

Query Rewriting Using Views in the Presence of Inclusion Dependencies

Qingyuan Bai Jun Hong Michael F. McTear

School of Computing and Mathematics,
University of Ulster at Jordanstown,
Newtownabbey, Co. Antrim BT37 0QB, UK
{q.bai, j.hong, m.mctear}@ulster.ac.uk

ABSTRACT

Query rewriting using views is an essential issue in data integration. A number of algorithms, e.g., the bucket algorithm, the inverse rules algorithm, the SVB algorithm and the MiniCon algorithm, have been proposed to address this issue. These algorithms can be divided into two categories: bucket-based algorithms and inverse rule-based algorithms. Some inverse rule-based algorithms have considered the problem of query rewriting in the presence of inclusion dependencies. However, there has been no bucket-based algorithm so far for the problem. All the previous bucket-based algorithms may miss query rewritings in the presence of inclusion dependencies. In this paper, we extend the MiniCon algorithm to the presence of inclusion dependencies. In the MiniCon algorithm, a view can be used in a non-redundant rewriting of a query only if at least one subgoal in the query is covered by a subgoal in the view. In the presence of inclusion dependencies, when no subgoal in a view directly covers the query subgoal we can apply the chase procedure and rule to the subgoals of the query or view that contains the chase reachable subgoals to get a revised query or view. The condition required by the MiniCon algorithm is then satisfied. We can therefore avoid the problem of missing rewritings with the previous bucket-based algorithms. We prove that our extended algorithm can find the maximally-contained rewriting of a conjunctive query using a set of conjunctive views in the presence of inclusion dependencies. Our extension of the MiniCon algorithm does not involve a significant increase in computational complexity and our new algorithm remains scalable.

Categories and Subject Descriptors

H.2.4 [Information Systems]: Systems - *query processing*;

H.2.5 [Information Systems]: Heterogeneous Databases

General Terms: Algorithms

Keywords: Query rewriting using views, inclusion dependencies, data integration systems.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WIDM'03, November 7–8, 2003, New Orleans, Louisiana, USA.
Copyright 2003 ACM 1-58113-725-7/03/0011...\$5.00.

1. INTRODUCTION

Query rewriting using views is an essential issue in data integration. There are two main approaches to addressing this issue, i.e., Global As View (GAV) and Local As View (LAV). In the LAV approach, data sources are described by views that are defined over a mediated schema, and user queries are also posed in terms of the mediated schema. The LAV approach to query rewriting is closely related to the problem of answering queries using views, which has recently received considerable attention due to its relevance to a wide variety of database applications [9].

In data integration, views describe a set of autonomous heterogeneous data sources. In this context we usually try to find the maximally-contained rewriting that provides the best possible answers for a given query. Many algorithms including the bucket algorithm [10,11], the inverse rules algorithm [2,14], the Shared Variables Bucket (SVB) algorithm [12], and the MiniCon algorithm [13], have been proposed. These algorithms can be divided into two groups, bucket-based algorithms and inverse rules-based algorithms. The algorithms based on inverse rules [6,7] have considered the problem of query rewriting in the presence of inclusion dependencies in a mediated schema. However, there has been no bucket-based algorithm for the problem. The previous bucket-based algorithms can generate the maximally-contained query rewritings in the absence of inclusion dependencies. However, they might sometimes miss query rewritings in the presence of inclusion dependencies.

Example 1 (adapted from [7]): Assume that a mediated schema consists of the relations: *patient*(name, dob, address, insurer), *procedure*(patient_name, physician_name, procedure_name, time), *insurer*(company, address, phone), *event*(event_name, description, patient_name, location).

