Using Constraints to Describe Source Contents in Data Integration Systems

Chen Li, *University of California, Irvine*

In data integration systems, information sources often have constraints, such as "all houses stored at a source have a unique address." These constraints are useful in computing answers to queries and facilitating information exchange.

Data integration supports 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[1] in which a user poses a query to a mediator that retrieves data from underlying sources to answer the query. In such an environment, sources can have various constraints on their contents, which the system can use in query processing and optimization.[2]

With this in mind, my colleagues and I at the University of California, Irvine's Raccoon Project looked at how to best describe these constraints so that they can be used to answer queries. We classified various constraints; analyzed their motivations, limitations, and advantages; and looked at how to manipulate them. This research gave us greater insight into how we can utilize constraints more effectively.

An example: Housing information

For instance, consider the mediation system in Figure 1, which provides information about houses. The system integrates two sources that have Orange County, California, housing information. They have these simplified schemas:

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

The sources have these 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 were built before 1995.
- C4: Each house in Orange County has a unique (street, zip) address.

These constraints carry a rich set of semantics that the system can use in query processing. For instance, consider these queries that ask for houses in Orange County:

- Q1 asks for houses under $230,000. We don't need to access S1 owing to C1.
- Q2 asks for houses built after 1998. We don't need to access S2 owing to C3.
- Q3 asks for the price and year of Orange County houses. We can take the natural join of these two sources on the street and zip attributes to compute answers. We can't compute the answers in this way if C4 doesn't hold true.

We consider the *local-as-view* (also known as source-centric) approach to data integration under the open-world assumption. The LAV approach uses a collection of *global predicates* 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, a global predicate house(street, city, zip, price, year, sqft) describes the source contents and user queries. (On the contrary, in the *global-as-view* approach, users pose queries directly on global views that are defined on source relations. Jeffrey D. Ullman compares these two approaches in detail.[3]) Many systems have adopted the LAV approach, particularly because of its good scalability; whenever a source description changes, we can just inform the mediator without modifying any global predicates

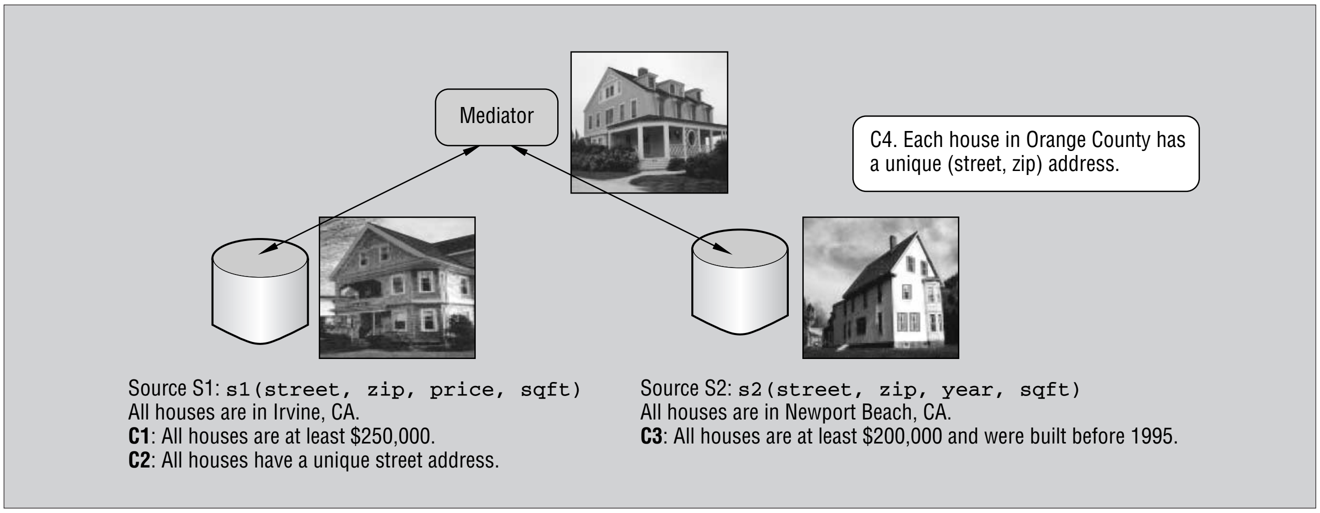

Mediator
C4. Each house in Orange County has a unique (street, zip) address.
Source S1: s1(street, zip, price, sqft)
All houses are in Irvine, CA.
C1: All houses are at least $250,000.
C2: All houses have a unique street address.
Source S2: s2(street, zip, year, sqft)
All houses are in Newport Beach, CA.
C3: All houses are at least $200,000 and were built before 1995.

Figure 1. A data integration system with source descriptions and content constraints.

or other source views. Under the open-world assumption, each source has partial information about a particular domain. For instance, there are always some houses in Orange County whose information isn't stored in S1 and S2.

