

Techniques for query reformulation, query merging, and information reconciliation – Part A

(FINAL, 22/05/2003)

Abstract – The purpose of this document is to define a general technique for query reformulation and query processing within one SINode in Sewasie. The structure of one SINode and the process of query reformulation is based on the idea that an SINode in Sewasie is considered as a data integration system framework with a global schema with key and foreign key constraints. Queries are posed in terms of the global schema, in particular as unions of conjunctive queries. Query processing is constituted by several steps. The first step is realized through an algorithm for expanding the query according the integrity constraints in the global schema. The second step materializes the global classes that are relevant for the query, based on the data stored in the local classes. The third step materializes the corresponding local classes, by accessing the relevant external sources. Finally, the fourth step evaluates the expanded query over the materialized global classes.

Contents

1	Executive summary	4
2	Query management in the context of SEWASIE	5
3	Query processing within one SINode: The general approach	5
3.1	The structure of an SINode	5
3.2	Semantics of an SINode	7
3.3	Queries posed to one SINode	8
3.4	Query processing: overview	8
4	Query expansion	9
4.1	Preliminary definitions	9
4.2	The algorithm FKrewrite	10
4.3	Soundness and completeness	11
4.4	Example	12
5	Global class materialization	13
5.1	Full Disjunction	16
5.2	Single Class Query Rewriting	17
6	Local class materialization	20
7	Evaluation of the expanded query	20
8	Conclusions	20

1 Executive summary

This document is the first part (Part A) of the deliverable D3.2: “Techniques for query reformulation, query merging, and information reconciliation”.

The goal of the document is to illustrate the basic strategy to answer queries posed to an SINode in the Sewasie architecture. The rest of the document is organized in 7 sections. In Section 2 we present a general description of query management in the context of Sewasie. Section 3 presents both the structure of an SINode, and the general overview of query processing within an SINode. The remaining sections illustrate the characteristics of the various steps of query processing. In particular, Section 4 describes the algorithm for expanding the query according the integrity constraints in the global schema, Section 5 and 6 illustrate the process of materializing the global classes, and the local classes, respectively, and Section 7 presents the method for evaluating the expanded query based on the materialized global classes.

While the present document concentrates on the task of query answering within one SINode, there are several aspects of query processing in Sewasie that are left out in the present analysis. These aspects will be the subject of the second part (part B) of this deliverable.

2 Query management in the context of SEWASIE

As we illustrated in [6], in Sewasie, the query agent is the carrier of the user query from the user interface to the SINodes, where concrete data are located. One basic task of the query agent is to interact with the brokering agents, in order to solve the query. Starting from a specified brokering agent, the query agent initiates a process constituted by a series of interactions with the various brokering agents, with the goal of getting information on the matter of interest from the SINodes managed by these brokering agents (see Section 6 of [4]).

A typical interaction between a query agent and a brokering agent may imply that the brokering agent will provide directions to relevant SINodes and information on SINode contents, or reference the query agent to other brokering agents. The query agent will then move to such SINodes and query them, or may move on to the other brokering agents to ask them for directions again. During this process, the query agent is informed by the brokering agents about which SINodes contain relevant data, so that the query agent may access such SINodes, collect partial answers and integrate them. It follows that the query agent should be able to:

1. interact with relevant brokering agents,
2. ask the various SINodes the right information, according to a specific query plan.

The problem of designing techniques to be used by the query agent in order to decide which brokering agents to contact (item 1 above), and how to interact with them, will be addressed in the second part (part B) of this deliverable. Such second part will be produced later on in the project, in particular at month 25, when the second release of the prototype will be delivered.

As for item (2) above, it is essential that the query asked to the SINode is answered correctly and completely by the SINode itself. This task is accomplished by the Query manager of the SINode. In this part (part A) of the deliverable, we will concentrate only on this task. In other words, in the rest of this document, we deal with the problem of answering a query posed to an SINode. As we said before, this task is carried out by the query manager of the SINode of interest, and is characterized as follows: Given a query posed in terms of the virtual global view associated to the SINode, derive a query plan that is able to correctly access the data sources under the control of that SINode, and execute the query according to this plan.