In addition, assume that all procedures with patients' names are also recorded in the *event* table, that is, the following inclusion dependency holds:

$$\text{procedure}(\text{procedure_name}, \text{patient_name}) \subseteq \text{event}(\text{event_name}, \text{patient_name}).$$

There are two views:

$$V_1(X,Z,W) :- \text{procedure}(X,U_1,Z,W).$$
$$V_2(X,U,S) :- \text{patient}(X,S_1,U,S_2), \text{insurer}(S_2,S,S_3).$$

Suppose that there is a query:

$$Q(E) :- \text{patient}(W_0,W_1,W_2,W_3), \text{event}(E,W_4, W_0,W_5).$$

o previous bucket-based rewriting algorithms can generate any rewriting of Q because no view contains the *event* relation. However, we can generate the following rewriting of Q in the presence of the inclusion dependency:

$$Q'(E) :- V_1(W_0, E, W), V_2(W_0, W_2, S).$$

In [13], experimental results show that the MiniCon algorithm, which is based on use of buckets, is more efficient than any algorithm based on use of inverse rules.

In this paper, we focus on the extension of the MiniCon algorithm to deal with the presence of inclusion dependencies in a mediated schema. In the MiniCon algorithm, a view can be used in a non-redundant rewriting of a query only if at least one subgoal in the query is covered by a subgoal in the view. In the presence of inclusion dependencies, when no subgoal in a view directly covers the query subgoal we can apply the chase procedure and rule to the subgoals of the query or view that contains the chase reachable subgoals to get a revised query or view. The condition required by the MiniCon algorithm is then satisfied. We can therefore avoid the problem of missing rewritings with the previous bucket-based algorithms. We prove that our extended algorithm can find the maximally-contained rewriting of a conjunctive query using a set of conjunctive views in the presence of inclusion dependencies. Our extension of the MiniCon algorithm does not involve a significant increase in computational complexity and our new algorithm remains scalable.

The rest of the paper is organized as follows. In Section 2, we formally define the problem. In Section 3, we present our algorithm to handle query rewritings in the presence of inclusion dependencies. We analyze computational complexity of our algorithm and prove its correctness. In Section 4, we have a brief look at the previous algorithms. We finally conclude and discuss our future work.

2. PRELIMINARIES

2.1 Query Containment and Answering Queries Using Views

We consider the problem of answering conjunctive queries using views. A conjunctive query has the form:

$$Q(\bar{X}) :- R_1(\bar{X}_1), \dots, R_k(\bar{X}_k)$$

where $R_1(\bar{X}_1), \dots, R_k(\bar{X}_k)$ are database relations. We require

that the query be safe, i.e., $\bar{X} \subseteq \bar{X}_1 \cup \dots \cup \bar{X}_k$. The variables in

\bar{X} are the distinguished variables, and the others are existential variables. A view is a named query.

We say that a query Q_1 is contained in the Q_2 , denoted by $Q_1 \subseteq Q_2$, if the answers to Q_1 are a subset of the answers to Q_2 for any database instance. The containment mapping provides a necessary and sufficient condition for testing query containment. A mapping ϕ from the variables of Q_2 to the variables of Q_1 is a containment mapping if (1) ϕ maps every subgoal in the body of Q_2 to a subgoal in the body of Q_1 , and (2) ϕ maps the head of Q_2 to the head of Q_1 . The query Q_2 contains Q_1 if and only if there is a containment mapping from Q_2 to Q_1 .

Given a query Q and a set of view definitions $\mathbf{V} = \{V_1, \dots, V_n\}$, a rewriting of Q using these views is a query expression Q' whose body predicates are only those in \mathbf{V} . Note that the views are not assumed to contain all the tuples in their definitions since the data sources are autonomous. Moreover, we cannot always find an equivalent rewriting of the query using views because data sources may not contain all the answers to the query. Instead, we consider the problem of finding maximally-contained rewritings.

Definition 2.1 (Maximally-contained rewriting) Q' is a maximally-contained rewriting of a query Q using the views $\mathbf{V} = V_1, \dots, V_n$ with respect to a query language \mathbf{L} if

- (1) Q' is a rewriting of Q such that $Q' \subseteq Q$,
- (2) there is no other rewriting Q_1 of Q , such that $Q_1 \subseteq Q$ and $Q' \subset Q_1$.

