is a unique binding pattern for each an IDB predicate, and of course that the binding patterns of view uses are respected. The assumption of unique binding patterns for IDB predicates is made without loss of generality, as we can always rewrite a program to be in this form (see Ullman [1989]).

## III. Related Work

Yang and Larson [1987] look at a broad class of queries (select-project-join queries) and implemented a system for finding equivalents in terms of views that were themselves SPJ queries. They develop a notion of mappings that look like a generalization of containment mappings for conjunctive queries but are too restrictive (they are one-one rather than many-one) and search for solutions only within this space. The question of implementing conjunctive queries by use of views was studied recently by Chaudhuri, Krishnamurthy, Potamianos, and Shim [1994]. In Levy, Mendelzon, Sagiv, and Srivastava [1995] the problem posed by Chaudhuri et al. is addressed by providing a decision algorithm for the question of whether a solution exists; we discuss the relationship of this algorithm to our work in Section IV.

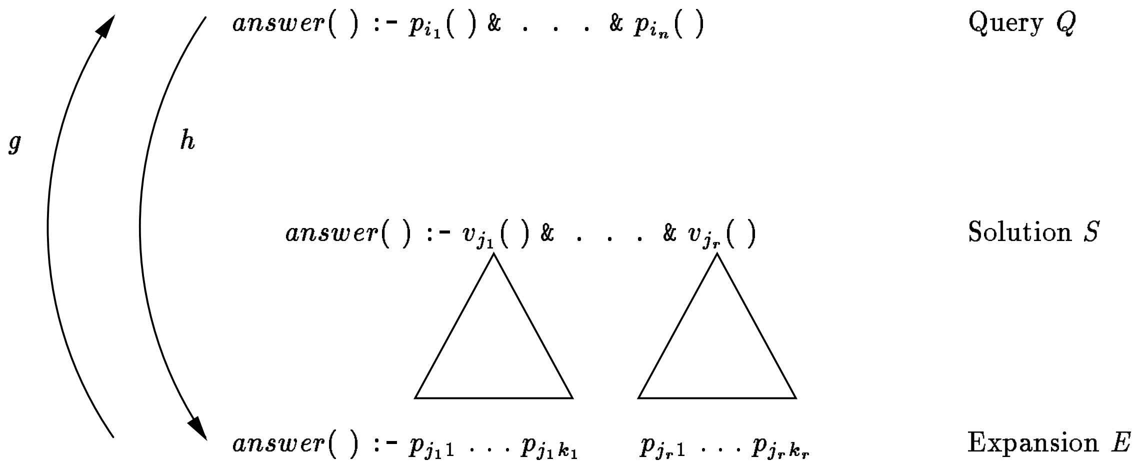


**Fig. 3.** A conjunctive query and its solution.

None of the papers mentioned above involve the use of binding patterns. However, Chaudhuri and Shim [1993] consider "foreign functions" in queries. The latter are related to our notion of views with binding patterns, since foreign functions require a particular binding pattern for use and therefore induce a constraint on queries that they appear at a position in which all required arguments are bound.

## IV. Testing for Solutions to Conjunctive Queries

The fundamental question to ask is: given a collection of views and given a query, is there a solution for the query in terms of the views? For the case that the views, query, and solution are conjunctive queries without arithmetic comparisons, we can give such a decision procedure and in fact can show the problem is in $\mathcal{NP}$ (and therefore $\mathcal{NP}$-complete, since simpler problems are $\mathcal{NP}$-complete).

First, we should note that the problem is not obviously decidable. While we can easily check that the condition on binding patterns is satisfied, and testing whether two conjunctive queries are equivalent is in $\mathcal{NP}$, there is no a priori limit on the size of a solution relative to the query. Levy et al. [1995] solve the problem without binding patterns by showing that if there is a solution, then there is a solution that uses no more view-subgoals in the solution than there are subgoals in the query. However, their result does not apply to our problem, and in fact Example 5 should convince us that their result is not true in our setting. That is, although the query has only two subgoals, the smallest solution requires three subgoals. However, we can prove a slightly weakened form of the result mentioned above.

