D2I

Integrazione, Warehousing e Mining di sorgenti eterogenee

Programma di ricerca (cofinanziato dal MURST, esercizio 2000)

Survey on methods for query rewriting and query answering using views

Diego Calvanese, Domenico Lembo, Maurizio Lenzerini

D1.R5 **30 aprile 2001**

Sommario

A Data Integration System is constituted by three main components: source schemas, a global schema and a mapping between the two. There exist two main approaches for specifying the mapping: in the *local-as-view* (LAV) approach the source structures are defined as views over the global schema; on the contrary in the *global-as-view* (GAV) approach each global concept is defined in terms of a view over the source schemas. The problem of query processing is to find efficient methods for answering queries posed to the global schema on the basis of the data stored at sources. In LAV there exist two approaches to query processing: by *query rewriting*, in which one tries to compute a rewriting of the query in terms of the views and then evaluates such a rewriting, and by *query answering*, in which one aims at directly answering the query based on the view extensions. In GAV, existing systems deal with query processing by simply unfolding each global concept in the query with its definition in terms of the sources. In this paper, we survey the most important query processing algorithms proposed in the literature for LAV, and we describe the principal GAV data integration systems and the form of query processing they adopt.

Tema	Tema 1: Integrazione di dati provenienti da sorgenti eterogenee
Codice	D1.R5
Data	30 aprile 2001
Tipo di prodotto	Rapporto Tecnico
Numero di pagine	25
Unità responsabile	RM
Unità coinvolte	RM
Autore da contattare	Domenico Lembo Dipartimento di Informatica e Sistemistica Università di Roma “La Sapienza” Via Salaria 113, 00198 Roma, Italia lembo@dis.uniroma1.it

Survey on methods for query rewriting and query answering using views

Diego Calvanese, Domenico Lembo, Maurizio Lenzerini

30 aprile 2001

Abstract

A Data Integration System is constituted by three main components: source schemas, a global schema and a mapping between the two. There exist two main approaches for specifying the mapping: in the *local-as-view* (LAV) approach the source structures are defined as views over the global schema; on the contrary in the *global-as-view* (GAV) approach each global concept is defined in terms of a view over the source schemas. The problem of query processing is to find efficient methods for answering queries posed to the global schema on the basis of the data stored at sources. In LAV there exist two approaches to query processing: by *query rewriting*, in which one tries to compute a rewriting of the query in terms of the views and then evaluates such a rewriting, and by *query answering*, in which one aims at directly answering the query based on the view extensions. In GAV, existing systems deal with query processing by simply unfolding each global concept in the query with its definition in terms of the sources. In this paper, we survey the most important query processing algorithms proposed in the literature for LAV, and we describe the principal GAV data integration systems and the form of query processing they adopt.

1 Introduction

Information integration is the problem of combining the data residing at different sources, and providing the user with a unified schema of these data, called global schema. The global schema is therefore a reconciled view of the information, which can be queried by the user. It can be thought as a set of virtual relations, in the sense that their extensions are not actually stored anywhere. A data integration system frees the user from having to locate the sources relevant to a query, interact with each source in isolation, and manually combine the data from different sources.

The interest in this kind of systems has been continuously growing in the last years. Many organizations face the problem of integrating data residing at several sources. Companies that build a Data Warehouse, a Data Mining, or an Enterprise Resource Planning system must address this problem. Also, integrating data in the World Wide Web is the subject of several investigations and projects nowadays. Finally, applications requiring accessing or re-engineering legacy systems must deal with the problem of integrating data stored in different sources.

The design of a data integration system is a very complex task, which comprises several different issues, including the following:

1. heterogeneity of the sources,

2. mapping between the global schema and the sources,

3. limitations on the mechanisms for accessing the sources,

4. materialized vs. virtual integration,

5. data cleaning and reconciliation,

6. how to process queries expressed on the global schema.

Problem (1) arises because sources are typically heterogeneous, meaning that they adopt different models and systems for storing data. This poses challenging problems in specifying the global view. The goal is to design such a view so as provide an appropriate abstraction of all the data residing at the sources. One aspect deserving special attention is the choice of the language used to express the global schema. Since such a view should mediate among different representations of overlapping worlds, the language should provide flexible and powerful representation mechanisms.

With regard to Problem (2), two basic approaches have been used to specify the mapping between the sources and the global schema. The first approach, called *global-as-view* (or query-based), requires that the global schema is expressed in terms of the data sources. More precisely, to every concept of the global schema, a view over the data sources is associated, so that its meaning is specified in terms of the data residing at the sources. The second approach, called *local-as-view* (or source-based), requires the global schema to be specified independently from the sources. The relationships between the global schema and the sources are established by defining every source as a view over the global schema. Thus, in the local-as-view approach, we specify the meaning of the sources in terms of the concepts in the global schema. It is clear that the latter approach favors the extensibility of the integration system, and provides a more appropriate setting for its maintenance. For example, adding a new source to the system requires only to provide the definition of the source, and does not necessarily involve changes in the global view. On the contrary, in the global-as-view approach, adding a new source may in principle require changing the definition of the concepts in the global schema. Recently, [36] proposed a new approach to specify the mapping between the global schema and the source schemas, called the *global-local-as-view* (GLAV) approach, since it combines the expressive power of both LAV and GAV. In the following we do not discuss the GLAV approach, and concentrate on the LAV and GAV approaches.

Problem (3) refers to the fact, that, both in the local-as-view and in the global-as-view approach, it may happen that a source presents some limitations on the types of accesses it supports. A typical example is a web source accessible through a form where one of the fields must necessarily be filled in by the user. Such a situation can be modeled by specifying the source as a relation supporting only queries with a selection on a column. Suitable notations have been proposed for such situations [71], and the consequences of these access limitations on query processing in the integration systems have been investigated in several papers [71, 64, 35, 84, 83, 62, 63].

Problem (4) deals with a further criterion that one should take into account in the design of a data integration system. In particular, with respect to the data explicitely managed by the system, one can follow two different approaches, called *materialized* and *virtual*. In the materialized approach, the system computes the extensions of the concepts in the global schema by replicating the data at the sources. In the virtual approach, data residing at the sources are accessed during query processing, but they are not replicated in the integration system. Obviously, in the materialized approach, the problem of refreshing the materialized views in order to keep them up-to-date is a major issue [46]. In the following, we only deal with the virtual approach.

Whereas the construction of the global schema concerns the intentional level of the data integration system, Problem (5) refers to a number of issues arising when considering integration at the extensional/instance level. A first issue in this context is the interpretation and merging of the data provided by the sources. Interpreting data can be regarded as the task of casting them into a common representation. Moreover, the data returned by various sources need to be converted/reconciled/combined to provide the data integration system with the requested information. The complexity of this reconciliation step is due to several problems, such as possible mismatches between data referring to the same real world object, possible errors in the data stored in the sources, possible inconsistencies between values representing the properties of the real world objects in different sources [37]. The above task is known in the literature as *Data Cleaning and Reconciliation*, and the interested reader is referred to [37, 11, 7] for more details on this subject.

Finally, Problem (6) is concerned with one of the most important issues in a data integration system, i.e., the choice of the method for computing the answer to queries posed in terms of the global schema. The main issue is that the system should be able to re-express such queries in terms of a suitable set of queries posed to the sources.

In the rest of the paper, we concentrate on Problem (6), namely, query processing in a data integration system specified by means of the LAV or the GAV approach. We first provide a logical formalization of the problem in terms of a general framework for data integration systems, comprising several source schemas, a global schema, and a mapping between the two. Then we discuss the problem of *query processing*, i.e., compute the answer to a query over the global schema only on the basis of the data residing at the sources, both in LAV and in GAV:

- In LAV, most of the proposed solutions are based on query answering by query rewriting. The problem of *query rewriting (using views)* consists in reformulating the query into a (possibly) equivalent expression, called *rewriting*, that refers only to the source structures. Once the rewriting of the query has been computed, it can be directly evaluated over the source to obtain the answer to the query. We observe that, besides data integration, the problem of query rewriting is relevant in several fields, including data warehousing [80], query optimization [22], and supporting physical data independence [75]. A more direct approach is that of *query answering using views*, in which besides the query and the mapping definitions, we are also given the extensions of the views over the global schema. The goal is to compute the set of tuples that are the answer set of the query in all databases that are consistent with the information on the views. For the LAV approach we survey the most important query processing algorithms proposed in the literature.

- In GAV, most of the solutions proposed in the literature essentially reduce to *unfolding* the query over the global schema by substituting each global relation with its definition in terms of the sources. We describe some GAV data integration systems, the specific assumptions they introduce upon the general framework, and the form of query processing they adopt.

The paper is organized as follows. Section 2 presents the basic notions about the relational model, which is the one we adopt. Section 3 illustrates the problem of modeling a data integration system in LAV (subsection 3.1) and GAV (subsection 3.2). Section 4 presents view-based query processing in LAV (subsection 4.1) and GAV (subsection 4.2). Section 5 concludes the paper.

2 The relational model: basic notions

Since we assume to work with relational databases, we illustrate here the basic notions of the relational model that will be used in our framework. In the relational model, predicate symbols are used to denote the relations in the database, whereas constant symbols denote the objects and the values stored in relations. We assume to have a fixed (infinite) alphabet Γ of constants, and we consider only databases over such an alphabet. We adopt the so-called *unique name assumption*, i.e., we assume that different constants denote different objects.

