Towards Efficient Semantic Data Integration

Marco Ruzzi
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
`ruzzi@dis.uniroma1.it`

October 20, 2005

1 Introduction

Data are the most relevant part of an information system. The need to handle huge amounts of data leads software companies towards the development of tools that allow several types of organization to effectively face the *on demand* challenge [1, 14]: in fact, data changes and this happens continuously and faster than ever before. Moreover, data come from an increasing number and variety of information sources, and the data management infrastructure of a generic business entity should be able to share its data with other entities, integrating them with information coming from elsewhere, also surmounting various kind of heterogeneity.

Several efforts are made in this direction from both the software development [15, 2] and the academic field [18, 24]: research in *information integration*, in fact, aims to provide robust and effective solutions to such kind of problem. Unfortunately, "information integration" often assumes very different meanings. Software industry mostly faces the matter from a procedural point of view: *data federation* tools [5], for example, adopt a database management system as a kind of middleware infrastructure that uses a set of software modules (wrappers) to access heterogeneous data sources. As another example, a *grid computing* tool [9] provides a framework that enables data owners to cope with the localization of data in distributed environments. On the other hand, academic research focuses on theoretical solutions to information integration issues, concerning topics like data modelling, expressiveness of formalisms, semantic characterization of the problem, which constitute very difficult tasks.

The present work is somehow located in the middle: more specifically, our aim is to find a way to use solutions coming from both the aforementioned fields, providing an efficient and effective solution to the information integration problem. Therefore, the main goal of our work is to build a novel and efficient system that is able to really access heterogeneous data sources, masking them under a common and expressive representation, and enabling users to efficiently retrieve useful answers to their queries. More in detail, we will exploit the capability of the IBM Information Integrator [2, 1] suite to access and integrate data and content as if it were a single resource, regardless of where the information resides, while retaining the autonomy and integrity of the sources' content. Then we will mask the integrated data under a mediated schema enriched with constraints, which represents better a variety of real world situations. To do this, we will take advantage of some very recent solutions [13, 19] coming from the research in database theory, and will significantly extend them to suit our requirements.

This goal is very ambitious: in fact, although some research work are in progress [13, 11, 19, 6, 12], there are still many problems to solve. On one hand, commercial tools are rather far to meet the requirements coming from the semantic data integration theory; on the other hand, solutions provided for that problems are still inefficient [7, 8].

This report is structured as follows: in Section 2 we provide a formal specification of the data integration problem; in Section 3 we present the architecture of the system; in Section 4 we briefly introduce IBM Information Integrator, showing in Section 4.3 how to use it to achieve integration. In Section 5 we characterize the consistent query answering theory, and explain in Section 5.3 how to deal with constraints. Finally, in Section 6 we present the future work plan.

2 A logical framework for data integration

Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data [18, 21, 16]. Thus, a data integration system provides the user with a *global schema*, i.e. an abstraction of the users' interest domain, connecting it to the *sources schema* (a representation of the data sources) by means of a *mapping.* The adoption of a logical formalism in specifying the global schema enables users to focus on specifying the intensional aspects of the integration (*what* they want), rather than thinking about the procedural aspects (*how* to obtain the answers). On one hand, through the specification of a global schema, the domain of interest can be described *independently* of the data sources structure. On the other hand, the mapping models the relation between the global schema and the sources.

In this section, we set up a logical framework for data integration[1]. We first formalize the notion of a data integration system, then we specify its semantics.

2.1 Syntax

We consider to have an infinite, fixed alphabet Γ of constants representing real world objects, and will take into account only databases having Γ as domain. Furthermore, we adopt the so-called unique name assumption, i.e., we assume that different constants denote different objects. Formally, a data integration system $\mathcal{I}$ is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where:

1. $\mathcal{G}$ is the global schema expressed in the relational model. In particular, $\mathcal{G}$ is a set of relations, each with an associated arity that indicates the number of its attributes: $\mathcal{G} = \{R_1(\mathbf{A_1}), R_2(\mathbf{A_2}), ..., R_n(\mathbf{A_n})\}$, where $\mathbf{A_i}$ is the sequence of attributes of R_i, for $i = 1..n$.

[1]Note that we do not consider integrity constraints (ICs) in this framework: it will be introduced in Section 5 when query answering problem is presented

2. $\mathcal{S}$ is the source schema, constituted by the schemas of the various sources. We assume that the sources are relational. Dealing with only relational sources is not restrictive, since we can always assume that suitable wrappers (as the IBM Information Integrator wrappers) present sources in the relational format. In particular, $\mathcal{S}$ is the union of sets of data source relations S_j: $\mathcal{S} = \bigcup_{j=1}^{m} S_j$, with $S_j = \{r_{j_1}(\mathbf{a_{j_1}}), r_{j_2}(\mathbf{a_{j_2}}), ..., r_{j_{n_j}}(\mathbf{a_{j_{n_j}}})\}$, where $\mathbf{a_{j_h}}$ is the sequence of attributes of the h-th relation of the j-th source, for $j = 1..m, h = 1..n_j$.

3. $\mathcal{M}$ is the mapping between the global and the source schema. In our framework, the mapping is defined in the Global-As-View (GAV) approach, i.e. each global relation in $\mathcal{G}$ is associated with a *view*, i.e., a query over the sources. We assume that views in the mapping are specified through conjunctive queries (CQ). We recall that a CQ q of arity n is a rule of the form:

$$q(x_1, ..., x_n) \leftarrow conj(x_1, ..., x_n, y_1, ..., y_m)$$

where: $conj(x_1, ..., x_n,\ y_1, ..., y_m)$ is a conjunction of atoms involving the variables $x_1, ..., x_n, y_1, ...,\ y_m$ and a set of constants of Γ. We denote the atom $q(x_1, ..., x_n)$ as $head(q)$, while $body(q)$ denotes the set $conj(x_1, ..., x_n, y_1, ..., y_m)$. A GAV mapping is therefore a set of CQ q, where $head(q)$ is a relation symbol of $\mathcal{G}$ and each atom in $body(q)$ is a relation symbol of $\mathcal{S}$. We assume that, for each relation symbol R in $\mathcal{G}$, at most one CQ in $\mathcal{M}$ uses R in its head. Such a CQ will have the form:

$$R(\vec{x}) \leftarrow r_1(\vec{x_1}, \vec{y_1}), ..r_k(\vec{x_k}, \vec{y_k})$$

where $r_h \in \mathcal{S}$ for $h = 1..k$, and $\vec{x} = \bigcup_{h=1}^{k} \vec{x_h}$. We denote as $\rho(R)$ such a CQ, if it exists.

Finally, a query over the global schema is a formula that is intended to extract a set of tuples of elements of Γ. We assume that the language used to specify queries is union of conjunctive queries (UCQ). We recall that a UCQ of arity n is a set of conjunctive queries $\mathcal{Q}$ such that each $q \in \mathcal{Q}$ has the same arity n and uses the same predicate symbol in the head. A *query over $\mathcal{I}$* is a UCQ that only uses relation symbols of $\mathcal{G}$ in the body of the rules. An example of data integration system specification follows.

Example 1 We build up a data integration system specification $\mathcal{I}$ considering three data sources: the first one stores information coming from the Registry Office concerning citizens and enterprises, the second one holds geographical information about the cities and their dislocation, the third one stores information coming from the Land Register about ownerships (buildings and lands). The source schema $\mathcal{S}$ that represents these sources within the system contains four relations:

$$\begin{aligned}
s_1 &= \textsf{citizen}(\text{ssn, name, citycode}) \\
s_2 &= \textsf{enterprise}(\text{ssn, name, citycode, emp\_number}) \\
s_3 &= \textsf{city}(\text{code, name, country}) \\
s_4 &= \textsf{ownerships}(\text{code, owner\_ssn, address, type})
\end{aligned}$$

We want to extract from these sources, only information about owners and their buildings. We can define the global schema $\mathcal{G}$ with two relations

$$\begin{array}{rcl} R_1 & = & \mathsf{owner}(\text{name, city}) \\ R_2 & = & \mathsf{building}(\text{address, owner}) \end{array}$$

and the mapping $\mathcal{M}$ can be defined as follows

$$\begin{array}{rcl} v_1 & = & \mathsf{owner}(X, Y) \leftarrow \mathsf{citizen}(W_1, X, Z), \mathsf{city}(Z, Y, W_2). \\ v_2 & = & \mathsf{owner}(X, Y) \leftarrow \mathsf{enterprise}(W_1, X, Z, W_2), \mathsf{city}(Z, Y, W_3). \\ v_3 & = & \mathsf{building}(X, Y) \leftarrow \mathsf{ownership}(W_1, Y, X, 'building'). \end{array}$$

A possible query q that asks for owners and address of buildings of Rome can be

$$q \quad = \quad \mathsf{Q}(X, Y) \leftarrow \mathsf{owner}(X, 'Rome'), \mathsf{building}(Y, X).$$

2.2 Semantics

A database instance (or simply database) $\mathcal{C}$ for a relational schema DB is a set of facts of the form $r(t)$ where r is a relation of arity n in DB and t is an n-tuple of constants of Γ. We denote as $r^{\mathcal{C}}$ the set $\{t \mid r(t) \in \mathcal{C}\}$ and with $q^{\mathcal{C}}$ the result of the evaluation of the query q (expressed over DB) on $\mathcal{C}$.

