Describing and Utilizing Constraints to Answer Queries in Data-Integration Systems

Chen Li
Information and Computer Science
University of California, Irvine, CA 92697
chenli@ics.uci.edu

Abstract

In data-integration systems, information sources often have various constraints such as "all houses stored at a source have a unique address." These constraints are very useful to compute answers to queries. In this paper we study how to describe these constraints, so that they can be utilized in query processing and optimization. We consider the local-as-view approach to data integration (under the open-world assumption), in which source contents and user queries are formulated on predefined global predicates. In this approach, source contents and constraints often exist before the global predicates are designed. We discuss two different levels of describing constraints: local constraints are defined sources, while global constraints are on global predicates. We formally define two types of global constraints, namely general global constraints and source-derived global constraints. We present the advantages of having these constraints, and discuss open problems that need more research investigations.

1 Introduction

The purpose of data integration is to support seamless access to autonomous, heterogeneous information sources, such as legacy databases, corporate databases connected by intranets, and sources on the Web. Many data-integration systems adopt a mediation architecture [Wiederhold, 1992], in which a user poses a query to a *mediator* that retrieves data from underlying sources to answer the query. A *wrapper* on a source is used to perform data translation and local query processing [Hammer *et al.*, 1997].

In such an environment, sources can have various constraints on their contents, which can be utilized in query processing and optimization [Godfrey *et al.*, 2001]. For instance, consider two data sources that have information about houses in Orange County. They have the following (obviously simplified) schemas. (The attribute names are self explanatory.)[1]

Source S1: s1(street, zip, price, sqft), Irvine houses
Source S2: s2(street, zip, year, sqft), Newport houses

[1]This "housing" example is used throughout this paper.

The sources have the following constraints. (**C1**) All houses at source S1 are at least $\$250,000$. (**C2**) All houses at source S1 have a unique street address. (**C3**) All houses at source S2 are at least $\$200,000$, and they are all built before year 1995. (**C4**) Each house in Orange County has a unique (street, zip) address.

These constraints carry a rich set of semantics, which can be utilized in query processing. For instance, consider the following queries that ask for houses in Orange County.

- Query **Q1** asks for houses under $\$230,000$. We do not need to access S1 due to constraint C1.
- Query **Q2** asks for houses built after year 1998. We do not need to access S2 due to constraint C3.
- Query **Q3** asks for the price and year of houses in Orange County. We can take the natural join of these two sources on the street and zip attributes to compute answers. Notice that we cannot compute the answers in this way if constraint C4 does not hold.

In this paper we study how to describe various source constraints in data-integration systems that can be utilized to answer queries. We consider the local-as-view (LAV, a.k.a. source-centric) approach to data integration under the open-world assumption ("OWA") [Lenzerini, 2002]. In the LAV approach, a collection of *global predicates* are used to describe source contents as views and formulate user queries. Given a user query, the system decides how to answer the query by synthesizing source views. In the housing example, we can use a global predicate `house(street, city, zip, price, year, sqft)` to describe the source contents and user queries. (On the contrary, in the *GAV approach*, user queries are posed directly on global views that are defined on source relations. These two approaches are compared in [Ullman, 1997].) The LAV approach is widely adopted by many systems, particularly due to its good scalability — whenever a source description changes, we can just inform the mediator about the change without modifying any global predicates or other source views. Under the open-world assumption, each source view has *some* (not all) records satisfying the view definition on a database of the global predicates.

Under these "LAV-OWA" assumptions, source contents and constraints could exist a-priori, before the global predicates are introduced. Some source constraints, such as C1,

C2, and C3 above, exist for some data sources only, and they do not hold on the global predicates. Other constraints, such as C4 above, are true for all sources in the system, and it is better to represent them as "global constraints," i.e., constraints that hold for the global predicates. As an example, constraint C4 can be represented by the fact that (street, zip) form a key for the predicate `house`. By using global constraints we can represent general information about the contents and constraints of the underlying sources. Notice that some global constraints can be implied from the source constraints. For instance, in the housing example, even though in general Orange County houses could have a large range of prices and built years, as answering user queries using the two given sources' contents is concerned, we can just assume that all the houses stored at these two sources are at least $\$200,000$.