A *relational schema* $\mathcal{C}$ is constituted by:

- An alphabet $\mathcal{A}$ of *predicate* (or relation) symbols, each one with an associated arity denoting the number of arguments of the predicate (or attributes of the relation).

- A set of *integrity constraints*, i.e., assertions on the symbols of the alphabet $\mathcal{A}$ that are intended to be satisfied in every database coherent with the schema.

A *relational database* (or simply, database, DB) $\mathcal{DB}$ over a schema $\mathcal{C}$ is simply a set of relations with constants as atomic values. We have one relation of arity n for each predicate symbol of arity n in the alphabet $\mathcal{A}$. The relation $r^{\mathcal{DB}}$ in $\mathcal{DB}$ corresponding to the predicate symbol R_i is constituted by a set of tuples of constants, those that satisfy the predicate R_i. A database $\mathcal{DB}$ over a schema $\mathcal{C}$ is said to be *legal* if every constraint of $\mathcal{C}$ is satisfied by $\mathcal{DB}$. The notion of satisfaction depends on the type of constraints defined over the schema.

A relational query is a formula that specifies a set of data to be retrieved from a database. In this work, we restrict our analysis to the class of unions of conjunctive queries. A *union of conjunctive queries* (called simply query) q of arity n over the schema $\mathcal{C}$ is written in the form

$$Q(\vec{\mathbf{x}}) \;\leftarrow\; conj_1(\vec{\mathbf{x}},\vec{\mathbf{y}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{x}},\vec{\mathbf{y}}_m)$$

where

- q belongs to a new alphabet $\mathcal{Q}$ (the alphabet of queries, that is disjoint from both Γ and $\mathcal{A}$),

- for each i, $conj_i(\vec{\mathbf{x}},\vec{\mathbf{y}}_i)$ is a conjunction of atoms involving the variables $\vec{\mathbf{x}} = X_1,\ldots,X_n$ and $\vec{\mathbf{y}}_i = Y_{i,1},\ldots,Y_{i,n_i}$, and constants from Γ,

- the predicate symbols of the atoms are in $\mathcal{C}$, and

- the number of variables of $\vec{\mathbf{x}}$ is called the *arity* of q, and is the arity of the relation denoted by the query q.

Given a database $\mathcal{DB}$, the answer of q over $\mathcal{DB}$, denoted $q^{\mathcal{DB}}$, is the set of n-tuples of constants $(c_1,\ldots,c_n)$, such that, when substituting each c_i for x_i, the formula

$$\exists\vec{\mathbf{y}}_1.conj_1(\vec{\mathbf{x}},\vec{\mathbf{y}}_1) \vee \cdots \vee \exists\vec{\mathbf{y}}_m.conj_m(\vec{\mathbf{x}},\vec{\mathbf{y}}_m)$$

evaluates to true in $\mathcal{DB}$.

3 Modeling the Data Integration System

In this section we set up a logical framework for data integration. The main components of a data integration system are the sources, the global schema and the mapping between the two. Formally, a data integration system $\mathcal{I}$ is a triple $\langle\mathcal{G},\mathcal{S},\mathcal{M}\rangle$, where:

- $\mathcal{S}$ is the source schema, constituted by the schemas of the various sources that are part of the data integration system. We assume that the sources are relational. Such an assumption is not restrictive since we can assume that suitable *wrappers* present the data at the sources in a relational form. Each source is modeled by a set of relations and we denote with $\mathcal{A}^{\mathcal{S}}$ the alphabet of the relational symbols of the sources.

- $\mathcal{G}$ is the *global schema*, expressed in the relational model in the global language $\mathcal{L}_{\mathcal{G}}$ over the alphabet $\mathcal{A}^{\mathcal{G}}$. Obviously, $\mathcal{A}^{\mathcal{G}}$ is disjoint from the alphabet of the source schemas $\mathcal{A}^{\mathcal{S}}$. The language $\mathcal{L}_{\mathcal{G}}$ determines the expressiveness allowed for specifying the global schema, i.e., the set of constraints that can be defined over it.

- $\mathcal{M}$ is the *mapping* between the sources and the global schema. The precise form of the mapping depends on the approach adopted for integration, and will be discussed later.

Intuitively, to specify the semantics of a data integration system, we have to start with a set of data at the sources, and, given such data at the sources, we have to specify which are the data satisfying the global schema. Thus, for assigning semantics to a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we start by considering a *legal source database* for $\mathcal{I}$, i.e., a database $\mathcal{D}$ for the source schema $\mathcal{S}$ that satisfies all integrity constraints of $\mathcal{S}$. Based on $\mathcal{D}$, we now specify which is the information content of the global schema $\mathcal{G}$. We call *global database* for $\mathcal{I}$ any database for $\mathcal{G}$. If $\mathcal{G}$ is constituted by the constraints $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$, we say that a global database *satisfies* $\mathcal{G}$ if it satisfies $\mathcal{C}_1, \ldots, \mathcal{C}_n$.

A global database $\mathcal{B}$ for $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ is said to be *coherent with $\mathcal{I}$ wrt $\mathcal{D}$*, if:

- $\mathcal{B}$ satisfies $\mathcal{G}$.
- $\mathcal{B}$ satisfies the mapping $\mathcal{M}$, that is its tuples respect the relationships defined between the global relations and the source structures. The precise meaning of satisfying a mapping depends on the specific form of the mapping and will be explained later when we will introduce the different approaches proposed to define such a mapping.

We denote with $coh(\mathcal{I}, \mathcal{D})$ the set of all databases for $\mathcal{G}$ coherent with $\mathcal{I}$ wrt to $\mathcal{D}$.

From the above definitions, it is easy to see that given a legal source database $\mathcal{D}$, in general several global coherent databases exist. In specific cases, no global coherent database may exist, for example if the set of constraints of the global schema is inconsistent. Also, there may be exactly one coherent database. We will give some examples in Subsections 3.1 and 3.2, where we discuss the specific forms of the mapping.

Finally we consider *queries* posed to a data integration system and define their semantics. Each query is issued over the global schema, and is expressed in a query language $\mathcal{L}_Q$ over the alphabet $\mathcal{A}^{\mathcal{G}}$. A query is intended to provide the specification of which data to extract from the global database represented in the integration system. In general, if Q is a query of arity n and $\mathcal{DB} \in coh(\mathcal{I}, \mathcal{D})$, we denote with $ans(Q, \mathcal{DB})$ the set of n-tuples that satisfy Q in $\mathcal{DB}$. Since several coherent databases may exist, we are interested in computing the set of tuples that satisfy Q in all databases in $coh(\mathcal{I}, \mathcal{D})$. Formally, we call *certain answers of Q wrt $\mathcal{I}, \mathcal{D}$* the set $cert(Q, \mathcal{I}, \mathcal{D})$ of n-tuples t such that $t \in ans(Q, \mathcal{DB})$ for every database $\mathcal{DB} \in coh(\mathcal{I}, \mathcal{D})$.

A tuple is said a *possible answer* if it is an answer only for some coherent database. Formally, we call *possible answers of Q wrt $\mathcal{I}, \mathcal{D}$*, the set $poss(Q, \mathcal{I}, \mathcal{D})$ of n-tuples t such that $t \in ans(Q, \mathcal{DB})$ for some database $\mathcal{DB} \in coh(\mathcal{I}, \mathcal{D})$.

In a data integration system $\mathcal{I}$, answering queries is essentially an extended form of reasoning in the presence of incomplete information [78]. Indeed, when we answer the query, we know only the extensions of the sources, and this provides us with only partial information on the global database. Moreover, since the query language may admit various forms of incomplete information (due to union, for instance), there are in general several possible databases coherent with $\mathcal{I}$.

We did not explain, until now, the way in which the mapping between the sources and the global schema is specified. There are two main approaches proposed to define such a mapping, namely the *local-as-view* (LAV) approach, and the *global-as-view* (GAV) approach. In the following we discuss both scenarios more in detail.

3.1 Local-as-view framework

In the LAV approach, the meaning of the sources is specified in terms of the concepts of the global schema. More exactly the mapping $\mathcal{M}$ between the sources and the globals schema is provided in terms of a set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$ over the global schema, one for each source. Associated to each view V_i we have:

- A definition $def(V_i)$ in terms of a query $V_i(\vec{\mathbf{x}}) \leftarrow v_i(\vec{\mathbf{x}}, \vec{\mathbf{y}})$, where $v_i(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ is expressed in the language $\mathcal{L}_\mathcal{V}$ over the alphabet $\mathcal{A}^\mathcal{G}$. The arity of $\vec{\mathbf{x}}$ determines the arity of the view V_i.
- The extension $V_i^\mathcal{D}$ over a given legal source database $\mathcal{D}$, which is a set of tuples of constants providing information about the content of the source. The arity of each tuple is the same as that of V_i.
- A specification $as(V_i)$ of which *assumption* to adopt for the view V_i, i.e., how to interpret the content $V_i^\mathcal{D}$ of the source with respect to the actual set of tuples in a global database $\mathcal{DB}$ that satisfy the view definition, i.e., $(def(V_i))^{\mathcal{DB}}$. We describe below the various possibilities that we consider for $as(V_i)$.

