IMPROVING VIEW SELECTION IN QUERY REWRITING USING DOMAIN SEMANTICS

Qingyuan Bai[1], Jun Hong[2], and Michael F. McTear[1]
[1]*School of Computing and Mathematics, University of Ulster, Newtownabbey, Co.Antrim, BT37 0QB, UK*

[2]*School of Computer Science, Queen's University of Belfast, Belfast, BT7 1NN, UK*

Keywords: Data integration, query rewriting using views, domain semantics, view selection

Abstract: Query rewriting using views is an important issue in data integration. Several algorithms have been proposed, such as the bucket algorithm, the inverse rules algorithm, the SVB algorithm, and the MiniCon algorithm. These algorithms can be divided into two categories. The algorithms of the first category are based on use of buckets while the ones of the second category are based on use of inverse rules. The bucket-based algorithms have not considered the effects of integrity constraints, such as domain semantics, functional and inclusion dependencies. As a result, they might miss query rewritings or generate redundant query rewritings in the presence of these constraints. A bucket-based algorithm consists of two steps. The first step is called view selection that selects views relevant to a given query and puts the views into the corresponding buckets. The second step is to generate all the possible query rewritings by combining a view from each bucket. In this paper, we consider an improvement of view selection in the bucket-based algorithms using domain semantics. We use the resolution method to generate a pseudo residue for each view given a set of domain semantics. Given a query, the pseudo residue of each view is compared with it and any conflict that exists can be found. As a result, irrelevant views can be removed even before a bucket-based algorithm is used.

1 INTRODUCTION

In a data integration system, data sources are autonomous, distributed, and heterogeneous. Usually, a logical virtual mediated schema is used to make queries and describe the contents of the data source. The actual data is however stored in the data sources. To answer a user query, we need to reformulate it into new queries over the data source schemas in order to get access to the data sources. This process is called query rewriting.

In general, there are two main approaches to query rewriting, i.e., Global As View (GAV) and Local As View (LAV). As stated in (Levy, 2001), the LAV approach is more suitable for a data integration system in a dynamic environment. Hence, we will focus on the LAV approach. Query rewriting using views on the LAV approach is closely related to the problem of answering queries using views, which has recently received considerable attention (Levy, 2001).

So far, there have been a number of rewriting algorithms presented. These algorithms can be divided into two categories, bucket-based algorithms and inverse rule-based algorithms. They can generate all the query rewritings in the absence of integrity constraints. However, if there are any integrity constraints in a mediated schema, a bucket-based algorithm might miss query rewritings or generate redundant query rewritings, because it does not consider the effects of these integrity constraints.

In the context of traditional databases, integrity constraints are the rules enforced on a database schema. From integrity constraints, some relationships among the relations in a database schema can be inferred. As mentioned previously, in a data integration system, data sources are described in terms of a mediated schema. Therefore, if there are integrity constraints in the mediated schema, some special relationships among the data sources can be found, which are useful in query processing.
There are three most ubiquitous types of integrity constraints enforced on a database schema: domain semantics, functional dependencies, and inclusion dependencies (DSs, FDs and INDs for short respectively). There have been some inverse rule-based rewriting algorithms that address the problem

Bai Q., Hong J. and F. McTear M. (2004).
IMPROVING VIEW SELECTION IN QUERY REWRITING USING DOMAIN SEMANTICS.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 177-183
DOI: 10.5220/0002623101770183

of query rewriting using views in the presence of functional and/or inclusion dependencies (Duschka *et al* 2000, Gryz, 98, 99). A logic-based approach for the problem of query rewriting using views (Grant and Minker, 2002) is presented, where there are functional and inclusion dependencies in a mediated schema and a resolution method is used to generate all the possible query rewritings. However, there has been no bucket-based algorithm to address the problem of query rewriting in the presence of domain semantics. This paper will address this issue. The following two examples show that the MiniCon algorithm, which is a bucket-based algorithm and the best one among the existing algorithms (Pottinger and Levy, 2000), generates a redundant rewriting or misses a query rewriting because domain semantics has not been considered.