2.2 The MiniCon Algorithm([13])

Since we aim to extend the MiniCon algorithm in the presence of inclusion dependencies, let us first have a brief look at it. In the MiniCon algorithm, MiniCon Descriptions (MCDs) contain the unification information between a given query Q and a view V , and take the roles of buckets in the bucket algorithm. The unification information in a MCD over a view shows that what subgoals G in Q are covered by the corresponding subgoals in the view. The unification is done by finding a partial query containment mapping. The advantage of the MiniCon algorithm over the bucket algorithm is that the shared variables, which appear at least twice in the body of Q , are considered when unifying. For example, suppose that we have a query:

$$Q(X) :- R(X, Y), S(Y, Z)$$

and there are two views:

$$V_1(U) :- R(U, V), T(V). \\ V_2(Z) :- S(W, Z).$$

where, R, S, T are relations in a mediated schema.

When unifying Q with V_1 , the MiniCon algorithm tries to find the containment mapping from Q to V_1 but none of subgoals in Q are covered by V_1 because there is a shared variable Y in Q . Without considering the effects of shared variables, the bucket algorithm generates the following candidate query rewriting:

$$Q'(X) :- V_1(X), V_2(Z).$$

which is not a valid one. The MiniCon algorithm, however, will not generate this invalid query rewriting.

After all the possible MCDs are formed, the MiniCon algorithm generates all the query rewritings by combining views in a subset of MCDs if all the subgoals in Q can be covered by the union of the corresponding and disjoint subgoals in the MCDs. Thus the second advantage of the MiniCon algorithm over the bucket algorithm is that it first tries to find a set of minimum MCDs $C_i, i=1, \dots, k$, such that $i \neq j, G_i \cap G_j = \emptyset$, and $G_1 \cup G_2 \cup \dots \cup G_k = \text{subgoals}(Q)$, before combining $V_i, i=1, \dots, k$. This guarantees that none of the query rewritings generated is redundant and the union of all the generated query rewritings is the maximally-contained rewriting.

2.3 Inclusion Dependencies

An m -ary inclusion dependency (IND) is a formal statement of the form $R[A_1, \dots, A_m] \subseteq S[B_1, \dots, B_m]$ where R and S are relation schemes, A_1, \dots, A_m and B_1, \dots, B_m are attributes in R and S respectively. A database obeys the IND $R[A_1, \dots, A_m] \subseteq S[B_1, \dots, B_m]$ if for every tuple $\langle a_1, \dots, a_m \rangle$ that occurs in columns A_1, \dots, A_m of relation R , there is a tuple of relation S that contains $\langle a_1, \dots, a_m \rangle$ in columns B_1, \dots, B_m . Sets of inclusion dependencies are denoted by Δ .

When the relations in a schema satisfy a set of inclusion dependencies Δ , we define our notion of query containment as query containment relative to Δ . Query Q_1 is contained in query Q_2 relative to Δ , denoted $Q_1 \subseteq_{\Delta} Q_2$, if for each database D satisfying the inclusion dependencies in Δ , $Q_1(D) \subseteq Q_2(D)$. Maximal query containment relative to Δ can be defined accordingly.

3. QUERY REWRITING USING VIEWS IN THE PRESENCE OF INCLUSION DEPENDENCIES

Now we consider the problem of query rewriting using views in the presence of inclusion dependencies. In the context of databases, inclusion dependencies sometimes exist in the database schema. These dependencies provide special relationships between different relations in the schema. As stated above, the data sources/views in data integration systems are defined over the relations in a mediated schema. As a result, these dependencies could also reveal some relationships between different data sources/views. Thus, this issue has practical significance.

3.1 Chase Procedure and Rule of Inclusion Dependencies

We first give two definitions on the relations between inclusion dependencies and queries. Note that views are named queries. Therefore, these definitions are applicable to views as well.