The following three assumptions have been considered in the literature [1, 38, 52, 13] [1]:

- *Sound Views.* When a view V_i is *sound* (denoted with $as(V_i) = sound$), its extension provides any subset of the tuples satisfying the corresponding definition. In other words, from the fact that a tuple is in $V_i^\mathcal{D}$ one can conclude that it satisfies the view, while from the fact that a tuple is not in $V_i^\mathcal{D}$ one cannot conclude that it does not satisfy the view. Formally, a database $\mathcal{DB}$ is *coherent with the sound view* V_i, if $V_i^\mathcal{D} \subseteq (def(V_i))^{\mathcal{DB}}$.
- *Complete Views.* When a view V_i is *complete* (denoted with $as(V_i) = complete$), its extension provides any superset of the tuples satisfying the corresponding definition. In other words, from the fact that a tuple is in $V_i^\mathcal{D}$ one cannot conclude that such a tuple satisfies the view. On the other hand, from the fact that a tuple is not in $V_i^\mathcal{D}$ one can conclude that such a tuple does not satisfy the view. Formally, a database $\mathcal{DB}$ is *coherent with the complete view* V_i, if $V_i^\mathcal{D} \supseteq (def(V_i))^{\mathcal{DB}}$.
- *Exact Views.* When a view V_i is *exact* (denoted with $as(V_i) = exact$), its extension is exactly the set of tuples satisfying the corresponding definition. Formally, a database $\mathcal{DB}$ is *coherent with the exact view* V_i, if $V_i^\mathcal{D} = (def(V_i))^{\mathcal{DB}}$

Such a definition of satisfaction of the mapping $\mathcal{M}$ completes the definition of *global coherent database* for the LAV approach. In general there can be many global coherent databases, for example when some views are sound, as demonstrated by the following example.

Example 3.1 Consider a data integration system $\mathcal{I}$ comprising two sources $\mathsf{s}_1(Name, City)$, which stores information about people and cities in which they reside and $\mathsf{s}_2(City, Country)$, which stores information about countries where cities are located.

Suppose the global schema $\mathcal{G}$ contains only the relation $\mathsf{residence}(Name, City, Country)$ and that the mapping $\mathcal{M}$ is as follows:

$$\begin{array}{rcl} \mathsf{s}_1(N, C) & \leftarrow & \mathsf{residence}(N, C, Co) \\ \mathsf{s}_2(C, Co) & \leftarrow & \mathsf{residence}(N, C, Co) \end{array}$$

and that we have an alphabet containing (among other symbols) three constants *Pablo*, *Rome* and *Italy*.[2]

Finally consider a source legal database $\mathcal{D}$ in which $\mathsf{s}_1^\mathcal{D} = \{(Pablo, Rome)\}$ and $\mathsf{s}_2^\mathcal{D} = \{(Rome, Italy)\}$, and assume that there are no constraints imposed by a schema.

[1] In some papers, for example [13], different assumptions on the domain of the database are also taken into account.

[2] We consider only databases over a fixed alphabet of constants (see section 2).

If both views are sound, there are several global databases coherent with the global schema and the mapping, since we only know that the global relation $\mathsf{residence}$ contains some tuples that have *Pablo* as their *Name* component and *Rome* as their *City* component, and some tuples that have *Rome* as their *City* component and *Italy* as their *Country* component. Therefore, the query $Q_c(x, y, z) \leftarrow \mathsf{residence}(x, y, z)$ asking for all residence informations would return an empty answer, i.e., $cert(Q_c, \mathcal{I}, \mathcal{D}) = \emptyset$.

However, if both views are exact, we can conclude that all $\mathsf{residence}$ tuples have *Pablo* as their *Name* component, *Rome* as their *City* component and *Italy* as their *Country* component. Hence $(Pablo, Rome, Italy)$ is the only tuple returned by the query Q_c, i.e., $cert(Q_c, \mathcal{I}, \mathcal{D}) = \{(Pablo, Rome, Italy)\}$. Note that in this case we have exactly one database coherent with the global schema and the mapping. ■

Even in the case where all views are exact we can have that several global databases exist, for example in the case where some global relations or attributes are not mentioned in any mapping from the sources.

When all views are exact it can happen that no global virtual database exists. This is the case when the data at the sources do not satisfy all constraints in the global schema.

Example 3.2 Suppose now to add to our running example a source $\mathsf{s}_3(Name, Country)$, which stores information about people citizenship, and add a global relation $\mathsf{citizenship}(Name, Country)$. Complete our mapping with the view

$$\mathsf{s}_3(N, Co) \quad \leftarrow \quad \mathsf{citizenship}(N, Co)$$

and assume to define a foreign key constraint between the component *Name* of the relation $\mathsf{residence}$ and component *Name* in the relation $\mathsf{citizenship}$. The constraint is satisfied by a database $\mathcal{DB}$ if for every tuple t_1 in $(\mathsf{residence})^{\mathcal{DB}}$ there is a tuple t_2 in $(\mathsf{citizenship})^{\mathcal{DB}}$ such that the *Name* component of t_1 agree with the *Name* component in t_2. Suppose to add to our alphabet the constants *John*, *USA*, and *Spain*. Consider the same source legal database $\mathcal{D}$ of Example 3.1, and suppose that $\mathsf{s}_3^{\mathcal{D}} = \{(\mathsf{John}, \mathsf{USA})\}$. If both views are exact, no global virtual database exists which is coherent with the global schema and the mapping, since s_3 doesn't store *Pablo*'s citizenship. On the contrary, if all views are sound, we have several coherent databases, that is all the databases that store *Pablo*'s citizenship. For example, consider the following extension for $\mathsf{citizenship}$:

Name	*Country*
John	*USA*
Pablo	*Spain*

■

3.2 Global-as-view framework

The GAV approach requires that the global schema is defined in terms of the data sources. More exactly, every relation of the global schema is expressed as a view over the sources, so that its meaning is specified in terms of the data residing at the sources. Note that, while in the LAV approach adding a new source to the system requires only to provide the definition of the source, and does not necessarily involve changes in the global schema, in the GAV approach, it may in principle requires changing the definition of the concepts in the global schema.

Formally, the mapping between the global schema and the sources is provided in terms of a set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$ over the source schemas, one for each relation in the global schema. Associated to each view V_i we have:

- A definition $def(V_i)$ in terms of a query $V_i(\vec{\mathbf{x}}) \leftarrow v_i(\vec{\mathbf{x}}, \vec{\mathbf{y}})$, where $v_i(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ is expressed in the language $\mathcal{L}_\mathcal{V}$ over the source alphabet $\mathcal{A}^\mathcal{S}$. The arity of $\vec{\mathbf{x}}$ determines the arity of the view V_i.

- A set $(def(V_i))^\mathcal{D}$ of tuples of constants returned from the evaluation of the view definition over a given legal source database $\mathcal{D}$. The arity of each returned tuple is the same as that of V_i.

- A specification $as(V_i)$ of which *assumption* to adopt for the view V_i, i.e., how to interpret the actual set of tuples in a global virtual database $\mathcal{DB}$ that satisfy V_i, i.e., $V_i^{\mathcal{DB}}$, with respect to $(def(V_i))^\mathcal{D}$. As in the LAV approach, one can consider views that are *sound*, *complete*, or *exact*:

  - when $as(V_i) = sound$, we are specifying that all tuples retrieved by $def(V_i)$ satisfy V_i.
  - when $as(V_i) = complete$, we are specifying that no tuples other than those retrieved by $def(V_i)$ satisfy V_i.
  - when $as(V_i) = exact$, we are specifying that the set of tuples that satisfy V_i is exactly the set of tuples retrieved by V_i.

  Formally a global virtual database $\mathcal{DB}$ is *coherent* with

  - the sound view V_i if $V_i^{\mathcal{DB}} \supseteq (def(V_i))^\mathcal{D}$,
  - the complete view V_i if $V_i^{\mathcal{DB}} \subseteq (def(V_i))^\mathcal{D}$,
  - the exact view V_i if $V_i^{\mathcal{DB}} = (def(V_i))^\mathcal{D}$,

Even if most of the GAV data integration systems described in the literature consider a single database for the global schema, due to the presence of integrity constraints, in general, we will have to account for a set of global databases. Given a legal source database, several situations are possible:

- No coherent global database exists. This happens, for example, when the data at the sources retrieved by the view definitions do not satisfy the constraints of the global schema.

- Several coherent global databases exist. This happens, for example, when all views are sound, and the data at the sources retrieved by the view definitions do not satisfy the constraints of the global schema. In this case, it may happen that several ways exist to add suitable tuples to the relations of $\mathcal{G}$ in order to satisfy constraints. Each such way yields a coherent global database.

- Exactly one coherent global database exists. This happens, for example, when all views are exact, and the data at the sources retrieved by the view definitions satisfy all the constraints of the global schema.

The following example illustrates the above considerations for the case where foreign key constraints are used in the global schema.