In this paper we discuss two ways to describe source constraints: either on the sources ("local constraints"), or on the global predicates ("global constraints"). While the meaning of local constraints is easily defined (Section 2), the semantics of global constraints are more subtle, since global predicates are *virtual*. We formally define two types of global constraints, namely general global constraints and source-derived global constraints. We present the advantages of having these constraints (Section 3). We conclude the paper by discussing some open problems that we are currently investigating (Section 4).

This work is conducted in the Raccoon Project on distributed data integration and sharing at University of California, Irvine [Raccoon, 2002].

1.1 Related Work

Integrity-constraint specification and checking have been studied intensively by researchers in deductive databases [Ling, 1987]. Using integrity constraints to speedup query processing has also been studied by many papers, especially in the area of deductive databases and object-oriented databases. In particular, the goal of *semantic query optimization* is to use semantic information (such as integrity constraints or deductive rules) to speedup the processing of queries [Chakravarthy *et al.*, 1990]. These techniques can be used in the data-integration context. In addition, these studies assume that the constraints are given. Since in a data-integration system, the constraints on global predicates might not exist a-priori, solutions that can compute these global constraints can make these techniques applicable in data integration. Furthermore, since the constraints could be described at both the source and the global levels, we need more optimization techniques that are suitable for this context. For instance, how to compute global constraints from local constraints, or vice versa, even before user queries are posed? As a related study, the recent work in [Hsu and Knoblock, 2000] presents a semantic query optimization approach to optimizing query plans of heterogeneous multidatabase systems. It provides global optimization for query plans as well as local optimization for subqueries that retrieve data from individual database sources.

Researchers have studied the problem of resolving constraint conflicts while integrating the conceptual schema of multiple autonomous data sources modeled using the entity-relationship (ER) (e.g., [Lee and Ling, 1997]). The assumption in most of the previous work is that we need to design the global schema to integrate data sources and resolve their constraint conflicts. Some of these techniques could be applied to solve our problems. Furthermore, in our work, by taking the LAV approach to data integration, we assume that the global predicates have been designed, and we need a systematic way to compute constraints on the global predicates.

2 Local Constraints

There are various constraints in heterogeneous sources in a data-integration environment. The following are a few common classes.

- Range constraints: such as "housePrice $\geq$ 200,000", "houseYear $\leq$ 1995", and "carPrice between [3000, 8000]".
- Enumeration constraints: such as the value of a "state" attribute can be only of the 50 state names in the US.
- Functional Dependencies: for instance, a source relation `hotel(hotelName, address, tel)` has a functional dependency `hotelName` $\rightarrow$ `(address, tel)`. Here we use the symbol "$\rightarrow$" to represent a functional dependency.
- Other constraints such as keys and referential-integrity (foreign keys).

In the LAV approach, source contents are defined on global predicates as views. Some of these source constraints could be described in the view definitions, if the view language is expressive enough. Consider the two data sources in the housing example. We could define the source contents and some of the constraints as follows:

$$\begin{aligned} s1(S,Z,P,F) :- \quad & house(S, irvine, Z, P, Y, F), \\ & P \geq 250,000. \\ s2(S,Z,Y,F) :- \quad & house(S, newport, Z, P, Y, F), \\ & P \geq 200,000, Y \leq 1995. \end{aligned}$$

There are several reasons we want to consider constraints separately from the view and query language. First, the language may not be expressive enough to describe all possible constraints. For example, the above view definitions use select-project-join queries with comparisons, a.k.a. conjunctive queries with built-in predicates [Ullman, 1988]. This query language cannot describe constraints such as enumerations and functional dependencies. Second, incorporating constraints in the query language could complicate the process of answering user queries using source views, since the complexity of this process heavily depends on the language. For instance, it is known that if user queries and source views are all conjunctive queries using the global predicates, the problem of finding a conjunctive rewriting for a query using the views is in NP [Levy *et al.*, 1995]. If the user queries and views can have arithmetic comparisons, we need at least recursive queries to compute answers to queries [Afrati *et al.*, 2002], and the complexity increases significantly. Thus we are interested in realistic cases where the process can be much

simplified. Third, describing constraints separately from the query language can allow us to do reasoning about them more easily.

Some of source constraints can be naturally represented as *local constraints*. Each local constraint is defined on one data source only. A constraint C for a data source S is a set of conditions, such that for any database instance of source S, these conditions must be satisfied by the tuples in the database. For instance, the first three constraints in the housing example can be represented as three local constraints:

C1 for S1: Price $\geq$ 250,000.
C2 for S1: street $\rightarrow$ (zip, price, sqft).
C3 for S2: Price $\geq$ 200,000, Year $\geq$ 1995.

