# Efficient Query Processing in XML-Based Information Integration

Dongfeng Chen
Computer Science Dept.
NC State University
dchen3@ncsu.edu

Rada Chirkova
Computer Science Dept.
NC State University
chirkova@csc.ncsu.edu

Fereidoon Sadri
Dept. of Computer Science
UNC Greensboro
sadri@uncg.edu

Timo J. Salo
IBM RTP
Research Triangle Park, NC
tjsalo@us.ibm.com

## ABSTRACT

The problem of decentralized data sharing is relevant for a wide range of applications and is still a source of major theoretical and practical challenges, in spite of many years of sustained research in information integration. We focus on the challenge of efficiency of query evaluation in information-integrations systems, with the objective of developing query-processing strategies that are widely applicable and easy to implement in real-life applications. In our algorithms we take into account important features of today's data-sharing applications, namely: XML as likely interface to or representation for data sources; potential for information overlap across data sources; and the need for inter-source processing (i.e., joins of data across data sources) in many applications.

Our proposed approaches are applicable both to the original stored data and to materialized views, including restructured views, which are a framework for representing, e.g., the *pivot* operation available in many database-management systems. To the best of our knowledge, our methods are the first to account for the practical issues of information overlap across data sources and of inter-source processing. While most of our algorithms are platform- and implementation-independent, we also propose XML-specific optimization techniques that allow for system-level tuning of query-processing performance. Finally, using real-life datasets and our implementation of an information-integration system shell, we provide experimental results that demonstrate that our algorithms are efficient and competitive in the information-integration setting.

## 1. INTRODUCTION

The need for decentralized data sharing arises naturally in a wide range of applications, including enterprise data management, scientific projects undertaken across universities or research labs in domains such as biology and astronomy, and data sharing among governmental databases. Historically, significant research effort has been directed toward information-integration systems that query data sources through a single central point with a fixed *mediated* schema [39]. In the mediator-based framework, the need to establish the mediated schema and translation rules, or *mappings,* between the data sources and the central mediator is a major bottleneck in integration efforts for real-life applications [37].

Peer data-management systems [18] address some aspects of this schema-mediation problem by having physical peers use agreed-on *local* mappings between their schemas. At the same time, the need to compose peer-to-peer schema mappings in query evaluation may result in inferior query-processing efficiency. (Please see [37] and Section 5 for the details.) On the other hand, query processing in mediator-based information-integration systems is typically addressed in research literature at the level of rewriting queries posed on the mediator using the given mappings (see, e.g., [39]), rather than at the level of providing query-execution strategies that would be efficient in implemented systems.

Systems built for real-life applications provide further evidence of the importance of the query-processing challenge in information integration. For instance, many ontology-based [30] information-integration systems [14, 21, 29, 33] use the "materialization approach," by importing into a single central data repository the information from all the data sources, usually in the form of RDF or OWL data, and then use Semantic-Web [35] tools and languages to give users access to the information accumulated in such a central data warehouse. Other such systems evaluate queries by simply identifying relevant sources and processing the query on each source independently. Note that both approaches assume that the problem of establishing mappings between the data sources and the mediator has been solved successfully.

In general, there could be a number of ways to process user queries, with widely different performances. No single query-processing strategy would be optimum for all queries and cases. Rather, an intelligent query-optimization approach would be to be able to choose from a number of alternatives. In this paper we focus on the problem of efficiency of query evaluation in information-integrations systems, with the objective of developing query-processing strategies that are widely applicable and easy to implement in real-life applications. In our algorithms we take into account the following important features of today's data-sharing applications:

- *XML as representation for data sources:* The recent advent of XML as a standard for online data interchange (and perhaps even for data storage, see, e.g., the datasets of [10, 36]) holds much promise toward promoting interoperability and data integration.
- *Overlapping information in data sources:* In many practical applications (e.g., banking or medical information

systems) data sources may overlap on the data they store, such as information about user accounts in individual bank branches.

- *Inter-source processing:* Some applications (e.g., banking or data sharing among governmental agencies) require evaluation of queries that involve joins of data stored across data sources.

While our theoretical results and most of our proposed techniques are implementation neutral and thus applicable in a variety of settings for information-integration systems, we are also introducing platform-specific approaches (such as our semantic optimization for XQuery[1] in Section 2.5) that allow for system-level tuning of query-processing performance.

To compare and rate the proposed approaches, we have obtained experimental results (see Section 4) using our information-integration system shell [28] that incorporates an implementation of all the algorithms and optimizations proposed in this paper. This software system enables interaction between (i) data sources that store data using the XML data model, and (ii) relational mediators whose schemas conform to the Semantic Model introduced in [22]. In the semantic-model approach, the data at each source are viewed as a collection of (logical) binary relations, which is called *semantic-model view*. (See Appendix B for illustrative examples.) To include a data source in the integration effort, the source owner provides mappings from the source data to the semantic-model view. The semantic-model approach is consistent with recent work on large-scale information integration [13, 25], as well as with approaches based on ontological modeling [29, 30] in the Semantic-Web initiative. For methods for building source-to-mediator mappings in this setting, we refer the reader to [11, 12, 32] and to references therein. Our system can also serve as a building block in a three-tier architecture [28] for information integration, which is the subject of our ongoing work. The purpose of the architecture is to decouple query processing and optimization into "intra-coordinator processing" (largely covered in this paper) and "inter-coordinator processing" (conceptually analogous to query processing in peer-to-peer systems), thus keeping the optimization choices relatively local and improving the overall query-processing performance of the system.

**Our specific contributions** are as follows:

1. We propose query-processing algorithms for information integration; the algorithms do *not* involve building source-to-mediator mappings (in contrast with the methods of item 3 here), and are platform- and implementation-independent. To the best of our knowledge, our methods are the first that account for information overlap and for inter-source processing.
2. We present theoretical results that allow for further reduction of inter-source processing by using information about integrity constraints in the data sources.
3. We develop an end-to-end suite of algorithms for efficient query processing in presence of *materialized restructured views* [4] (as exemplified by, e.g., operation *unfold*, or *pivot* [7, 24]); the algorithms are applicable to views materialized in both mediators and data sources.
4. We propose XML-specific optimization techniques that allow for system-level tuning of query-processing performance.
5. Using real-life datasets and our implementation of an information-integration system shell [28], we report experimental results that demonstrate that our algorithms are efficient and competitive.

[1]Our proposed semantic-optimization techniques are also more widely applicable to general XQuery optimization.

The remainder of the paper is structured as follows: In the main text, we first describe our basic query-processing algorithms in Section 2. Then, Section 3 introduces algorithms using materialized restructured views. Section 4 presents our experimental results; we conclude with an overview of related work in Section 5. Appendix A presents our formal results, while Appendix B illustrates the semantic-model approach [22] adopted in our implementation [28].

# 2. QUERY-PROCESSING ALGORITHMS

In this section we present our basic query-processing algorithms for information integration. These algorithms are applicable both to the original stored data (in the data sources) and to materialized views (either in the data sources or in the mediator), as long as all the relevant source-to-mediator mappings have already been constructed. (Contrast this with the approaches presented in Section 3.) All our algorithms, with the exception of semantic optimization for XQuery (see Section 2.5), are platform- and implementation-independent. To the best of our knowledge, our methods are the first that account for information overlap across data sources, as well as for inter-source processing.