In order to assign semantics to a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we start considering a source database for $\mathcal{I}$, i.e., a database $\mathcal{D}$ for the source schema $\mathcal{S}$. We call global database for $\mathcal{I}$ any database for $\mathcal{G}$. Given a source database $\mathcal{D}$ for $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ the semantics *sem* of $\mathcal{I}$ w.r.t. $\mathcal{D}$, $sem(\mathcal{I}, \mathcal{D})$, is the set of *global databases* $\mathcal{B}$ for $\mathcal{I}$ such that $\mathcal{B}$ satisfies $\mathcal{M}$ with respect to $\mathcal{D}$. In particular, $\mathcal{B}$ is such that for each view $\rho_{\mathcal{M}}(R)$ in the GAV mapping $\mathcal{M}$, all the tuples that satisfy $\rho_{\mathcal{M}}(R)$ in $\mathcal{D}$ satisfy the global schema relation R, i.e. $\rho_{\mathcal{M}}(R)^{\mathcal{D}} \subseteq R^{\mathcal{B}}$.

Finally, we specify the semantics of queries posed to a data integration system. Such queries are expressed in terms of the symbols in the global schema of $\mathcal{I}$. Formally, given a source database $\mathcal{D}$ for $\mathcal{I}$, we call *certain answers* $q^{\mathcal{I},\mathcal{D}}$ to a query q with respect to $\mathcal{I}$ and $\mathcal{D}$, the set of tuples t of objects in Γ such that $t \in q^{\mathcal{B}}$ for every global database $\mathcal{B}$ for $\mathcal{I}$ with respect to $\mathcal{D}$, i.e $q^{\mathcal{I},\mathcal{D}} = \{t \mid \forall \mathcal{B} \in sem(\mathcal{I}, \mathcal{D}),\, t \in q^{\mathcal{B}}\}$.

3 System Architecture

In this section we describe the architecture of a novel system for efficient semantic data integration (depicted in Figure 1), based on a commercial tool for information integration, and explain its behavior.

Informally, the system operates in two modes. The first, called *setup*, is performed by the *Specification Processor* module and is enabled when a new data integration system is processed: the system reads the specification provided by the designer, and performs a series of actions in order to set up the environment. During this phase, a set of SQL statements are produced that enable IBM Information Integrator to access the sources; the connection with the sources is ensured by exploiting *nicknames*: a nickname is an alias that allows IBM Information Integrator to interact with a source as if it were a relational table. In other words, all the sources appear to be part of a unique *federated* database (see Section 4 for details). Furthermore, during the setup phase, the GAV mapping assertions are taken into account to produce a set of views definitions which

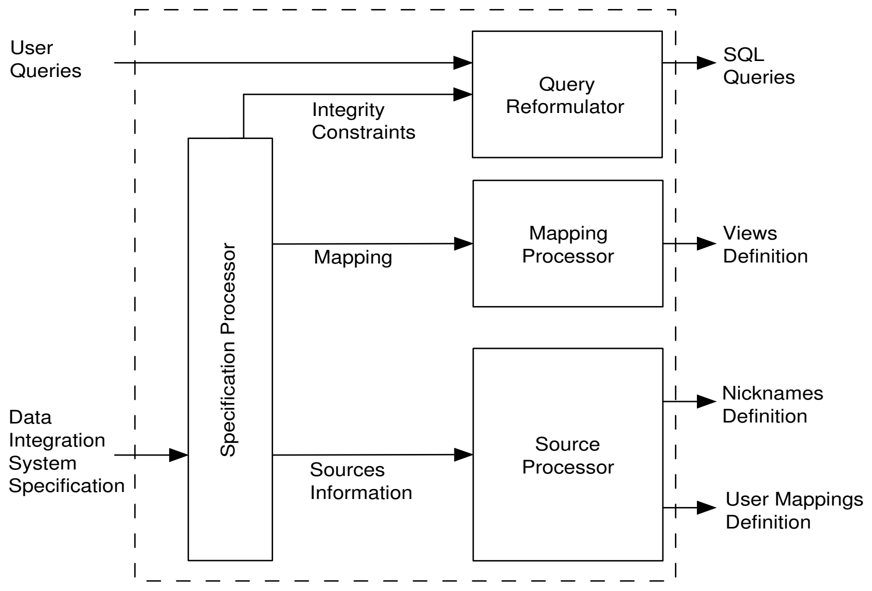


Figure 1: System architecture

are used to effectively integrate data stored in different sources. At the end of the setup phase we have a nickname for each element of the global schema. As we explain below, users will access data of different sources in a single query by using nicknames.

The second modality (*run*), is enabled every time a user issues a query to the system. Queries are posed over the elements of the global schema, and are processed in two steps: first, the query is rewritten into a first-order logic sentence, that encodes the integrity constraints expressed over the global schema together with the query (see Section 5.3). Such first-order sentence is then translated into SQL and passed to the SQL engine in order to retrieve the answers. Roughly speaking we impose constraints over nicknames by encoding them into the queries.

A more detailed description of the modules composing the system follows.

3.1 Specification processor

The Specification Processor analyzes the data integration system specification (global schema, sources schema, mapping) provided by the designer. The module extracts information about sources and mapping, passing it respectively to the *Source Processor* and to the *Mapping Processor*. It also stores the system information in a local meta-data repository, used to retrieve system information for future uses.

3.2 Sources processor

To be accessed, sources need to be "wrapped". This module receives sources' details from the Specification Processor and produces some SQL statements: as we will explain better in the following, such SQL statements are expressed in a DB2 dialect that allow IBM Information Integrator to interact with the sources. For those types of sources for which some permissions are required (user name and password, for example), the Source

Processor exploit its sub-module *Permission Manager*. It handles the information needed to ensure source access for those sources that require them, and produces SQL statements for user mappings creation.

3.3 Mapping processor

Reads the mapping specification provided by the Specification Processor and builds the view definitions used to encode the mapping of the data integration system specification. The definition are built according to the “internal technique” described in Section 4.4.

3.4 Query Reformulator

Is the module that encodes integrity constraint expressed over the global schema into the query. As mentioned at the beginning of this section, the module operates in two phases. The query is first translated into a first order logic sentences, according to the algorithm presented in Figure 8, that encodes both key dependencies and exclusion dependencies into the rewriting. Then the first order sentences is translated into SQL statements and passed to the SQL engine of IBM Information Integrator.

4 Exploiting the data federation approach of IBM Information Integrator

Data federation tools enable data from multiple heterogeneous data sources to appear as if it was contained in a single *federated* database [15]. Besides uniform access, performance and resource requirements are also considered in order to suitably face typical needs as joins or aggregations of different data sources.

In this section, we provide an overview of a commercial solution. We choose IBM Information Integrator, since it is known to provide a very efficient federation of heterogeneous data sources [5]. Then, starting from a critics of such tool, we discuss the limits of data federation with respect to data integration.

4.1 Accessing sources: IBM Information Integrator’s architecture

IBM Information Integrator (IBMII) is the IBM’s solution to the data federation problem [2, 1]. It provides a single uniform access to a large variety of heterogeneous data sources. More precisely, it provides different wrappers (implemented by different libraries) for many popular data sources, such as relational sources (Oracle, MS SQL Server, ...) as well as semi-structured XML sources and many common specialized data sources (such as an Excel table). Note that several homogeneous data sources can refer to the same wrapper. Then, inside a particular data source, data sets of interest are modelled as *nicknames*, that constitute basically a virtual view on the set of data. For example, inside a relational source, IBMII associates a nickname to each relation of interest in the source, while, for a non-relational XML source, IBMII associates a nickname to a fragment of the XML document characterized by an XPATH expression.

The IBMII wrapper architecture enables the federation of heterogenous data sources, maintaining the state information for each of them, managing connections, decomposing queries into fragments that each source can handle and managing transactions across data sources. It supplies the infrastructure to model multiple data sets and multiple operations, letting specialized functions supported by the external source to be invoked in a query even though DB2 does not natively support the function. Furthermore , it includes a flexible framework to provide input to the query optimizer about the cost of data source operations and the size of data that will be returned, supplying at the same time information based on the immediate context of the query. Finally, wrappers participate as resource managers [15] in a DB2 transaction.

4.2 Limits of data federation

Let us consider the IBMII's approach to information integration presented above. As a federation tool, note that IBMII does not let the designer define an arbitrary description of the domain of interest, through which the users can access external sources. In particular, we have identified two main limitating features of the product:

1. As we already mentioned, IBMII models an external source data set as a nickname that consists of a virtual view on the data set. Therefore, IBMII establishes a one-to-one correspondence between source data sets and nicknames, instead of letting the designer define a correspondence between a concept of interest, represented as a unique relation, and a view over the multiple source data sets.