In particular, for C1, the condition "Price $\geq$ 250,000" should be satisfied by *any* database instance of source S1, not just for a particular instance. We might be tempted to represent the constraint C4 as two local constraints:

C4 for S1: (street, zip) $\rightarrow$ (price, sqft).
C4 for S2: (street, zip) $\rightarrow$ (year, sqft).

However, they do not correctly describe C4. These two *local* functional dependencies do not disallow the case where the two sources have two different houses that have the same street and zip code. That is, it is still possible that source S1 has a record `s1(main street, 92697, $300,000, 2500)`, while S2 has a record `s2(main street, 92697, $360,000, 2800)` (for a different house). Clearly this case is not allowed by C4. As a consequence, some queries may not be answered properly. Thus, if we replace C4 with these two local constraints, we cannot take the natural join of S1 and S2 to answer the query Q3. The limitations of local constraints show the need to use global constraints.

3 Global Constraints

There are two types of global constraints, namely general global constraints and source-derived global constraints. We first define them formally, then present the advantages of using them.

Let $s_1, \ldots, s_k$ be k sources in a data-integration system. Let $P = \{p_1, \ldots, p_n\}$ be a set of global predicates, on which the contents of each source s_i are defined. For simplicity, we assume that each source s_i has a single view v_i defined on the global predicates. (Notice that some constraints such as referential integrity need multiple tables at a source.)

3.1 General Global Constraints

A *general global constraint* is a condition that should be satisfied by *any* database instance of the global predicates P. For example, the constraint C4 in the housing example can be represented as the following (general global) constraint G_1:

G_1: (street, zip) form a key of the `house` predicate, i.e., (street, zip) $\rightarrow$ (city, price, year, sqft)

General global constraints can be introduced during the design phase of such a data-integration system. They are used to capture the semantics of the application domain, no matter how many sources are in the system. That is, even if new sources join or existing ones leave the system, it is assumed that these constraints should be satisfied by any database instance of the global predicates. These constraints can be utilized to answer queries, as shown by the query Q3.

3.2 Source-Derived Global Constraints

The second type of global constraints is called "source-derived global constraints." We first use the housing example to motivate this concept. Consider the following view definition for source s1.

$$S1(S, Z, P, F) \text{ :- } house(S, irvine, Z, P, Y, F).$$

Under the OWA, this view means that for each tuple $t_1 = \langle s, z, p, f \rangle$ at source S1, there must exist a tuple $t_2 = \langle s, c, z, p, y, f \rangle$ in the predicate `house`, such that the city value `c = 'irvine'`, even though we do not know the exact value of year y. Notice that S1 may not have the information about *all* the houses in Irvine. On the other hand, we can only infer the existence of tuples such as t_2 (for all tuples in s_1), and we do not know whether other tuples exist or not. Thus we can just assume that other tuples do not exist (except those that be derived from other source view instances). Given a database instance of all the source views, we use the derived tuples on the global predicates (such as t_2) to compute answers to a query.

Given these two sources, since we are certain about those derived tuples on the global predicate, we can introduce the corresponding constraints that must be satisfied by these derived tuples. Due to the local constraints C1 and C3, the following is a condition that is satisfied by all the derived tuples for the global predicate `house`:

$$G_2\text{: price} \geq 200,000.$$

Such a constraint is a source-derived global predicate.

Now we define source-derived global constraints formally. We first revisit the semantics of a source view definition. We use the language of conjunctive queries (select-project-join queries) as an example, even though the concept can be generalized to other languages. Consider a view defined as the following conjunctive query, where each r_i is a global predicate.

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

Under the open-world assumption, this view means that for each tuple t_v in an instance of the view, there are tuples $t_1, \ldots, t_n$ in the predicates $r_1, \ldots, r_n$ respectively, such that this query v will produce (at least) tuple t_v using these n tuples [Calvanese *et al.*, 2000].

