# Theory of Answering Queries Using Views *

**Alon Y. Halevy**†
Department of Computer Science and Engineering
University of Washington
Seattle, WA, 98195
alon@cs.washington.edu

## Abstract

The problem of answering queries using views is to find efficient methods of answering a query using a set of previously materialized views over the database, rather than accessing the database relations. The problem has recently received significant attention because of its relevance to a wide variety of data management problems, such as query optimization, the maintenance of physical data independence, data integration and data warehousing. This article surveys the theoretical issues concerning the problem of answering queries using views.

## 1 Introduction

The problem of answering queries using views (a.k.a. rewriting queries using views) has recently received significant attention because of its relevance to a wide variety of data management problems: query optimization, maintenance of physical data independence, data integration and data warehouse design. Informally speaking, the problem is the following. Suppose we are given a query $Q$ over a database schema, and a set of view definitions $V_1, \ldots, V_n$ over the same schema. Is it possible to answer the query $Q$ using *only* the answers to the views $V_1, \ldots, V_n$? If we can access both the views and the database relations, what is the cheapest query execution plan for answering $Q$? Alternatively, what is the maximal set of answers for $Q$ that we can obtain from the views? This article surveys the theoretical aspects of answering queries using views. For the more comprehensive survey from which this material was drawn, see [30].

The first context in which we encounter the problem of answering queries using views is query optimization and database design. In the context of query optimization, computing a query using previously materialized views can speed up query processing because part of the computation necessary for the query may have already been done while computing the views. Such savings are especially significant in decision support applications when the views and queries contain grouping and aggregation. In the context of database design, view definitions provide a mechanism for supporting the independence of the *physical* view of the data and its *logical* view. This independence enables us to modify the storage schema of the data (i.e., the physical view) without changing its logical schema. Hence, several authors have proposed to describe the storage schema as a set of views over the logical schema [45, 43, 24, 19]. Given these descriptions of the storage, the problem of computing a query execution plan (which, of course, must access the physical storage) involves figuring out how to use the views to answer the query.

A second context in which our problem arises is data integration. A data integration system provides a uniform interface to a multitude of data sources. Users of a data integration system pose queries against a *mediated schema,* rather than the schemas in which the data is actually stored. The system catalog includes a set of *source descriptions,* that provide semantic mappings between the relations in the mediated schema and those in the data sources. As a result, the problem of reformulating a user query, posed over the mediated schema, into a query that refers directly to the source schemas becomes the problem of answering queries using views. The solutions to the problem of answering queries using views differ in this context because the number of views (i.e., sources) tends to be much larger, and the sources need not contain the *complete* extensions of the views.

Other contexts in which the problem of answering queries using views arises are data warehouse de-

*Database Principles Column.
Column editor: Leonid Libkin, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 08974. E-mail: libkin@research.bell-labs.com.

†The author previously known as Levy.

sign [5, 15], web-site design [23] and semantic data caching [17].

The many applications of the problem of answering queries using views has spurred a flurry of research, ranging from theoretical foundations to algorithm design and implementation in several commercial systems. The treatment of the problem has differed on several dimensions, including the specific query language considered, whether we consider an *equivalent* or *maximally-contained* rewriting of the query, and whether the views are assumed to contain *all* the tuples in their definitions. An important distinction that has been made in the theoretical work is between the problem of *query rewriting* where the output is a query expression for computing the answers from the views, and the problem of *query answering,* where the result is the set of all possible answers we can obtain for the queries from the views.

Section 2 formally defines the problem and its different dimensions. Section 3 surveys the work on the rewriting problem. Section 4 discusses the work on query answering, and Section 5 describes treatments of several query-language extensions. This article is limited to the theoretical aspects of the problems. The complete survey [30] discusses the large body of work on incorporating materialized views into query optimizers, specific algorithms for answering queries using views and a more complete treatment of queries with grouping and aggregation.