**Lemma 1:** If $Q$ is a conjunctive query (with binding pattern for the head) with $n$ subgoals and $m$ different variables, and we are given a collection of views that are conjunctive queries with binding patterns for the head, then if there is a conjunctive-query solution to $Q$ there is a solution with at most $m+n$ subgoals, using at most $m$ different variables.

**Proof:** Consider Fig. 3, which suggests the query $Q$, some solution $S$, and the expansion $E$ of that solution. We also show containment mappings $h$ from $Q$ to $E$ and $g$ from $E$ to $Q$, that together demonstrate $Q \equiv E$.

We begin by grouping variables of $E$ into equivalence classes by defining $X \equiv Y$ if $g(X) = g(Y)$. There are at most $m$ classes, since that is the number of variables of $Q$.

Next, we need to shorten $S$ so it has at most $n + m$ subgoals and is still a solution. We retain a subgoal $G$ of $S$ if either

1. One of the subgoals of $E$ that come from $G$ is the target of a subgoal of $Q$ under the containment mapping $h$.

2. One of the variables $X$ in $G$ is the *leader* of its equivalence class of variables according to our partition of the variables of $E$, meaning that $X$ is the leftmost occurrence in $S$ of any of the members of $X$'s equivalence class. (Note

that our safety assumption for view definitions assures that every variable in $S$ appears in $E$.)

There are at most $n$ subgoals of type (1) and at most $m$ subgoals of type (2) in $S$. We construct $S'$ from $S$ by:

a) Delete from $S$ any subgoal that does not meet at least one of the two conditions above. We are left with at most $m + n$ subgoals.

b) Replace all variables in an equivalence class by their leader. We are left with at most $m$ variables.

We claim $S'$ is also a solution. First, the binding-pattern condition is met. The reason is that we have retained the subgoals of $S$ that introduce the leader of each group and have equated all nonleader variables to their leader. Thus, the only occurrences of variables that are not bound must be leaders, and these occurrences were not bound in $S$ either.

Finally, we must show that the expansion of $S'$, which we shall call $E'$, is equivalent to $Q$. $E'$ differs from $E$ in three ways.

*i.* Some local variables (those introduced in the expansion of a view) may be different. The names of these nondistinguished variables is unimportant, and we can change $E'$ to make them the same as those in $E$.

*ii.* Subgoals in $E$ may be missing in $E'$, because we deleted the subgoal of $S$ from which they came. However, we did not delete any subgoal that is the target of a subgoal of $Q$ under mapping $h$, so this change does not prevent $h$ from being a containment mapping from $Q$ to $E'$.† Deletion of subgoals in $E$ obviously does not prevent $g$ from being a containment mapping from $E'$ to $Q$.

*iii.* Variables that are distinct in $E$ may be identified in $E'$. This change cannot prevent $h$ from being a containment mapping, although we may have to change its value for some variables of $Q$. It could prevent $g$ from being a containment mapping, but we were careful to equate only variables that had the same image under $g$.

Thus, $g$ and $h$, modified as in (*iii*), are containment mappings showing the equivalence of $Q$ and $E'$. That in turn shows that $S'$ is a solution with at most $n + m$ subgoals and at most $m$ distinct variables, proving the lemma. □

**Theorem 1:** There is a nondeterministic-polynomial-time algorithm that, given views defined by conjunctive queries and binding patterns and a query of the same type, decides if there is a conjunctive-query solution to the query for these views and finds it if so.

**Proof:** Guess a solution that has no more subgoals than the sum of the number of subgoals and variables in the query and uses no more variables than does the query. Check that the binding-pattern condition is satisfied in deterministic polynomial time. If so, guess containment mappings and check their correctness in nondeterministic polynomial time. □

Thus the problem of finding a solution for the conjunctive query case is $\mathcal{NP}$-complete, but no worse. Since queries tend to be short, we regard this result as positive, suggesting that a real implementation of the idea is feasible.