Definition 3.1 (The Chase $_{\Delta}$ Procedure and Rule [8])

If an inclusion dependency $R[A_1, \dots, A_m] \subseteq S[B_1, \dots, B_m]$ is in Δ , and $R(X_1, \dots, X_M)$, $M \geq m$, is a subgoal of a query Q , and $S(Y_1, \dots, Y_N)$, $N \geq m$, does not appear in Q , we say this inclusion dependency is applicable. The chase $_{\Delta}$ rule is to add a new subgoal $S(Y_1, \dots, Y_N)$ to Q , where $S[B_1, \dots, B_m] = R[A_1, \dots, A_m]$ and for $S[C]$, $C \neq B_i$, $1 \leq i \leq m$, is a new variable that does not appear in Q .

Definition 3.2 (Δ -Equivalence [7])

Let Q and Q' be queries. Q is chase $_{\Delta}$ reachable from Q' if $Q \equiv Q'$ or there exist Q_1, \dots, Q_n , such that:

$Q \equiv Q_1$, Q_{i+1} is derived from Q_i , $1 \leq i \leq n-1$, by applying a single chase $_{\Delta}$ step to it, and $Q \equiv Q_n$.

3.2 Further Analysis of the MiniCon Algorithm with an Illustrative Example

Given a conjunctive query Q and a set of views V_1, \dots, V_n , the MiniCon algorithm first tries to form a MCD for Q over each of these views V_i , $i=1, 2, \dots, n$, by unifying subgoals in Q with the

corresponding subgoals in V_i . It fails to form a MCD for Q over a view V_i when one of the following cases occurs:

- (1) No subgoal in Q appears in V_i ;
- (2) There is a violation in containment mapping, for example, a distinguished variable of Q can not be mapped to a distinguished variable of V_i ;
- (3) There is an inconsistency between built-in predicates in Q and ones in V_i if there exist any built-in predicates in Q and V_i .

Regarding the above case (1), even though no subgoal in a query appears in any view, we can still make use of inclusion dependencies to get a revised view or query so that some query subgoals can be mapped to the corresponding view subgoals. As shown in Example 1, though the above case (1) occurs, a correct rewriting still exists. We now use Example 1 to show how the correct rewriting that is missed by the MiniCon algorithm can be generated in the presence of inclusion dependencies.

Continue with Example 1, view V_1 is Δ -equivalent to the following view by applying a single chase $_{\Delta}$ step:

$V'_1(X, Z, W) :- \text{procedure}(X, U_3, Z, W), \text{event}(Z, D, X, L)$.

Using the MiniCon algorithm with V'_1 (instead of V_1) and V_2 , we can generate the following MCDs shown in Table 1, where \rightarrow_m denotes a mapping, H denotes a head homomorphism on a view, ϕ denotes a containment mapping from Q to the view, and G denotes the set of subgoals covered by the view:

Table 1: The MCDs for Q over the chased views.

View	H	ϕ	G
$V'_1(X, Z, W)$	$X \rightarrow_m X, Z \rightarrow_m Z, W \rightarrow_m W$	$E \rightarrow_m Z, W_4 \rightarrow_m D, W_0 \rightarrow_m X, W_5 \rightarrow_m L$	event
$V_2(X, U, S)$	$X \rightarrow_m X, U \rightarrow_m U, S \rightarrow_m S$	$W_0 \rightarrow_m X, W_1 \rightarrow_m S_1, W_2 \rightarrow_m U, W_3 \rightarrow_m S_2$	patient

We can then generate the following rewriting of Q :

$Q_{\Delta}(E) :- V'_1(W_0, E, W), V_2(W_0, W_2, S)$.