Example 3.3 Consider a data integration system $\mathcal{I}$ in which the source schema $\mathcal{S}$, the alphabet of constants, the global relations, and the integrity constraints are as in Example 3.2. Suppose to have the following GAV mapping $\mathcal{M}$

$$\begin{array}{rcl} \mathsf{residence}(N, C, Co) & \leftarrow & \mathsf{s}_1(N, C,) \wedge \mathsf{s}_2(C, Co) \\ \mathsf{citizenship}(N, Co) & \leftarrow & \mathsf{s}_3(N, Co) \end{array}$$

and the source legal database $\mathcal{D}$ in which $s_1^{\mathcal{D}} = \{(Pablo, Rome)\}$, $s_2^{\mathcal{D}} = \{(Rome, Italy)\}$ and $s_3^{\mathcal{D}} = \{(Pablo, Spain)\}$

In such a case the data at the sources respect the foreign key constraint. So, if the views are exact, exactly one global coherent database exists. Therefore, given the query $Q_c(x, y, z) \leftarrow \mathsf{residence}(x, y, z)$ we have that $cert(Q_c, \mathcal{I}, \mathcal{D}) = \{(Pablo, Rome, Italy)\}$. Consider a different situation in which s_1 and s_2 have the same extensions as above, but now $s_3^{\mathcal{D}} = \{(John, USA)\}$. Since the data at the source does not respect the foreign key constraint, no global coherent database exists. On the other hand, if the views are sound we can obtain coherent databases by adding the tuple (*Pablo*, *Spain*) to the relation citizenship. Note that we still have $cert(Q_c, \mathcal{I}, \mathcal{D}) = \{(Pablo, Rome, Italy)\}$. ■

4 Query processing

In this section we deal with the problem of query processing, i.e., how to compute the answer to a query posed over the global schema on the basis of the data stored at the sources and exploiting the information available about the data integration system. We examine some algorithms and approaches proposed in the literature for query processing both in LAV and in GAV integration systems. For the LAV framework we distinguish between two existing approaches: query processing by *query rewriting*, which aims at reformulating the query in terms of the source relations and then evaluating the computed rewriting over the source extensions, and *query answering using views*, which aims at directly answering the query on the basis of both the view definitions and the source extensions. For the GAV framework we describe some data integration systems in which query processing essentially amounts to unfolding.

4.1 LAV approach

The main issue for query processing in a LAV integration system is deciding how to decompose the query on the global schema into a set of subqueries on the sources, based on the meaning of the sources expressed in terms of the relations in the global schema. Since in LAV the mapping between the sources and the global schema is described as a set of views over the global schema, query processing amounts to finding a way to answer a query posed over a database using a set of views over the same database. This problem, called *answering queries using views*, is widely studied in the literature, since it has applications in many areas. In query optimization [22], the problem is relevant because using materialized views may speed up query processing. A data warehouse can be seen as a set of materialized views, and, therefore, query processing reduces to query answering using views [43, 74, 82, 42]. In the context of database design, using views provides means for maintaining the physical perspective of the data independent from its logical perspective [75]. It is also relevant for query processing in distributed databases [47] and in federated databases [60]. Finally, since the views provide partial knowledge on the database, answering queries using views can be seen as a special case of query answering with incomplete information [45, 1].

4.1.1 Query rewriting

The most common approach proposed in the literature to deal with the problem of query processing in data integration systems is by means of *query rewriting*. In query rewriting, a query and a set of view definitions over the global schema are provided, and the goal is to reformulate the query into an expression, the *rewriting*, that refers only to the views, and supplies the answer to the query. A query Q is expressed in a certain query language $\mathcal{L}_Q$ over the global alphabet $\mathcal{A}^{\mathcal{G}}$, while the rewriting is expressed in a query language $\mathcal{L}'_Q$ over the source alphabet $\mathcal{A}^{\mathcal{S}}$. $\mathcal{L}_Q$ and $\mathcal{L}'_Q$ can be different and hence of different expressive power. Query processing via query rewriting is divided in two steps, where the first one consists in reformulating the query in terms of the given query language $\mathcal{L}'_Q$ over $\mathcal{A}^{\mathcal{S}}$, and the second one evaluates the rewriting over the view extensions.

We first introduce the notion of containment between Datalog queries whose EDB predicates are the relations in $\mathcal{A}^{\mathcal{G}}$. Given such a query Q and a global database $\mathcal{DB}$, the answer $Q^{\mathcal{DB}}$ of Q over $\mathcal{DB}$ is the minimal fixpoint model of Q and $\mathcal{DB}$. Given two queries Q_1 and Q_2, we say that Q_1 is *contained in* Q_2, denoted by $Q_1 \subseteq Q_2$, if for all databases $\mathcal{DB}$ we have that $Q_1^{\mathcal{DB}} \subseteq Q_2^{\mathcal{DB}}$. We say that Q_1 and Q_2 are *equivalent* if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.

The problem of query containment has been studied in various settings. In [21], NP-completeness has been established for conjunctive queries, and in [25] a multi-parameter analysis has been performed for the same case, showing that the intractability is due to certain types of cycles in the queries. In [48, 77], Π_2^p-completeness of containment of conjunctive queries with inequalities was proved, and in [72] the case of queries with the union and difference operators was studied. For various classes of Datalog queries with inequalities, decidability and undecidability results were presented in [23] and [77], respectively. Query containment under constraints has also been the subject of several investigations. For example, decidability of conjunctive query containment was investigated in [3] under functional and multi-valued dependencies, in [27] under functional and inclusion dependencies, in [20, 59, 61] under constraints representing *is-a* hierarchies and complex objects, and in [28] in the case of constraints represented as Datalog programs. For queries over semistructured data, query containment for conjunctive regular path queries, in which the atoms in the body are regular expressions over binary predicates, has been studied in [8, 34], and EXPSPACE completeness has been established in [14].

To formally define the notion of rewriting, we consider queries and views defining the source relations that are unions of conjunctive queries over the global schema.

Given such a query Q and set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$ over $\mathcal{A}^{\mathcal{G}}$, the query Q_r is a *rewriting* of Q using $\mathcal{V}$ if

- $Q_r \cup \mathcal{V}$ is contained in Q, and
- Q_r does not refer to the predicates of $\mathcal{A}^{\mathcal{G}}$, i.e., the relations of the global schema appear only in the view definitions and not in the bodies of the clauses of Q_r.

We illustrate the notion of rewriting with the following example, adapted from [54], in which queries are conjunctive queries.

Example 4.1 Consider two sources, $\mathsf{r}_1(\mathit{Title}, \mathit{Year}, \mathit{Director})$, which stores information about movies filmed by european directors since 1960, and $\mathsf{r}_2(\mathit{Title}, \mathit{Critique})$, which stores information about movie reviews since 1990.

Suppose we have the following global schema

$$\begin{array}{l} \mathsf{movie}(\mathit{Title}, \mathit{Year}, \mathit{Director}) \\ \mathsf{european}(\mathit{Director}) \\ \mathsf{review}(\mathit{Title}, \mathit{Critique}) \end{array}$$

and the mapping

$$\begin{array}{rcl} \mathsf{r}_1(T,Y,D) & \leftarrow & \mathsf{movie}(T,Y,D) \wedge \mathsf{european}(D) \wedge Y \geq 1960 \\ \mathsf{r}_2(T,R) & \leftarrow & \mathsf{movie}(T,Y,D) \wedge \mathsf{review}(T,R) \wedge Y \geq 1990 \end{array}$$

Consider the query

$$\mathsf{q}(T,R) \quad \leftarrow \quad \mathsf{movie}(T,1998,D) \wedge \mathsf{review}(T,R)$$

asking for reviews of movies filmed in 1998. A rewriting of such a query is

$$\mathsf{qr}(T,R) \quad \leftarrow \quad \mathsf{r}_1(T,1998,D) \wedge \mathsf{r}_2(T,R)$$

Indeed, the body of qr refers only to the source relations, and hence can be directly evaluated over the source extensions. Moreover, qr together with the view definitions is equivalent to the following query over the global schema

$$\mathsf{qr}(T, R) \quad \leftarrow \quad \mathsf{movie}(T, 1998, D) \wedge \mathsf{european}(D) \wedge \mathsf{review}(T, R)$$

in which we deleted the atoms $1998 \geq 1960$ and $1998 \geq 1990$, which evaluate to true. It is immediate to verify that qr is contained in q. ■

In the example above, since source r_1 provides only movies filmed by European directors, the computed rewriting does not return exactly the data about reviews of movies filmed in 1998, requested by the original query. Indeed, qr is not equivalent to q. In general, the set of available sources may not store all the data needed to answer a query, and therefore the goal is to find a query expression that provides all the answers that can be obtained from the views. So, whereas in different contexts, i.e., query optimization or maintaining physical data independence [53], the focus is on finding rewritings that are logically equivalent to the original query, in data integration systems one is interested in *maximally contained* rewritings [30], formally defined as follows.