### Can Solutions be More General Than Conjunctive Queries?

We can extend Theorem 1 to include the possibility that, while the views and query are conjunctive queries with binding patterns, the solution is a datalog program. As discussed in Section II, this datalog program must respect binding patterns within its rules. Therefore, it should come as no surprise that:

**Theorem 2:** If there is a valid datalog program solution to the problem of Theorem 1, then there is a valid conjunctive-query solution, and it can be found in $\mathcal{NP}$-time.

**Proof:** (sketch) A datalog program can be expanded into a (possibly infinite) union of conjunctive queries. We can show that each of these must satisfy the binding-pattern condition for CQs. If a CQ is equivalent to a union consisting of the expansions of the CQs generated from the datalog program, then it must be equivalent to one of them (Sagiv and Yannakakis [1981]). □

† This argument is "borrowed" from Levy et al. [1995], where it is used to prove their result about $n$ subgoals being sufficient when binding patterns are not considered.

## V. Conjunctive Queries With Arithmetic Comparisons

We can extend Theorem 1 to conjunctive queries with arithmetic comparisons, although we lose the membership in $\mathcal{NP}$ and the linear bound on the size of solutions.

**Lemma 2:** Suppose views, queries, and solutions are CQs with arithmetic comparisons (and binding patterns for the heads, of course). There is a doubly exponential function $f(s)$ such that if $s$ is the size of a given query $Q$ and set of views, there is a solution $S$ for $Q$ in terms of the views of size at most $f(s)$.

**Proof:** (sketch) Again think of Fig. 3 and an expansion $E$ for solution $S$. We can use the technique of Klug [1988], where we consider all possible orderings of the variables of the query $Q$ when testing whether $Q \subseteq E$.

There are at most $s$ variables in $Q$, therefore at most $s!$ orders of these variables. For each order, there may or may not be equality between each pair of consecutive variables, thus the number of orders with equality is at most $s!2^{s-1}$, an exponential in $s$. For each of these orders, there must be some containment mapping from $E$ to $Q$ that respects the order.

Define two variables of $E$ to be *equivalent* if they map to the same variable of $Q$ for every order. There is "only" a doubly exponential number of equivalence classes, i.e., at most $c(s) = s^{s!2^{s-1}}$ classes.

Modify $S$ by identifying variables that are equivalent. As in Lemma 1, safety of CQs assures that variables of $S$ appear in $E$. Eliminate all but the first of identical subgoals in $S$, to form $S'$.

We claim $S'$ has at most $sc(s)^s$ subgoals. The reason is that for a subgoal of $S$ we may pick the predicate (view name) in at most $s$ ways. There are at most $s$ arguments to any view, so the variables appearing there can be chosen in $c(s)^s$ ways.

Since $c(s)$ is doubly exponential, $c(s)^s$ is also doubly exponential. Thus, we may pick $f(s)$ to be the length of a solution with $sc(s)^s$ subgoals; $f(s)$ will also be doubly exponential and is a bound on the length of $S'$.

The containment mappings from $E$ to $Q$ can be used without change to show that $Q$ is contained in the expansion $E'$ of $S'$, since whenever variables are identified they have the same result when any of the mappings involved are applied.

The binding pattern requirements of $S'$ are surely satisfied, since we never removed a subgoal of $S$ that contains the first occurrence of a variable. Also, the identification of variables cannot make any binding go from bound to free.

Finally, $E' \subseteq E$, because $E'$ is $E$ with additional restrictions (the equality of some variables) and the deletion of some obviously redundant subgoals.