# 2 Problem Definition

We use datalog notation throughout the paper. We consider mostly *conjunctive queries*, which have the form

$$q(\bar{X}) :- r_1(\bar{X}_1),\ \ldots,\ r_n(\bar{X}_n)$$

where $q$, and $r_1, \ldots, r_n$ are predicate names. The predicate names $r_1, \ldots, r_n$ refer to database relations. The atom $q(\bar{X})$ is called the *head* of the query, and $q$ refers to the answer relation. The atoms $r_1(\bar{X}_1), \ldots, r_n(\bar{X}_n)$ are the *subgoals* in the body of the query. The tuples $\bar{X}, \bar{X}_1, \ldots, \bar{X}_n$ contain either variables or constants. We require that the query be *safe*, i.e., that $\bar{X} \subseteq \bar{X}_1 \cup \ldots \cup \bar{X}_n$ (that is, every variable that appears in the head must also appear in the body).

Queries may also contain subgoals whose predicates are arithmetic comparisons $<, \leq, =, \neq$. In this case, we require that if a variable $X$ appears in a subgoal of a comparison predicate, then $X$ must also appear in an ordinary subgoal.

In our discussion, we denote the result of computing the query $Q$ over the database $D$ by $Q(D)$. We often refer to queries that reference named views (e.g., in query rewritings); in that case, $Q(D)$ refers to the result of computing $Q$ after the views have been computed from $D$. Throughout this paper we assume set semantics for our databases and query results.

The notions of query containment and query equivalence enable comparison between different reformulations of a query. They will be used when we test the correctness of a rewriting of a query in terms of a set of views. The problems of query containment and equivalence have been studied extensively in the literature and deserve a survey of their own.

**Definition 2.1** (**Query containment and equivalence**) A query $Q_1$ is said to be contained in a query $Q_2$, denoted by $Q_1 \sqsubseteq Q_2$, if for all databases $D$, the set of tuples computed for $Q_1$ is a subset of those computed for $Q_2$, i.e., $Q_1(D) \subseteq Q_2(D)$. The two queries are said to be equivalent if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$. □

## 2.1 Query Rewriting

Given a query $Q$ and view definitions $V_1, \ldots, V_m$, a rewriting of $Q$ using the views is a query expression $Q'$ whose subgoals are either view relations $V_1, \ldots, V_m$, or comparison predicates. In practice, we may also be interested in rewritings that can also refer to the database relations, but conceptually this case does not introduce any new difficulties.

We distinguish two types of query rewritings: *equivalent rewritings* and *maximally-contained rewritings.* In the contexts of query optimization and maintainance of physical data independence we usually consider equivalent rewritings.

**Definition 2.2** (**Equivalent rewritings**) Let $Q$ be a query and $\mathcal{V} = V_1, \ldots, V_m$ be a set of view definitions. The query $Q'$ is an equivalent rewriting of $Q$ using $\mathcal{V}$ if:

- the subgoals of $Q'$ are either relations in $\mathcal{V}$, or comparison predicates, and
- $Q'$ is equivalent to $Q$.

□

**Example 2.1** Consider a simple database schema in which the relation cites(p1,p2) stores pairs of publications where p1 cites p2, and the relation sameTopic stores pairs of publications that are on the same topic. The unary relations inSIGMOD and inVLDB store papers published in SIGMOD and VLDB respectively. The following query asks for pairs of papers on the same topic that also cite each other.

Q(x,y):- sameTopic(x,y), cites(x,y), cites(y,x)

Assume we have the following views. The view V1 stores pairs of papers that cite each other, and V2 stores pairs of papers on the same topic and each of which cites at least one other paper.

V1(a,b):- cites(a,b), cites(b,a)
V2(c,d) :- sameTopic(c,d), cites(c,c1), cites(d,d1)
The following is an equivalent rewriting of $Q$:
Q'(x,y):- V1(x,y), V2(x,y) .
To check that Q' is an equivalent rewriting, we can unfold the view definitions to obtain Q", and show that Q is equivalent to Q" by finding containment mappings [13] between $Q$ and $Q''$.
Q"(x,y):- cites(x,y), cites(y,x), sameTopic(x,y),
cites(x,x1), cites(y,y1) □

In the context of data integration, we use views to describe the contents of the data sources. Given a query $Q$, the data integration system first needs to reformulate $Q$ to refer to the data sources, i.e., the views. In this context, we cannot always find an equivalent rewriting of the query using the views, because the sources do not provide all the necessary data. Hence, we consider the problem of finding the *maximally-contained* rewriting. Note that maximality of a rewriting is defined w.r.t. a particular query language:

**Definition 2.3** (**Maximally-contained rewritings**) Let $Q$ be a query, $\mathcal{V} = V_1, \ldots, V_m$ be a set of view definitions, and $\mathcal{L}$ be a query language. The query $Q'$ is a maximally-contained rewriting of $Q$ using $\mathcal{V}$ w.r.t. $\mathcal{L}$ if:

- $Q'$ is a query in $\mathcal{L}$ that refers only to the views in $\mathcal{V}$ or comparison predicates,
- $Q'$ is contained in $Q$, and
- there is no rewriting $Q_1 \in \mathcal{L}$, such that $Q' \subseteq Q_1 \subseteq Q$ and $Q_1$ is not equivalent to $Q'$.

□

When a rewriting $Q'$ is contained in $Q$ but is not a maximally-contained rewriting we refer to it as a contained rewriting.

**Example 2.2** Suppose that in our domain we have the following two data sources, S1 and S2, containing pairs of SIGMOD (respectively VLDB) papers that cite each other. The sources can be described as follows:
S1(a,b):- cites(a,b), cites(b,a), inSIGMOD(a),
inSIGMOD(b)
S2(a,b):- cites(a,b), cites(b,a), inVLDB(a),
inVLDB(b)
Given the query from the previous example and the sources S1, S2 and V2, the best rewriting we can compute is:
q'(x,y):- S1(x,y), V2(x,y)
q'(x,y):- S2(x,y), V2(x,y)

Note that this rewriting is a union of conjunctive queries, describing multiple ways of obtaining answers to the query from the available sources. The rewriting is not an equivalent rewriting, since it misses any pair of papers that is not both in SIGMOD or both in VLDB, but we don't have data sources to provide us such pairs. Furthermore, since the sources are not guaranteed to have all the tuples in the definition of the view, our rewritings need to consider different views that may have similar definitions. Specifically, if there were another source S3 with an identical description as S1, we would also have to consider query plans in which S1 is replaced by S3. □

A more fundamental question we can consider is how to find *all* the possible answers to the query, given a set of view definitions and their extensions. Finding a rewriting of the query using the views and then evaluating the rewriting over the views is clearly one candidate algorithm. If the rewriting is equivalent to the query, then we are guaranteed to find all the possible answers. However, a maximally-contained rewriting of a query using a set of views does not always provide all the possible answers that can be obtained from the views. Intuitively, the reason for this is that a rewriting is maximally-contained only w.r.t. a specific query language, and hence there may sometimes be a query in a more expressive language that may provide more answers.

The problem of finding all the answers to a query given a set of views is formalized in [1] by the notion of *certain answers*. The definition distinguishes the case in which the view extensions are assumed to be complete (closed-world assumption) from the case in which the views may be partial (open-world).

**Definition 2.4** (**Certain answers**) Let $Q$ be a query and $\mathcal{V} = V_1, \ldots, V_m$ be a set of view definitions over the database schema $R_1, \ldots, R_n$. Let the sets of tuples $v_1, \ldots, v_m$ be extensions of the views $V_1, \ldots, V_m$, respectively.

The tuple $\bar{a}$ is a *certain answer* to the query $Q$ under the closed-world assumption given $v_1, \ldots, v_m$ if $\bar{a} \in Q(D)$ for all databases $D$ such that $V_i(D) = v_i$ for every $i$, $1 \leq i \leq m$.

The tuple $\bar{a}$ is a *certain answer* to the query $Q$ under the open-world assumption given $v_1, \ldots, v_m$ if $\bar{a} \in Q(D)$ for all databases $D$ such that $V_i(D) \supseteq v_i$ for every $i$, $1 \leq i \leq m$. □

The intuition behind the definition of certain answers is the following. The extensions of $V_1, \ldots, V_n$ do not define a unique extension of the database relations. Hence, given the extension of the views we have only partial information about the real state of the database. A tuple is a certain answer of the query $Q$ if it is an answer for *any* of the possible database extensions that are consistent with the given extensions of the views.