It can be proved that the above rewriting is the maximally-contained rewriting. This follows from the MiniCon algorithm. It can also be proved that $V'_1(X, Z, W)$ is Δ -equivalent to $V_1(X, Z, W)$. This follows from Definitions 3.1 and 3.2. \square

3.3 Query Rewriting Algorithm

Our algorithm proceeds in two stages. First, for each subgoal of Q , we check whether MCDs for Q over views can be formed. If so, these MCDs will be used to generate possible query rewritings in the same way as in the MiniCon algorithm. Otherwise, we check whether Case (1) occurs. If so, we apply the chase $_{\Delta}$ procedure and rule to Q or some views to get Δ -equivalent query or Δ -equivalent views if we can make use of inclusion dependencies. Second, we apply the MiniCon algorithm to these Δ -equivalent query or Δ -equivalent views to form the corresponding MCDs.

Algorithm: Query Rewriting Using Views in the Presence of Inclusion Dependencies

Input: A conjunctive query Q , a set of the views V_1, V_2, \dots, V_n , a set of inclusion dependencies Δ in the form of $R[A_1, \dots, A_m] \subseteq S[B_1, \dots, B_m]$.

Output: Q_Δ 's, whose union is the maximally-contained rewriting of Q relative to inclusion dependency Δ .

Method:

Step 1: Forming the MCDs using the MiniCon Algorithm

For each subgoal of Q , form MCDs over each view. If Case (1) occurs, go to Step 2.

Step 2: Forming the MCDs in the presence of inclusion dependencies

Note that if Q contains neither R nor S in any inclusion dependencies in Δ , then none of the inclusion dependencies could be used. Thus, only the following three cases need to be considered: Q contains R , S or both, where R and S are the left side and right side of an inclusion dependency respectively.

For each subgoal R_i , $1 \leq i \leq k$, in Q , we check whether R_i is R or S , where R and S are the relations in an inclusion dependency $R[A_1, \dots, A_m] \subseteq S[B_1, \dots, B_m]$ in Δ .

Case 1: $R_i = R$ // The chase_Δ procedure can be applied to Q .

For each view V_i , $1 \leq i \leq n$, check whether V_i contains R or S .

If V_i contains R , then V_i has been processed in Step 1,

Else if V_i contains S , then

(1) get $Q' = \text{chase}_\Delta(Q)$ by applying the chase_Δ procedure to Q ;

(2) form MCDs over Q' using the MiniCon algorithm.

Case 2: $R_i = S$ // The chase_Δ procedure cannot be applied to Q .

For each view V_i , $1 \leq i \leq n$, check whether V_i contains R or S .

If V_i contains R , then

(1) get $V'_i = \text{chase}_\Delta(V_i)$ by applying the chase_Δ procedure to V_i ;

(2) form MCDs using the MiniCon algorithm.

Else if V_i contains S , V_i has been processed in Step 1.

Case 3: Q contains both R and S .

If necessary, remove S from Q , which is called the minimum of Q in [7], and then Case 3 is changed into Case 1.

Step 3: Generating the maximally-contained rewriting using the MCDs formed in Steps 1 and 2.

The MCDs formed in Steps 1 and 2 are used to generate query rewritings in the same way as in the MiniCon algorithm [13]. Then V'_i is replaced with V_i in the obtained query rewritings if V'_i is obtained from V as a result of Case 2 in Step 2. The union of all the query rewritings generated is the maximally-contained rewritings relative to inclusion dependencies Δ .

Note that when Case 1 in Step 2 occurs, the containment mapping is considered from the chased query Q' to view V_i . When generating query rewritings, subgoals R and S in Q' should be considered as the same.

3.4 Computational Complexity and Correctness of Our Algorithm

3.4.1 Computational Complexity of Our Algorithm

The computational complexity in Step 1 of our algorithm is the same as the MiniCon algorithm. In Step 2 of our algorithm, the computation consists of two parts. The first part of computation is to check whether the chase_Δ procedure can be applied to Q or some views. In the worst case, the computational complexity of

this process is $|Q| * |\Delta| * n$, where $|Q|$, $|\Delta|$, and n are the number of subgoals in Q , the number of inclusion dependencies in Δ and the number of views respectively. The second part of computation is to form MCDs for the chased query or views, which is the same as the MiniCon algorithm. The computation in Step 3 of our algorithm is also the same as the MiniCon algorithm. Thus, the total increased computational complexity in our algorithm has only polynomial time complexity, i.e., $|Q| * |\Delta| * n$.