We also introduce theoretical results that allow for further reduction of inter-source processing by using information about integrity constraints in the data sources. Specifically, these results allow us (1) to determine when no inter-source processing is needed, and, (2) in case inter-source processing cannot be avoided, to determine minimal equivalence sets of subqueries that are adequate to provide the complete answer to a user query. Please see Appendix A for the details.

## 2.1 The Materialization Approach

This is the base approach against which we evaluate other query-processing strategies. In the simple *materialization approach,* we precompute those mediator-based data that are relevant to the user query, and execute the query on the materialized data. Example 1 illustrates the approach.

EXAMPLE 1. *Consider a system with four information sources and a mediator. Suppose query* $Q = \Pi_{B,C}\ \sigma_P(r \bowtie s)$ *involves relations* $r(A, B)$ *and* $s(A, C)$ *in the mediator schema, with selection condition* $P$*. Assume XML data sources; Figure 1 shows a small part of two of the sources. Note that the sources have different schemas.*

```
...                          ...
  <X>                          <Z>
    <A>a1</A>                    <C>c3</C>
    <B>b1</B>                    <W>
    <Y>                             <A>a2</A>
      <C>c1</C>                  </W>
      <C>c2</C>                </Z>
    </Y>                       <Z>
  </X>                           <C>c4</C>
  <X>                            <W>
    <A>a2</A>                       <A>a3</A>
    <B>b2</B>                       <B>b3</B>
  </X>                           </W>
...                            </Z>
                               ...
Source 1                     Source 2
```

**Figure 1: Part of information in two data sources.**

*In the materialization approach, we create two materialized relations (fragments) for $r(A, B)$ and $s(A, C)$ in each source. The conditions of predicate $P$ that involve only $r$ or $s$ are enforced at this point. The queries to create these fragments are generated using the mapping rules for each source. The materialized relations ($r_1$ and $s_1$ for Source 1, $r_2$ and $s_2$ for Source 2) are sent to the mediator. The mediator merges these relations and executes the user query on them.* ■

## 2.2 Subquery-Based Approach

This approach is based on generating and executing local and inter-source subqueries for the user query, and merging their results to obtain the query answer. A *local subquery* is executed on data from a single source, while an *inter-source subquery* is executed on data from multiple sources.

EXAMPLE 2. *Consider the user query $\Pi_{B,C}\ \sigma_P(r \bowtie s)$ of Example 1. We have (up to) $4^2 = 16$ subqueries.*[2] *Only four of these subqueries, $\Pi_{B,C}\ \sigma_P(r_i \bowtie s_i), i = 1, \ldots, 4$, are local; the remaining twelve are inter-source subqueries. The local subqueries are translated to queries on the source schemas and executed locally; the results are sent to the mediator. For an inter-source subquery such as $\Pi_{B,C}\ \sigma_P(r_1 \bowtie s_2)$, either the data for $r_1$ is sent to Source 2, or the data for $s_2$ is sent to Source 1. (We have implemented in [28] algorithms for data transmission and subquery execution for XML data sources.) Finally, the mediator merges all partial results in order to obtain the final answer to the user query.* ■

Algorithm 1 provides the specific pseudocode for the XQuery version of generating inter-source subqueries.

**Algorithm 1**: Inter-source subquery generation

```
input  : User query Q, set of source-to-mediator mappings
output: Inter-source subquery(XQuery) Q'
foreach predicate in Q do
    create one variable p for predicate, specifying data locations;
    for each attribute in predicate, create one variable using p;
end
construct a FOR clause based on the above variables;
copy the WHERE clause from Q into Q';
replace binary predicates in the WHERE clause with the
  corresponding variables in the FOR clause;
foreach head in Q's head elements do
    generate its RETURN string with the corresponding
    variables in the FOR clause;
end
concatenate the FOR, WHERE, and RETURN clauses;
return XQuery Q' with FLWOR expressions;
```

## 2.3 The Optimized-Subquery Approach

The *optimized-subquery approach* uses our formal results to eliminate, to the extent possible, inter-source processing. Based on key and foreign-key constraints that are relevant to the query, all or some of inter-source subqueries may be redundant and will not be evaluated. The savings can be substantial; for instance, given a query involving $k$ mediator-based relations in a system with $n$ data sources, there are only $n$ local subqueries, while the number of inter-source subqueries can be as large as $n^k - n$. The details are presented in Appendix A.

[2]If source $i$ has no data for relation $r$ then all subqueries involving fragment $r_i$ are empty and need not be executed.

EXAMPLE 3. *In the setting of Example 2, suppose that attribute $A$ is the key for $r$ and that a foreign-key constraint holds from $s.A$ to $r.A$. Then, by Theorem 1 (see Appendix A), no inter-source processing is needed. This reduces the processing from sixteen to four (all local) subqueries that are translated and executed locally on the data sources.* ■

## 2.4 The Wrapper Approach

In the *wrapper approach,* we generate only one local query per data source; the query extracts from the source the minimum amount of information that is needed to answer the user query. We call this the "wrapper" approach because this extraction can be viewed as a (query-specific) wrapper that collects the needed information from each source. Compared to the approaches based on subqueries (Sections 2.2–2.3), the information extracted from each source in the wrapper approach is richer than the result of the local subquery on the same source, thus making it possible to obtain the full answer to the user query by further processing. Intuitively, the information collected by the local (wrapper) query corresponds to the full outer-join of the relations involved; see Example 4 for a more detailed discussion. In a large class of applications, an efficient *chase*-based algorithm [38] can be applied to the extracted information to obtain the full answer to the user query.

**Algorithm 2**: The Wrapper Approach

```
input  : User query, set of sources Srcs and their mappings
output: Single XML document Doc
foreach source in Srcs do
    create a local subquery s for source;
    execute s locally, then send result to the mediator;
end
merge the local results at the mediator;
if isInterSourceProcessingNeeded() then
    chaseSteps();
end
eliminate duplicates in the query answer;
save the final answer into XML document Doc;
return Doc;
```

EXAMPLE 4. *Consider the user query $\Pi_{B,C}\ \sigma_P(r \bowtie s)$ of Example 2. Suppose that $A$ is the key of $r$, but no foreign-key constraint holds from $s$ to $r$ (thus, according to Theorem 2 of Appendix A, inter-source processing* is *needed.) In the wrapper approach, each source $i$ generates a relation $t_i(A, B, C)$ corresponding to the full outer-join of $r_i$ and $s_i$ and sends it to the mediator. The mediator combines there relations, applies the chase, and enforces the query conditions and projections. In our example, Source 1 (see Figure 1) has the following tuples (among others): (a1, b1) and (a2, b2) for $r_1$, and (a1, c1) and (a1, c2) for $s_1$. Hence $t_1$ contains (a1, b1, c1), (a1, b1, c2), and (a2, b2, null). Source 2 has tuple (a3, b3) for $r_2$ and tuples (a2, c3) and (a3, c4) for $s_2$. Then $t_2$ contains (a2, null, c3) and (a3, b3, c4). The result of unioning $t_1$ and $t_2$ and chasing with respect to the key constraint generates a new tuple (a2, b2, c3) (among others) in the result. The final step is to enforce predicate $P$ and to project onto output attributes $B, C$. The answer contains (b2, c3) (unless filtered out by $P$).* ■

Note that in the subquery-based approaches (Sections 2.2–2.3), the answer (b2, c3) of Example 4 is generated by inter-source subquery $\Pi_{B,C}\ \sigma_P(r_1 \bowtie s_2)$. Thus, while executing

only local queries, the wrapper approach is able to generate at the mediator the results of inter-source subqueries.