**Example 2.3** Consider a database schema that includes the single relation e(X,Y) and consider the following views T1 and T2 and query:
T1(x):- e(x,y)
T2(y):- e(x,y)
q(x,y):- e(x,y)
Suppose the extension of T1 is $\{(a)\}$ and the extension of T2 is $\{(b)\}$. Under the closed-world assumption the tuple $(a, b)$ is a certain answer to q. However, under the open-world assumption $(a, b)$ is not a certain answer. □

**Dimensions of the problem:** The treatments of the problem of answering queries using views have been considered along several dimensions. Two of the dimensions mentioned above are equivalent versus maximally-contained rewritings, and whether the views are assumed to be complete or not. Clearly, the algorithms and results regarding answering queries using views depend crucially on the language used for expressing the views and the queries. The problem has been considered for conjunctive queries, queries with union, aggregation, recursion, and query languages for object-oriented and semi-structured databases. In addition, the problem has been considered in the presence of integrity constraints on the database and access-pattern limitations, and constraints expressed in Description Logics. Finally, a key dimension is the expected output of the algorithm. Given a query $Q$, and a set of views $\mathcal{V}$, there are three possible outputs an algorithm may produce: (1) an expression $Q'$ that references the views and is either equivalent to or contained in $Q$, (2) a query execution plan for answering $Q$ using the views (and possibly the database relations), or (3) the answers to $Q$. In the latter case, we assume the extensions of the views $\mathcal{V}$ are given as well.

# 3 The Rewriting Problem

In this section we consider the problem of rewriting a query using a set of views, i.e., algorithms that produce query expressions as their output. The first question one can ask about an algorithm for rewriting queries using views is whether the algorithm is sound and complete: given a query $Q$ and a set of views $\mathcal{V}$, is there an algorithm that will find a rewriting of $Q$ using $\mathcal{V}$ when one exists, and what is the complexity of that problem. The first answer to this question was given for the class of queries and views expressed as conjunctive queries [35]. In that paper it was shown that when the query does not contain comparison predicates and has $n$ subgoals, then there exists an equivalent conjunctive rewriting of $Q$ using $\mathcal{V}$ only if there is a rewriting with at most $n$ subgoals. An immediate corollary is that the problem of finding and equivalent rewriting of a query using a set of views is in NP, because it suffices to guess a rewriting and check its correctness.[1]

In [35] it is also shown that the problem of finding a rewriting is NP-hard for two independent reasons: (1) the number of possible ways to map a single view into the query, and (2) the number of ways to combine the mappings of different views into the query. In [18] it is shown that the problem of finding a contained rewriting is also NP-complete. In [14] the authors exploit the close connection between the containment and rewriting problems, and show several polynomial-time cases of the rewriting problems, corresponding to analogous cases for the problem of query containment.

The bound on the size of the rewriting (and therefore on the search space of rewritings) has led to a succession of algorithms that attempt to efficiently search the space. Three of these algorithms, (the Bucket Algorithm [36], the Inverse-rules Algorithm [22], and the MiniCon Algorithm [41]) that focus on query rewriting for the data integration context, are experimentally compared in [41], showing that the MiniCon outperforms the other two and scales up to hundreds of views.

## 3.1 Access Pattern Limitations

In the context of data integration, where data sources are modeled as views, we may have limitations on the possible access paths to the data. For example, when querying the Internet Movie Database, we cannot simply ask for all the tuples in the database. Instead, we must supply one of several inputs, (e.g., actor name or director), and obtain the set of movies in which they are involved.

We can model limited access paths by attaching a set of adornments to every data source. If a source is

[1] Note that checking the correctness of a rewriting is NP-complete, however, the guess of a rewriting can be extended to a guess for containment mappings showing the equivalence of the rewriting and of the query.

modeled by a view with $n$ attributes, then an adornment consists of a string of length $n$, composed of the letters $b$ (bound) and $f$ (free). The meaning of the letter $b$ in an adornment is that the source *must* be given values for the attribute in that position. The meaning of the letter $f$ in an adornment is that the source doesn't have to be given a value for the attribute in that position. For example, an adornment $V(A,B)^{bf}$ means that tuples of $V$ can be obtained only by providing values for the attribute $A$.