3.4.2 Correctness of Our Algorithm

From the correctness of the MiniCon algorithm, it follows that the union of the query rewritings obtained in our algorithm is the maximally-contained rewriting relative to inclusion dependencies Δ , because the original views and Q are Δ -equivalent to those new views and $\text{chase}_\Delta(Q)$ obtained by a single chase_Δ step. \square

In Example 1, we have shown the process of Case 2 in our algorithm. Example 2 below shows the process of Case 1 in our algorithm.

Example 2: Assume that a mediated schema consists of the relations: $R(A_1, A_2, A_3)$, $S(A_4, A_5)$, and $T(A_1, A_6)$, and there is an inclusion dependency $\{R[A_3] \subseteq S[A_4]\}$.

There are two views:

$V_1(W) :- S(W, W_2)$.

$V_2(U) :- T(U, Z)$.

Suppose that there is a query:

$Q(X) :- R(X, W_0, W_1), T(X, Y)$

By using the MiniCon algorithm, we can immediately form a MCD for subgoal T in Q over V_2 . By applying a single chase_Δ step to Q , we have:

$Q'(X) :- R(X, W_0, W_1), S(W_1, W_3), T(X, Y)$.

Thus, we can form a MCD for subgoal S in Q' over V_1 . All the MCDs are as shown in Table 2.

Table 2: The MCDs for the revised Q over the views.

View	H	ϕ	G
$V_1(W)$	$W \rightarrow_m W$	$W_1 \rightarrow_m W, W_3 \rightarrow_m W_2$	S
$V_2(U)$	$U \rightarrow_m U$	$X \rightarrow_m U, Y \rightarrow_m Z$	T

We can then get the following query rewriting:

$Q_\Delta(X) :- V_1(W_1), V_2(X)$.

The MiniCon algorithm cannot generate such a query rewriting because it does not take into account the existence of inclusion dependencies. \square

4. RELATED WORK

A comprehensive survey on query rewriting using views can be found in [9]. In this section, we have a brief look at the related algorithms.

4.1 Algorithms Based on Use of Buckets

The bucket algorithm [10,11] proceeds in two stages. Initially, a bucket is created for each subgoal of a query. A view is put in the bucket for a subgoal if it can be unified with the subgoal in

the query. Next, candidate query plans are generated by picking one view from each bucket, which are then verified using containment tests.

In the SVB algorithm [12], given a query Q , two types of buckets are created. The first type of bucket is called single-subgoal bucket, which is built in the same way as in the bucket algorithm. The second type of bucket is called shared-variable bucket, which is created by checking the containment mapping from a set of subgoals in Q containing a shared variable to some subgoals in a view. Once all the buckets are created, the algorithm constructs rewritings by combining views from buckets that contain disjoint sets of subgoals of Q .

The best bucket-based algorithm is the MiniCon algorithm that has been described in Section 2. In [13], it is shown that the MiniCon algorithm significantly outperforms both the bucket and SVB algorithms and can be scaled up to hundreds of views. All the bucket-based algorithms have not considered query rewriting using views in the presence of inclusion dependencies. Our algorithm extends the MiniCon algorithm, in which the presence of inclusion dependencies has been dealt with while retaining the main characteristics of the MiniCon algorithm. Our extension of the MiniCon algorithm does not involve a significant increase in computational complexity and remains scalable.

4.2 Algorithms Based on Use of Inverse Rules

The key idea underlying the inverse rules algorithm [2,14] is to first construct a set of rules called inverse rules that invert the view definitions, and then replace existential variables in the view definitions with Skolem functions in the heads of the inverse rules. The rewriting of a query Q using the set of views V is simply the composition of Q and the inverse rules for V by using the transformation method in [2] or the u-join method in [14].