Since $E \equiv Q$ is assumed, we have $E' \subseteq E \equiv Q \subseteq E'$. Thus, all three of $Q$, $E$, and $E'$ are equivalent, proving that $S'$ is a solution. Since the size of $S'$ is "only" doubly exponential in the size of $Q$ and the view definitions, the proof is complete. □

**Theorem 3:** If queries, views, and solutions are CQs with comparisons and binding patterns, there is a nondeterministic, doubly exponential-time decision procedure to find a solution to a given query or determine that none exists.

**Proof:** Guess a solution $S$ of size at most $f(s)$, where $f$ is the function described in Lemma 2 and $s$ is the size of the query and views. Expand $S$ and guess containment mappings between the query and the expansion. □

## VI. Conclusions

In this paper, we introduced the problem of answering a query using query templates (views) that are restricted to use certain binding patterns for their distinguished variables. Our motivation for studying this problem is the automatic generation of translators from declarative specifications, as in the Tsimmis project. We presented some preliminary results showing that the decision problem is decidable for conjunctive queries. We proved bounds on the size of the solution (if it exists), which leads to a non-deterministic polynomial-time algorithm for the case with no arithmetic comparisons and nondeterministic exponential time when arithmetic comparisons are permitted.

We are currently investigating restrictions on the structure of the query and/or the views that lead to practical and efficient algorithms for both these problems. We are also considering algorithms and heuristics for obtaining the cheapest solution under certain reasonable cost metrics.

## VII. References


Chandra, A. K. and P. M. Merlin [1977]. "Optimal implementation of conjunctive queries in relational databases," *Proc. Ninth Annual ACM Symposium on the Theory of Computing*, pp. 77–90.

Chaudhuri, S., R. Krishnamurthy, S. Potamianos, and K. Shim [1994]. "Optimizing queries with materialized views," HPL-DTD-94-16, Hewlett Packard Research Laboratories, Palo Alto, CA.

Chaudhuri, S. and K. Shim [1993]. "Query optimization in the presence of foreign functions," *Proc. International Conference on Very Large Data Bases*, pp. 529–542.

Chawathe, S., et al. [1994]. "The TSIMMIS project: integration of heterogeneous information sources," IPSJ, Tokyo, Oct., 1994. Available by anonymous ftp from `db.stanford.edu` as file `pub/chawathe/1994/tsimmis-overview.ps`

Gupta, A., Y. Sagiv, J. D. Ullman, and J. Widom [1994]. "Constraint checking with partial information," *Proc. Thirteenth ACM Symposium on Principles of Database Systems*, pp. 45–55.

Klug, A. [1988]. "On conjunctive queries containing inequalities," *J. ACM* **35**:1, pp. 146–160.

Levy, A., A. Mendelzon, Y. Sagiv, and D. Srivastava [1995]. "Answering queries using views," these proceedings.

Levy, A. and Y. Sagiv [1993]. "Queries independent of updates," *Proc. International Conference on Very Large Data Bases*, pp. 171–181.

Papakonstantinou, Y., H. Garcia-Molina, and J. Widom [1994]. "Object exchange across heterogeneous information sources," unpublished memorandum to appear in 1995 ICDE. Available by anonymous ftp from `db.stanford.edu` as file

`pub/papakonstantinou/1994/object-`
exchange-heterogeneous-is.ps

Sagiv, Y. and M. Yannakakis [1981]. "Equivalence among relational expressions with the union and difference operators," *J. ACM* **27**:4, pp. 633–655.

Ullman, J. D. [1988]. *Principles of Database and Knowledge-Base Systems, Vol. I: Classical Database Systems*, Computer Science Press, New York.

Ullman, J. D. [1989]. *Principles of Database and Knowledge-Base Systems, Vol. II: The New Technologies*, Computer Science Press, New York.

Yang, H. Z. and P. A. Larson [1987]. "Query transformation for PSJ queries," *Proc. International Conference on Very Large Data Bases*, pp. 245–254.

Zhang, X. and M. Z. Ozsoyoglu [1993]. "On efficient reasoning with implication constraints," *Proc. Third DOOD Conference*, pp. 236–252.