Algorithm 2 provides the pseudocode for the wrapper approach for XML data sources. Function *isInterSourceProcessingNeeded*() is based on our formal results (see Appendix A), and function *chaseSteps*() is based on the results described in [38].

## 2.5 Semantic Optimization for XQuery

In the approaches presented in Sections 2.2–2.4, the system generates queries to be executed on local data (e.g., XQuery queries on XML data). In our implementation framework of [28], user queries tend to have a relatively large number of joins of binary relations in the semantic-model view (see Appendix B for illustrative examples). When the above translation algorithms are adapted to this setting, they create one variable in the XQuery query for each binary relation in the user query. Our platform-specific semantic optimization rewrites XQuery queries into more efficient equivalent queries with fewer joins and variables. The algorithm uses information from mapping rules and from key constraints of the binary relations in the semantic model. Instead of creating one variable for each binary relation in the user query, the new algorithm generates a single variable for *all* binary relations that have a common "glue" variable (Appendix B) in their mappings and the same key. Consider the following illustration of our semantic-optimization rewriting process.

EXAMPLE 5. *Suppose an application involves multiple data sources with information on stocks (see Figure 2 and Section 4 for the background). The semantic-model view for this application contains binary relations* `k-ticker, k-year, k-month, k-day, k-price, k-priceType`*, where* `k` *is the unique key. Each relation name refers to a nonkey stock attribute. For example,* `k-ticker` *has two attributes:* `k` *is a unique key, and* `ticker` *is the ticker (stock) id. For each stock, four price types are recorded: open, close, high, and low. A mapping rule for* `k-ticker` *in an XML source could be:*

```
k-ticker($X, $Y) <- /stocks/stock $G,
                    $G/@uid $X, $G/ticker $Y
```

*As an example for the improvement obtained by semantic query rewriting, consider a user query that produces the average closing price for IBM in October 2005. This query uses five of the above binary relations, and a corresponding XQuery will have five variables on the XML stocks document. But since these five binary relations have the common glue variable* `/stocks/stock` *and the same key* `k`*, we would obtain the following local subquery with just one variable:*

```
LET $br := /stocks/stock[year=2005 and month='Oct' and
  ticker='IBM' and prices/priceType = 1 ]
RETURN avg($br/prices/price)
```

The effect of our semantic optimization, as demonstrated by the experimental evaluation in Section 4, can be significant (up to two orders of magnitude for certain queries).

## 2.6 Merging XML Data

Given two or more XML documents on the same schema, our *merge algorithm* produces one XML document on the same schema; the document contains all the data from the input documents. In case merging the input documents is not possible, the algorithm outputs the discrepancies that hindered the merge operation. The algorithm can be used as a subroutine in platform-specific instantiations of the other algorithms proposed in this section. The pseudocode for merging XML data is shown in Algorithm 3; due to the space limit for this paper, the pseudocode omits some cases considered by the algorithm.

**Algorithm 3**: The Merge Algorithm

```
   input : XML documents and their schemas
   output: Single XML document D
 1 parseSchema(fileStr);
 2 D = mergeAll(dirStr);
 3 Procedure parseSchema(schemaStr)
 4 retrieve keys, unique nodes and other constraints from schema;
 5 treewalk all elements from root and classify elements into types
   (single required leaf, single optional leaf, etc...);
 6 Function mergeAll(dirStr)
 7 foreach XML in the directory dirStr do
 8     DocumentnextDoc = getDoc(XML);
 9     rsDoc = mergeTwoDocs(rsDoc, nextDoc);
10 end
11 write rsDoc into an XML document D;
12 return D
13 Function mergeTwoDocs(rsDoc, nextDoc)
14 get root element root1 from rsDoc;
15 get root element root2 from nextDoc;
16 foreach element under root1 do
17     mergeElements(element, root2, rsDoc, nextDoc);
18 end
19 return rsDoc;
20 Procedure mergeElements(e1, eleInNextDoc, rsDoc, nextDoc)
21 determine e1's type typeVal;
22 switch typeVal do
23     case Single Required Leaf
24         check existence of corresponding element e2 in nextDoc;
25         compare e1's value with e2's value;
26         if e2 does not exist or same value as e1: print ERROR;
27         break;
28     end
29     case Single Optional Leaf
30         if e2 exists, compare e1's value with e2's value;
31         break;
32     end
33     case Single Required NonLeaf
34         check existence and size of eleInNextDoc in nextDoc;
35         ele2 = eleInNextDoc's child;
36         foreach element under e1 do
37             mergeElements(element, ele2, rsDoc, nextDoc);
38         end
39         break;
40     end
41     ...
42     otherwise
43         print ERROR
44         break;
45     end
46 end
```

As an example, consider several XML documents of *personnel* information. Certain natural consistency constraints are expected to hold on this kind of data. (For example, each individual has a social security number, SSN, that uniquely identifies the person's name and date of birth. A person may have multiple phone numbers, but the date of birth should be unique, etc.) The output of running the merge algorithm on such personnel data should contain all individuals mentioned in the input files. In addition, the information for one individual (determined by the same SSN in different inputs) is combined in the output document in the intuitive way: The date of birth of the same individual from different inputs, if known, should be identical. If not, merge is not possible and a discrepancy in the date of birth of this individual is identified. Further, phone numbers from multiple inputs for the same individual are all included in the

merged information for that individual. In our information-integration system shell [28], merge is halted if a discrepancy is detected. Many other approaches are possible, including approaches that use information about the degree of reliability of sources to guide the merge in presence of discrepancies, and can be incorporated with relative ease.

# 3. QUERY OPTIMIZATION USING MATERIALIZED RESTRUCTURED VIEWS

In this section we present an end-to-end suite of algorithms for efficient query processing in presence of materialized restructured views [4].[3] An important part of the suite is an algorithm for providing mappings between the definitions of restructured views and the mediator schema; the algorithm uses as inputs the mappings that are available to the algorithms of Section 2. All the algorithms presented in this section are applicable to views materialized both in mediators (resulting in a version of the materialization approach of Section 2.1) and in data sources (resulting in a version of the subquery-based approaches of Sections 2.2–2.3). Our experimental results demonstrate that using restructured views may result in orders-of-magnitude improvement in query-processing time for certain classes of queries.