3 Query processing within one SINode: The general approach

A query issued to an SINode is managed by the query manager associated to such SINode. The **Query Manager** of an SINode is the coordinated set of functions which take an incoming query, define a decomposition of the query according with the mapping of the global virtual view of the SINode onto the specific data sources available and relevant for the query, sends the queries to the wrappers in charge of the data sources, collects their answers, performs any residual filtering as necessary, and finally delivers whatever is left to the requesting query agent.

For all the issues regarding the structure of the SINodes, especially those related to the representation of the ontology managed by SINodes, we refer the reader to [5]. In the next subsection, we provide an overview of the most important features related to query processing. We then specify the formal semantics of an SINode, and the notion of query posed to an SINode. Finally, we provide an overview of query processing within one SINode.

3.1 The structure of an SINode

We conceive a single SINode as a data integration system based on a so-called global schema. In other words, each SINode combines the data residing at different sources, and provide the external user with a unified view of these data. Such a unified view is represented by the global schema (also called Global Virtual View), and provides a reconciled view of all data, which can

be queried by the user. Obviously, one of the main task in the design of an SINode is to establish the mapping between the sources and the global schema. Such a mapping should be suitably taken into account in formalizing the notion of SINode.

It follows that, from the perspective of the query manager, the main components of an SINode are the global schema, the sources, and the mapping. Therefore, we formalize a SINode as follows.

Definition 1 An *SINode* \mathcal{I} is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where

- \mathcal{G} is the *global schema*, expressed in a language $\mathcal{L}_{\mathcal{G}}$ over an alphabet $\mathcal{A}_{\mathcal{G}}$.
- \mathcal{S} is the *source schema*, expressed in a language $\mathcal{L}_{\mathcal{S}}$ over an alphabet $\mathcal{A}_{\mathcal{S}}$. The alphabet $\mathcal{A}_{\mathcal{S}}$ includes a symbol for each source accessible by the SINode.
- \mathcal{M} is the *mapping* between \mathcal{G} and \mathcal{S} .

Next, we discuss how the three components are specialized in the context of Sewasie.

Global schema The global schema is expressed in terms of an object-oriented data model. However, such schema is translated into the relational model. Therefore, the language $\mathcal{L}_{\mathcal{G}}$ in our context is the relational model, and the alphabet $\mathcal{A}_{\mathcal{G}}$ is the set of global class names, with the obvious correspondence of one relation for each global class. More precisely, the language $\mathcal{L}_{\mathcal{G}}$ in our context is constituted by the relational model with two kinds of constraints:

- *Key constraints*: given a relation r in the schema, a key constraint over r is expressed in the form $key(r) = \mathbf{A}$, where \mathbf{A} is a set of attributes of r . Such a constraint is satisfied in a database \mathcal{DB} if for each $t_1, t_2 \in extr\mathcal{DB}$, with $t_1 \neq t_2$, we have $t_1[\mathbf{A}] \neq t_2[\mathbf{A}]$, where $t[\mathbf{A}]$ is the projection of the tuple t over \mathbf{A} . In the following, we assume that every relation in the global schema has one and only one key. Also, we assume without loss of generality, that the attributes constituting the key of a relation r are the first h attributes, i.e., $key(r) = \{1, \dots, h\}$, for an $h \leq arity(r)$.
- *Foreign key constraints*: a foreign key constraint is a statement of the form $r_1[\mathbf{A}] \subseteq r_2[\mathbf{B}]$, where r_1, r_2 are relations, \mathbf{A} is a sequence of distinct attributes of r_1 , and \mathbf{B} is $key(r_2)$, i.e., the sequence $[1, \dots, h]$ constituting the key of r_2 . Such a constraint is satisfied in a database \mathcal{DB} if for each tuple t_1 in $extr_1\mathcal{DB}$ there exists a tuple t_2 in $extr_2\mathcal{DB}$ such that $t_1[\mathbf{A}] = t_2[\mathbf{B}]$.