Given a database instance $I_s = \{T_1, \ldots, T_k\}$ of the k source views $v_1, \ldots, v_k$, in which T_i is a table instance of view v_i, let D be a database instance of the global predicates derived from I_s according to the view definitions. The database instance D could have some unknown values. However, to make sure join attributes share the same value in the records to produce a record in I_s, these join values could use the same unknown values. (This observation is reflected as using functional symbols in the inverse-rule algorithm in [Duschka and Genesereth, 1997].) We call D a *derived database* of I_s.

Definition 3.1 (**Source-Derived Global Constraints**) Let source views $v_1, \ldots, v_k$ have local constraints $\mathcal{C}_i, \ldots, \mathcal{C}_k$, respectively. A source-derived global predicate G of these source views is a condition that must be satisfied by *any* derived database of *any* database instance of the source views that satisfy the local constraints $\mathcal{C}_i, \ldots, \mathcal{C}_k$. □

In the housing example, "price $\geq 200,000$" is a source-derived global constraint, since it is satisfied by any house tuple derived from any database instance of the two sources. On the contrary, "price $\geq 300,000$" is not, since there could be a house (derived from the a database instance of the two sources) that has a price lower than this one. Similarly, the constraint C2 cannot derive a global constraint "street $\rightarrow$ (zip, price, sqft)", since this functional dependency might not be true for a derived database of the house predicate.

There are two important observations about source-derived global constraints.

- Such a constraint may not hold for the particular domain in general. As an example, there can be houses in Orange County that are cheaper than $\$200,000$, violating the constraint G_2. However, since these houses cannot be derived from any database instance of the two sources, for the purpose of computing answers to queries using these two sources, we do not need to consider these houses.
- If a source joins the system with its own local constraints, the existing source-derived global constraints may change. For instance, if a source s3 also has house information, and the prices of all its houses are at least $\$150,0000$, then constraint G_2 will become "price $\geq 150,000$." If a source s4 does not have any local constraint on its houses, then G_2 will become invalid. On the contrary, if sources leave the system, we may get more global constraints, or existing ones may can become more restrictive (e.g., "price $\geq 150,000$" could become "price $\geq 180,000$").

3.3 Motivation of Using Global Constraints

There are many advantages of using global constraints, rather than keeping local constraints only. First, some source constraints cannot be expressed as local constraints, as shown by the constraint C4, which can be easily described as a (general) global constraint. General global constraints are easier and more natural to represent conditions of tuples at the global-predicate level.

Second, global constraints provide a general description of the data at the sources, and this description can be very useful for users to understand the source contents. Knowing that constraint G_2 is true, users can better submit queries to meet their needs.

Third, global constraints can make query processing more efficient. Given the global predicate G_2, if a query asks for houses cheaper than $\$130,000$, without checking the source contents and constraints, the mediator can immediately know that the answer is empty.

Four, a data-integration system can be used as a subcomponent in a larger-scale system. We could have a hierarchy of data-integration systems, in which each node in the hierarchy could be a system that has multiple sources. We could also have a network of data-integration systems, each participant of which consists of multiple sources. This architecture can happen in a peer-based data-integration system such as the Raccoon project [Raccoon, 2002], the Piazza project [Gribble *et al.*, 2001], and the AMOS II system [Katchaounov *et al.*, 2003]. Having the global constraints for each such a component will be very critical for other components to know the contents of those sources within this component. This information can be used effectively in query answering and routing.

4 Open Problems

There are many open problems on how to describe constraints in data-integration systems, so that their rich semantics can help the system answer queries. One problem is how to compute source-derived global constraints from local constraints. Since such a system can be very dynamic — sources can come and go — we need a systematic way to calculate these global constraints automatically. In developing the algorithms, we need to consider different kinds of constraints.

Take range constraints as an example. One possible approach could be the following. For each local constraint for a source view, we use the view definition to compute a constraint on the predicates used in the view. We do the computations for all the local constraints. Then for those computed constraints on the global predicates, we check if some of them are consistent, and then choose the "most relaxing constraint." For instance, from local constraint C1 we can get a constraint "price $\geq \$250,000$" on the house predicate. From local constraint C3 we can get a constraint "price $\geq \$200,000$" on the house predicate. Combining these two new constraints we can get "price $\geq \$200,000$", which is the valid source-derived global constraint G_2.

The computation process can become complicated if the view definitions have complex operations such as joins and projections. For instance, if we want to "map" the local constraint C3 to the global predicate, we might need the following constraint on predicate house:

> If city = 'irvine', then functional dependency "street $\rightarrow$ (zip, price, sqft)" is true.