Given a query Q, a set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$, and a query language $\mathcal{L}_Q$, the query Q_m is a *maximally contained rewriting* of Q using $\mathcal{V}$ wrt $\mathcal{L}_Q$ if

- $Q_m \cup \mathcal{V}$ is a rewriting of Q
- there is no rewriting $Q' \neq Q_m$ in $\mathcal{L}_Q$ such that $Q_m \cup \mathcal{V} \subseteq Q' \cup \mathcal{V}$

There are many studies that are concerned with the problem of query rewriting using views, under different assumptions, both for form of the queries and the views and also wrt the general integration framework described in Section 3:

- queries are conjunctive queries [55, 71, 70]
- queries are recursive queries [30, 31, 2]
- queries are description logics queries [4, 11]
- queries for semistructured data [12, 68, 15, 17]
- queries with aggregates [39, 26]
- presence of limitations in accessing the sources [32, 71, 50, 58, 35]
- presence of functional dependencies [40, 32, 31] and inclusion dependencies [41]

Finally, it is worth noting that, since in data integration systems the relations of the global schema are virtual and we cannot extract data from them, we are interested in rewritings that use only the views, reflected by the definition above. In [55] such rewritings are called *complete*, and in the following we refer only to complete rewritings.

In the following we present two query rewriting algorithms that have been proposed in the literature in order to solve the query answering problem.

4.1.2 The bucket algorithm

The bucket algorithm [56, 57] is a query rewriting algorithm which works in the case where the query is a union of conjunctive queries and the views are conjunctive queries.

The algorithm is based on the fact that in such a case, given a query Q, if an equivalent rewriting of Q exists, then such a rewriting has at most as many atoms as Q [55]. Such a result leads immediately to a nondeterministic polynomial-time algorithm for finding equivalent conjunctive rewritings: guess a *candidate rewriting* and then check whether it is equivalent to the original query. The check is done by looking for a *containment mapping* between the query and the rewriting [21], which is an NP-complete problem. Note that the number of candidate rewritings is exponential in the size of the query.

The *bucket algorithm* aims at computing all the rewritings that are contained in (and not necessarily equivalent to) the original query, by pruning the space of candidate rewritings. The algorithm was proposed in the context of the Information Manifold (IM) system [60], a project developed at AT&T. IM handles the presence of inclusion and functional dependencies over the global schema and limitations in accessing the sources, and uses conjunctive queries as the language for describing the sources and querying the system.

To compute the rewriting of a query Q, the bucket algorithm proceeds in two steps:

1. for each atom g in Q, create a bucket that contains the views from which tuples of g can be retrieved, i.e., the views whose definition contains an atom to which g can be mapped in a rewriting of the query;

2. consider as candidate rewriting each conjunctive query obtained by combining one view from each bucket, and check by means of a containment algorithm whether such a query is contained in Q. If so, the candidate rewriting is added to the answer.

If the candidate rewriting is not contained in Q, before discarding it, the algorithm checks if it can be modified by adding comparison predicates in such a way that it is contained in Q. The proof that the bucket algorithm generates the maximal contained rewriting when the query language is union of conjunctive queries, is given in [38].

Note that, on the basis of the results in [55], the algorithm considers only rewritings that have at most the same number of atoms as the original query. As shown in the same paper and in [71], the given bound on the size of the rewriting does not hold in the presence of arithmetic comparison predicates, functional dependencies over the global schema, or limitations in accessing the sources. In such cases we have to consider rewritings that are longer than the original query. More precisely, let p be the number of atoms in the query Q:

- In the presence of functional dependencies, a minimal rewriting has at most $p + d$ atoms, where d is the sum of the arities of the atoms in Q;

- In the presence of limitations in accessing the sources, a minimal rewriting has at most $p + m$ atoms, where m is the number of different variables in Q;

- In the presence of comparison predicates, the size of a minimal rewriting is at most exponential in the size of Q.

According to [56] the bucket algorithm, in practice, does not miss solutions because of the length of the rewritings it considers, but other results [30, 41, 40] demonstrate that in the presence of functional dependencies and limitations in accessing the sources, union of conjunctive queries does not suffice to obtain the maximal contained rewritings, and one needs to resort to recursive rewritings. We refer the interested reader to [30, 32, 50] for a more detailed treatment of the problem.

4.1.3 The inverse rules algorithm

The previous algorithm searches the space of possible candidate rewritings by using buckets and then checks whether each computed rewriting is contained in the original query. In the following we describe a different algorithm, namely the *inverse rules algorithm* [30], that generates a rewriting (query plan) in time that is polynomial in the size of the query. The algorithm was developed in the context of the Infomaster system [29], an information integration tool of the Standford University, based on a language for representing first order logic expressions, called *Knowledge Interchange Format*. The inverse rules algorithm constructs a set of rules that invert the view definitions and provides the system with an inverse mapping which establish how to obtain the data of the global concepts from the data of the sources. The basic idea is to replace existential variables in the body of each view definition by a Skolem function.

Example 4.2 Given a view definition

$$\mathsf{v}(X) \quad \leftarrow \quad \mathsf{a}_1(X,Y) \land \mathsf{a}_2(X,Y)$$

the set of inverse rules obtained from it is

$$\begin{aligned} \mathsf{a}_1(X, f(X)) &\quad \leftarrow \quad \mathsf{v}(X) \\ \mathsf{a}_2(X, f(X)) &\quad \leftarrow \quad \mathsf{v}(X) \end{aligned}$$

■

Given a non-recursive Datalog query Q and a set of view definitions $\mathcal{V}$, the rewriting is the Datalog program consisting of both the query and the inverse rules obtained from $\mathcal{V}$.

Note that in the skolemization phase, we just introduce function symbols in the head of the inverse rules and never introduce a symbol within another, which leads to a finite evaluation process. Since bottom-up evaluation of the rewriting can produce tuples with function symbols, these need to be discarded. A polynomial time procedure to eliminate these function symbols by adding new predicates is described in [30]. It is also shown that the inverse rules algorithm returns a maximally contained rewriting wrt union of conjunctive queries, in time that is polynomial in the size of the query[3]. Even if the computational cost of constructing the query plan is polynomial, the obtained rewriting contains rules which may cause accessing views that are irrelevant for the query. [51] show that the problem of eliminating irrelevant rules has exponential time complexity. A second drawback of the inverse rules algorithm is that evaluating the inverse rules over the source extension may invert some useful computation done to produce the views [69]. However, the inverse rules algorithm can handle also recursive Datalog queries, the presence of functional dependencies over the global schema, or the presence of limitations in accessing the sources, by extending the obtained query plan with other specific rules.

Two other algorithm developed for the case where queries and views are conjunctive queries are the *MiniCon* algorithm [69] and the *unification-join algorithm* [70]. Combining ideas both from the bucket algorithm and from the inverse rules algorithm, MiniCon scales out and outperforms both algorithms. Experimental results related to the performance of MiniCon, also in presence of arithmetical comparison predicates, are reported in [69]. The unification-join algorithm is an exponential-time query answering algorithm based on a skolemization phase and on the representation of conjunctive queries by means of hypergraphs. [70] presents also a polynomial time algorithm for the large and natural subclass of acyclic conjunctive queries. Finally, [40, 41] extend the algorithm by taking into account also inclusion and functional dependencies expressed over the schema.

[3] Note that from a non-recursive Datalog program one can easily obtain an equivalent union of conjunctive queries.

4.1.4 Query answering

In general, in a data integration system, given a legal source database $\mathcal{D}$, i.e., an extension for the views, we are interested in computing the set of certain answers for a given query Q (see Section 3). In the query rewriting approach, first Q is reformulated (not taking into account the source extensions), and only in a second step the computed rewriting is evaluated over $\mathcal{D}$. So, the capability of the rewriting to retrieve the set of certain answers depends on the given query language in which it is expressed. Such a situation can constitute a limitation on computing the set of certain answers for Q.

A more general approach to query processing, namely *view-based query answering* [1, 38, 13, 15, 10], is that to consider, besides the query and the view definitions, also the extensions of the views. In view-based query answering, we do not pose any limit to query processing, and the only goal is to compute the set of certain answers to the query by exploiting all possible information, in particular the view extensions.

Unfortunately, many of the papers concerning query processing do not distinguish between query answering and query rewriting using views, and give raise to a sort of confusion between the two notions. Part of the problem comes from the fact that when the query and the views are conjunctive queries, there are algorithms, like the bucket or the inverse rules algorithm, for computing the best possible rewriting as union of conjunctive queries. Therefore the best possible rewriting (independently of the query language) is basically expressible in the same language as the original query and views. However for other query languages this is not the case. So, in spite of the large amount of work on the subject, the relationship between query rewriting and query answering using views is not completely clarified yet.

According to [16, 17], to focus on this relationship we have to abstract from the language used to express the rewriting, thus generalizing the notion of rewriting considered in the literature. [16, 17] define a *rewriting* of a query Q with respect to a set $\mathcal{V}$ of views as a function that, given the extensions of the views, i.e., a legal source database $\mathcal{D}$, returns a set of tuples that are contained in the certain answers of the query Q wrt $\mathcal{V}$ and $\mathcal{D}$, i.e., in the answer set of Q for every global database coherent with the views and $\mathcal{D}$. The rewriting that returns precisely the set of certain answers for each legal source database is called the *perfect* rewriting of the query wrt the views.

Observe that, by evaluating the perfect rewriting over given view extensions, one obtains the same set of tuples provided by query answering using views. Hence, the perfect rewriting is the best rewriting that one can obtain, given the available information on both the definitions and the extensions of the views.