2. Furthermore, if the source is relational, the nickname schemas are identical to those of the modelled data source relations, which means that both have the same number, name and type of attributes. Even if the definition of nickname schemas is a little more flexible for non-relational data sources, there are still several rules that the designer has to follow, which limit the expressiveness of the correspondence even inside a single source data set.

The mentioned limits are typical of data federation. Indeed, in this kind of tools, the designer is provided with a view of source data sets that is source-dependent, since it reflects basically the source data sets structure. In the next section, we present a formal approach to the problem of data integration showing how this theory can fulfil the gap enforced by such limitations.

4.3 Efficient data integration through data federation

As illustrated in the previous section, data integration supplies a higher level of abstraction with respect to data federation. In particular, as shown in Figure 2, a data integration system can be considered as composed of: (i) a global schema, that constitutes the intensional description of data of interest, (ii) a source schema, that consists basically of a federated schema, (iii) a set of source data sets, that represent the real data, (iv) a set of GAV mappings that model the relation between the global relations and the federated relations and (v) a set of wrappers that implement the correspondence one-to-one between the federated relations and the source data sets. Such a scenario can be seen as a specialization of the well known wrapper-mediator architecture [24].

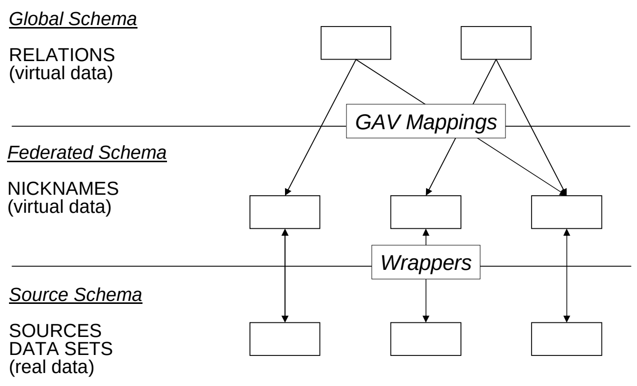


Figure 2: Levels of integration

In the following section, we first present an overview of our approach to realize efficient data integration system relying on a federation tool; then, we define two different techniques based on such an approach; finally, we discuss the correctness of both the techniques.

In particular, the solution we propose has the aim to implement an efficient data integration system that offers all the optimization techniques supplied by the federation tool, besides the expressiveness of the global schema. Note that, in the specification of the logical framework for data integration, we assume to deal with a relational source schema $\mathcal{S}$. Indeed, through this assumption, we implicitly assume to take advantage of the mechanism of wrappers supplied by the federation tool.

Let us use IBMII in order to implement a data integration system. We need first to build the source schema $\mathcal{S}$, by creating a set of nicknames. Then, we need to implement two modules:

1. a *compiler*, that is responsible of compiling the data integration system $\mathcal{I}$ in a IBMII instance $\mathcal{I}'$;

2. a *translator*, that is responsible to translate a query Q over the data integration system $\mathcal{I}$ in a query Q' over the corresponding IBMII instance $\mathcal{I}'$.

An overview of the architecture of the entire system is given in Figure 3.

Suppose to start from a IBMII instance that contains all the nicknames in $\mathcal{S}$. There are two possible techniques for the implementation of both modules presented above:

- **Internal technique:** the idea is to rely upon the IBMII management of views. The compiler builds a IBMII instance that contains a view V_R, for each global relation R. V_R is defined by the query $\rho(R)$ expressed in the data integration system mapping.

- **External technique:** the idea is to focus the process on the user queries, rather than on the system specification, implementing an *unfolding* technique [24] that takes into account the correspondences expressed in the mappings.

Note that, in the following, for the lack of space, we will omit the rules that specify the translation between the logic representation of both mapping and queries, and the corresponding SQL statement issued to the DBMS.

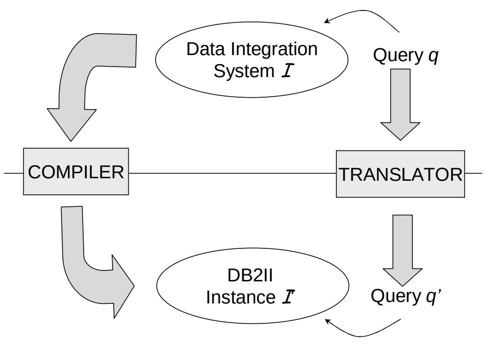

Data Integration System $\mathcal{I}$
Query q
COMPILER
TRANSLATOR
DB2II Instance $\mathcal{I}'$
Query q'

Figure 3: System architecture

```
$\mathcal{V} := \{\}$;
**for each**$R(\mathbf{A}) \in \mathcal{G}$ **do**
  $\mathcal{V} := \mathcal{V} \cup \langle V_R(\mathbf{A}), V_R \leftarrow body(\rho(R))\rangle$
```

Figure 4: $compile_i(\mathcal{I})$ algorithm.

4.4 Internal technique: implementing data integration using IBMII views

Since IBMII relies on DB2 UDB (Universal Database), the DBMS of IBM, a possible solution to realize data integration is to take advantage of the management of views of IBMII, in order to implement the correspondences expressed in the mapping. In particular, the IBMII instance $\mathcal{I}_i{}'$ that results from the compilation of an input data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ is characterized by a set of views $\mathcal{V} = \{W\}$, where each view is represented as a couple $W = \langle V(\mathbf{A}), \Psi(V)\rangle$ such that:

- $V(\mathbf{A})$ is the view schema, where $\mathbf{A}$ is a vector of attributes and V is the view name;
- $\Psi(V)$ is the conjunctive query on $\mathcal{S}$ that defines the view; $\Psi(V)$ has the form:
  $$V(\vec{x}) \leftarrow S_1(\vec{x_1}, \vec{y_1}), ..S_k(\vec{x_k}, \vec{y_k})$$
  where $\vec{x} = \bigcup_{h=1}^{k} \vec{x_h}$ and the components of the vectors $\vec{x}$ and $\vec{y_h}$ are variables or constants of the set Γ, for $h = 1..k$.

The compilation algorithm $compile_i(\mathcal{I})$, shown in Figure 4, generates the set $\mathcal{V}$ of views of $\mathcal{I}_i{}'$, by applying the following rule: if $R(\mathbf{A}) \in \mathcal{G}$, then insert into $\mathcal{V}$ the couple $W = \langle V_R(\mathbf{A}), \Psi(V_R)\rangle$, where:

- $\rho(R) = R(\vec{x}) \leftarrow r_1(\vec{x_1}, \vec{y_1}), ..r_k(\vec{x_k}, \vec{y_k})$,
- $\Psi(V_R) = V_R(\vec{x}) \leftarrow r_1(\vec{x_1}, \vec{y_1}), ..r_k(\vec{x_k}, \vec{y_k})$.

Now, after having created in the IBMII instance $\mathcal{I}_i{}'$ the set of views $\mathcal{V}$, we need to provide a translation mechanism that, given a query a conjunctive Q over $\mathcal{I}$, returns a corresponding query $Q_i{}'$ over the set of views $\mathcal{V}$ that belong to $\mathcal{I}_i{}'$. The translation algorithm $translate_i(Q, \mathcal{I})$, shown in Figure 5, generates the result, by applying the following two syntactical rules:

```
$q_{aux} := \{\}$;
for each $g \in body(Q)$ such that
$g = R(\vec{x_i}, \vec{y_i})$ do
  $q_{aux} := q_{aux} \cup V_R(\vec{x_i}, \vec{y_i})$
$Q' := head(Q) \leftarrow q_{aux}$
```

Figure 5: $translate_i(Q, \mathcal{I})$ algorithm.

1. the head of the query ${Q_i}'$ is identical to the head of the query Q;
2. for each atom $R(\vec{x_1}, \vec{y_1})$ in the body of Q, insert the atom $V_R(\vec{x_1}, \vec{y_1})$ in the body of ${Q_i}'$.

Example 2 Referring to the system specification $\mathcal{I}$ exposed in Example 1, we show the compiled IBMII instance $\mathcal{I}_i{}'$:

$$\begin{array}{rclcl}
\langle V_{\mathsf{owner}}(name, city) & , & \{V_{\mathsf{owner}}(X, Y) & \leftarrow & \mathsf{citizen}(W_1, X, Z), \mathsf{city}(Z, Y, W_2).,\\
 & & V_{\mathsf{owner}}(X, Y) & \leftarrow & \mathsf{enterprise}(W_1, X, Z, W_2),\\
 & & & & \mathsf{city}(Z, Y, W_3).\}\rangle\\
\langle V_{\mathsf{building}}(address, owner) & , & V_{\mathsf{building}}(X, Y) & \leftarrow & \mathsf{ownership}(W_1, Y, X, 'building').\rangle
\end{array}$$

Note that the view definition for V_{owner} is a union of conjunctive queries. The translation for the user query Q to be posed on the defined views is

$$Q_i \quad = \quad \mathsf{Q}(X, Y) \leftarrow V_{\mathsf{owner}}(X, 'Rome'), V_{\mathsf{building}}(Y, X).$$

The following theorem shows the correctness of the internal technique.