Source schema The source schema is again expressed in terms of the relational model. Therefore, the language $\mathcal{L}_{\mathcal{S}}$ in our context is the relational model, and the alphabet $\mathcal{A}_{\mathcal{S}}$ is the set of local class name, with the obvious correspondence of one relation for each local class. It is important to note that local classes are internal entities representing external sources, in the sense that each local class has a link with a corresponding external source, where external data are loacted. The correspondence between local classes and external sources is realized by means of a further layer (that we can call the **wrapping layer**), that associates to each local class a query over the external sources. In this part (part A) of the deliverable, we will not delve into the details of the wrapping layer. We will simply assume in the following that the wrapping layer is able to specify how to extract the data from the sources, so as to correctly populate the local classes. A more detailed description of the wrapping layer will be presented in the second part (part B) of this deliverable.

Mapping In Sewasie, the various SINodes follow the Global-as-view (GAV) approach. In the GAV approach, the mapping \mathcal{M} associates to each element g in \mathcal{G} a query $q_{\mathcal{S}}$ over \mathcal{S} . The following properties further characterize how the mapping is specified in Sewasie.

- To every global class C a set of local classes is associated. Only data coming from such local classes will be used to get instances for the class C .

- Sources are considered sound, which implies that the global classes may have more instances than those retrieved from the data of the local sources, according to the mapping.
- The query associated to C retrieves the data ensuring that, in every instance of the global schema, no two objects with the same value of the key of C exist. This property is particularly important, because it implies that the mapping incorporates data cleaning/transformation functions, that are activated when data are loaded into the global classes.

3.2 Semantics of an SInode

As for the semantics of an SInode, a database (DB) for a schema \mathcal{G} is simply a collection of sets of objects, one set for each global class in the alphabet of \mathcal{G} . Note that, since every global class has a key, there is an obvious correspondence between objects of a global class C and tuples of the relation represented by C : indeed, there is one object for every value of the key.

In order to assign semantics to an SInode $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we start by considering a *local database* for \mathcal{I} , i.e., a database \mathcal{D} that conforms to the local classes 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} .

Definition 2 A global database \mathcal{B} for \mathcal{I} is said to be *legal with respect to \mathcal{D}* , if:

- \mathcal{B} is legal with respect to \mathcal{G} , i.e., \mathcal{B} satisfies all the constraints of \mathcal{G} ;
- \mathcal{B} satisfies the mapping \mathcal{M} with respect to \mathcal{D} .

The notion of \mathcal{B} satisfying the mapping \mathcal{M} with respect to \mathcal{D} depends on the approach adopted for assigning meaning to the mapping. In the GAV approach, the one adopted in our context, the mapping \mathcal{M} associates to each element g in \mathcal{G} a query q_S over \mathcal{S} . Therefore, a GAV mapping is a set of assertions, one for each element g of \mathcal{G} , of the form

$$q_S \rightsquigarrow g$$

Definition 3 A database \mathcal{B} satisfies the assertion $q_S \rightsquigarrow g$ with respect to a local database \mathcal{D} if

$$q_S^{\mathcal{D}} \subseteq g^{\mathcal{B}}$$

where $q_S^{\mathcal{D}}$ is the result of evaluating the query q_S over the source database \mathcal{D} .

Roughly speaking, the logical characterization of GAV is therefore through the first order assertions

$$\forall \mathbf{x} \, q_S(\mathbf{x}) \rightarrow g(\mathbf{x})$$

It is interesting to observe that the implicit assumption in many GAV proposals is the one of exact views. Indeed, in a setting where all the views are exact, there are no constraints in the global schema, and a first order query language is used as $\mathcal{L}_{\mathcal{M},\mathcal{S}}$, a GAV data integration system enjoys what we can call the “single database property”, i.e., it is characterized by a single database, namely the global database that is obtained by associating to each element the set of tuples computed by the corresponding view over the sources. This motivates the widely shared intuition that query processing in GAV is easier than in LAV. However, it should be pointed out that the single database property only holds in such a restricted setting.