Several works have considered the problem of answering queries using views when the views are also associated with adornments describing limited access patterns. In [42] it is shown that the bound given in [35] on the length of a possible rewriting does not hold anymore. To illustrate, we have the following views with their associated adornments:

CitationDB$^{bf}$(X,Y) :- cites(X,Y)
CitingPapers$^{f}$(X) :- cites(X,Y)

CitationDB requires that the first argument be given as input, while the CitingPapers source does not have limitations on access patterns. Suppose we have the following query asking for all the papers citing paper #001:

Q(X) :- cites(X,001)

The bound given in [35] would require that if there exists a rewriting, then there is one with at most one atom, the size of the query. However, the only possible rewriting in this case is:

q(X) :- CitingPapers(X), CitationDB(X,001).

In [42] the authors show that in the presence of access-pattern limitations it is sufficient to consider a slightly larger bound on the size of the rewriting: $n+v$, where $n$ is the number of subgoals in the query and $v$ is the number of variables in the query. Hence, the problem of finding an equivalent rewriting of the query using a set of views is still NP-complete.

The situation becomes more complicated when we consider maximally-contained rewritings. As the following example given in [33] shows, there may be *no* bound on the size of a rewriting. In the following example, the source DBSource stores the set of papers in the database field, and has no access-pattern limitations. The second source, when given a paper, will return all the papers that are cited by it. The third source, when given a paper (in databases or any other field), returns whether the paper is an award winner or not. The source descriptions are the following:

DBSource$^{f}$(X) :- DBpapers(X)
CitationDB$^{bf}$(X,Y) :- cites(X,Y)
AwardDB$^{b}$(X) :- AwardPaper(X)

The query asks for all the papers that won awards:

Q(X) :- AwardPaper(X).

Since the source AwardDB requires its input to be bound, we cannot query it directly. One way to get solutions to the query is to obtain the set of all database papers from the source DBSource, and perform a dependent join with the source AwardDB. Another way would be to begin by retrieving the papers in DBSource, join the result with the source CitationDB to obtain all papers cited by papers in DBSource, and then join the result with the source AwardDB. As the rewritings below show, we can follow any length of citation chains beginning with papers in DBSource and obtain answers to the query that were possibly not obtained by shorter chains. Hence, there is no bound on the length of a rewriting of the query using the views.

Q'(X) :- DBSource(X), AwardDB(X)
Q'(X) :- DBSource(U), CitationDB(U,$X_1$), ..., CitationDB($X_n$,X), AwardDB(X).

Fortunately, as shown in [20], we can still find a finite rewriting of the query using the views, albeit a recursive one. The following datalog rewriting will obtain all the possible answers from the above views.

papers(X) :- DBsource(X)
papers(X) :- papers(Y), CitationDB(Y,X)
Q'(X) :- papers(X), AwardDB(X).

The key in constructing the program is to define a new intermediate relation papers whose extension is the set of all papers reachable by citation chains from papers in databases, and is defined by a transitive closure over the view CitationDB. In [20] it is shown that a maximally-contained rewriting of the query using the views can always be obtained with a recursive rewriting, when the views are conjunctive and don't contain comparison predicates. In [25] and [34] the authors describe additional optimizations to this basic algorithm. Additional cases in which recursive rewritings may be needed are when the query is recursive, in the presence of functional dependencies on the database schema [20], when views contain unions [3] (though even recursion is does not always suffice here), and the case where additional semantic information about class hierarchies on objects is expressed using description logics [6].

# 4 Query answering

If $Q'$ is an *equivalent-rewriting* of a query $Q$ using the views $\mathcal{V}$, then it will always produce the same result as $Q$, independent of the state of the database or of the views. In particular, this means that $Q'$ will always produce all the certain answers to $Q$ for any possible database. Recall that an answer to $Q$ is certain given the extensions $v_1, \ldots, v_n$ of the views $V_1, \ldots, V_n$, if it would be an answer of $Q$ for *any* database that would give rise to those view extensions.

When $Q'$ is a *maximally-contained rewriting* of $Q$ using the views $\mathcal{V}$ it may produce only a subset of the answers of $Q$ for a given state of the database. The maximality of $Q'$ is defined only w.r.t. the other possible rewritings in a particular query language $\mathcal{L}$ that we consider for $Q'$. Hence, the question that remains is how to find all the certain answers, whether we do it by applying some rewritten query to the views or by some other algorithm.