Theorem 3 *Given a data integration system specification $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, a database instance $\mathcal{D}$ for $\mathcal{I}$, a conjunctive query Q over $\mathcal{G}$ and a tuple $\bar{t}$, $\bar{t} \in q^{\mathcal{I},\mathcal{D}}$ if and only if $\bar{t}$ belongs to the set of answers we obtain by querying the compiled IBMII instance $\mathcal{I}_i{}'$ by means of the query ${Q_i}'$, where $\mathcal{I}_i{}'$ is the IBMII instance obtained from the compilation of $\mathcal{I}$ by means of the algorithm $compile_i(\mathcal{I})$, and ${Q_i}'$ is the query obtained from the translation obtained of Q translation by means of the algorithm $translate_i(Q, \mathcal{I})$.*

4.5 External technique: external management of global schema and mapping

In the second solution, we specify a technique for the implementation of a data integration system specification by means of external data structure. In short, we leave inside the IBMII instance $\mathcal{I}_e{}'$ only the federated schema $\mathcal{S}$, that is, the set of nicknames that wrap the sources, maintaining, by appropriate outer structure, the global schema and the mapping between global relations and nicknames. Therefore, in this solution, the compiler behaves like the identity function, i.e. the IBMII instance $\mathcal{I}_e{}'$ is constituted only by $\mathcal{S}$.

As we said in the previous sections, the system user poses his queries on the global schema. In order to process user's requests, we have to show a translation mechanism that allows the queries to be issued on the compiled IBMII instance $\mathcal{I}_e{}'$. Informally, this

```
$q_{aux}$ := $q$;
**for each** $g$ in $body(q)$
$v$ := mapping_by_head($g$);
  $\sigma$ := unify($g$, $head(v)$);
  **if** ($\sigma$ == {})
    **return** NULL;
  **else**
    $q_{aux}$ := $\sigma$[replace($q, g, body(v)$)];
  **end if**
**end for**
**return** $q_{aux}$;
```

Figure 6: $translate_e(q, \mathcal{I})$ algorithm

may be done by substituting each atom appearing in the body of the query with the body of its corresponding mapping definition. This process can be seen as an extension of the well-known *unfolding* algorithm [24], and can be formalized as follows. Given a conjunctive query q over the relational symbols of $\mathcal{G}$, where $\mathcal{G}$ is the global schema of $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we define *translated query* for $\mathcal{I}'$ the query $q' = translate_e(q, \mathcal{I})$ in which $q' = translate_e(q, \mathcal{I})$ is a new query expressed over the relational symbols of $\mathcal{S}$.

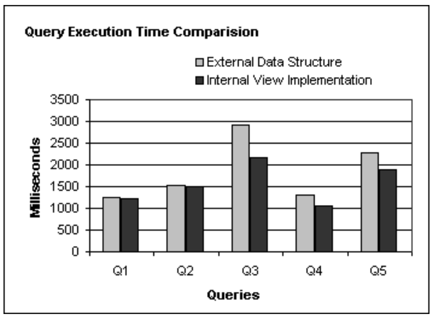
In Figure 6 we show the unfolding algorithm we adopt for query reformulation. Such an algorithm makes use of some subroutines: `mapping_by_head(`g`)`, that given an atom g returns the mapping view whose head contains g; `unify(`g_1`,`g_2`)`, that unifies the variables of g_1 and g_2, returning, if an unifier can be found, a set of substitutions σ that makes both the atoms equal. Moreover, the unfolding algorithm uses the subroutine `replace(`q`,`g`,`$conj$`)`, that, given a conjunctive query q, one of its body atoms g, and a conjunction $conj$, replaces each atom g of the body of the query q with the conjunction $conj$.

Example 4 We consider again the system specification $\mathcal{I}$ of Example 1. As we have said before, there is no need of compiling the specification, but we have to unfold the query q over the source relations. Based on the unfolding algorithm, we obtain the following union of conjunctive query

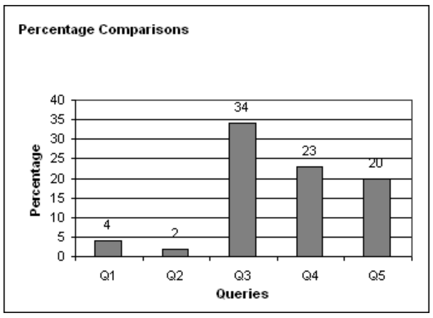
$$\begin{array}{lcl} Q'(X,Y) & \leftarrow & \mathsf{citizen}(W_1, X, Z), \mathsf{city}(Z, Y, W_2), \\ & & \mathsf{ownership}(W_3, Y, X, 'building'). \\ Q'(X,Y) & \leftarrow & \mathsf{enterprise}(W_1, X, Z, W_2), \mathsf{city}(Z, Y, W_3), \\ & & \mathsf{ownership}(W_4, Y, X, 'building'). \end{array}$$

The following theorem shows the correctness of the external technique.

Theorem 5 *Given a data integration system specification $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, a database instance $\mathcal{D}$ for $\mathcal{I}$, a conjunctive query Q over $\mathcal{G}$ and a tuple $\bar{t}$, $\bar{t} \in q^{\mathcal{I},\mathcal{D}}$ if and only if $\bar{t}$ belongs to the set of answers we obtain by querying the IBMII instance ${\mathcal{I}_e}'$, where ${\mathcal{I}_e}'$ is constituted only by $\mathcal{S}$, and ${Q_e}'$ is the query obtained from the translation obtained from Q by means of the $translate_e(Q, \mathcal{I})$ algorithm.*



(a) Query execution times



(b) Percentage comparisons

Figure 7: Data integration through data federation

4.6 Experimental results

In order to test the feasibility of both the techniques presented in the previous sections, we have carried out some experiments on real data, coming from the University of Rome “La Sapienza”. The experiment scenario comprises three data sources significantly overlapping: (i) a DB2 database instance holding administrative information, i.e. registry, exams and career information about students (204014 tuples)); (ii) a Microsoft SQLServer database instance storing information about exam plan of students (28790 tuples); (iii) some XML documents containing information about exams of students of the computer science diploma course (5836 tuples). Note that the number of tuples refers to the federated tables, resulting from wrapping.

Starting from these sources, we have built a GAV data integration system whose specification, that we omit for the lack of space, contains 10 global relations and 27 source relations. In order to test the efficiency of our solutions regardless of network traffic and delays, we have carried out the experiments on local instances of the data, so as to have a truthful comparison of both the techniques. We have conducted the experiments on an Double Intel Pentium IV Xeon machine, with 3 GHz processor clock frequency, equipped with 2 GB of RAM memory and a 30 GB SCSI hard disk at 7200 RPM; the machine runs the Windows XP operating system.

We have run 5 test queries on both the specifications: one obtained by the internal technique presented in Section 4.4 and the other one generated with the external technique proposed in Section 4.5. We observe that the internal technique based on IBMII views definition is faster than the external one, which implements a query rewriting algorithm. Furthermore, the gain obtained with the first technique is major for queries that present an higher number of joins. This is due to the capability of the IBMII query engine to efficiently process views, taking into account source statistics and some access optimization. Comparative execution times are shown in Figure 7, where the percentage difference between the lower and the faster technique is also presented.

5 How to deal with integrity constraints: Inconsistent databases and consistent answers

The system architecture introduced so far, strongly separates two of the hardest tasks of query answering in data integration systems: (i) accessing the data sources and (ii) processing the integrity constraints. Hence, in this section, we introduce a formal framework for consistent query answering in a single relational database setting, which directly applies to our data integration scenario where nicknames mask the sources layer (see Section 3 for details).

5.1 Syntax

A *database schema* $\mathcal{S}$ is a triple $\langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$, where:

- $\mathcal{A}$ is a relational signature.
- $\mathcal{K}$ is a set of *key dependencies* over $\mathcal{A}$. A key dependency (KD) over $\mathcal{A}$ is an expression of the form $key(r) = \{i_1, \ldots, i_k\}$, where r is a relation of $\mathcal{A}$, and, if n is the arity of r, $1 \leq i_j \leq n$ for each j such that $1 \leq j \leq k$. We assume that at most one KD is specified over a relation r.
- $\mathcal{E}$ is a set of *exclusion dependencies* over $\mathcal{A}$. An exclusion dependency (ED) over $\mathcal{A}$ is an expression of the form $r_1[i_1, \ldots, i_k] \cap r_2[j_1, \ldots, j_k] = \emptyset$, where r_1, r_2 are relations of $\mathcal{A}$, and, if n_1 and n_2 are the arities of r_1 and r_2 respectively, for each ℓ such that $1 \leq \ell \leq k$, $1 \leq i_\ell \leq n_1$ and $1 \leq j_\ell \leq n_2$.