Clearly this constraint is not a pure functional dependency, since it is "constant based." In order to capture various source constraints, we may need to introduce more expressive constraints.

Another problem is how to verify if a given database instance of source views satisfies a general global constraint. Recall that such a constraint is introduced in the design phase of the system, assuming it is satisfied by any database of the global predicates. We need to compute the corresponding constraints that need to be satisfied by source relations. This process is just the opposite of the process of inferring global constraints from local constraints. We need efficient algorithms for this process and the verification.

A third problem is how to utilize local constraints and global constraints to answer queries effectively and efficiently. There has been intensive study on using constraints to answer queries (e.g., [Chakravarthy *et al.*, 1990;

Gryz, 1999]). For instance, the query Q3 could be answered using the two sources by utilizing the global constraint G_1. It is interesting and challenging to see how to use these techniques to process and optimize queries in the context of data integration. Notice that because the constraints could be described at both the source and the global levels, new techniques need to be developed in this new context [Hsu and Knoblock, 2000].

In summary, we believe that these problems are very important for data-integration systems, and they need more research investigations. We are currently studying some of these problems.

Acknowledgments: We thank Jia Li for the fruitful discussions on topics related to this work.

References

[Afrati *et al.*, 2002] Foto Afrati, Chen Li, and Prasenjit Mitra. Answering queries using views with arithmetic comparisons. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2002.

[Calvanese *et al.*, 2000] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views over description logics knowledge bases. In *PODS*, pages 386–391, July - August 2000.

[Chakravarthy *et al.*, 1990] Upen S. Chakravarthy, John Grant, and Jack Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems (TODS)*, 15(2):162–207, 1990.

[Duschka and Genesereth, 1997] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 109–116, 1997.

[Godfrey *et al.*, 2001] Parke Godfrey, Jarek Gryz, and Calisto Zuzarte. Exploiting constraint-like data characterizations in query optimization. *ACM SIGMOD Record*, 30(2):582–592, 2001.

[Gribble *et al.*, 2001] Steven Gribble, Alon Halevy, Zachary Ives, Maya Rodrig, and Dan Suciu. What can databases do for peer-to-peer? In *WebDB*, 2001.

[Gryz, 1999] Jarek Gryz. Query rewriting using views in the presence of functional and inclusion dependencies. *Information Systems*, 24(7):597–612, 1999.

[Hammer *et al.*, 1997] Joachim Hammer, Hector Garcia-Molina, Svetlozar Nestorov, Ramana Yerneni, Markus M. Breunig, and Vasilis Vassalos. Template-based wrappers in the TSIMMIS system. *ACM SIGMOD International Conference on Management of Data*, pages 532–535, 1997.

[Hsu and Knoblock, 2000] Chun-Nan Hsu and Craig A. Knoblock. Semantic query optimization for query plans of heterogeneous multidatabase systems. *Knowledge and Data Engineering*, 12(6):959–978, 2000.

[Katchaounov *et al.*, 2003] Timour Katchaounov, Vanja Josifovski, and Tore Risch. Scalable view expansion in a peer mediator system. In *8th International Conference on Database Systems for Advanced Applications (DASFAA)*, 2003.

[Lee and Ling, 1997] Mong-Li Lee and Tok Wang Ling. Resolving constraint conflicts in the integration of entity-relationship schemas. In *Conceptual Modeling - ER*, volume 1331 of *Lecture Notes in Computer Science*, pages 394–407, 1997.

[Lenzerini, 2002] Maurizio Lenzerini. Data integration: A theoretical perspective. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 233–246, 2002.

[Levy *et al.*, 1995] Alon Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, 1995.

[Ling, 1987] Tok Wang Ling. Integrity Constraint Checking in Deductive Databases Using the Prolog Not-Predicate. *Data and Knowledge Engineering*, 2:145–168, 1987.

[Raccoon, 2002] Raccoon. The Raccoon Project on Distributed Data Integration and Sharing, Information and Computer Science, University of California, Irvine, 2002.

[Ullman, 1988] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems, Volumes I: Classical Database Systems*. Computer Science Press, New York, 1988.

[Ullman, 1997] Jeffrey D. Ullman. Information integration using logical views. In *International Conference on Database Theory*, pages 19–40, 1997.

[Wiederhold, 1992] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.