Example 1 (Adapted from (Mitra, 2001)). Suppose that there are five data sources as follows:

V_1(Seller):- car(Car_1), sells(Seller, Car_1). // List of car sellers.

V_2(Car_2, S_2):- car(Car_2), sells(S_2, Car_2). // List of cars and their sellers.

V_3(Car_3):- dealer(D_3), located(D_3, "CA"), sells(D_3, Car_3). // List of cars that CA's Dealers sell.

V_4(D_4, State):- dealer(D_4), located(D_4, $State_4$). // List of dealers and their states.

V_5(Union):- member(D_5, Union), dealer(D_5), located(D_5, "CA"). // CA's Dealer Union.

The logical predicates, *car(carType), dealer(Dealername), located(Dealername, City)* and *sells(Dealername, carType)*, are defined in a mediated schema.

Assume that there exists domain semantics:
(x≠ "TESCO")← located(x,y),sells(x,z), y= "CA".
A query is made:
Q(X):- car(X), dealer(D), located(D, State), sells(D,X), D= "TESCO".
Using the MiniCon algorithm, we can get the MCDs for each view as follows (For simplicity, we list only two components of each MCD, where G is a set of subgoals of Q covered by the MCD):

Table 1: The MCDs for each view in the query (g_i,i=1,2,3,4, represents the ith subgoal of Q)

V	G	V	G	V	G
V_2	g_1	V_2	g_4	V_3	g_2,g_3,g_4

V	G	V	G
V_4	g_2	V_4	g_3

Two query rewritings can be formed by combining MCDs as follows:
Q_1(X):-V_2(X,D),V_4(D, "CA"), D= "TESCO".
Q_2(X):-V_2(X,D),V_3(X), D= "TESCO".
However, in the presence of domain semantics, the join of V_2(X,D) and V_3(X) is null (actually V_3(X) should not appear in any query rewriting) and Q_2 should be discarded. But, the MiniCon algorithm fails to find this. This example shows that domain semantics is useful for removing redundant query rewritings.

Example 2. Suppose that there is a relation *Student* in a mediated schema:
Student(S_ID, SName, Gender, Dept, RegDate).
There are two data sources:
V_1(SName):- Student(S_ID, SName, Gender, Dept, RegDate),1000<S_ID, S_ID<2000.
V_2(SName):- Student(S_ID, SName, Gender, Dept, RegDate), 2000<S_ID, S_ID<4000.
The domain semantics is represented in the form of rules as follows:

(1) 1000<S_ID, S_ID<2000 ⇔ Dept= "CS";
(2) 2000<S_ID, S_ID <3000 ⇔ Dept= "EN";
(3) 3000<S_ID, S_ID <4000 ⇔ Dept= "BI";

These rules tell us that S_IDs in a particular department are restricted to a specific range.

A query is to ask for students' names from the *Computer Science* department, i.e.,
Q(SName):- Student(S_ID, SName, Gender, Dept, RegDate), Dept= "CS".
The MiniCon algorithm fails to form any MCD for the given query over either V_1 or V_2, because comparisons are not consistent between either Q and V_1 or Q and V_2. However, we can see that there exists a query rewriting as follows:
Q'(SName):- V_1(SName).
The reason is that we can substitute the comparison, Dept= "CS", in Q with equivalent comparisons, 1000<S_ID, S_ID <2000, as shown in domain semantics (1). Then using the MiniCon algorithm, we can get the above rewriting for the query.

The rest of the paper is organized as follows. In the next section, we have a brief overview of the related work. In Section 3, preliminaries of query rewriting using views and semantics query optimization are given. In Section 4, we discuss query rewriting using views in the presence of semantic constraints. In Section 5, we conclude the paper.