A *term* is either a variable or a constant symbol. An *atom* is an expression of the form $p(t_1, \ldots, t_n)$ where p is a relation symbol of arity n and $t_1, \ldots, t_n$ is a sequence of n terms (either variables or constants). An atom is called *fact* if all the terms occurring in it are constants. A *database instance* $\mathcal{D}$ for $\mathcal{S}$ is a set of facts over $\mathcal{A}$. We denote as $r^{\mathcal{D}}$ the set $\{t \mid r(t) \ \in \mathcal{D}\}$.

A *conjunctive query* of arity n is an expression of the form $h(x_1, \ldots, x_n) \coloneq a_1, \ldots, a_m$, where the atom $h(x_1, \ldots, x_n)$, is called the *head* of the query (denoted by *head*(q)), and $a_1, \ldots, a_m$, called the *body* of the query (and denoted by *body*(q)), is a set of atoms, such that all the variables occurring in the query head also occur in the query body. In a conjunctive query q, we say that a variable is a *head variable* if it occurs in the query head, while we say that a variable is *existential* if it only occurs in the query body. Moreover, we call an existential variable *shared* if it occurs at least twice in the query body (otherwise we say that it is *non-shared*). A *FOL query* of arity n is an expression of the form $\{x_1, \ldots, x_n \mid \Phi(x_1, \ldots, x_n)\}$, where $x_1, \ldots, x_n$ are variable symbols and Φ is a first-order formula with free variables $x_1, \ldots, x_n$.

5.2 Semantics

First, we briefly recall the standard evaluation of queries over a database instance. Let q be the CQ $h(x_1, \ldots, x_n) \coloneq a_1, \ldots, a_m$ and let $t = \langle c_1, \ldots, c_n \rangle$ be a tuple of constants. A set of facts I is an *image of* t *w.r.t.* q if there exists a substitution σ of the variables occurring in q such that $\sigma(head(q)) = h(t)$ and $\sigma(body(q)) = I$. Given a database instance $\mathcal{D}$, we denote by $q^{\mathcal{D}}$ the evaluation of q over $\mathcal{D}$, i.e., $q^{\mathcal{D}}$ is the set of tuples t such that there exists an image I of t w.r.t. q such that $I \subseteq \mathcal{D}$.

Given a FOL query q and a database instance $\mathcal{D}$, we denote by $q^{\mathcal{D}}$ the evaluation of q over $\mathcal{D}$, i.e., $q^{\mathcal{D}} = \{t_1, \ldots, t_n \mid \mathcal{D} \models \Phi(t_1, \ldots, t_n)\}$, where each t_i is a constant symbol and $\Phi(t_1, \ldots, t_n)$ is the first-order sentence obtained from Φ by replacing each free variable x_i with the constant t_i.

Then, we define the semantics of queries over inconsistent databases. A database instance $\mathcal{D}$ *violates* the KD $key(r) = \{i_1, \ldots, i_k\}$ iff there exist two *distinct* facts $r(c_1, \ldots, c_n)$, $r(d_1, \ldots, d_n)$ in $\mathcal{D}$ such that $c_{i_j} = d_{i_j}$ for each j such that $1 \leq j \leq k$. Moreover, $\mathcal{D}$ violates the ED $r_1[i_1, \ldots, i_k] \cap r_2[j_1, \ldots, j_k] = \emptyset$ iff there exist two facts $r_1(c_1, \ldots, c_n)$, $r_2(d_1, \ldots, d_m)$ in $\mathcal{D}$ such that $c_{i_\ell} = d_{j_\ell}$ for each ℓ such that $1 \leq \ell \leq k$.

Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$ be a database schema. A database instance $\mathcal{D}$ is *legal for* $\mathcal{S}$ if $\mathcal{D}$ does not violate any KD in $\mathcal{K}$ and does not violate any ED in $\mathcal{E}$.

A set of ground atoms $\mathcal{D}'$ is a *repair* of $\mathcal{D}$ under $\mathcal{S}$ iff: (i) $\mathcal{D}' \subseteq \mathcal{D}$; (ii) $\mathcal{D}'$ is legal for $\mathcal{S}$; (iii) for each $\mathcal{D}''$ such that $\mathcal{D}' \subset \mathcal{D}'' \subseteq \mathcal{D}$, $\mathcal{D}''$ is not legal for $\mathcal{S}$. In words, a repair for $\mathcal{D}$ under $\mathcal{S}$ is a maximal subset of $\mathcal{D}$ that is legal for $\mathcal{S}$.

Let q be a CQ. A tuple t is a *consistent answer* to q in $\mathcal{D}$ under $\mathcal{S}$ iff, for each repair $\mathcal{D}'$ of $\mathcal{D}$ under $\mathcal{S}$, $t \in q^{\mathcal{D}'}$.

Example 6 Consider the database schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$, where $\mathcal{A}$ comprises the relations $Journal(title, editor)$, $ConfPr(title, editor)$ and $Editor(name, country)$, $\mathcal{K}$ comprises the dependencies $key(Journal) = \{1\}$, $key(ConfPr) = \{1\}$, $key(Editor) = \{1\}$, $\mathcal{E}$ comprises the dependency $Journal[1] \cap ConfPr[1] = \emptyset$. Consider the database instance $\mathcal{D}$ described below

$$\begin{gathered}\{Journal(\mathsf{TODS}, \mathsf{ACM}), Journal(\mathsf{TODS}, \mathsf{IEEE}),\\ Editor(\mathsf{ACM}, \mathsf{USA}), ConfPr(\mathsf{PODS05}, \mathsf{ACM}),\\ ConfPr(\mathsf{PODS05}, \mathsf{SV}), Editor(\mathsf{IEEE}, \mathsf{USA})\}.\end{gathered}$$

It is easy to see that $\mathcal{D}$ is not consistent with the KDs on *Journal* and *ConfPr* of $\mathcal{S}$. Then, the repairs of $\mathcal{D}$ under $\mathcal{S}$ are:

$$\{Journal(\mathsf{TODS}, \mathsf{ACM}), ConfPr(\mathsf{PODS05}, \mathsf{ACM}),\\ \qquad Editor(\mathsf{ACM}, \mathsf{USA}), Editor(\mathsf{IEEE}, \mathsf{USA})\}$$

$$\{Journal(\mathsf{TODS}, \mathsf{ACM}), ConfPr(\mathsf{PODS05}, \mathsf{SV}),\\ \qquad Editor(\mathsf{ACM}, \mathsf{USA}), Editor(\mathsf{IEEE}, \mathsf{USA})\}$$

$$\{Journal(\mathsf{TODS}, \mathsf{IEEE}), ConfPr(\mathsf{PODS05}, \mathsf{ACM}),\\ \qquad Editor(\mathsf{ACM}, \mathsf{USA}), Editor(\mathsf{IEEE}, \mathsf{USA})\}$$

$$\{Journal(\mathsf{TODS}, \mathsf{IEEE}), ConfPr(\mathsf{PODS05}, \mathsf{SV}),\\ \qquad Editor(\mathsf{ACM}, \mathsf{USA}), Editor(\mathsf{IEEE}, \mathsf{USA})\}.$$

Let $q(x, z) \coloneq Journal(x, y), Editor(y, z)$ be a user query. The consistent answers to q in $\mathcal{D}$ under $\mathcal{S}$ are $\{\langle \mathsf{TODS}, \mathsf{USA} \rangle\}$. □

5.3 Query answering

In this section we present a technique to deal with integrity constraints by encoding them into the users' queries. As introduced in Section 3, queries are first rewritten in terms of a first-order logic formula that encapsulates integrity constraints expressed over the global schema, and then translated into an SQL statement.

It is important to note that, besides the first-order technique shown in this paper, some other approaches have been presented that exploit logic programs [17, 20, 4] to retrieve consistent answers from an inconsistent database. Experiments conducted in Section 5.5 compare the relative efficiency of such standard techniques with respect to the first-order technique presented below.

In the following, first we characterize the computational complexity of the general problem, and present **FOLRewrite**, the algorithm that produces the first-order rewriting of the queries.

The problem of computing consistent answers to conjunctive queries over inconsistent databases in the presence of KDs (under the repair semantics introduced in Section 5) is coNP-hard in data complexity [7, 10]. In the following, we prove that such a problem is coNP-hard in data complexity also for schemas in which only EDs occur[2].