## 3.1 Restructured Views: An Overview

The concepts of restructuring and restructured views stem from a number of projects, including those described in [16, 20, 23]. In a restructured view, some data from the base table(s) are represented as metadata — that is, schema information, such as table and attribute names — or vice versa. Intuitively, by moving data values that appear frequently and repeatedly in a relation to metadata in the view, we can represent the information in the view more compactly than via regular view. This size reduction, together with the physical clustering that results from the restructuring, may yield impressive improvements in query-processing time [4].

Consider, for example, the `Stocks` relation of Figure 2, which lists information about stocks over a number of years. Figure 3 shows `stocksByTicker`, a restructured view which represents the same information as `Stocks`, but the tickers (`ibm`, `msft`, ... , `dell`) now play the role of attribute names, and the stock values for a given price type are organized "horizontally" into a single tuple for each day. Note that each tuple of the view `stocksByTicker` represents the information from many tuples of the base table `Stocks`. For example, if the `Stocks` table stores information about 20 stock tickers, then each tuple of `stocksByTicker` represents the same information as the corresponding 20 tuples in the `Stocks` table. This form of restructuring has been called the *unfold* (or *pivot*) operation [7, 24]. Another form of restructuring, known as the *split* operation [24], is analogous to horizontal partitioning based on the values of an attribute.

## 3.2 Using Restructured Views in Information Integration

We now discuss our approach to integrating restructured views into the information-integration framework.

### 3.2.1 Defining restructured views

Restructured views can be defined by s-LOG mappings [4] from the mediator schema. For example, the `stocksByTicker` view of Figure 3 can be defined as:

[3][4] focuses on restructured views in centralized databases.

| ticker | year | month | day | priceType | price |
|---|---|---|---|---|---|
| ibm | 2005 | Oct | 25 | open | 91.43 |
| ibm | 2005 | Oct | 25 | close | 92.14 |
| ibm | 2005 | Oct | 25 | low | 91.06 |
| ibm | 2005 | Oct | 25 | high | 92.55 |
| ... | ... | ... | ... | ... | ... |
| msft | 2005 | Oct | 25 | open | 88.92 |
| msft | 2005 | Oct | 25 | close | 90.12 |
| ... | ... | ... | ... | ... | ... |

**Figure 2: Base relation `Stocks`.**

| year | month | day | priceType | ibm | msft | ··· | dell |
|---|---|---|---|---|---|---|---|
| 2005 | Oct | 25 | open | 91.43 | 88.92 | ··· | 56.27 |
| 2005 | Oct | 25 | close | 92.14 | 90.12 | ··· | 55.78 |
| 2005 | Oct | 25 | low | 91.06 | 88.58 | ··· | 53.47 |
| 2005 | Oct | 25 | high | 92.55 | 91.63 | ··· | 56.55 |
| ... | ... | ... | ... | ... | ... | ... | |

**Figure 3: Restructured view `stocksByTicker` based on the `Stocks` data of Figure 2.**

```
stocksByTicker (year:Y,month:M,day:D,priceType:R,T:P) <-
k-ticker(k:K,ticker:T),k-year(k:K,year:Y),k-month(k:K,month:M),
k-day(k:K,day:D),k-priceType(k:K,priceType:R),k-price(k:K,price:P).
```

In order to further reduce the size of the restructured view, we decompose it into a set of restructured views, one for each ticker: `stocksByIBM`, `stocksByMSFT`, and so on. For instance, the s-LOG rule for `stocksByIBM` is as follows:

```
stocksByIBM (year:Y,month:M,day:D,priceType:R,ibm:P) <-
k-ticker(k:K,ticker:'ibm'),k-year(k:K,year:Y),k-month(k:K,month:M),
k-day(k:K,day:D),k-priceType(k:K,priceType:R),k-price(k:K,price:P).
```

Similarly, if we take `priceType` (whose value is one of "open", "close", "high" and "low") and `price` as parameters for the pivot operation, we get a restructured view `stocksByPrice Type(ticker,year,month,day,open,close,min,max)`. We can also define *aggregate* restructured views. As an example, consider the following view definition that stores the monthly average closing prices for the tickers.

```
monthlyAvgByTicker (year:Y,month:M,priceType:R,T:avg(P)) <-
k-ticker(k:K,ticker:T), k-year(k:K,year:Y), k-month(k:K,month:M),
k-priceType(k:K,priceType:'close'), k-price(k:K,price:P).
```

### 3.2.2 View materialization

Once restructured views are defined (e.g., by the database administrator), one can use our algorithm for automatically generating queries that materialize the view answers. The algorithm composes the mappings from each data source to the mediator schema with the mapping from the mediator schema to the restructured view; the latter can be obtained automatically from s-LOG view definitions. The pseudocode for the XML-based setting is given by Algorithm 4.

As an illustration, for the view `stocksByIBM` defined above one would obtain the query:

```
<stocksByIBM>{
FOR $br in /stocks/stock
WHERE $br/ticker = 'ibm'
RETURN
  <stock>
    {$br/year}
    {$br/month}
    {$br/day}
    {$br/priceType}
    <ibm>{$br/price/text()}</ibm>
  </stock>
}</stocksByIBM>
```

**Algorithm 4**: Materialization of restructured views

**input** : Source-to-mediator mappings and definition of restructured view
**output**: Query Q to materialize the restructured-view answer

1 Examine the relations defining the restructured view, and their key constraints;
2 Construct FOR/LET clauses based on the relations appearing in the view definition;
3 Generate the WHERE clause if there is any constraint in the view definition;
4 Set up the RETURN clause from the head of the view definition;
5 Concatenate the FOR/LET, WHERE, and RETURN clauses;
6 Return query Q.

### 3.2.3 Query rewriting

We now propose an algorithm for taking advantage of restructured views, by using them to rewrite user queries. Algorithms for view usability and query rewriting for materialized restructured views have been presented in [4], and are used in steps 2 and 3 of (the relational version of) Algorithm 5 that summarizes our approach. The novelty for the information-integration setting is in an extension of the algorithm (not discussed here due to the space limit) to the case of restructured views defined in XML-based data sources. Another extension of the algorithm allows one to obtain rewritings involving multiple views at a time.

**Algorithm 5**: Query rewriting using restructured views

**input** : User query $Q$ and set of restructured views $V$
**output**: Equivalent query $Q'$ that uses views in $V$

1 **foreach** *view $v$ in $V$* **do**
2  **if** *$Q$ can be rewritten into an equivalent query that uses $v$* **then**
3  Rewrite $Q$ using $v$;
4  Output the rewritten query;
5  **end**
6 **end**

EXAMPLE 6. *The following user query lists the average closing price for IBM in October 2005:*

```
Q(monthlyAvg:avg(P)) <-
   k-ticker(k:K, ticker:'IBM'), k-year(k:K, year:2005),
   k-month(k:K, month:'Oct'), k-priceType(k:K, priceType:'Close'),
   k-price(k:K, price:P)
```

*Our proposed algorithm can be used to rewrite Q into a query that uses view* `monthlyAvgByTicker (select t.ibm from monthlyAvgByTicker where t.year = 2005 and t.month = 'oct')`:

```
select m.oct
from   monthlyAvgByMonth m
where  m.year = 2005 and m.ticker = 'ibm'
```

■

# 4. EXPERIMENTAL EVALUATION

In this section we present the results of our evaluation of the impact of our query-processing techniques (introduced in Sections 2 and 3) on the query performance in our information-integration system shell [28]. Our experimental results demonstrate that our algorithms are efficient and competitive.