2 RELATED WORK

As stated in Section 1, there are two categories of query rewriting algorithms, i.e., bucket-based rewriting algorithms and inverse rule-based rewriting algorithms. The key idea underlying the inverse rule-based algorithms 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 a set of views **V** is simply the composition of Q and the inverse rules for **V** using the transformation method (Duschka *et al*, 2000), the unification-join method (Qian, 1996), or the resolution method (Grant and Minker, 2002).

A bucket-based algorithm consists of two stages. The first stage is called view selection that selects the views relevant to a given query and puts the views into the corresponding buckets. The second stage is to generate all the possible query rewritings by combining a view from each bucket. View selection is based on containment mapping from the given query to each view. There are three representative algorithms.

Bucket algorithm (Levy *et al*, 1996a, 1996b):

Given a query, a bucket is first created for each subgoal of the query. A view is put in the bucket if it can be unified with the subgoal in the query. Next, candidate query plans are generated by combining a view from each of the buckets. These plans are then verified using containment tests.

SVB algorithm (Mitra, 2001):

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

MiniCon algorithm (Pottinger and Levy, 2000):

In the first phase of the MiniCon algorithm, a MiniCon Description (MCD for short) for a query Q over a view V is formed to contain a set of subgoals in Q and the mapping information. In fact, a MCD takes the role of a bucket in the bucket algorithm and the SVB algorithm. The MCDs and the minimum MCDs in the MiniCon algorithm correspond to the single-subgoal buckets and the shared-variable buckets in the SVB algorithm respectively. In the second phase, the MiniCon algorithm combines the MCDs to generate query rewritings.

In summary, view selection is not needed in inverse rule-based rewriting algorithm, but it needs to be done in the first stage of bucket-based algorithms. As shown in the previous section, the problems of missing query rewritings and generating redundant query rewritings in bucket-based algorithms might occur if domain semantics is not taken into account.

3 PRILIMINARIES

3.1 Domain Semantics and Residues

Domain Semantics

In the context of databases, integrity constraints are in the forms of three main practical types of constraints, i.e., domain semantics, functional dependencies, and inclusion dependencies. The functional and inclusion dependencies are mainly used for the design of database schemas, e.g., normalization of schemas, data duplication. Domain semantics is relevant to the knowledge of a specific application domain. In this paper, we only consider the effects of domain semantics in query rewriting. In some cases, a query may even be answered without accessing a database if sufficient knowledge is contained in the domain semantics.

Domain semantics is represented in this paper using the following types of rules:

D_1 (Equivalence Proposition): $CQ_1 \Leftrightarrow CQ_2$.
D_2(Dependency rule): $R.CQ_1 \Leftarrow S.CQ_2$.
D_3 (Production rule): $CQ_1 \Leftarrow R(X), S(Y), CQ_2$.
where, CQ_i (i=1,2) refers to the comparison expressions whose variables appear in some relations in a mediated schema.

D_1 means that two expressions are equivalent. D_2 means that if CQ_2 holds, CQ_1 should be satisfied in a database schema. The variables in CQ_1 and CQ_2 are in either relation R or relations R and S. The right-hand side of D_3 is a conjunction of two or more relations. CQ_1 in D_i, i=2,3, can be null.

Residues

Residues are used in semantic query optimization (Chakravarthy *et al*, 1990) to eliminate redundant joins in a given query. In this paper, we exploit residues to remove the views irrelevant to a query.

The notion of residues, associated with the concept of subsumption, is used for semantic query optimization in the presence of integrity constraints. A clause C_1 subsumes a clause C_2 if there is a substitution θ such that $C_1\theta$ is a sub-clause of C_2. The refutation tree is used to test subsumption between C_1 and C_2. C_1 subsumes C_2 if and only if there is a refutation tree that ends with the null clause.

In general, a null clause can not be obtained because integrity constraints rarely subsume relations. But integrity constraints may partially subsume a relation, leaving a fragment at the bottom of the refutation tree. Such a fragment is called a residue representing an interaction between a relation and an integrity constraint.