The question of finding all the certain answers is considered in detail in [1, 26]. In their analysis they distinguish the case of the *open-world assumption* from that of the *closed-world assumption*. With the closed-world assumption, the extensions of the views are assumed to contain *all* the tuples that would result from applying the view definition to the database. Under the open-world assumption, the extensions of the views may be missing tuples (but they may not have incorrect tuples).

In [1] it is shown that under the open-world assumption, in many practical cases, finding all the certain answers can be done in polynomial time. In these cases, the certain answers are found by applying the maximally-contained rewriting to the extensions of the views. However, the problem becomes co-NP-hard as soon as we allow union in the language for defining the views, or allow the predicate $\neq$ in the language defining the query.

Under the closed-world assumption the situation is worse. Even when both the views and the query are defined by conjunctive queries without comparison predicates, the problem of finding all certain answers is already co-NP-hard.

It is interesting to note the connection established in [1] between the problem of finding all certain answers and computation with conditional tables [31]. The partial information about the database that is available from a set of views can be encoded as a conditional table using the formalism studied in [31]. This connection also points at an important property of using views as a formalism for describing data sources, specifically, the ability of the formalism to capture partial information about data sources.

The work in [26] also considers the case where the views may either be incomplete, complete, or contain incorrect tuples. It is shown that without comparison predicates in the views or the query, when either all the views are complete or all of them may contain incorrect tuples, finding all certain answers can be done in polynomial time in the size of the view extensions. In other cases, the problem is co-NP-hard.

Finally, [39] considers the problem of *relative query containment*: is the set of certain answers of a query $Q_1$ always contained in the set of certain answers of a query $Q_2$. The paper shows that for the conjunctive queries and views with no comparison predicates the problem is $\Pi_2^p$-complete, and that the problem is still decidable in the presence of access pattern limitations.

# 5 Query language extensions

We briefly discuss results on answering queries using views for several query language extensions.

**Semi-structured data:** The emergence of XML as a standard for sharing data on the WWW has spurred significant interest in building systems for integrating XML data from multiple sources. The emerging formalisms for modeling XML data are variations on labeled directed graphs, which have also been used to model semi-structured data [2]. The model of labeled directed graphs is especially well suited for modeling the irregularity and the lack of schema which are inherent in XML data.

Several works have started considering the problem of answering queries using views when the views and queries are expressed in a language for querying semi-structured data. There are two main difficulties that arise in this context. First, such query languages enable using *regular path expressions* in the query, to express navigational queries over data whose structure is not well known a priori. Regular path expressions essentially provide a very limited kind of recursion in the query language. In [9] the authors consider the problem of rewriting a regular path query using a set of regular path views, and show that the problem is in 2EXPTIME (and checking whether the rewriting is an equivalent one is in 2EXPSPACE). In [10] the authors consider the problem of finding all the certain answers when queries and views are expressed using regular path expressions, and show that the problem is co-NP-complete when data complexity (i.e., size of the view extensions) is considered. In [11] the authors extend the results of [9, 10] to path expressions that include the inverse operator, allowing both forward and backward traversals in a graph.

The second problem that arises in this context stems from the rich restructuring capabilities which enable the creation of arbitrary graphs in the output. The output graphs can also include nodes that did not exist in the input data. In [40] the authors consider the rewriting problem in the case where the query can create *answer trees*, and queries do not involve regular path expressions with recursion.

**Infinite number of views:** Two works have considered the problem of answering queries using views in the presence of an *infinite* number of views [37, 44]. The motivation for this seemingly curious problem is that when a data source has the capability to perform *local* processing, it can be modeled by a (possibly infinite) set of views it can supply, rather than a single one. Hence, to answer queries using such sources, one need not only choose which sources to query, but also which query to send to it out of the set of possible queries it can answer. In [37, 44] it is shown that in certain important cases the problem of answering a query using an infinite set of views are decidable. Of particular note is the case in which the set of views that a source can answer is described by the finite unfoldings of a datalog program.