Theorem 7 *Let $\mathcal{S} = \langle \mathcal{A}, \emptyset, \mathcal{E} \rangle$ be a database schema containing only EDs, $\mathcal{D}$ a database instance for $\mathcal{S}$, q a CQ of arity n over $\mathcal{S}$, and t an n-tuple of constants. The problem of establishing whether t is a consistent answer to q in $\mathcal{D}$ under $\mathcal{S}$ is coNP-hard with respect to data complexity.*

Proof (sketch). We prove coNP-hardness by reducing the 3-colorability problem to the complement of our problem. Consider a graph $G = \langle V, E \rangle$ with a set of vertices V and edges E. We define a relational schema $\mathcal{S} = \langle \mathcal{A}, \emptyset, \mathcal{E} \rangle$ where $\mathcal{A}$ consists of the relation *edge* of arity 2, and the relation *col* of arity 5, and $\mathcal{E}$ contains the dependencies $col[3] \cap col[4] = \emptyset$, $col[3] \cap col[5] = \emptyset$, $col[4] \cap col[5] = \emptyset$. The instance $\mathcal{D}$ is defined as follows:

$$\begin{aligned}\mathcal{D} \;=\; & \{col(n, 1, n, \_, \_), col(n, 2, \_, n, \_), col(n, 3, \_, \_, n) \mid \\ & n \in V\} \cup \{edge(x, y) \mid \langle x, y \rangle \in E\}.\end{aligned}$$

Where each occurrence of the meta-symbol $\_$ denotes a different constant not occurring elsewhere in the database. Intuitively, to represent the fact that vertex $n \in V$ is assigned with color $i \in \{1, 2, 3\}$, $\mathcal{D}$ assigns to *col* a tuple in which i occurs as second component and n occurs as first and also as $2 + i$-th component. The EDs of $\mathcal{S}$ impose that consistent instances assign no more than one color to each node. Finally, we define the query

$$q \;\leftarrow\; edge(x, y), col(x, z, w_1, w_2, w_3), col(y, z, w_4, w_5, w_6).$$

On the basis of the above construction it is possible to show that G is 3-colorable (i.e., for each pair of adjacent vertices, the vertices are associated with different colors) if and only if the empty tuple $\langle\rangle$ is not a consistent answer to q in $\mathcal{D}$ under $\mathcal{S}$ (i.e., the boolean query q has a negative answer). □

[2]We consider the decision problem associated to query answering (see e.g., [10])

5.4 FOL Rewriting

Let us now consider a different approach to consistent query answering, which aims at identifying *subclasses* of queries for which the problem is tractable. This is the line followed in [3, 10, 13]. In particular, in [13] the authors define a subclass of CQs, called $\mathcal{C}_{tree}$, for which they prove tractability of consistent query answering in the presence of KDs, and provide a FOL rewriting technique. The class $\mathcal{C}_{tree}$ is based on the notion of join graph: a *join graph of a query q* is the graph that contains (i) a node N_i for every atom in the query body, (ii) an arc from N_i to N_j iff an existential shared variable occurs in a non-key position in N_i and occurs also in N_j, (iii) an arc from N_i to N_i iff an existential shared variable occurs at least twice in N_i, and one occurrence is in a non-key position. According to [13], $\mathcal{C}_{tree}$ is the class of conjunctive queries (a) without repeated relation symbols, (b) in which every join condition involves the entire key of at least one relation and (c) whose join graph is acyclic. As pointed out in [13], this class of queries is very common, since cycles are rarely present in queries used in practice. However, no repeated symbols may occur in the queries, and queries must have joins from non-key attributes of a relation to the entire key of another one.

We now extend the work of [13] as follows:

- We refine the class $\mathcal{C}_{tree}$ by allowing join conditions in which not necessarily the entire key of one relation has to be involved, but it is sufficient that, for each pair of attributes, at least one attribute must belong to a key (i.e., we allow for joins involving portions of key). In such a way, we obtain a new class, called $\mathcal{C}^+_{tree}$, larger than $\mathcal{C}_{tree}$, for which consistent query answering is polynomial in the presence of KDs. In other words, $\mathcal{C}^+_{tree}$ is the class of conjunctive queries for which only condition (a) and (c) above hold.
- We refine the class $\mathcal{C}^+_{tree}$ in order to obtain a class of queries, called *$\mathcal{KE}$-simple*, for which consistent query answering is polynomial in the presence of both KDs and also EDs.
- We provide a new algorithm for computing the FOL rewriting for *$\mathcal{KE}$-simple* queries. In the algorithm, we exploit the notion of join graph of [13], but we enrich the structure of the graph by associating to each node an adornment which specifies the different nature of terms in the atoms (see below), in order to deal with *$\mathcal{KE}$-simple* queries.

Let us describe in detail our technique. Henceforth, given a CQ q, we denote by R_q the set of relation symbols occurring in *body*(q). Given a database schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$ and a CQ q, we denote by $O_{\mathcal{E}}(q)$ the set of relation symbols $O_{\mathcal{E}}(q) = \{s \mid r[j_1, \ldots, j_k] \cap s[\ell_1, \ldots, \ell_k] = \emptyset \in \mathcal{E} \text{ and } r \in R_q\}$. In words, $O_{\mathcal{E}}(q)$ contains each relation symbol $s \in \mathcal{A}$ such that there exists an exclusion dependency between s and r in $\mathcal{E}$, where r is a relation symbol occurring in *body*(q).

Definition 8 Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$ be a database schema. A conjunctive query q is *$\mathcal{KE}$-simple* if $q \in \mathcal{C}^+_{tree}$, and

- there exists no pair of relation symbols r, s in $O_{\mathcal{E}}(q)$ such that there exists an exclusion dependency between r and s in $\mathcal{E}$,

```
**Algorithm** FolTree($N$,$\mathcal{E}$)
**Input:** node $N$ of $JG(q)$; set of EDs $\mathcal{E}$
**Output:** FOL formula
**begin**
  let $a = r(x_1/t_1, \ldots, x_n/t_n)$ be the label of $N$;
  **for** $i := 1$ **to** $n$ **do**
    **if** $t_i \in \{KB, B\}$ **then** $v_i := x_i$
    **else** $v_i := y_i$, where $y_i$ is a new variable
  **if** each argument of $a$ is of type $B$ or $KB$ **then** $f_1 := r(x_1, \ldots, x_n)$
  **else begin**
    let $i_1, \ldots, i_m$ be the positions of the arguments of $a$ of type $S$, $U$, $KU$;
    $f_1 := \exists y_{i_1}, \ldots, y_{i_m}.\ r(v_1, \ldots, v_n)$
  **end**;
  **for each** ED $r[j_1, \ldots, j_k] \cap s[\ell_1, \ldots, \ell_k] = \emptyset \in \mathcal{E}$ **do**
  **begin**
    let $m$ be the arity of $s$;
    **for** $i := 1$ **to** $m$ **do**
      **if** $i \in \{\ell_1, \ldots, \ell_k\}$ **then if** $i = \ell_c$ **then** $z_i = v_{j_c}$
      **else** $z_i = y'_i$ where $y'_i$ is a new variable;
    let $y'_{i_1}, \ldots, y'_{i_k}$ be the new variables above introduced;
    $f_1 = f_1 \wedge \neg\exists y'_{i_1}, \ldots, y'_{i_k}.\ s(z_1, \ldots, z_m)$
  **end**
  **if** there exists no argument in $a$ of type $B$ or $S$ **then return** $f_1$
  **else begin**
    let $p_1, \ldots, p_c$ be the positions of the arguments of $a$ of type $U$, $S$ or $B$;
    let $\ell_1, \ldots, \ell_h$ be the positions of the arguments of $a$ of type $B$;
    **for** $i := 1$ **to** $c$ **do**
      **if** $t_{p_i} = S$ **then** $z_{p_i} := x_{p_i}$ **else** $z_{p_i} := y''_i$, where $y''_i$ is a new variable
    **for** $i := 1$ **to** $n$ **do**
      **if** $t_i \in \{KB, KU\}$ **then** $w_i := v_i$ **else** $w_i := z_i$;
    $f_2 := \forall z_{p_1}, \ldots, z_{p_c}.\ r(w_1, \ldots, w_n) \rightarrow \left( \bigwedge_{N' \in jgsucc(N)} \mathsf{FolTree}(N') \right) \wedge \bigwedge_{i \in \{\ell_1, \ldots, \ell_h\}} w_i = x_i$
    **return** $f_1 \wedge f_2$
  **end**
**end**
```

Figure 8: The algorithm FolTree

- there exists no relation symbol r in $O_{\mathcal{E}}(q)$ such that there exists $r[i_1, \ldots, i_k] \cap s[j_1, \ldots, j_k] = \emptyset$ in $\mathcal{E}$, and either $key(r) \not\supseteq \{i_1, \ldots, i_k\}$ or $key(s) \not\supseteq \{j_1, \ldots, j_k\}$, where s is a relation symbol in R_q.