Definition 1. An integrity constraint IC partially subsumes an atom A, if and only if IC does not subsume ¬A, but a sub-clause of IC+ (expansion of IC) subsumes ¬A. Let C be the clause at the bottom of a refutation tree. Then $(C^-)\theta^{-1}$ (C^- is a result of reducing C) is a residue of IC and A.

3.2 Query Containment and Query Rewriting Using Views

Queries and views

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

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

where $R_1(\bar{X}_1),...,R_k(\bar{X}_k)$ are the subgoals referred to database relations, C_Q is a comparison expression. $Q(\bar{X})$ is the head of the query. The tuples $\bar{X}, \bar{X}_1,...,\bar{X}_k$ contain either variables or constants. We require that the query be safe, i.e., $\bar{X} \subseteq \bar{X}_1 \cup ... \cup \bar{X}_k$. The variables in $\bar{X}$ are distinguished variables, and the others are existential variables. We use Vars(Q), Q(D) to refer to all variables in Q and the evaluation of Q over a database instance D respectively.

A view is a named query. If the query results are stored, we refer to them as a materialized view and the result set as the extension of the view.

Query containment and equivalence

The concepts of query containment and equivalence enable us to make a comparison between queries and rewritings. We say that a query Q_1 is contained in another query 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. Containment mappings provide a necessary and sufficient condition for testing query containment. A mapping φ from Vars(Q_2) to Vars(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. The query Q_1 is equivalent to Q_2 if and only if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.

Answering queries using views

Given a query Q and a set of view definitions **V**=$V_1,...,V_m$, a rewriting of Q using the views is a query expression Q' whose body predicates are only from V1,...,V_m.

Note that the views are not assumed to contain all the tuples in their definitions since the data sources are managed autonomously. Moreover, we cannot always find an equivalent rewriting of the query using the 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(Maximally-contained rewriting): Q' is a maximally-contained rewriting of a query Q using views **V** with respect to a query language **L** if
(1) for any database D, Q' ⊆ Q, and
(2) there is no other query rewriting Q'' in the language **L**, such that for every above database D, Q'' ⊆ Q, and Q' ⊆ Q''.

4 VIEW SELECTION IN QUERY REWRITING USING DOMAIN SEMANTICS

Assume that there are a set of views V_i, (i=1,2,...,n, n≤m) and a set of domain semantics in the three types of rules as described in Section 3.1. As an equivalence proposition, D_1 will be added to the relevant views without losing any information. D_2 and D_3 enforce constraints on certain views. As stated later, we will resolute each of D_2 and D_3 with every corresponding view V_i to get a so called pseudo residue PR_i for V_i. A pseudo residue is a fragment at the bottom of the refutation tree, representing an interaction between a view and a D_2 or D_3. PR_i takes a role of the residue, but whether it can play a role in view selection also depends on a given query. According to (Chakravarthy *et al*, 1990), each relation has a residue for each integrity constraint, which results in several SCAs (semantically constrained axioms) in a view. However, in this paper, each D_2 or D_3 is viewed as a whole and there is only one pseudo residue for each D_2 or D_3 over a view.

4.1 Computing a Pseudo Residue

Computing a pseudo residue for V_i, (i=1,....,n) is based on the resolution method as follows:

Case 1. For each D_1(Equivalence Proposition): $CQ_1 \Leftrightarrow CQ_2$, where the variables of both CQ_1 and CQ_2 are in a relation R, we check whether view V_i contains R. If not, then the constraint can not be enforced on V_i. Otherwise we check whether CQ_1 (or CQ_2) appears in V_i. If so, a pseudo residue is CQ_2 (or CQ_1), denoted by ER_i=CQ_2 (or ER_i =CQ_1).

Case 2. For each D_2 (Dependency rule): R.$CQ_1 \Leftarrow$ S.CQ_2, we check whether view V_i contains R and S. If not, then the constraint can not be enforced on V_i. Otherwise, we check whether CQ_2 is consistent with V_i. If so, $PR_i = CQ_1$.