In particular, the possibility of specifying constraints in \mathcal{G} greatly enhances the modeling power of GAV systems, especially in those situations where the global schema is intended to be expressed in terms of a conceptual data model, or in terms of an ontology [1]. In these cases, the language $\mathcal{L}_{\mathcal{G}}$ is in fact sufficiently powerful to allow for specifying, either implicitly or explicitly, various forms of integrity constraints on the global database.

3.3 Queries posed to one SInode

Queries to \mathcal{I} are posed in terms of the global schema \mathcal{G} , and are expressed in a query language \mathcal{L}_Q over the alphabet \mathcal{A}_Q . A query is intended to provide the specification of which data to extract from the virtual database represented by the SInode. We will restrict our attention to unions of conjunctive queries. In other words, the language \mathcal{L}_Q in the context of Sewasie is the language of unions of conjunctive queries. Formally, a *conjunctive query* q of *arity* n is written in the rule-based form

$$q(x_1, \dots, x_n) \leftarrow \text{body}(x_1, \dots, x_n, y_1, \dots, y_m)$$

where:

- q belongs to a new alphabet Q (the alphabet of queries),
- $\text{body}(x_1, \dots, x_n, y_1, \dots, y_m)$ is a conjunction of atoms involving the variables $x_1, \dots, x_n, y_1, \dots, y_m$, and a set of constants from Γ , and
- the predicate symbols of the atoms are in \mathcal{G} .

Note that, since all variables x_1, \dots, x_n in the head appear also in the body, we are dealing with *safe* conjunctive queries.

We now specify the semantics of queries posed to an SInode. As we said before, such queries are expressed in terms of the symbols in the global schema of \mathcal{I} .

Definition 4 If q is a query of arity n and \mathcal{DB} is a database, we denote with $q^{\mathcal{DB}}$ the set of tuples (of arity n) in \mathcal{DB} that satisfy q .

Definition 5 Given a local database \mathcal{D} for \mathcal{I} , the answer $q^{\mathcal{I}, \mathcal{D}}$ to a query q with respect to \mathcal{I} and \mathcal{D} , is the set of tuples t such that $t \in q^{\mathcal{B}}$ for every global database \mathcal{B} that is legal for \mathcal{I} with respect to \mathcal{D} , i.e. such that t is an answer to q over every database \mathcal{B} that is legal for \mathcal{I} with respect to \mathcal{D} . The set $q^{\mathcal{I}, \mathcal{D}}$ is called the set of *certain answers* to q in with respect to \mathcal{I} and \mathcal{D} .

Note that, from the point of view of logic, finding certain answers is a logical implication problem: check whether it logically follows from the information on the sources that t satisfies the query.

The answer to a conjunctive query q of arity n over a database \mathcal{B} for \mathcal{G} , denoted $q^{\mathcal{B}}$, is the set of n -tuples of constants (c_1, \dots, c_n) , such that, when substituting each c_i for x_i , the formula

$$\exists(y_1, \dots, y_m). \text{body}(x_1, \dots, x_n, y_1, \dots, y_m)$$

evaluates to true in \mathcal{DB} . Note that the answer to q over \mathcal{B} is a relation whose arity is equal to the arity of the query q . Finally, the answer to a union of conjunctive queries over a database \mathcal{B} for \mathcal{G} is the union of the answers to the component conjunctive queries.

3.4 Query processing: overview

Query processing within one SInode is constituted by the following phases:

1. **Query Expansion:** the query is expanded to take into account the integrity constraints in the global schema.
2. **Global class materialization:** the atoms in the expanded query are materialized by taking into account the mapping to the local classes.
3. **Local class materialization:** in order to materialize the global classes, another materialization step is required, namely the one that retrieves the data from the external sources, and stores them into the local classes.
4. **Evaluation of the expanded query:** when all the global classes that are relevant for the query are materialized, the expanded query is submitted to the SQL Engine to obtain the answer of the original query.