## 4.1 Experimental Setup

We carried out two sets of experiments on different datasets and queries. In our first experiments we used the setup of [37], with the modification that in addition to the queries used in [37] we defined extra queries that would require more inter-source processing. The setup includes the "DB-research" dataset, which contains data sources pertaining to several universities, research organizations, and publication information from sources such as CiteSeer [6], DBLP [10], and SIGMOD [36]. Our mediator schema (which uses the semantic-model view, see Appendix B for an overview) for the DB-research dataset, shown in Figure 4, is based on the *academic department ontology* [1] from the DAML ontology library [8]; the queries are shown in Figure 5. (The first eight queries are from [37]; Q9 is our extra query.)

```
[proceeding]
proceeding-title
proceeding-year
proceeding-location
proceeding-gc(general-chair)
proceeding-pc(program-chair)
proceeding-member(program-committee)

[paper]                  [project and person]
paper-title              project-leader
paper-author             project-member
paper-conference         project-paper
paper-cite               project-area
paper-status             project-topic
                         person-adviser
[review]                 person-advisee
review-reviewer          person-affiliation
review-rating            person-homepage
review-paper
review-comment
```

**Figure 4: Semantic model for DB-Research.**

```
Q1: Find all XML-related projects
Q2: Find all projects involving a given person
Q3: Find all co-authors of a given researcher
Q4: Find all papers by a given pair of authors
Q5: Find papers by faculty leading XML projects
Q6: Find J. Shanmugasundaram's VLDB'99 paper
Q7: Find recent PC chairs and their papers
Q8: Find UC Berkeley people and their web pages
Q9: List persons who are PC chairs at a conference
    both in 2003 and in 2004
```

**Figure 5: Queries for the DB-research experiments.**

In our second set of experiments we downloaded stock information from the Internet for various stocks (tickers), and loaded it into sources with different XML schemas. We also developed a simple semantic-model view based on an ontology for stocks consisting of binary relations `k-ticker`, `k-year`, `k-month`, `k-day`, `k-price`, and `k-priceType`, where `k` is a unique key. (See Example 5 for more information.) User queries for these experiments are listed in Figure 6.

The experiments were executed using PostgreSQL [31] version 8.1.3 and SAXON [34] version 8 on a 2.0 GHz Pentium M computer with 768 MB memory and 40 GB hard disk running Windows XP Pro. All runtimes reported in this section are the average of five runs of the queries.

## 4.2 Experimental Results

Figures 7 and 8 illustrate the performance of our algorithms for the DB-Research dataset experiments. The approaches are as discussed in Section 2. The semantically optimized versions of some of the algorithms, shown in the graphs as `Subqueries*`, `Optimized Subqueries*` and `Wrapper*`, apply the semantic optimization to the queries then

```
Q1: List the closing price for ibm on 2005/11/21
Q2: List the average closing price for ibm in
    October 2005
Q3: List stocks that had high price at least 20%
    higher than low price at least once in 2005
Q4: List dates in 2005 when ibm's closing price
    was at least 50% higher than msft's
Q5: List tickers that show a rapid increase in the
    fourth quarter of 2005 (those whose average
    closing price in the 4th quarter of 2005 has
    at least 10% increase each month)
Q6: List the maximum closing price for ibm in
    October 2005
Q7: List days when ibm's price decreased rapidly
    by 10% or more in two consecutive days
```

**Figure 6: User queries for the Stocks experiments.**

execute them. Figure 8 also shows optimization using the restructured-views technique on a subset of the queries.

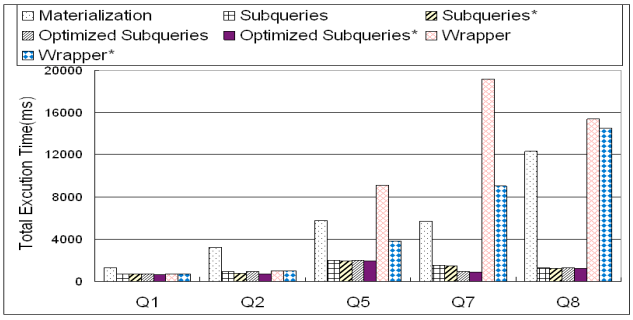


**Figure 7: Experimental results (1) for DB-Research, with all runtimes given in milliseconds.**

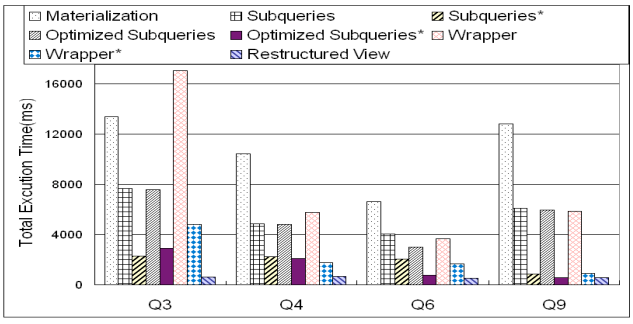


**Figure 8: Experimental results (2) for DB-Research, with all runtimes given in milliseconds.**

The horizontal (X) axes of Figures 7 and 8 list the queries, and the vertical (Y) axes the respective execution times in milliseconds. As shown in the figures, the `Subqueries*` and `Optimized-Subqueries*` techniques seem to be the algorithms of choice for most cases (Figure 7), except for cases where optimization using materialized restructured views is applicable and yields better performance (Figure 8). The step of materializing relations in `Materialization` and the chase steps in `Wrapper` were time consuming in this experiment. Sometimes (e.g., for *Q4*) the `Wrapper*` method is more efficient than `Optimized Subqueries*`, in cases where the data are very regular at each source, and each source contributes to all relations in a user query.

Our second set of experiments uses the `Stocks` dataset (see Figure 2) and is primarily intended to show the impact of optimization using restructured views. The restructured views that were used in our experiments are listed in Figure 9, see Section 3 for some of the definitions.

```
stocksByOpen    stocksByIBM    monthlyAvgByIBM    monthlyAvgByJan
stocksByClose   stocksByMSFT   monthlyAvgByMSFT   monthlyAvgByFeb
stocksByLow     stocksByCSCO   monthlyAvgByCSCO    ...
stocksByHigh     ...             ...              monthlyAvgByDec
```

**Figure 9: Restructured views for the Stocks dataset.**

We carried out extensive experiments to evaluate the effect of query optimization using restructured views. The results, reported in Figure 10, clearly show the superiority of query optimization using our restructured-view algorithm over the other options. Note that the Y-axis, the execution time (measured in milliseconds), is logarithmic. Hence the improvement over the other techniques amounts to one to three orders of magnitude. Note also that the semantic optimization (applied on top of the optimized-subquery approach), which uses key-constraint information to optimize the XQuery subqueries, yields an order of magnitude or better performance than that of the (unmodified) optimized-subquery approach.[4] We expect the gains obtained by the optimized algorithm to be typical for all user queries with moderate to large number of joins. In addition, our optimization using restructured views can be applied whenever the data exhibit certain regularity properties, thus resulting in substantial size reduction via restructured views.