Case 3. For each D_3 (Production rule): $CQ_1 \Leftarrow$ R(X),S(Y), CQ_2, we check whether view V_i contains the join of relations R and S. If not, then the constraint can not be enforced in V_i. Otherwise, we check whether CQ_2 is consistent with V_i. If so, $PR_i = CQ_1$.

We use the resolution method to get the PR_i of V_i for each D_3. We construct a linear refutation tree with the body of V_i as the root, using at each step an element of the right side of D_3 in resolution. If the tree ends with empty or the subgoals only in V_i, then PR_i =CQ_1. Figure 1 shows the process.

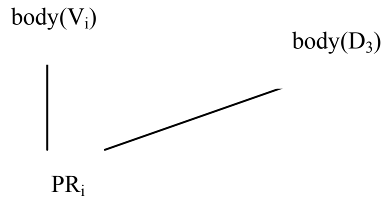

Figure 1: Computing a pseudo residue of V_i

We show the process of resolution between a view and a D_3 using the following example.

Example 3. Continuing with Example 1, we can get the pseudo residues for each view as follows:

View	V_1	V_2	V_3	V_4	V_5
PR_i	{}	{}	{D≠ "TESCO"}	{}	{}

For simplicity, we show only the process of computing the pseudo residue of V_3. In this example, CQ_1 ={located.x ≠ "TESCO"} and CQ_2 ={ located.y ="CA"}.

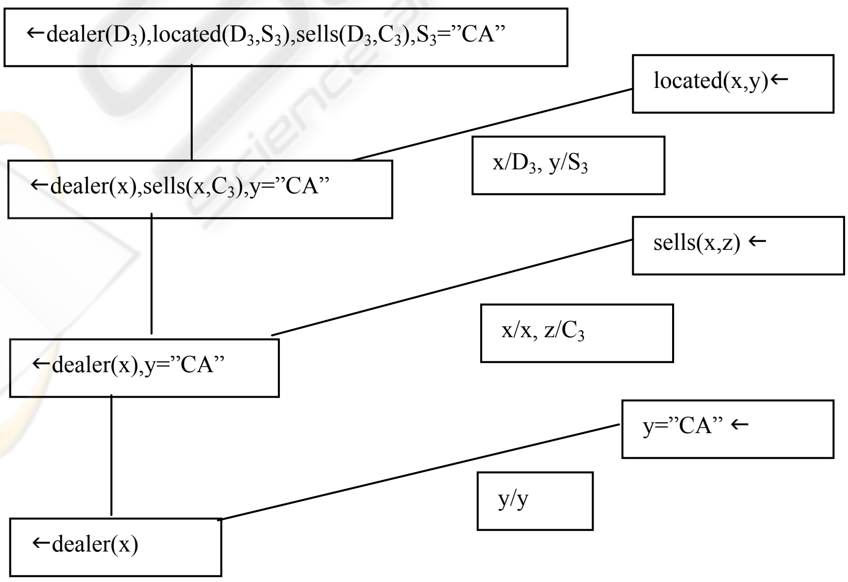

Figure 2: Computing the pseudo residue of V_3

Thus, PR_3 = {located.x ≠"TESCO"}. For each of the other views V_i (i=1,2,4,5), a linear refutation tree would not end with such subgoals that appear only in the views, then the pseudo residues of the views are null, denoted by {}.

4.2 View Selection in Query Rewriting Using Views

As mentioned in Section 2, the first step in a bucket-based algorithm is to select the views relevant to a given query. For a given query Q, using domain semantics we may remove the irrelevant views or pick up certain relevant views before using any bucket-based rewriting algorithm. As a result, the soundness and completeness of query rewriting algorithms can be improved.