An immediate consequence of the relationship between perfect rewriting and query answering is that the data complexity of evaluating the perfect rewriting over the view extensions is the same as the data complexity of answering queries using views[4].

Typically, one is interested in queries that can be evaluated in PTIME (i.e., are PTIME functions in data complexity), and hence one would like rewritings to be PTIME as well. For queries and views that are conjunctive queries (without union), the perfect rewriting is a union of conjunctive queries and hence is PTIME [1]. In general this is not the case. The complexity of answering queries using views for different languages (both for the query and for the views) is studied in [1]. The authors deal with the two assumptions that all the views are sound (called *open world assumption* in the paper), or that all the views are exact (called *closed world assumption* in the paper). Under open world assumption they show that in the case where the views are conjunctive queries, the problem becomes coNP-hard already in the case where the query is a conjunctive query with inequality. Instead, considering queries that are conjunctive queries, the problem is already coNP-hard when the views are positive queries. As

[4] According to [79] the data complexity is the complexity of the problem with respect to the size of the view extensions only. Instead the query complexity is defined with respect to the size of view definitions and the query.

shown in [16, 17], the problem is already coNP-hard for very simple query languages containing union. It is also shown that characterizing which instances of query rewriting admit a perfect rewriting that is PTIME, is a very difficult problem due to its connection with a longstanding open problem for constraint-satisfaction problem [49, 33]. Under the different assumption that the views are exact (closed world assumption) the problem is coNP already in the case that views and queries are conjunctive queries [1]. Note that in the above reported results about query rewriting, we implicitly considered the open world assumption. Finally, [1] sketch an effective way to compute the certain answers by representing the global database by means of conditional tables and querying them using the techniques for databases with incomplete information [45].

The problem of view-based query answering in the context where constraints over the global schema are expressed in terms of description logics, and queries and views are unions of conjunctive queries over the global schema is addressed in [9].

4.2 GAV approach

In a GAV integration system each relation of the global schema is expressed in terms of a view over the source structures. Hence, to answer a query Q, formulated in terms of the global schema, it is sufficient to *unfold* Q by replacing each global relation with the corresponding view [76]. This is illustrated in the following example.

Example 4.3 Consider the same sources and the same global schema as in Example 4.1, but suppose to have the following GAV mapping

$$\begin{array}{rcl} \mathsf{movie}(T,Y,D) & \leftarrow & \mathsf{r}_1(T,Y,D) \\ \mathsf{european}(D) & \leftarrow & \mathsf{r}_1(T,Y,D) \\ \mathsf{review}(T,R) & \leftarrow & \mathsf{r}_2(T,R) \end{array}$$

In such a situation the query

$$\mathsf{q}(T,R) \quad \leftarrow \quad \mathsf{movie}(T,1998,D) \wedge \mathsf{review}(T,R)$$

asking for reviews of movies filmed in 1998, is processed by expanding the atoms according to their definitions. Such an expansion computes the same reformulation as the one in Example 4.1

$$\mathsf{q}(T,R) \quad \leftarrow \quad \mathsf{r}_1(T,1998,D) \wedge \mathsf{r}_2(T,R)$$

■

Even if the process of unfolding is a simple mechanism to reformulate the original query Q in terms of the source alphabet, defining the views, needed to expand the global concepts involved in Q, implies to understand the precise relationships between the sources. This is generally a non trivial task. We can say that, while in the LAV approach the query reformulation (rewriting) process turns out to be realized at run time, in the GAV approach it is realized at design time. As a consequence, the main issue in a GAV integration system is the definition of *wrappers* and *mediators* [81]. As already mentioned, wrappers are software components that present data at the sources in a suitable form adopted in the integration system, and handle the access to the data taking into account source capabilities to answer queries posed to the sources [64, 83]. Mediators are modules that establish how to access and merge data residing at the sources, i.e., synthesize the view definitions associated to the global relations.

In most GAV integration systems the notion of global schema is actually missing, in the sense that the global schema is simply the collection of relations exported by the mediators and no rules or integrity constraints can actually be expressed over the global concepts. The mediators simply realize an explicit layer between the users' application and the data sources. Furthermore, the systems consider only the assumption that views over source schemas are exact (closed world assumption). In such a situation exactly one global database coherent with the integration system and the source extensions exists (see Section 3), i.e., the database computed by populating the views on the basis of the definitions synthesized at mediators. Wrt the general integration framework presented in Section 3, such assumptions greatly simplify the problem of computing the answer to a query (query processing reduces to unfolding). At the same time, however, they make the GAV integration systems unable to properly take into account actual constraints imposed by the application domain.

In the sequel we describe some integration systems that follow the GAV approach in the simplified setting we have described above.

4.2.1 The TSIMMIS Project

TSIMMIS (The Stanford-IBM Manager of Multiple Information Sources) is a joint project of the Stanford University and the Almaden IBM database research group [24]. It is based on an architecture that presents a hierarchy of wrappers and mediators, in which wrappers convert data from each source into a common data model called OEM (Object Exchange Model) and mediators combine and integrate data exported by wrappers or by other mediators. Hence, the global schema $\mathcal{G}$ is essentially constituted by the set of OEM objects exported by wrappers and mediators. Mediators are defined in terms of a logical language called MSL (Mediator Specification Language), which is essentially Datalog extended to support OEM objects. OEM is a semistructured and self-describing data model, in which each object has associated a label, a type for the value of the object and a value (or a set of values). Users' queries are posed in terms of objects synthesized at a mediator or directly exported by a wrapper. They are expressed in MSL or in a specific query language called LOREL (Lightweight Object REpository Language), an object-oriented extension of SQL.

Each query is processed by a module, the Mediator Specification Interpreter (MSI) [67, 83], consisting of three main components:

- The *View Expander*, which uses mediator specification to reformulate the query into a *logical plan* by expanding the objects exported by the mediator according to their definitions. The logical plan is a set of MSL rules which refer to information at the sources.
- The *Plan Generator*, also called *Cost-Based Optimizer*, which develops a *physical plan* specifying which queries will be sent to the sources, the order in which they will be processed, and how the results of the queries will be combined in order to derive the answer to the original query.
- The *Execution engine*, which executes the physical plan and produces the answer.

The problem of query processing in TSIMMIS in the presence of limitations in accessing the sources is addressed in [64] by devising a more complex Plan Generator comprising three modules:

- a *matcher*, which retrieves queries that can process part of the logical plan;
- a *sequencer*, which pieces together the selected source queries in order to construct feasible plans;
- an *optimizer*, which selects the most efficient feasible plan.

It has to be stressed that, in TSIMMIS, no global integration is ever performed. Each mediator performs integration indipendently. As a result, for example, a certain concept may be seen in completely different and even inconsistent ways by different mediators. Such form of integration can be called *query-based*, since each mediator supports a certain set of queries, i.e., those related to the view it provides.

4.2.2 The Garlic Project

The Garlic project [18], developed at IBM's Almaden Research Center, provides the user with an integrated data perspective by means of an architecture comprising a middleware layer for query processing and data access software called *Query Services & RunTime System*. The middleware layer presents an object-oriented data model based on the ODMG standard [19] that allows data from various information sources to be represented uniformly. In such a case the global schema is simply constituted by the union of local schemas, and no integrity constraints are defined over the OMDG objects. The objects are exported by the wrappers using the Garlic Data Language (GDL), which is based on the standard Object Definition Language (ODL). Each wrapper describes data at a certain source in the OMDG format and gives descriptions of source capabilities to answer queries in terms of the query plans it supports. Note that the notion of mediator, central in the TSIMMIS system, is missing in Garlic, and most of the mediator tasks, as the integration of objects from different sources, are submitted to the wrappers. Users pose queries in terms of the objects of the global schema in an object-oriented query language which is an object-extended dialect of SQL. The *Query Services & RunTime System* produces a query execution plan in which a user's query is decomposed in a set of sub-queries to the sources, by expanding each involved object with its definition provided by the relative wrapper. The query execution plan is obtained by means of some sequence of parsing, semantic checking, query reformulation and optimization, based on the information, provided by a system catalog, about where data is stored, what wrapper it is associated with, its schema, any available statistics, etc. Each involved wrapper translates the sub-queries into the source's native query language taking into account the query processing power of the source. The description of the source's query capabilities is used by the query optimizer to create a set of feasible plans and to select the best for execution.

4.2.3 The Squirrel Project

In the approaches to data integration described so far, the data at the sources are not replicated in the integration system (virtual approach). In contrast, in the materialized approach, the system computes the extension of the concepts in the global schema and maintains it in a local repository. Maintenance of replicated data against updates to the sources is a central aspect in this context and the effectiveness of maintenance affects timeliness and availability of data.

The *Squirrel* System, developed at the University of Colorado [87, 86, 85, 44], provides a GAV framework for data integration based on the notion of *integration mediator*. Integration mediators are active modules that support data integration using a hybrid of virtual and materialized data approaches.

The initial work on Squirrel [87, 85] does not consider virtual views and studies the problems related to data materialization. A key feature of Squirrel mediators, which consist of software components implementing materialized integrated views over multiple sources, is their ability to incrementally maintain the integrated views by relying on the active capabilities of sources. More precisely, at startup the mediator sends to the source databases a specification of the incremental update information needed to maintain the views and expects sources to actively provide such information.