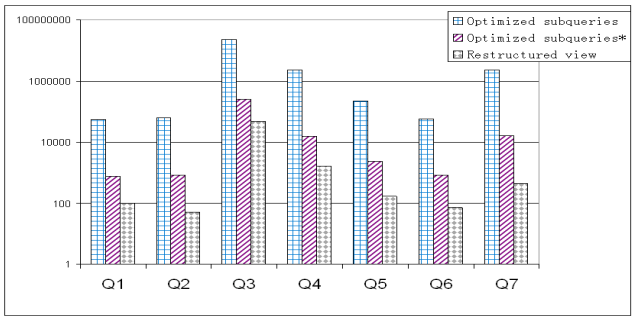


**Figure 10: Performance comparison (on the Stocks dataset) for optimization using restructured views.**

For each query of Figure 6, the restructured views used for Figure 10 are as follows (please see Section 3 for the details on some of the view definitions):

- `Q1.` Q1 requires a simple selection. Either view `stocksByIBM` or `stocksByClose` is usable for Q1. We chose `stocksByIBM` for Q1 in the experiment.
- `Q2.` Q2 involves a simple selection and aggregation. The view `monthlyAvgByIBM` is used in Q2's rewriting.
- `Q3.` Q3 has self joins. Both `stocksByHigh` and `stocksByLow` are utilized for Q3.

[4]Depending on the allocated memory, large numbers of variables in an XQuery query may have a pronounced effect on execution time by SAXON.

Q4. Similarly, we take advantage of both `stocksByIBM` and `stocksByMSFT`.

Q5. In Q5's implementation using restructured views, three views are involved: `monthlyAvgByOct`, `monthlyAvgByNov` and `monthlyAvgByDec`.

Q6. Q6 looks for the maximum closing price for IBM in October 2005. We use the MAX aggregation function on `stocksByIBM`.

Q7. This query has a self join on `stocksByIBM`.

# 5. RELATED WORK

In recent years there has been a surge of interest in information integration, as well as in interoperability and its applications (see, e.g., [17] and references therein.) The importance of large-scale integration (web-scale integration) and the "pay-as-you-go" paradigm in such environments have been observed in many recent publications (see, e.g., [25]). There has been a revitalization of ontological modeling as a result of the W3C Semantic Web initiative [35]. Some information-integration systems have been proposed and implemented using ontological-modeling concepts. For example, the ICS-FORTH Semantic Web Integration Middleware (SWIM) [5] uses Semantic-Web tools for integration. The main differences between these and our approach are in our use of database tools and concepts, and in our emphasis on query optimization. Further, we pay special attention to queries that require data from multiple sources (inter-source processing), while this important issue seems to have not been addressed in SWIM or other projects.

In a recent report on indexing large volumes of data [13], the authors advocate a "triple" model as a global model for all data. Triples are closely related to RDF and ontologies, and to the semantic-model approach (see Appendix B) used in our implementation [28], demonstrating that our semantic-model views are valid and natural for modeling the information contents of information sources.

The Piazza and related projects (see [18] and references therein) cover various aspects of large-scale data integration, including (1) peer-based data management, (2) schema mapping, and (3) theoretical foundations, indexing, and access control. The main idea behind data integration/interoperability in Piazza is that users provide mappings between pairs of information sources. There is no need to provide mappings for all pairs. In fact, all that is needed is that the graph that represents sources (via nodes) and available mappings be connected. Mappings between any two sources can then be obtained by composing the pairwise mappings along a path connecting the two sources [26, 37]. While this approach works well for sources belonging to the same application domain and with similar data, in other scenarios the composition process might result in information loss. In contrast, our experimental setting is based on mappings from data sources to simple ontology-based semantic-model views and can be applied to federations of sources with different (as well as similar) information. In our approach mappings are local: No knowledge about other sources is needed. Another advantage of our approach is that the system is more resilient to erroneous mappings, in that only one source is affected by an erroneous mapping. In contrast, an erroneous mapping in Piazza affects all query processing that uses a path that includes the mapping.

The Clio [27] and Hyperion [2] projects have developed tools for automating common data and structure-management tasks underlying many data-integration, translation, transformation, and evolution tasks. The thrust of these projects has been on supporting schema management, such as generating, matching, and mapping queries between schemas in multi-source and peer-to-peer systems. The architecture is similar to that of Piazza: A query is submitted at a peer, which passes the query, possibly in translated form, to (some of) its acquaintances, which repeat this process. In our project, schema mappings are always between a simple semantic model and a general schema. We may benefit from some of the discoveries of these projects, but in general, we do not need the sophisticated techniques for general schema matching and mapping.

Our work is also related to works on distributed query processing and optimization (see, e.g., [19] and references therein.) Most of the related projects address relational and object-oriented systems, and study the mechanisms and tradeoffs among data shipping, query shipping, and hybrids of these approaches (e.g., [15]). Our work differs from these in that we concentrate on XML sources and, besides, our data shipping (i.e., our materialization and wrapper approaches) transmit views derived from XML data, rather than (transmit) base tables. In that respect, our work resembles semantic-caching approaches (e.g., [**?**]), except that we use significantly different and more sophisticated mappings to extract the data to transmit.

## Acknowledgements

We are grateful to Igor Tatarinov and Alon Halevy for providing us with the complete experimental setup for [37], and to Natasha Noy for the helpful discussions on ontology-based information mediation. This work has been supported by NSF grants Career 0447742 and IIS 0307072 and by NCSU CACC grant 07-01.

# 6. REFERENCES

[1] Academic Department Ontology. `http://www.daml.org/ontologies/65`.

[2] Marcelo Arenas, Vasiliki Kantere, Anastasios Kementsietsidis, Iluju Kiringa, Renée J. Miller, and John Mylopoulos. The Hyperion project: From data integration to data coordination. *SIGMOD Record*, 32(3):53–58, 2003. Special issue on Peer to Peer Data Management.

[3] Dongfeng Chen, Rada Chirkova, and Fereidoon Sadri. Designing an information integration and interoperability system — first steps. Technical Report NCSU CSC TR-2006-30. Available at `http://www4.ncsu.edu/~rychirko/Papers`, October 2006.

[4] Rada Chirkova and Fereidoon Sadri. Query optimization using restructured views. In *Proceedings of International Conference on Information and Knowledge Management*, pages 642–651, 2006.

[5] Vassilis Christophides, Gregory Karvounarakis, Aimilia Magkanaraki, Dimitris Plexousakis, and Val Tannen. The ICS-FORTH Semantic Web integration middleware (SWIM). In *IEEE Data Engineering Bulletin*, pages 11–18, 2003.

[6] CiteSeer. `http://citeseer.ist.psu.edu/`.