Given a query Q, for each V_i (i=1,2,…,n), if PR_i of V_i is not compatible with C_Q in Q, then the view V_i is irrelevant to Q and should not be considered when rewriting the query. Note that each pseudo residue is in the form of a comparison expression and so are both C_Q in Q and ones in V_i, we need only to check the consistency between two comparison expressions. Two comparison expressions, CQ_1 and CQ_2, are comparable if their variables are in the same relations. CQ_1 is consistent with CQ_2 if they are equivalent or their conjunct, $CQ_1 \wedge CQ_2$, is not always false for any values of the variables involved. In the following algorithm, all comparison expressions are comparable. Otherwise, the corresponding process would have stopped.

Our algorithm consists of two steps. In the first step, some irrelevant views, in the presence of domain semantics, with respect to Q are removed by comparing C_Q in Q with the pseudo residues of views. In the second step, other irrelevant views, in terms of unification, with respect to Q are removed by unifying Q with the definitions of views, which is used in any previous bucket-based rewriting algorithm.

Algorithm: View Selection in Query Rewriting Using Domain Semantics
Input: A given query Q, a set of views V_i (i=1,2,...,n) associated with a set of pseudo residues PR_i (i=1,2,...,n).
Output: A set of buckets or MCDS containing views V_j (j=1,2,...,t, t≤n) which are relevant to Q both in the presence of domain semantics and in terms of unification.

Methods:

Step 1: Removing the irrelevant views with respect to Q by comparing C_Q with pseudo residues.
V={ V_i (i=1,2,...,n)}.
For each view V_i, (i=1,...,n) associated with a pseudo residues PR_i, we proceed according to the following cases:
Case 1: The pseudo residue is ER_i: We check whether C_Q in Q is consistent with CQ_1 or CQ_2 in D_1. If not, the view V_i is irrelevant to Q, i.e., **V**= **V**−{ V_i }. Otherwise, ER_i is added into the definition of view V_i.
Case 2: The pseudo residue is PR_i: We check whether C_Q in Q is consistent with PR_i of V_i. If not, the view V_i is irrelevant to Q, i.e., **V**= **V**−{ V_i }.

Step 2: Selecting relevant views with respect to Q by unifying Q with the definitions of views.
For each view V_i in **V**, a set of the buckets is built according to the SVB algorithm or a set of the MCDs is formed according to the MiniCon algorithm. This procedure is based on unification from Q to views. The views in the buckets or the MCDs are relevant to Q in the presence of domain semantics and in terms of unification.
End.

Example 4. Continuing with Example 3, the pseudo residue of V_3 is PR_3 = {located.x ≠"TESCO"}. Note that x is the first argument in relation *located*, resulting in the pseudo residue of V_3 is D ≠"TESCO". It conflicts with the comparison in Q. Hence, the view V_3 is irrelevant to the given query Q in the presence of domain semantics. For the rest of views, we use the MiniCon algorithm to form the MCDs as follows:

Table 2: The MCDs for V_1, V_2, V_4, and V_5

V	G	V	G	V	G	V	G
V_2	g_1	V_2	g_4	V_4	g_2	V_4	g_3

There are two views relevant to Q in terms of unification. However, there is only one query rewriting formed by combining MCDs so that all subgoals of Q can be covered.

Q_1(X):-V_2(X,D),V_4(D, "CA"), D= "TESCO".

5 CONCLUSIONS

In previous bucket-based rewriting algorithms, for a given query Q, view selection is done by unifying Q with the definition of views. In other words, the meaning of “relevant to Q” is in terms of unification. We found that there is another explanation about “relevant to Q”, i.e., in the presence of domain semantics. That is, we can remove the irrelevant views which could not be found in any bucket-based algorithm. Also, in some cases, we can avoid the problem of missing relevant views, which occurs in bucket-based algorithms.