The following sections are devoted to explaining the main features of the above described steps of our query processing strategy.

4 Query expansion

Query expansion amounts to rewriting the query Q posed to a global schema \mathcal{G} into a new query Q' , in such a way that all the knowledge about the constraints in \mathcal{G} has been “compiled” into Q' . The goal of the rewriting step in our approach is made explicit in the following definition. The definition makes use of the notion of retrieved global database.

Definition 6 If \mathcal{D} is a local database for the data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, then the **retrieved global database** $\mathcal{M}(\mathcal{D})$ is the global database obtained by computing, for every element of \mathcal{G} , the query associated to it by \mathcal{M} over the local database \mathcal{D} .

We remind the reader that, according to what we said before, we assume that, for every \mathcal{D} , the retrieved global database $\mathcal{M}(\mathcal{D})$ satisfies all key constraints in \mathcal{G} .

Definition 7 Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system of an SINode, and let Q be a query (union of conjunctive queries) over \mathcal{G} . Then Q_1 is called a **perfect rewriting of Q wrt \mathcal{I}** if, for every local database \mathcal{D} , $Q^{\mathcal{I}, \mathcal{D}} = Q_1^{\mathcal{M}(\mathcal{D})}$.

In this section we briefly recall from [2, 3] the main properties of an algorithm that computes a perfect rewriting of a query Q wrt \mathcal{I} , or, simply, wrt to a global schema \mathcal{G} . The above definition states that the algorithm rewrites a query expressed over the global schema of an SINode into a new query in such a way that the result of the expanded query over a schema obtained by the original global schema by ignoring the integrity constraints, is the same as the result of the original query over the original global schema. The algorithm is described in more detail in [3].

As we said before, we assume that the mapping is defined in such a way that no key constraint is violated when we load data from the local classes to the global classes. It follows that key constraints are taken into account by the global class materializer, in particular through the cleaning strategy it implements. It follows that the only integrity constraints we should consider in the following are only the foreign key constraints.

4.1 Preliminary definitions

Given a conjunctive query q , we say that a variable X is *unbound* in q if it occurs only once in q , otherwise we say that X is *bound* in q . Notice that variables occurring in the head of the query are necessarily bound, since each of them must also occur in the query body. A *bound term* is either a bound variable or a constant.

Following the standard notation used in deductive databases, we adopt a special symbol for all unbound variables in the query q : specifically, we use the special term “ ξ ” for such variables (deductive database systems use the symbol “ $_$ ” to this purpose).

Definition 8 Given an atom $g = s(X_1, \dots, X_n)$ and a foreign key constraint (FK) $I = r[i_1, \dots, i_k] \subseteq s[j_1, \dots, j_k]$, we say that I is *applicable to g* if the following conditions hold:

1. for each ℓ such that $1 \leq \ell \leq n$, if $X_\ell \neq \xi$ then there exists h such that $j_h = \ell$;
2. there exists a variable substitution σ such that $\sigma(s[j_h]) = \sigma(s[j_{h'}])$ for each h, h' such that $i_h = i_{h'}$. If such a σ exists, we denote with $\sigma_{g, I}$ the most general substitution that verifies this condition.

Moreover, if I is applicable to g , we denote with $gr(g, I)$ the atom $r(Y_1, \dots, Y_m)$ (m is the arity of r in Ψ) where for each ℓ such that $1 \leq \ell \leq m$, $Y_\ell = \sigma_{g, I}(X_{j_h})$ if there exists h such that $i_h = \ell$, otherwise $Y_\ell = \xi$.

Roughly speaking, a FK I is applicable to an atom g if the relation symbol of g corresponds to the symbol in the right-hand side of I and if all the attributes for which bound terms appear in g are propagated by the FK I (condition 1 of the above definition). Moreover, if the left-hand side of the FK contains repeated attributes, then the FK is applicable only if the corresponding terms in

g unify (condition 2 of the above definition). When I is applicable to g , $gr(g, I)$ denotes the atom obtained from g by using I as a rewriting rule, whose direction is right-to-left.