Under these LAV and open-world assumptions, source contents and constraints could exist a priori even before we introduce the global predicates. Some source constraints, such as C1, C2, and C3, exist for only some data sources and don't hold on the global predicates. Other constraints, such as C4, are true for all sources and better represented as *global constraints*—constraints that hold for the global predicates. For example, the fact that (street, zip) form a key for the predicate house can represent C4. By using global constraints, we can represent general information about the underlying sources' contents and constraints. The source constraints can imply some global constraints. For instance, even though Orange County houses might generally have a large range of prices and years built, for answering user queries using the two given sources' contents, we can assume that all houses stored at these two sources are at least $200,000.

At a high level, there are two ways to describe source constraints: either on the sources (*local constraints*) or on the global predicates (*global constraints*). Although local constraints are easy to define, the semantics of global constraints are subtler because global predicates are virtual. That is, in the LAV approach to data integration, the global predicates don't have relations that store real records.

Local constraints

Heterogeneous sources in a data integration environment have various constraints. A few common classes are

- Range constraints (such as housePrice $\geq$ 200,000, houseYear $\leq$ 1995, and carPrice between [3000, 8000])
- Enumeration constraints (for example, the value of a state attribute can be only from the 50 US state names)
- Functional dependencies (for example, a source relation hotel(hotelName, address, tel) has a functional dependency hotelName $\rightarrow$ (address, tel)["$\rightarrow$" represents a functional dependency])
- Other constraints such as referential integrity (foreign keys) and inclusion constraints (for example, all the house addresses from table *R* are a subset of all those addresses from table *S*)

Some of these source constraints could be described in the source view definitions if the view language were expressive enough. Consider the two data sources in the housing example. We could define the source contents and some of the constraints in this way:

S1(s, z, p, f) :- house(s, irvine, z, p, y, f), $p \geq 250{,}000$.
S2(s, z, y, f) :- house(s, newport, z, p, y, f), $p \geq 200{,}000$, $y \leq 1995$.

We want to consider constraints separately from the view and query language for several reasons. First, the language might not be expressive enough to describe all possible constraints. For example, the view definitions just listed use select-project-join queries with comparisons, also known as conjunctive queries with built-in predicates. This query language can't 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 because its complexity heavily depends on the language. For instance, 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 NP-complete.[4] If the user queries and views can have arithmetic comparisons, we need at least recursive queries to compute answers to queries,[5] and the complexity increases significantly. So, we're interested in realistic cases where we can simplify the process.

Third, describing constraints separately from the query language can let us reason about them more easily.

Some source constraints can naturally be represented as local constraints. Each local constraint is defined on one data source only. A local constraint C for a data source S is a set of conditions such that for any database instance of S, the tuples in the database must satisfy these conditions. For instance, we can represent the first three constraints in the housing example 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 $\leq$ 1995.

Related Work

Researchers in deductive databases have intensively studied integrity-constraint specification and checking. Using integrity constraints to speed up query processing has also been the topic of many papers, especially in the area of deductive and object-oriented databases. In particular, *semantic query optimization* uses semantic information (such as integrity constraints or deductive rules) to speed up query processing.[1] We can use these techniques for data integration. In addition, these studies assume that the constraints are given. Because the constraints on global predicates might not exist a priori in a data integration system, solutions that can compute these global constraints can make these techniques applicable in data integration. Furthermore, because we can describe the constraints at both the source and global levels, we need more optimization techniques that are suitable for this context. For instance, how do we compute global constraints from local constraints, or vice versa, even before we pose user queries? Chun-Nan Hsu and Craig A. Knoblock present a semantic query-optimization approach for heterogeneous multidatabase systems.[2] It provides global optimization for query plans and 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 using the entity relationship model,[3] assuming that we need to design the global schema to integrate data sources and resolve their constraint conflicts. They studied how to compute constraints from local sources to global schemas and how to resolve conflicts.[4]

References

1. U.S. Chakravarthy, J. Grant, and J. Minker, "Logic-Based Approach to Semantic Query Optimization," *ACM Trans. Database Systems* (TODS), vol. 15, no. 2, June 1990, pp. 162–207.
2. C.N. Hsu and C.A. Knoblock, "Semantic Query Optimization for Query Plans of Heterogeneous Multidatabase Systems," *IEEE Trans. Knowledge and Data Eng.*, vol. 12, no. 6, Nov./Dec. 2001, pp. 959–978.
3. M.L. Lee and T.W. Ling, "Resolving Constraint Conflicts in the Integration of Entity-Relationship Schemas," *Proc. 16th Int'l Conf. Conceptual Modeling* (ER 97), LNCS 1,331, Springer-Verlag, 1997, pp. 394–407.
4. M.W.W. Vermeer and P.M.G. Apers, "The Role of Integrity Constraints in Database Interoperation," *Proc. 22nd Int'l Conf. Very Large Databases* (VLDB 96), Morgan Kaufmann,1996, pp. 425–435.