Integration by means of hybrid virtual or materialized views is addressed in [44] in the context of the relational model. However, the presented techniques can also be applied in the context of object-oriented database models. In the sequel we refer to classes and objects to indicate data exported by mediators. The architecture of a Squirrel mediator described in [44] consists of components that deal with the problem of refreshing the materialized portion of the supported view, namely the update queue and the Incremental Update Processor (IUP), and components related to the problem of query processing, namely the Query Processor (QP) and the Virtual Attribute Processor (VAP). The QP provides the interface of the integrated view to the users. When it receives a user's query, it tries to answer the query on the basis of the materialized portion of the view, maintained in a local store. If the QP needs virtual data to answer the query, the VAP constructs temporary relations containing the relevant data. To obtain temporary relations the VAP uses information provided by a Virtual Decomposition Plan (VDP) maintained in the local store. The notion of VDP is analogous to that of Query Decomposition Plan in query optimization. More specifically, the VDP specifies the classes that the mediator maintains (materialized, virtual, or hybrid), and provides the basic structure for retrieving data from the sources or supporting incremental maintenance[5].

Either for the materialized version of the mediator or for the hybrid one, an automatic generator of Squirrel integrators has been developed. Such a module takes as input a specification of the mediator expressed in a high-level Integration Specification Language (ISL). A specification in ISL includes a description of the relevant subschemas of the source databases, and the match criteria between objects of families of corresponding classes, in particular a list of the classes that are matched and a binary matching predicate specifying correspondences between objects of two classes. The output of this module is an implementation of the mediator in the *Heraclitus* language, a database programming language whose main feature is the ability of representing and manipulating collections of updates to the current database state (deltas). An object-oriented extension of Heraclitus, called *H20*, is used in [86].

The problem of object matching in Squirrel mediators is addressed in [87]. In particular, a framework is presented for supporting intricate object identifier (OID) match criteria, such as key-based matching, lookup-table-based matching, historical-based-matching, and comparison-based matching. The last criterion allows for both considering attributes other than keys in object matching, and using arbitrary boolean functions in the specification of object matching.

4.2.4 The MOMIS Project

The MOMIS system [5, 6], jointly developed at the University of Milano and the University of Modena and Reggio Emilia, provides semi-automatic techniques for the extraction and the representation of properties holding in a single source schema (*intraschema relationships*), or between different source schemas (*interschema relationships*), and for schema clustering and integration, to identify candidates to integration and synthesize candidates into an integrated global schema. The relationships are both intensional (e.g., lexical relationships extracted according to a Wordnet supplied ontology) and extensional (e.g., lexical relationships which strengthen the corresponding intensional relationships), either defined by the designer or automatically inferred by the system. The integration process is based on a source independent object-oriented model called ODM_{I^3}, used for modeling structured and semistructured data sources in a common way. The model is described by means of the ODL_{I^3} language. In MOMIS mediators are composed of two modules:

1. the *Global Schema Builder*, which constructs the global schema by integrating the ODL_{I^3} source descriptions provided by the wrappers, and by exploiting the intraschema and the interschema relationships.

2. the *Query Manager*, which performs query processing and optimization. The Query Manager exploits extensional relationships to first identify all sources whose data are needed to answer a user's query posed over the global schema. Then it reformulates the original query into queries to the single sources, sends the obtained sub-queries to the wrappers,

[5]For a description of the update process and the relative update queue and IUP see [44].

which execute them and report the results to the Query Manager. Finally, the Query Manager combines the single results to provide the answer to the original query.

Currently, the query processing aspect is in a preliminary stage of development, and still needs further investigation.

4.2.5 The DIKE Project

The DIKE system [65], developed at the University of Calabria in collaboration with the University of Reggio Calabria, exploits a conceptual model called SDR-Network [66, 73] in order to uniformly handle and represent heterogeneous data sources. Each source has an associated SDR-Network, constituting of a set of nodes representing concepts and a set of arcs representing relationships between pairs of concepts. DIKE provides the user with an algorithm for data source integration which takes as input two SDR-Networks, which are either associated directly to the sources or are derived from previously constructed SDR-Networks, and computes an output *global* SDR-Network. The process is carried out by exploiting synonymies, and homonymies between arcs and nodes of the SDR-Networks, and similarity relationships between sub-nets. To each derived node (resp., arc) a mapping is associated that describes the way the node has been obtained from one or more nodes (resp., arcs) belonging to the original SDR-Networks. Finally, to each mapping between nodes a view is associated that allows to obtain the instances of the target node of the mapping, from the instances of the source nodes of the mapping. At the moment the system is under development and does not present a completely defined method to answer queries posed over the global SDR-Networks. To answer a user's query it simply uses the views associated to the global nodes involved in the query to retrieve its instances from the nodes stored at the sources.

4.2.6 Limitations in accessing the sources

Limitations on how sources can be accessed significantly complicate query processing also in the GAV approach, since in this case simply unfolding the global relations in the query with their definitions is in general not sufficient. In fact, to answer queries over such sources one generally needs to start from a set of constants (provided e.g., by the user who fills a form, or taken from a source without access limitations) to be used to bind attributes. Such bindings are used to access sources and thus obtain new constants which in turn can be used for new accesses. Hence, as shown in [62, 63], query answering in GAV in the presence of limited access patterns in general requires the evaluation of a recursive query plan, which can be suitably expressed in Datalog.

Since source accesses are costly, an important issue is how to minimize the number of accesses to the sources while still being guaranteed to obtain all possible answers to a query. [62, 63] discuss several optimizations that can be made at compile time, during query plan generation.

5 Conclusions

We have described a general framework for data integration systems, which provide access to a set of heterogeneous sources by means of a global schema in terms of which users' queries are posed. The source schemas are related to the global schema by means of a mapping. The problem of query processing in such a setting amounts to finding a way to answer a query posed over the global schema using only the information stored at the sources. We have discussed query processing in the context of the two relevant cases where the mapping between the global and the local schemas is specified following either the local-as-view (LAV) or the global-as-view (GAV) approach. While in the LAV approach each source relations is expressed in terms of a view over the global schema, in the GAV approach each global relation is associated with a view over the source relations. Since the mapping differs in the two frameworks, the problem assumes different aspects in LAV and GAV.

In the former approach query processing has been addressed by means of either *query rewriting*, which tries to reformulate the original query in terms of the sources and then evaluates the rewritten query over the data at the sources, or *query answering*, which aims at directly answering the query based on the view extensions. We have presented the algorithms proposed in the literature that deal with query rewriting and query answering in the LAV approach.

Integration in the GAV approach is treated in the literature only in the simplified settings in which no constraint is defined in the global schema. In such a setting query processing in the GAV approach reduces to simply *unfolding* the global relations in the query with their definitions. We have described some GAV integration systems that deal with the data integration problem in such a setting. We showed also that in the presence of constraints over the global schema unfolding is in general not sufficient.

References

[1] Serge Abiteboul and Oliver Duschka. Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 254–265, 1998.

[2] Foto N. Afrati, Manolis Gergatsoulis, and Theodoros Kavalieros. Answering queries using materialized views with disjunction. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 435–452. Springer-Verlag, 1999.

[3] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalence among relational expressions. *SIAM J. on Computing*, 8:218–246, 1979.

[4] Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset. Rewriting queries using views in description logics. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 99–108, 1997.

[5] D. Beneventano, S. Bergamaschi, S. Castano, A. Corni, R. Guidetti, G. Malvezzi, M. Melchiori, and M. Vincini. Information integration: the MOMIS project demonstration. In *Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB 2000)*, 2000.

[6] S. Bergamaschi, S. Castano, D. Beneventano, and M. Vincini. Retrieving and integrating data from multiple sources: the MOMIS approach. *Data and Knowledge Engineering*, 36:251–249, 2001.

[7] Mokrane Bouzeghoub and Maurizio Lenzerini. Special issue on data extraction, cleaning, and reconciliation. *Information Systems*, 2001. To appear.

[8] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[9] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. What can knowledge representation do for semi-structured data? In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI'98)*, pages 205–210, 1998.

[10] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 386–391, 2000.

[11] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Data integration in data warehousing. *Int. J. of Cooperative Information Systems*, 2001. To appear.

[12] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 194–204, 1999.

[13] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Answering regular path queries using views. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 389–398, 2000.

[14] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'2000)*, pages 176–185, 2000.

[15] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Query processing using views for regular path queries with inverse. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2000)*, pages 58–66, 2000.

[16] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing and constraint satisfaction. In *Proc. of the 15th IEEE Symp. on Logic in Computer Science (LICS 2000)*, pages 361–371, 2000.

[17] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. What is query rewriting? In *Proc. of the 7th Int. Workshop on Knowledge Representation meets Databases (KRDB 2000)*, pages 17–27. CEUR Electronic Workshop Proceedings, http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-29/, 2000.

[18] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers. Towards heterogeneous multimedia information systems: The Garlic approach. In *RIDE-DOM*, pages 124–131, 1995.

[19] Roderick G. G Cattell and Douglas K. Barry, editors. *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann, Los Altos, 1997.

[20] Edward P. F. Chan. Containment and minimization of positive conjunctive queries in oodb's. In *Proc. of the 11th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'92)*, pages 202–211, 1992.