In words, a query q is $\mathcal{KE}$-simple if it belongs to the class $\mathcal{C}^+_{tree}$, and if both there are no EDs between relations that are in $O_{\mathcal{E}}(q)$, and each ED between a relation $r \in R_q$ and a relation $s \in O_{\mathcal{E}}(q)$ does not involve non-key attributes of r or s. Notice that this last condition does not limit the applicability of our approach in many practical cases. For example, in relational databases obtained from ER-schemas, EDs are typically specified between keys.

For $\mathcal{KE}$-simple CQs, we present in the following a query rewriting algorithm which, given a query q, produces a FOL rewriting, whose evaluation over any database instance $\mathcal{D}$ for the database schema $\mathcal{S}$ returns the consistent answers to q in $\mathcal{D}$ under $\mathcal{S}$. The basic idea of the algorithm is to specify a set of conditions, expressible in FOL, that, if verified over a database instance $\mathcal{D}$, for a given tuple t, guarantee that in any repair of $\mathcal{D}$ there is an image of t w.r.t q, i.e., t is a consistent answer to q in $\mathcal{D}$. We point out that, for non-$\mathcal{KE}$-simple CQs, such conditions cannot be specified in FOL. Observe that, in our approach, the FOL rewriting is then in turn translated into SQL, and query evaluation is performed by means of standard DBMS query answering techniques. This further encoding does not present particular difficulties, and due to space limit we omit such transformation.

In order to construct our join graph we need the following definition.

Definition 9 Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$ be a database schema, q be a CQ, and $a = r(x_1, \ldots, x_n)$ be an atom (of arity n) occurring in R_q. Then, let $key(r) = \{i_1, \ldots, i_k\}$ belong to $\mathcal{K}$, and let $1 \leq i \leq n$. The *type of the i-th argument of a in q*, denoted by $type(a, i, q)$ is defined as follows:

1. If $i_1 \leq i \leq i_k$, then:
   - if x_i is a head variable of q, a constant, or an existential shared variable, then $type(a, i, q) = KB$;
   - if x_i is an existential non-shared variable of q, then $type(a, i, q) = KU$.
2. Otherwise ($i_1 \notin \{i_1, \ldots, i_k\}$):
   - if x_i is a head variable of q or a constant, then $type(a, i, q) = B$;
   - if x_i is an existential shared variable of q, then $type(a, i, q) = S$;
   - if x_i is an existential non-shared variable of q, then $type(a, i, q) = U$.

Terms typed by KB or B are called *bound terms*, otherwise they are called *unbound.* We call the *typing of a in q* the expression of the form $r(x_1/t_1, \ldots, x_n/t_n)$, where each t_i is the type of the argument x_i in q.

The following algorithm KEFolRewrite computes the FOL rewriting to a $\mathcal{KE}$-simple conjunctive query q. In the algorithm, $JG(q)$ denotes the join graph of q, in which each node N_i is labelled with the typing of the corresponding atom a_i in q. Furthermore, $roots(JG(q))$ denotes the set of nodes that are roots in $JG(q)$ (notice that for $\mathcal{KE}$-simple queries the join graph is a forest, since it is acyclic).

relations		**integrity constraints**	
$faculty/3$	$exam\_plan/10$	$key(faculty) = \{1,2\}$	$key(plan\_status) = \{1\}$
$course\_assignment/3$	$degree/5$	$key(exam\_plan) = \{1\}$	$key(positioning) = \{1\}$
$positioning/2$	$course/4$	$key(university) = \{1\}$	$key(prof\_data) = \{1\}$
$plan\_status/2$		$key(exam\_type) = \{1\}$	$key(degree) = \{1\}$
$prof\_data/3$		$key(course) = \{1\}$	$key(exam) = \{2\}$
$university/3$		$key(master\_exam) = \{1\}$	
$bachelor\_exam/2$		$key(bachelor\_exam) = \{1\}$	
$master\_exam/2$		$course\_assignment[2] \cap professor[1] = \emptyset$	
$exam\_type/2$		$master\_exam[1] \cap bachelor\_exam[1] = \emptyset$	
$exam/4$		$course[3,4] \cap bachelor\_exam[1,2] = \emptyset$	

Figure 9: A portion of the test database schema

```
**Algorithm** KEFolRewrite($q, \mathcal{S}$)
**Input:** $\mathcal{KE}$-simple CQ $q$ (whose head variables are $x_1, \ldots, x_n$);
        schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$
**Output:** FOL query (representing the rewriting of $q$)
**begin**
  compute $JG(q)$;
  **return** $\{x_1, \ldots, x_n \mid \bigwedge_{N \in roots(JG(q))} \mathsf{FolTree}(N, \mathcal{E})\}$
**end**
```

Basically, the algorithm builds the join graph of q and then builds the first-order query by invoking the algorithm FolTree on all the nodes that are roots of the join graph.

The algorithm FolTree is defined in Figure 8. Roughly speaking, the algorithm FolTree($N, \mathcal{E}$) returns a first-order formula that constitutes the encoding of the whole subtree of the join graph of the query whose root is the node N. To do that, the algorithm computes two subformulas f_1 and f_2. The formula f_1 contains an atom whose predicate is the predicate r labelling the node N, in which the unbound variables of r are renamed with new existentially quantified variables. Furthermore, f_1 contains an atom of the form $\neg\exists y'_{i_1}, \ldots, y'_{i_k} . s(z_1, \ldots, z_m)$ for each ED that involves r and a relation s. Intuitively, when evaluated over a database instance $\mathcal{D}$, each such atom checks that there are no facts of the form $s(t_s) \in \mathcal{D}$ that violate the ED together with a fact of the form $r(t_r) \in \mathcal{D}$, which is in an image I of a tuple t w.r.t. the input query q, i.e., the atom guarantees that I is not contradicted w.r.t. the ED.

The formula f_2 is empty only when all non-key arguments of the atom r are existential non-shared variables (i.e., of type U). Otherwise, the formula f_2 is a universally quantified implication. In such an implication, the antecedent is an atom whose predicate is r, and the consequent is a conjunction of equality conditions and other subformulas: more precisely, there is an equality condition for each non-key argument in r of type B, and a subformula for each successor N' of N in the join graph of q, computed by recursively invoking FolTree on N'. Intuitively, f_2 enforces the joins between r and each atom labelling the successors of r in the join graph of q. At the same time f_2 ensures that, when evaluated over a database instance $\mathcal{D}$, if there exists a fact of the form $r(\overline{t_r}) \in \mathcal{D}$ that violates the KD specified on r together with a fact of the form $r(t_r) \in \mathcal{D}$, which is in the image of a tuple t w.r.t. q, $r(\overline{t_r})$ belongs to another image of t w.r.t. q. In other words, the atom guarantees that in any repair there exists an image of t (w.r.t. the KD on r). Such a check is iterated for other KDs by recursively invoking FolTree. The following example illustrates the way the algorithm works.

Example 6 (contd.). It is easy to verify that the query $q(x,z) :\!-\ Journal(x,y), Editor(y,z)$ is $\mathcal{KE}$-simple.

$$Journal(x/KB, y/S)\ (N1) \longrightarrow (N2)\ Editor(y/KB, z/B)$$

Now, by applying the algorithm KEFolRewrite and FolTree we obtain:

$$\begin{array}{rcl} \mathsf{KEFolRewrite}(q) & = & \{x, z \mid \mathsf{FolTree}(N1)\} \\ \mathsf{FolTree}(N1) & = & \exists y_2.\ Journal(x,y_2) \land \neg\exists y_2'.\ ConfPr(x,y_2') \land \\ & & \forall y.\ Journal(x,y) \rightarrow (\mathsf{FolTree}(N2)) \\ \mathsf{FolTree}(N2) & = & Editor(y,z) \land \forall y_2''.\ Editor(y,y_2'') \rightarrow y_2'' = z. \end{array}$$

By evaluating the rewriting over $\mathcal{D}$ we get $\{\langle \mathsf{TODS}, \mathsf{USA}\rangle\}$, i.e., the set of consistent answers to q in $\mathcal{D}$ under $\mathcal{S}$. □

Next, we state soundness and completeness of the algorithm.

Theorem 10 *Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E}\rangle$ be a database schema, q be a $\mathcal{KE}$-simple conjunctive query over $\mathcal{S}$, and q_r be the FOL rewriting returned by KEFolRewrite(q). Then, for every database instance $\mathcal{D}$ for $\mathcal{S}$, a tuple t is a consistent answer to q in $\mathcal{D}$ under $\mathcal{S}$ iff $t \in q_r^{\mathcal{D}}$.*

As a corollary, consistent query answering for $\mathcal{KE}$-simple conjunctive queries over database schemas with KDs and EDs is polynomial in data complexity.

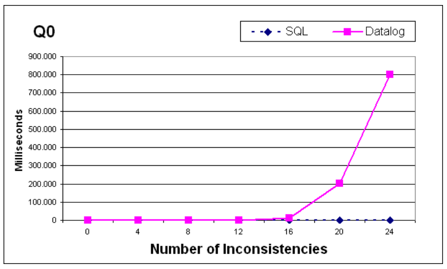
5.5 Experiments

We now present some experimental results comparing the FOL rewriting previously described and the standard Datalog$^\neg$ approach.