In this paper, we have aimed to solve the problems of missing query rewritings and redundant query rewritings in bucket-based rewriting algorithms so that we can improve the soundness and completeness of these algorithms. In the presence of domain semantics in a mediated schema, we first compute the pseudo residue for each constraint over the views using the resolution method. In fact, what we have done is to transfer the integrity constraints over the relations of the mediated schema into a rule over a view. As a result, for a given query, we can determine which view is irrelevant to the query, in the presence of domain semantics, by making a comparison between the pseudo residue of a view and the comparison expression of the query. The pseudo residues can be calculated in advanced, which means that the total increased computation in Step 1 in our algorithm is only in polynomial size of $|D|*|V|$, where $|D|$ and $|V|$ are the number of domain semantics in a mediated schema and of the views respectively. This process is useful for query rewriting, which has been shown by examples in Section 1.

REFERENCES

Arens, Y., Knoblock, C.A., Shen, W., 1996. Query Reformulation for Dynamic Information Integration. *In Journal of Intelligent Information Systems, Special Issue on Intelligent Information Integration,* 6(2/3):99--130.

Cali, A., Calvanese, D., Giacomo, G. D., Lenzerini, M., 2002. On the Role of Integrity Constraints in Data Integration. *In IEEE Data Engineering Bulletin, 25(3), Special Issue on Organizing and Discovering the Semantic Web,* 39-45.

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

Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim, K., 1995. Optimizing Queries with Materialized Views. *In Proceeding of the 11th International Conference on Data Engineering,* IEEE Computer Soc. Press, 190-200.

Duschka, O.M., Genesereth, M.R., Levy, A.Y., 2000. Recursive Query Plans for Data Integration. *In Journal of Logic Programming, special issue on Logic Based Heterogeneous Information Systems,* 43(1), 49-73.

Florescu, D., Raschid, L., Valduriez, P., 1996. Query Reformulation in Multidatabase Systems using Semantic Knowledge. *In International Journal of Cooperative Information Systems,* 5(1996), 431-468.

Fagin, R., Vardi, M.Y.,1986. The Theory of Data Dependencies—A Survey. *In Proceedings of Symposia in Applied Mathematics,* Volume 34, 19-71.

Godfrey, P., Grant, J., Gryz, J., Minker, J., 1998. Integrity Constraints: Semantics and Applications. *In Chapter 9 of Logics for Databases and Information Systems, J.Chomicki and G.Saake, editors, Kluwer Press,* 265-306.

Grant, J., Minker., J., 2002. A logic-based approach to data integration. *In TLP,* 2(3):323-368.

Gryz, J., 1998. An Algorithm for Query Folding with Functional Dependencies. *In Proceedings of the 7th International Symposium on Intelligent Information Systems ,* 7-16.

Gryz, J., 1999. Query rewriting using views in the presence of functional and inclusion dependencies. *In Information System,* 24(7):597-612.

Hsu, C., Knoblock, C.A., 2000. Semantic Query Optimization for Query Plans of Heterogeneous Multidatabase Systems. *In IEEE Transactions on Knowledge and Data Engineering,* 12(6):959--978.

Levy, A.Y., 2001. Answering Queries Using Views: A Survey. *In VLDB Journal,* 10(4), 270-294.

Levy, A.Y., Rajaraman, A., Ordille, J.J., 1996a. Querying Heterogeneous Information Sources Using Source Descriptions. *In Proceedings of the 22nd VLDB Conference,* 251--262.

Levy, A.Y., Rajaraman, A., Ordille, J.J., 1996b. Query-Answering Algorithms for Information Agents. *In Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, AAAI Press / MIT Press,* 40--47.

Mitra, P., 2001. An Algorithm for Answering Queries Efficiently Using Views. *In Proceedings of the 12th Australasian Database Conference,* 99-106.

Pottinger, R., Levy, A.Y., 2000. A Scalable Algorithm for Answering Queries Using Views. *In Proceedings of the International Conference on Very Large Data Bases(VLDB),* 484-495.

Qian, X., 1996. Query folding. *In Proceedings of the 12th IEEE International Conference on Data Engineering (ICDE’96),* 48-55.