[7] Conor Cunningham, Goetz Graefe, and César A. Galindo-Legaria. PIVOT and UNPIVOT: Optimization and execution strategies in an RDBMS. In *Proc. Int'l Conf. on Very Large Databases*, pages 998–1009, 2004.

[8] DAML Ontology Library. `http://www.daml.org/ontologies/`.

[9] Susan Davidson, Wenfei Fan, Carmem Hara, and Jing Qin. Propagating XML constraints to relations. In *Proc. IEEE Int'l Conf. on Data Engineering*, 2003.

[10] dblp. http://www.informatik.uni-trier.de/ ley/db/index.html.
[11] Hong Hai Do and Erhard Rahm:. COMA - a system for flexible combination of schema matching approaches. In *Proc. Int'l Conf. on Very Large Databases*, pages 610–621, 2002.
[12] AnHai Doan, Pedro Domingos, and Alon Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proc. ACM SIGMOD Int'l Conf.*, 2001.
[13] Xin Dong and Alon Halevy. Indexing dataspaces. In *Proc. ACM SIGMOD Int'l Conf.*, pages 43–54, 2007.
[14] Dieter Fensel. *Information Integration with Ontologies: Ontology Based Information Integration in an Industrial Setting*. John Wiley & Sons, 2005.
[15] Michael J. Franklin, Björn Thór Jónsson, and Donald Kossmann. Performance tradeoffs for client-server query processing. In *Proc. ACM SIGMOD Int'l Conf.*, pages 149–160, 1996.
[16] Mark Gyssens, Laks V. S. Lakshmanan, and Iyer N. Subramanian. Tables as a paradigm for querying and restructuring. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 93–103, 1996.
[17] Alon Y. Halevy. Data integration: A status report. In *Proceedings of German Database Conference (Datenbanksysteme für Business, Technologie und Web, BTW)*, pages 24–29, 2003.
[18] Alon Y. Halevy, Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu, and Igor Tatarinov. The Piazza peer data management system. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):787–798, 2004.
[19] Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, 2000.
[20] Ravi Krishnamurthy, Witold Litwin, and William Kent. Language features for interoperability of databases with schematic discrepancies. In *Proc. ACM SIGMOD Int'l Conf.*, pages 40–49, 1991.
[21] Zoé Lacroix, Louiqa Raschid, and Maria-Esther Vidal. Semantic model to integrate biological resources. In *ICDE Workshops*, 2006.
[22] Laks V. S. Lakshmanan and Fereidoon Sadri. Interoperability on XML data. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 146–163, 2003.
[23] Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *Proceedings of International Conference on Deductive and Object-Oriented Databases*, pages 81–100, 1993.
[24] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. On efficiently implementing SchemaSQL on a SQL database system. In *Proc. Int'l Conf. on Very Large Databases*, pages 471–482, 1999.
[25] Jayant Madhavan, Shirley Cohen, Xin Luna Dong, Alon Y. Halevy, Shawn R. Jeffery, David Ko, and Cong Yu. Web-scale data integration: You can afford to pay as you go. In *Proc. Conf. on Innovative Data Systems Research*, pages 342–350, 2007.
[26] Jayant Madhavan and Alon Y. Halevy. Composing mappings among data sources. In *Proc. Int'l Conf. on Very Large Databases*, pages 572–583, 2003.
[27] Renée J. Miller, Mauricio A. Hernández, Laura M. Haas, Ling-Ling Yan, C. T. Howard Ho, Ronald Fagin, and Lucian Popa. The Clio project: Managing heterogeneity. *SIGMOD Record*, 30(1), 2001.
[28] NCSU-UNCG Information-Integration Project. http://dbgroup.ncsu.edu/?page_id=205.
[29] Natalya F. Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, 2004.
[30] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology, 2001. http://ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mc
[31] PostgreSQL. http://www.postgresql.org/.
[32] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB J.*, 10(4):334–350, 2001.
[33] Satya Sanket Sahoo, Christopher Thomas, Amit P. Sheth, William S. York, and Samir Tartir. Knowledge modeling and its application in life sciences: a tale of two ontologies. In *Proceedings of the International WWW Conference*, pages 317–326, 2006.
[34] SAXONICA XSLT and XQuery Processing. http://www.saxonica.com/.
[35] Semantic Web. http://www.w3c.org/2001/sw/.
[36] SIGMOD. http://www.sigmod.org/.
[37] Igor Tatarinov and Alon Halevy. Efficient query reformulation in peer data management systems. In *Proc. ACM SIGMOD Int'l Conf.*, pages 539–550, 2004.
[38] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.
[39] Jeffrey D. Ullman. Information integration using logical views. In *Proceedings of International Conference on Database Theory*, pages 19–40, 1997.
[40] XML Path Language (XPath). http://www.w3c.org/TR/xpath.

# APPENDIX

# A. OUR FORMAL RESULTS

In this section we present theoretical results that allow for further reduction of inter-source processing, in approaches introduced in Section 2, by using information about integrity constraints in the data sources. For ease of presentation, the results are discussed for the semantic-model setting [22], which is outlined in Appendix B. At the same time, the extension of all the results to the general relational setting is straightforward.

## *Eliminating Inter-Source Subqueries*

Depending on the key and foreign-key constraints on the mediator-based predicates, we may not need any inter-source subqueries, or may only need a subset of all possible inter-source subqueries. The formal results introduced in this section play a significant role in query optimization in the semantic-model approach. Our first result addresses the question "when is inter-source processing not needed?" To motivate this investigation, let us first obtain an intuition about the amount of inter-source processing that may be needed: Consider a system with $n$ information sources, and a user query involving $k$ relations in the semantic model. The total number of possible subqueries, where the data for each of the $k$ relations comes from one of the $n$ sources, is $n^k$. Only $n$ of these are local, in the sense that all data come from the same source. The remaining $n^k - n$ subqueries may require some degree of inter-source processing. This is, of course, a worst-case scenario. In practice, even when the total number of sources is very large, a specific relation in the semantic-model view has a limited number of sources with data pertaining to that relation, reducing the possible inter-source queries to $m^k - n$, where $m << n$ is the number of sources with data for a given relation, on the average. Nevertheless, in large-scale information integration, where $n$ can be in the hundreds or even thousands or higher, this number can still be quite large. If we are able to identify the

minimum amount of inter-source processing that is required, and restrict our query evaluation to avoid any extra work, we can potentially achieve orders of magnitude faster query processing in large-scale information integration.

DEFINITION 1. (LOCAL-JOIN GRAPH) Given relations $r^i(A, B)$ and $r^j(B, C)$, we say $r^i$ and $r^j$ have the *local-join property* if the following conditions hold:

1. Key constraint: For every source $k$, $B$ is the key for the fragment $r_k^j$.
2. Foreign-key constraint: For every source $k$, there is a foreign-key constraint from $r_k^i(B)$ to $r_k^j(B)$.
3. Consistency constraint: If $r_k^j(b, c)$ and $r_l^j(b, c')$ hold at two sources $k$ and $l$, then $c = c'$.