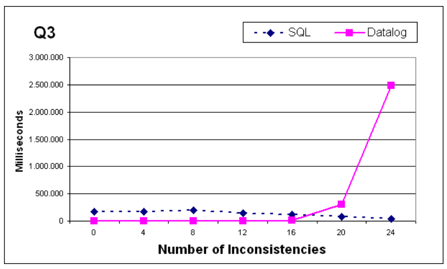
To perform the experiments, we implemented a rewriting module that translates CQs issued over the database schema into both FOL queries and Datalog$^\neg$ queries. FOL queries are in turn translated by the module into SQL queries. Then, we ran the SQL queries on a MySQL 4.1.10 instance of the test database, while we executed Datalog$^\neg$ queries on DLV [20]. The experiments were conducted on a double processor machine, with 3 GHz Pentium IV Xeon CPU and 2 GB of main memory, running the Linux operating system.

The test database holds information about the computer science engineering degrees of the university of Rome "La Sapienza" and contains 27 tables with an overall size of over 200.000 tuples. In Figure 9, we present the portion of the test database schema that is relevant for the queries (in the figure, "r/n" indicates that relation r is of arity n). Due to space limits, we only report details about three of the queries we tested:

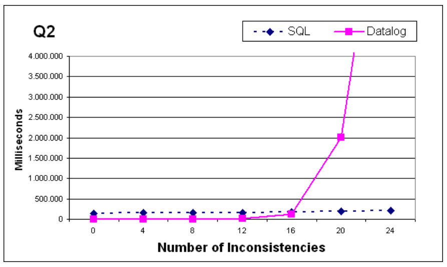
$$\begin{array}{rcl} Q_0 & = & q(C) : -faculty(C, U, 'INGEGNERIA'). \\ Q_2 & = & q(S, D, P) : -positioning(PS, P), plan\_status(ST, DE), \\ & & exam\_plan(C, S, PS, DT, ST, '1', U1, U2, U3, U4). \\ Q_3 & = & q(N, D, NP, CP) : -master\_exam(C, N, T, '5'), \\ & & exam\_type(T, D), \end{array}$$



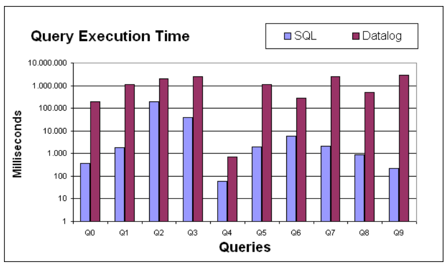
(a) Q_0 execution time



(b) Q_3 execution time



(c) Q_2 execution time



(d) SQL vs. Datalog$^\neg$

Figure 10: Experimental Results

The queries have been posed on various instances of the test database with an increasing number of pairs of tuples violating some ICs.

In Figure 7, some experimental results are summarized. In the charts 10(a), 10(b) and 10(c), the execution time of the SQL encoding and of the Datalog$^\neg$ program are compared for queries Q_0, Q_2, and Q_3. As expected, from a certain inconsistency level on, the execution time of the Datalog$^\neg$ encoding has an exponential blow-up; in contrast, the execution time for the SQL encoding is constant on the average, and for Q_3 (Figure 10(b)) it decreases: although this might be surprising, it turns out that some inconsistency allows the SQL engine to prune the search space for query answering.

Moreover, the chart presented in Figure 10(d) compares, on a logarithmic scale, the execution time of all queries at the highest inconsistency level. It shows that the SQL encoding is always more efficient when the degree of data inconsistency grows; however, it turns out that the method based on Datalog$^\neg$ and DLV proves particularly efficient in the presence of few data inconsistencies.

6 Work plan

The topics presented in this paper enforce the need to build a novel system that really integrates efficiently data coming from heterogeneous and distributed sources. In fact, as mentioned in the previous sections, all the available solutions to the information integration problem exhibit some limitations; while solutions coming from software development area mostly take care of procedural aspects like distribution and heterogeneity of sources, academic research in data integration mainly faces theoretical issues related to the expressiveness of the data models adopted and semantic characterization of query answering, among others.

The goal of this work is very ambitious: it aims to exploit solutions coming from both the aforementioned fields, in order to produce an efficient and effective solution to the information integration problem, built on commercial tools, that exploits the latest research results in information integration.

The work done so far led us to some partial results. In particular:

- the definition of algorithms for consistent query answering in data integration systems, that are able to deal with some type of integrity constraints (key constraints and exclusion dependencies) [17].
- The development of software modules implementing some of the techniques presented in this report, related to (i) consistent query answering [17, 6] and (ii) data integration via data federation [23];
- Several experimental results conducted on real data integration environments.

Of course, the largest part of the work has still to be done. More specifically:

- the extension of the first-order rewriting technique to the presence of more expressive forms of integrity constraints seems to be a very important improvement; in particular, the capability to handle foreign key and inclusion dependencies would make the first-order approach really significant.
- Other forms of constraints should be taken into account: in particular, domain constraints would add some useful features, by significantly enriching the expressiveness of the domains of interest handled by the system.
- The extension of the language adopted for the mapping assertions to more complex forms of mapping like LAV and GLAV mapping [22].
- Besides theoretical research issues, the implementation of the whole system is another significant step.
- In order to validate the work, several experiments on real cases scenarios have to be done.
- Last but not least, the whole approach should be made independent of the particular commercial tool adopted. To this aim, we plan to analyze the Oracle Information Integrator tool.

References

[1] IBM websphere information integrator: Accessing and integrating diverse data for the on demand business. *IBM White Paper*, 2005.

[2] Information infrastructure: delivering the advantage of information integration today. *IBM White Paper*, 2005.

[3] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 68–79, 1999.

[4] Pablo Barceló and Leopoldo E. Bertossi. Logic programs for querying inconsistent databases. In *Proc. of PADL 2003*, pages 208–222, 2003.

[5] Paolo Bruni, Francis Arnaudies, Amanda Bennett, Susanne Englert, and Gerhard Keplinger. Data federation with IBM DB2 Information Integrator V8.1, 2003. Redbook, `http://www.redbooks.ibm.com/redbooks/pdfs/sg247052.pdf`.

[6] A. Calì, D. Lembo, R. Rosati, and M. Ruzzi. Experimenting data integration with disatdis. In *Conference on Avdanced Information System Engineering*, 2004.

[7] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of the 22nd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2003)*, pages 260–271, 2003.

[8] Andrea Calì, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 16–21, 2003.

[9] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Riccardo Rosati, and Guido Vetere. A framework for peer-to-peer data integration on grids. In *Proceedings of the International Conference on Semantics of a Networked World: Semantics for Grid Databases (ICSNW'04)*, 2004.

[10] Jan Chomicki and Jerzy Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In *Inconsistency Tolerance*, pages 119–150, 2005.

[11] Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. Hippo: a system for computing consistent query answers to a class of sql queries. In *Proc. of EDBT 2004*, pages 841–844, 2004.

[12] Ariel Fuxman, Elham Fazli, and Renée J. Miller. ConQuer: Efficient management of inconsistent databases. In *Proc. of SIGMOD 2005*, 2005. To Appear.

[13] Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. In *International Conference on Database Theory, ICDT*, 2005.

[14] L.M. Haas. A researcher's dream. *DB2 Magazine*, 8(3):34–40, 2003.

[15] L.M. Haas, E.T. Lin, and M.A. Roth. Data integration through database federation. *IBM Systems Journal*, 41(4):578–596, 2002.

[16] Richard Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, pages 51–61, 1997.

[17] L. Griecoand D. Lembo, R. Rosati, and M. Ruzzi. Consistent query answering under key and exclusion dependencies: algorithms and experiments. In *ACM Fourteenth Conference on Knowledge and Information Management, CIKM*, 2005.

[18] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, pages 233–246, 2002.

[19] N. Leone, G. Greco, G. Ianni, V. Lio, G. Terracina, T. Eiter, W. Faber, M. Fink, G. Gottlob, R. Rosati, D. Lembo, M. Lenzerini, M. Ruzzi, E. Kalka, B. Nowicki, and W. Stani. The infomix system for advanced integration of incomplete and inconsistent data. In *SIGMOD Conference*, 2005.

[20] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. on Computational Logic*, 2005. To appear.

[21] Alon Y. Levy. Logic-based techniques in data integration. In *Logic-based artificial intelligence*, pages 575–595. Kluwer Academic Publishers, 2000.

[22] Jayant Madhavan and Alon Y. Halevy. Composing mappings among data sources. In *Proc. of the 29th Int. Conf. on Very Large Data Bases (VLDB 2003)*, pages 572–583, 2003.

[23] A. Poggi and M. Ruzzi. Filling the gap between data integration and data federation. In *12th Italian Symposium on Advanced Database Systems, SEBD*, 2004.

[24] 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, 1997.