So far, some rewriting algorithms based on use of inverse rules have been proposed for query rewriting in the presence of inclusion dependencies [6,7]. However, the issue addressed in [6,7] is different from what we have focused on in this paper. For instance, in [7] the issue addressed is mainly about how to generate new queries that are Δ -equivalent to or Δ -contained in the original query. Not only is our algorithm concerned with generating an Δ -equivalent or Δ -contained query from a query that contains chase_Δ reachable subgoals but also views that contain chase_Δ reachable subgoals if no Δ -equivalent or Δ -contained query can be generated. We apply the chase procedure to either the query or a view only once when appropriate instead of more than once as in [7].

5. CONCLUSIONS

As stated in Section 3, the presence of inclusion dependencies can be exploited to avoid the problem of missing query rewritings in the previous bucket-based algorithms. Based on this observation we have extended the MiniCon algorithm. We have analyzed the MiniCon algorithm and shown that the problem of missing query rewritings can be avoided when the presence of inclusion dependencies is taken into account. We have described how to apply the chase procedure to either a

query or a view. This enables our algorithm to form the MCDs that cannot be formed by the MiniCon algorithm.

Applying semantic query optimization to query rewriting using views is another important issue. The issue focuses on how to exploit semantic constraints in a mediated schema. There has been some work to address this issue using logic and inverse rules approaches. In future work, we intend to address the issue on query rewriting using views in the presence of semantic constraints in a mediated schema using a bucket-based approach.

REFERENCES

1. S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley Publishing Company, 1995.
2. O. M. Duschka, M. R. Genesereth, Answering Recursive Queries Using Views, In Proc. of 16th ACM Conference on Principles of Database Systems, PODS, Tucson, AZ, May 1997.
3. O. M. Duschka, M. R. Genesereth, A. Y. Levy, Recursive Query Plans for Data Integration. Journal of Logic Programming, special issue on Logic Based Heterogeneous Information Systems, 43(1),49-73, 2000.
4. J. Grant, J. Minker, A Logic-based Approach to Data Integration, TLP 2(3):323-368, 2002.
5. J. Gryz, An Algorithm for Query Folding with Inclusion Dependencies. Intelligent Information Systems VII Proceedings of the Workshop, Malbork, Poland, June, 1998.
6. J. Gryz, Query Folding with Inclusion Dependencies. In Proceedings of ICDE'98, Orlando, Florida, USA. IEEE Computer Society 1998, ISBN 0-8186-8289-2.
7. J. Gryz, Query Rewriting Using Views in the Presence of Functional and Inclusion Dependencies. Information Systems, 1999, 24(7):597-612.
8. D. S. Johnson, A. Klug, Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. Journal of Computer and System Sciences, 28:167-189 (1984).
9. A. Y. Levy, Answering Queries Using Views: A Survey. The VLDB Journal, 2001.
10. A. Y. Levy, A. Rajaraman, J. J. Ordille, Querying Heterogeneous Information Sources Using Source Descriptions. In Proceedings of the 22nd VLDB Conference, 1996.
11. A. Y. Levy, A. Rajaraman, J. J. Ordille, Query-Answering Algorithms for Information Agents. In Proceedings of AAAI, 1996.
12. P. Mitra, An Algorithm for Answering Queries Efficiently Using Views. In Proceedings of the 12th Australasian Database Conference, 2001.
13. R. Pottinger., A. Y. Levy.: A Scalable Algorithm for Answering Queries Using Views. In Proc. of the International Conference on Very Large Data Bases (VLDB), 2000.
14. X. Qian., Query folding. In Proceedings of the 12th IEEE International Conference on Data Engineering (ICDE'96), 48-55,1999.
15. J. Wang, M. Maher, R. Topor, Rewriting General Conjunctive Queries Using Views. In Proceedings of 13th Australasian Database Conference (ADC2002), 2002.