Let $r^1, \ldots, r^m$ be all the relations in the semantic-model view. The *local-join graph* is a directed graph $G = (N, E)$, where nodes $N$ corresponds to the relations $r^1, \ldots, r^m$, and $(r^i, r^j) \in E$ if $r^i$ and $r^j$ have the local-join property. ■

THEOREM 1. *Given a user query involving the natural join of two or more relations $r^1, \ldots, r^k$, if the local-join graph restricted to the query relations $\{r^1, \ldots, r^k\}$ contains a directed spanning tree, then no inter-source processing is needed for this query.* ■

We omit the proofs of Theorems 1 and 2 due to the space limit; both proofs can be found in [3].

EXAMPLE 7. *Consider user query $r(A, B) \bowtie s(A, C)$. Assume attribute $A$ is the key for $r(A, B)$, and a foreign-key constraint holds from $s.A$ to $r.A$. According to Definition 1 and Theorem 1, no inter-source processing is needed for this query. As a result, evaluating only the local subqueries is enough to obtain the exact (correct) answer to the query.* ■

Theorem 1 gives a sufficient condition, based on key and foreign-key constraints, for eliminating inter-source processing in the evaluation of a user query. The question naturally arises as to whether the condition of Theorem 1 is also necessary. In other words, if the restriction of the local-join graph to query relations does not have a directed spanning tree, does the evaluation of the query require evaluation of some inter-source subqueries? We should first mention that there are weaker semantic constraints (than key, foreign-key constraints) that may provide conditions for Theorem 1 [22]. However, these semantic constraints are, to the best of our knowledge, not available as standard features in commercial databases. Hence, we restrict ourselves to key and foreign-key constraints. This means that if there is no edge from $r^i$ to $r^j$ in the local-join graph, then no constraints of any form exist between $r^i$ and $r^j$. The following theorem addresses in the positive the issue of whether the conditions of Theorem 1 are also necessary.

THEOREM 2. *Given a user query involving the natural join of two or more relations $r^1, \ldots, r^k$, if the local-join graph restricted to the query relations $\{r^1, \ldots, r^k\}$ does not contain a directed spanning tree, then a database instance exists where at least one inter-source subquery is not subsumed by any local subqueries.* ■

### *Partitioning Inter-Source Subqueries*

In this section we discuss the problem of determining the set of subqueries that are needed for the evaluation of the user query. In particular, we present a counterintuitive result, namely that the set of needed subqueries is not unique, rather, there can be multiple *equivalence sets* of subqueries. More specifically, we show that the set of subqueries can be partitioned into (1) required subqueries, (2) redundant subqueries, with each subquery in this group being subsumed by a subquery in the required group, and (3) zero or more sets of equivalent subqueries, where we need to execute only one subquery from each equivalence class.

EXAMPLE 8. *Consider a user query involving the natural join of three relations $r(A, B)$, $s(A, C)$, $t(A, D)$. Further, assume that attribute $A$ is the key for $r$, and foreign-key constraints hold from $s.A$ and $t.A$ to $r.A$. Also assume the consistency constraint of Definition 1 holds for $r$. Note that the local-join graph, restricted to $r$, $s$, and $t$, has edges from $s$ and $t$ to $r$, and does not have a directed spanning tree.*

*There are $2^3 = 8$ subqueries. Two of them are local subqueries, namely, $r_1 \bowtie s_1 \bowtie t_1$ and $r_2 \bowtie s_2 \bowtie t_2$ (where $r_i$ represents the fragment of $r$ that comes from source $i$, similarly for $s$ and $t$.) It is easy to verify that, for this query,*

- *$r_1 \bowtie s_1 \bowtie t_1$ and $r_2 \bowtie s_2 \bowtie t_2$ are required;*
- *$r_1 \bowtie s_2 \bowtie t_2$ and $r_2 \bowtie s_1 \bowtie t_1$ are redundant: $r_1 \bowtie s_2 \bowtie t_2$ is subsumed by $r_2 \bowtie s_2 \bowtie t_2$, and $r_2 \bowtie s_1 \bowtie t_1$ is subsumed by $r_1 \bowtie s_1 \bowtie t_1$; and*
- *$r_1 \bowtie s_1 \bowtie t_2$ and $r_2 \bowtie s_1 \bowtie t_2$ are equivalent, and so are $r_1 \bowtie s_2 \bowtie t_1$ and $r_2 \bowtie s_2 \bowtie t_1$.*

*Hence, the user query can be evaluated fully by evaluating just four subqueries out of the total 8. There are four sets of such minimally-sufficient subqueries: Each set includes the two required subqueries, plus one subquery from each of the two equivalence classes in the third bullet above.* ■

## B. SEMANTIC-MODEL OVERVIEW

The examples in this section provide an illustration of the semantic-model [22] approach in our implementation [28].

EXAMPLE 9. *Consider a federation of catalog sales businesses. Here we concentrate on their warehousing operations. A possible ontology for this application may use objects* (concepts) *such as* `item`, `warehouse`, `city`, `state`, *and relationships* (properties) *such as* `item-name`, `item-warehouse`, `warehouse-city`, *and* `warehouse-state`. *The semantic-model view (i.e., the ontology-based mediator schema) consists of binary relations representing the relationships. Sources with heterogeneous models and schemas can model their warehousing operations using this semantic-model view. For example, the DTDs of two hypothetical XML sources are shown below.*[5] *We discuss the mappings from these schemas to the semantic-model view in Example 10.*

```
<!ELEMENT store (warehouse*)>
<!ELEMENT warehouse (city, state, item*)>
<!ELEMENT item (id, name, description)>
<!ATTLIST warehouse id ID #REQUIRED>

<!ELEMENT store (items, warehouses)>
<!ELEMENT items (item*)>
<!ELEMENT item (id, name, description)>
<!ELEMENT warehouses (warehouse*)>
<!ELEMENT warehouse (city, state)>
<!ATTLIST item warehouse-id IDREFS #REQUIRED>
<!ATTLIST warehouse id ID #REQUIRED>
```

[5]We omit declarations of elements of type `#PCDATA`.

■

The language we use to specify XML-to-semantic-model mappings is based on (a subset of) XPath [40] and is similar to mapping languages, also called "transformation rules" or "source-to-target dependencies" in the literature (see, e.g., [9]). A mapping for a binary relation $p$ has the following general form:

```
p($X, $Y) <- path1 $G, $G/path2 $X, $G/path3 $Y.
```

where `$X` and `$Y` correspond to the arguments of $p$. The variable `$G` in the body of the rule is called the "glue" variable, and is used to restrict (`$X`,`$Y`) pairs to have the same `$G` ancestor element in the document.

EXAMPLE 10. *Consider the first information source of Example 9. Some of the mapping rules that map data in this source to the semantic-model view are as follows:*

```
item-name($I,$N) <-
  /store/warehouse/item $X, $X/id $I, $X/name $N.
item-warehouse($I,$W) <-
  /store/warehouse $X, $X/item/id $I, $X/@id $W.
warehouse-state($W,$S) <-
  /store/warehouse $X, $X/@id $W, $X/state $S.
```