**Description Logics:** Description logics are a family of logics for modeling complex hierarchical structures (see [8] for a survey). A description logic enables to define sets of objects by specifying their properties, and then to reason about the relationship between these sets (e.g., subsumption, disjointness). Several works have considered the problem of answering queries using views when description logics are used to model the domain. In [6] it is shown that in general, answering queries using views in this context may be NP-hard, and presents cases in which we can obtain a maximally-contained rewriting of a query in recursive datalog. The complexity of answering queries using views for an expressive description logic (which also includes n-ary relations) is studied in [12].

**Other extensions:** In [4, 21] the authors consider the rewriting problem when the views may contain unions. The presence of inclusion dependencies on the database relations introduces several subtleties to the query rewriting problem, which are considered in [29]. Several works [16, 27, 28] consider the formal aspects of answering queries using views in the presence of grouping and aggregation. They present cases in which it can be shown that a rewriting algorithm can be complete, in the sense that it will find a rewriting if one exists. Their algorithms are based on insights into the problem of query containment for queries with grouping and aggregation. In [38], the author considers the query rewriting for a language that enables querying the schema and data uniformly, and hence, names of attributes in the data may become constants in the extensions of the views.

# 6 Conclusions

The problem of answering queries using views raises a multitude of challenges, ranging from theoretical foundations to considerations of a more practical nature. While algorithms for answering queries using views are already being incorporated into commercial database systems (e.g., [7, 46]), these algorithms will have even more importance in data integration systems and data warehouse design.

There are many issues that remain open in this realm. Of particular note are studying rewriting algorithms in the presence of a wider class of integrity constraints on both the database and view relations, and studying the effect of restructuring capabilities of query languages for querying semi-structured data.

Finally, I believe that the next challenge is to develop algorithms for *selecting* views to materialize in a data warehouse, web site, or environment in which data is spread over multiple (possibly mobile) devices (e.g., [32]). Even though there has been work on this problem, the research is still in its infancy. The wealth of techniques developed for answering queries using views will be key to developments in this realm.

## Acknowledgments

I would like to thank Phil Bernstein, Mike Carey, Leonid Li kin, Todd Millstein and Rachel Pottinger for valua le comments on this paper. I would like to acknowledge the support of a Sloan Fellowship and NSF Grants #IIS-9978567 and #IIS-9985114.

# References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS*, Seattle, WA, 1998.

[2] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web.* Morgan Kaufmann, 1999.

[3] Foto Afrati. Personal communication, 2000.

[4] Foto Afrati, M. Gergatsoulis, and Th. Kavalieros. Answering queries using materialized views with disjunctions. In *Proc. of ICDT*, 1999.

[5] Sanjay Agrawal, Surajit Chaudhuri, and Vivek Narasayya. Automated selection of materialized views and indexes in Microsoft SQL Server. In *Proc. of VLDB*, Cairo, Egypt, 2000.

[6] Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset. Rewriting queries using views in description logics. In *Proc. of PODS*, Tucson, Arizona., 1997.

[7] Randall Bello, Karl Dias, Alan Downing, James Feenan, Jim Finnerty, William Norcott, Harry Sun, Andrew

Witkowski, and Mohamed Ziauddin. Materialized views in Oracle. In *Proc. of VLDB*, pages 659–664, 1998.

[8] Alex Borgida. Description logics in data management. *IEEE Trans. on Know. and Data Engineering*, 7(5):671–682, 1995.

[9] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. of PODS*, 1999.

[10] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. Answering regular path queries using views. In *Proc. of ICDE*, 2000.

[11] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Vardi. View-based query processing for regular path queries with inverse. In *Proc. of PODS*, 2000.

[12] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views in description logics. In *Working notes of the KRDB Workshop*, 1999.

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

[14] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *Proc. of ICDT*, Delphi, Greece, 1997.

[15] Rada Chirkova and Michael Genesereth. Linearly bounded reformulations of conjunctive databases. In *Proc. of DOOD*, 2000.

[16] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. of PODS*, pages 155–166, 1999.

[17] Shaul Dar, Michael J. Franklin, Bjorn Jonsson, Divesh Srivastava, and Michael Tan. Semantic data caching and replacement. In *Proc. of VLDB*, pages 330–341, 1996.