For C1 in particular, the condition price ≥ 250,000 should be satisfied by any database instance of S1, not just for a particular instance. We might want to represent C4 as two local constraints:

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

However, they don't correctly describe C4. These two *local* functional dependencies don't disallow the case where the two sources have two different houses with the same street and zip code. That is, source S1 might have 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, C4 doesn't allow this case. So, if we replace C4 with these two local constraints, we can't take the natural join of S1 and S2 to answer Q3. Local constraints' limitations show the need to use global constraints.

Global constraints

The two kinds of global constraints are *general global constraints* and *source-derived global constraints*.

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, let's assume that each source S_i has a single view v_i defined on the global predicates. (Some constraints such as referential integrity need multiple tables at a source.)

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, we can represent C4 in the housing example as this (general global) constraint:

G1: (street, zip) form a key of the house predicate—that is, (street, zip) $\rightarrow$ (city, price, year, sqft)

The system designer can introduce general global constraints during the design of a data integration system. They capture the application domain's semantics no matter how many sources are in the system. That is, even if new sources join or existing ones leave the system, we assume that any database instance of the global predicates should satisfy these constraints. They can be used to answer queries, as Q3 shows.

Source-derived global constraints

Consider the following view definition for source S1:

S1(s, z, p, f) :- house(s, irvine, z, p, y, f)

Under the open-world assumption, this view means that for each tuple $t_1 = \langle s, z, p, f \rangle$ at source S1, a tuple $t_2 = \langle s, c, z, p, y, f \rangle$ must exist in the predicate house such that the city value c = irvine, even though we don't know year y's exact value. S1 might not have the information about all Irvine houses. On the other hand, we can infer only the existence of tuples such as t_2 (for all tuples in S1), and we don't know whether other tuples exist. So, we can just assume that other tuples don't exist (except those that are 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, because we're certain about those derived tuples on the global predicate, we can introduce the corresponding constraints these derived tuples must satisfy. Owing to the local constraints C1 and C3, the following is a condition that all the derived tuples for the global predicate house should satisfy:

G2: price ≥ 200,000.

G2 is a source-derived global constraint.

To formally define source-derived global constraints, let's 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:

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

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.

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. D could have some unknown values. However, to ensure that 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 (reflected by using functional symbols in the inverse-rule algorithm[6]). We call D a *derived database* of I_s.

Let source views $v_1, \ldots, v_k$ have local constraints $C_1, \ldots, C_k$, respectively. A source-derived global constraint 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 $C_1, \ldots, C_k$.

In the housing example, price ≥ 200,000 is a source-derived global constraint because it's satisfied by any house tuple derived from any database instance of the two sources. On the contrary, price ≥ 300,000 isn't, because there could be a house (derived from a database instance of the two sources) that has a price lower than this one. Similarly, C2 can't derive a global constraint street → (zip, price, sqft) because this functional dependency might not be true for a derived database of the house predicate.

Two observations about source-derived global constraints stand out as important.

First, such a constraint might not hold for the particular domain in general. For example, houses in Orange County could be cheaper than $200,000, violating constraint G2. However, because these houses can't be derived from any database instance of the two sources, we don't need to consider them for the purpose of computing answers to queries using these two sources.

Second, if a source joins the system with its own local constraints, the existing source-derived global constraints could change. For instance, if source S3 also has house information and the prices of all its houses are at least $150,000, then constraint G2 will become price ≥ 150,000. If S4 doesn't have any local constraint on its house prices, then G2 will become invalid. On the contrary, if sources leave the system, we might get more global constraints or existing ones might become more restrictive (for example, price ≥ 150,000 could become price ≥ 180,000).

Motivation to use global constraints

Global constraints have several advantages over local ones. First, some source constraints can't be expressed as local—as shown by C4, which can be easily described as a (general) global constraint. General global constraints easily and more naturally represent conditions of tuples for the global-predicate level.

Second, global constraints generally describe the data at the sources; this description can help users understand the source contents. Knowing that constraint G2 is true, users can better submit queries to meet their needs.

Third, global constraints can make query processing more efficient. Given the global predicate G2, if a query asks for houses cheaper than $130,000, the mediator can immediately know that the answer is empty without checking the source contents and constraints.

Fourth, a data integration system can serve as a subcomponent in a larger-scale system. We could have a hierarchy of data integration systems, in which each node 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. Having the global constraints for each component will be critical for other components to know the contents of the sources in this component. We can use this information effectively in query answering and routing.