[21] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of the 9th ACM Symp. on Theory of Computing (STOC'77)*, pages 77–90, 1977.

[22] S. Chaudhuri, S. Krishnamurthy, S. Potarnianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of the 11th IEEE Int. Conf. on Data Engineering (ICDE'95)*, Taipei (Taiwan), 1995.

[23] Surajit Chaudhuri and Moshe Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. In *Proc. of the 11th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'92)*, pages 55–66, 1992.

[24] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. of IPSI Conf. (IPSI'94)*, Tokyo (Japan), 1994.

[25] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, pages 56–70, 1997.

[26] Sara Cohen, Werner Nutt, and Alexander Serebrenik. Rewriting aggregate queries using views. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 155–166, 1999.

[27] Anthony C. Klug David S. Johnson. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. of Computer and System Sciences*, 28(1):167–189, 1984.

[28] Guozhu Dong and Jianwen Su. Conjunctive query containment with respect to views and constraints. *Information Processing Lett.*, 57(2):95–102, 1996.

[29] Oliver Duschka. *Query Planning and Optimization in Information Integration.* PhD thesis, Stanford University, 1997.

[30] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 109–116, 1997.

[31] Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. Recursive query plans for data integration. *J. of Logic Programming*, 43(1):49–73, 2000.

[32] Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. In *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, pages 778–784, 1997.

[33] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction. *SIAM J. on Computing*, 28:57–104, 1999.

[34] Daniela Florescu, Alon Levy, and Dan Suciu. Query containment for conjunctive queries with regular expressions. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 139–148, 1998.

[35] Daniela Florescu, Alon Y. Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 311–322, 1999.

[36] Marc Friedman, Alon Levy, and Todd Millstein. Navigational plans for data integration. In *Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI'99)*, pages 67–73. AAAI Press/The MIT Press, 1999.

[37] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. An extensible framework for data cleaning. Technical Report 3742, INRIA, Rocquencourt, 1999.

[38] Gösta Grahne and Alberto O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 332–347. Springer-Verlag, 1999.

[39] Stéphane Grumbach, Maurizio Rafanelli, and Leonardo Tininini. Querying aggregate data. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 174–184, 1999.

[40] Jarek Gryz. An algorithm for query folding with functional dependencies. In *Proc. of the 7th Int. Symp. on Intelligent Information Systems*, pages 7–16, 1998.

[41] Jarek Gryz. Query folding with inclusion dependencies. In *Proc. of the 14th IEEE Int. Conf. on Data Engineering (ICDE'98)*, pages 126–133, 1998.

[42] Himanshu Gupta, Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Index selection for OLAP. In *Proc. of the 13th IEEE Int. Conf. on Data Engineering (ICDE'97)*, pages 208–219, 1997.

[43] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implemeting data cubes efficiently. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 205–216, 1996.

[44] Richard Hull and Gang Zhou. A framework for supporting data integration using the materialized and virtual approaches. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 481–492, 1996.

[45] Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. of the ACM*, 31(4):761–791, 1984.

[46] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis, editors. *Fundamentals of Data Warehouses.* Springer-Verlag, 1999.

[47] Arthur M. Keller and Julie Basu. A predicate-based caching scheme for client-server database architectures. *Very Large Database J.*, 5(1):35–47, 1996.

[48] Anthony C. Klug. On conjunctive queries containing inequalities. *J. of the ACM*, 35(1):146–160, 1988.

[49] Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 205–213, 1998.

[50] Chung T. Kwok and Daniel Weld. Planning to gather information. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI'96)*, pages 32–39, 1996.

[51] Alon Levy, Richard E. Fikes, and Shuky Sagiv. Speeding up inference using relevance reasoning: A formalism and algorithms. *Artificial Intelligence*, 97(1–2), 1997.

[52] Alon Y. Levy. Obtaining complete answers from incomplete databases. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, pages 402–412, 1996.

[53] Alon Y. Levy. Answering queries using views: A survey. Technical report, University of Washinghton, 1999.

[54] Alon Y. Levy. Logic-based techniques in data integration. In Jack Minker, editor, *Logic Based Artificial Intelligence.* Kluwer Publishers, 2000.

[55] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95)*, pages 95–104, 1995.

[56] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query answering algorithms for information agents. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI'96)*, pages 40–47, 1996.

[57] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogenous information sources using source descriptions. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, 1996.

[58] Alon Y. Levy, Anand Rajaraman, and Jeffrey D. Ullman. Answering queries using limited external query processors. In *Proc. of the 15th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'96)*, pages 227–237, 1996.

[59] Alon Y. Levy and Marie-Christine Rousset. CARIN: A representation language combining Horn rules and description logics. In *Proc. of the 12th Eur. Conf. on Artificial Intelligence (ECAI'96)*, pages 323–327, 1996.

[60] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *J. of Intelligent Information Systems*, 5:121–143, 1995.

[61] Alon Y. Levy and Dan Suciu. Deciding containment for queries with complex objects. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 20–31, 1997.

[62] Chen Li and Edward Chang. Query planning with limited source capabilities. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 401–412, 2000.

[63] Chen Li and Edward Chang. On answering queries in the presence of limited access patterns. In *Proc. of the 8th Int. Conf. on Database Theory (ICDT 2001)*, 2001.

[64] Chen Li, Ramana Yerneni, Vasilis Vassalos, Hector Garcia-Molina, Yannis Papakonstantinou, Jeffrey D. Ullman, and Murty Valiveti. Capability based mediation in TSIMMIS. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 564–566, 1998.

[65] Luigi Palopoli, Giorgio Terracina, and Domenico Ursino. The system DIKE: Towards the semi-automatic synthesis of cooperative information systems and data warehouses. In *Proc. of Symposium on Advances in Databases and Information Systems (ADBIS-DASFAA 2000)*, pages 108–117, 2000.

[66] Luigi Palopoli, Giorgio Terracina, and Domenico Ursino. A graph-based approach for extracting terminological properties of elements of XML documents. In *Proc. of the 17th IEEE Int. Conf. on Data Engineering (ICDE 2001)*, pages 330–340. IEEE Computer Society Press, 2001.

[67] Yannis Papakonstantinou, Hector Garcia-Molina, and Jeffrey D. Ullman. MedMaker: A mediation system based on declarative specifications. In Stanley Y. W. Su, editor, *Proc. of the 12th IEEE Int. Conf. on Data Engineering (ICDE'96)*, pages 132–141, 1996.

[68] Yannis Papakonstantinou and Vasilis Vassalos. Query rewriting using semistructured views. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, 1999.

[69] Rachel Pottinger and Alon Y. Levy. A scalable algorithm for answering queries using views. In *Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB 2000)*, pages 484–495, 2000.

[70] Xiaolei Qian. Query folding. In *Proc. of the 12th IEEE Int. Conf. on Data Engineering (ICDE'96)*, pages 48–55, 1996.

[71] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. Answering queries using templates with binding patterns. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'95)*, 1995.

[72] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. of the ACM*, 27(4):633–655, 1980.

[73] Giorgio Terracina and Domenico Ursino. Deriving synonymies and homonymies of object classes in semi-structured information sources. In *Proc. of International Conference on Management of Data (COMAD 2000)*, pages 21–32. McGraw-Hill, New York, 2000.

[74] Dimitri Theodoratos and Timos Sellis. Data warehouse design. In *Proc. of the 23rd Int. Conf. on Very Large Data Bases (VLDB'97)*, pages 126–135, 1997.

[75] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for phyisical data independence. *Very Large Database J.*, 5(2):101–118, 1996.

[76] Jeffrey D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1997.

[77] Ron van der Meyden. *The Complexity of Querying Indefinite Information.* PhD thesis, Rutgers University, 1992.

[78] Ron van der Meyden. Logical approaches to incomplete information. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publisher, 1998.

[79] Moshe Y. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82)*, pages 137–146, 1982.

[80] Jennifer Widom. Special issue on materialized views and data warehousing. *IEEE Bulletin on Data Engineering*, 18(2), 1995.

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

[82] Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for materialized view design in data warehousing environment. In *Proc. of the 23rd Int. Conf. on Very Large Data Bases (VLDB'97)*, pages 136–145, 1997.

[83] Ramana Yerneni, Chen Li, Hector Garcia-Molina, and Jeffrey D. Ullman. Computing capabilities of mediators. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 443–454, 1999.

[84] Ramana Yerneni, Chen Li, Jeffrey D. Ullman, and Hector Garcia-Molina. Optimizing large join queries in mediation systems. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, pages 348–364, 1999.

[85] Gang Zhou, Richard Hull, and Roger King. Generating data integration mediators that use materializations. *J. of Intelligent Information Systems*, 6:199–221, 1996.

[86] Gang Zhou, Richard Hull, Roger King, and Jean-Claude Franchitti. Data integration and warehousing using H20. *IEEE Bull. of the Technical Committee on Data Engineering*, 18(2):29–40, 1995.

[87] Gang Zhou, Richard Hull, Roger King, and Jean-Claude Franchitti. Using object matching and materialization to integrate heterogeneous databases. In *Proc. of the 3rd Int. Conf. on Cooperative Information Systems (CoopIS'95)*, pages 4–18, 1995.