[18] Jan Van den Bussche. Two remarks on the complexity of answering queries using views. *To appear in Information Processing Letters*, 2000.

[19] Alin Deutsch, Lucian Popa, and Val Tannen. Physical data independence, constraints and optimization with universal plans. In *Proc. of VLDB*, 1999.

[20] Oliver Duschka, Michael Genesereth, and Alon Levy. Recursive query plans for data integration. newblock *Journal of Logic Programming*, 43(1):49–73, 2000.

[21] Oliver M. Duschka. *Query Planning and Optimization in Information Integration*. PhD thesis, Stanford University, Stanford, California, 1998.

[22] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. of PODS*, Tucson, Arizona., 1997.

[23] Daniela Florescu, Alon Levy, Dan Suciu, and Khaled Yagoub. Optimization of run-time management of data intensive web sites. In *Proc. of VLDB*, 1999.

[24] Daniela D. Florescu. *Search Spaces for Object-Oriented Query Optimization*. PhD thesis, Univerisity of Paris VI, France, 1996.

[25] M. Friedman and D. Weld. Efficient execution of information gathering plans. In *Proceedings of the International Joint Conference on Artificial Intelligence, Nagoya, Japan*, 1997.

[26] Gosta Grahne and Alberto O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proc. of ICDT*, 1999.

[27] Stephane Grumbach, Maurizio Rafanelli, and Leonardo Tininini. Querying aggregate data. In *Proc. of PODS*, pages 174–184, 1999.

[28] Stephane Grumbach and Leonardo Tininini. On the content of materialzed aggregate views. In *Proc. of PODS*, 2000.

[29] Jarek Gryz. Query folding with inclusion dependencies. In *Proc. of ICDE*, Orlando, Florida, 1998.

[30] Alon Y. Halevy. Answering queries using views: A survey. To appear in the VLDB Journal, 2000.

[31] T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[32] Z. Ives, A. Levy, J. Madhavan, R. Pottinger, S. Saroiu, I. Tatarinov, E. Jaslikowska, and T. Yeung. Self-organizing data sharing communities with SAGRES: System demonstration. In *Proc. of SIGMOD*, 2000.

[33] Chung T. Kwok and Daniel S. Weld. Planning to gather information. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, 1996.

[34] Eric Lambrecht, Subbarao Kambhampati, and Senthil Gnanaprakasam. Optimizing recursive information gathering plans. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1999.

[35] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proc. of PODS*, San Jose, CA, 1995.

[36] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, Bombay, India, 1996.

[37] Alon Y. Levy, Anand Rajaraman, and Jeffrey D. Ullman. Answering queries using limited external processors. In *Proc. of PODS*, Montreal, Canada, 1996.

[38] R. J. Miller. Using schematically heterogeneous structures. In *Proc. of SIGMOD*, pages 189–200, Seattle, WA, 1998.

[39] Todd Millstein, Alon Levy, and Marc Friedman. Query containment for data integration systems. In *Proc. of PODS*, Dallas, Texas, 2000.

[40] Yannis Papakonstantinou and Vasilis Vassalos. Query rewriting for semi-structured data. In *Proc. of SIGMOD*, 1999.

[41] Rachel Pottinger and Alon Levy. A scalable algorithm for answering queries using views. In *Proc. of VLDB*, Cairo, Egypt, 2000.

[42] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of PODS*, San Jose, CA, 1995.

[43] Odysseas G. Tsatalos, Marvin H. Solomon, and Yannis E. Ioannidis. The GMAP: A versatile tool for physical data independence. *VLDB Journal*, 5(2):101–118, 1996.

[44] Vasilis Vassalos and Yannis Papakonstantinou. Describing and using query capabilities of heterogeneous sources. In *Proc. of VLDB*, pages 256–265, Athens, Greece, 1997.

[45] H. Z. Yang and P. A. Larson. Query transformation for PSJ-queries. In *Proc. of VLDB*, pages 245–254, Brighton, England, 1987.

[46] Markos Zaharioudakis, Roberta Cochrane, George Lapis, Hamid Pirahesh, and Monica Urata. Answering complex SQL queries using automatic summary tables. In *Proc. of SIGMOD*, 2000.