We've encountered some problems regarding how to describe constraints so that their rich semantics help the system answer queries. One is how to compute source-derived global constraints from local constraints. Because such a system can be very dynamic—sources 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 for example. One possible approach could be that, 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 choose the most inclusive constraint. For instance, from local constraint C1, we can get a constraint price ≥ $250,000 on the house predicate. From local constraint C3, we can get a constraint price ≥ $200,000 on the house predicate. Combining these two new constraints, we can get price ≥ $200,000, which is the valid source-derived global constraint G2.

The computation process can get complicated if the view definitions have complex operations such as joins and projections. For instance, if we want to "map" the local constraint C2 to the global predicate, we might need this constraint on the predicate house:

> If city = irvine, then functional dependency street → (zip, price, sqft) is true.

This constraint clearly isn't a pure functional dependency, because it's constant based. To capture various source constraints, we might 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 during the system's design, assuming any database of the global predicates satisfies it. We need to compute the corresponding constraints source relations must satisfy. 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, about which intensive research exists, is how to utilize local constraints and global constraints to answer queries effectively and efficiently.[7] For instance, Q3 could be answered using the two sources by utilizing the global constraint G1. It's interesting and challenging to see how to use these techniques to process and optimize queries in the context of data integration. Because the constraints could be described at both the source and the global levels, new techniques need to be developed in this new context.[8] ■

Acknowledgments

My colleagues and I conducted this research at the University of California, Irvine's Raccoon Project (www.ics.uci.edu/~raccoon). I thank Jia Li for the fruitful discussions on this topic.

References

1. G. Wiederhold, "Mediators in the Architecture of Future Information Systems," *Computer*, vol. 25, no. 3, Mar. 1992, pp. 38–49.

2. P. Godfrey, J. Gryz, and C. Zuzarte, "Exploiting Constraint-Like Data Characterizations in Query Optimization," *Proc. 2001 ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, pp. 582–592.

3. J.D. Ullman, "Information Integration Using Logical Views," *Proc. 6th Int'l Conf. Database Theory* (ICDT 97), LNCS 1,186, Springer-Verlag, 1997, pp. 19–40.

4. A. Levy et al., "Answering Queries Using Views," *Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems* (PODS 1995), ACM Press, pp. 95–104.

5. F. Afrati, C. Li, and P. Mitra, "Answering Queries Using Views with Arithmetic Comparisons," *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems* (PODS 2002), ACM Press, pp. 209–220.

6. O.M. Duschka and M.R. Genesereth, "Answering Recursive Queries Using Views," *Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems* (PODS 1997), ACM Press, pp. 109–116.

7. U.S. Chakravarthy, J. Grant, and J. Minker, "Logic-Based Approach to Semantic Query Optimization," *ACM Trans. Database Systems* (TODS), vol. 15, no. 2, June 1990, pp. 162–207.

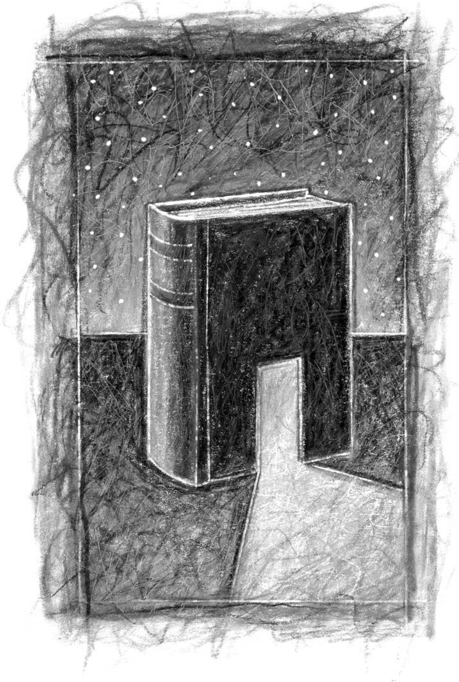
8. C.N. Hsu and C.A. Knoblock, "Semantic Query Optimization for Query Plans of Heterogeneous Multidatabase Systems," *IEEE Trans. Knowledge and Data Eng.*, vol. 12, no. 6, Nov./Dec. 2001, pp. 959–978.

The Author

Chen Li is an assistant professor in the School of Information and Computer Science at the University of California, Irvine. His research interests are in database and information systems, including data integration and sharing, data warehouses, data cleansing, multimedia databases, and XML. He received his PhD in computer science from Stanford University. He is a member of the ACM. Contact him at the Dept. of Information and Computer Science, Univ. of California, Irvine, CA 92697; chenli@ics.uci.edu; www.ics.uci.edu/~chenli.

For more information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.

BOOK REVIEWERS



Looking to contribute?
Don't have much time?

Review a book for *Distributed Systems Online*.
Volunteer your services to a great publication.
Gain knowledge (and get a **free book!**).

Select your book online at http://
dsonline.computer.org/books/list.htm

IEEE distributed systems ONLINE
Expert-authored articles and resources