For instance, let $I = r[1, 2] \subseteq s[1, 3]$, $g = s(X, \xi, c)$ and let r be of arity 4. I is applicable to g , since the attributes of s which contain bound terms (i.e., attributes 1 and 3) are propagated by the FK, and $gr(g, I) = r(X, c, \xi, \xi)$. As another example, consider now $I = r[1, 1] \subseteq s[1, 3]$, $g = s(X, \xi, c)$ and let r be of arity 4. I is applicable to g since the terms X and c unify and $\sigma_{g,I} = \{X \rightarrow c\}$. Therefore, $gr(g, I) = r(c, c, \xi, \xi)$.

Definition 9 Given an atom $g_1 = r(X_1, \dots, X_n)$ and an atom $g_2 = r(Y_1, \dots, Y_n)$, we say that g_1 and g_2 unify if there exists a variable substitution σ such that $\sigma(g_1) = \sigma(g_2)$. Each such a σ is called *unifier*. Moreover, if g_1 and g_2 unify, we denote as $\mathcal{U}(g_1, g_2)$ a *most general unifier (mgu)* of g_1 and g_2 (we recall that σ is a mgu if for every unifier σ' of g_1 and g_2 there exists a substitution γ such that $\sigma' = \gamma\sigma$).

Informally, two atoms unify if they can be made equal through a substitution of the variable symbols with other terms (either variables or constants) occurring in the atoms. We recall that each occurrence of ξ in the two atoms actually represents a different, new variable symbol, therefore each occurrence of ξ in g_1 and g_2 can be assigned to a different term.

For example, let $g_1 = r(X, \xi, c, \xi)$ and an atom $g_2 = r(\xi, Y, Z, \xi)$. Actually, g_1 and g_2 correspond to $g_1 = r(X, \xi_1, c, \xi_2)$, $g_2 = r(\xi_3, Y, Z, \xi_4)$. Then, g_1 and g_2 unify, and $\mathcal{U}(g_1, g_2) = \{\xi_3 \rightarrow X, \xi_1 \rightarrow Y, Z \rightarrow c, \xi_2 \rightarrow \xi_4\}$.

4.2 The algorithm FKrewrite

In Figure 1 we define the algorithm FKrewrite that computes the rewriting of a UCQ Q with respect to a set of foreign keys. In the algorithm, it is assumed that unbound variables in the input query Q are represented by the symbol ξ . Also, given a global schema \mathcal{G} , we denote with Ψ the schemas of relations in \mathcal{G} , and with Σ_I the foreign key constraints in \mathcal{G} .

Specifically, the algorithm generates a set of conjunctive queries that constitute a perfect rewriting of Q , by computing the closure of the set Q with respect to the following two rules:

1. if there exists a conjunctive query $q \in Q$ such that q contains two atoms g_1 and g_2 that unify, then the algorithm adds to Q the conjunctive query $\text{reduce}(q, g_1, g_2)$ obtained from q by the following algorithm:

Algorithm $\text{reduce}(q, g_1, g_2)$

Input: conjunctive query q , atoms $g_1, g_2 \in \text{body}(q)$ such that g_1 and g_2 unify

Output: reduced conjunctive query q'

$q' = q$;

$\sigma \leftarrow \mathcal{U}(g_1, g_2)$;

$\text{body}(q') \leftarrow \text{body}(q) - \{g_2\}$;

$q' \leftarrow \sigma(q')$;

$q' \leftarrow \tau(q')$;

return q'

Informally, the above algorithm reduce starts by eliminating g_2 from the query body; then, the substitution $\mathcal{U}(g_1, g_2)$ is applied to the whole query (both the head and the body), and finally the function τ is applied. Such a function replaces with ξ each variable symbol X such that there is a single occurrence of X in the query. The use of τ is necessary in order to guarantee that, in the generated query, each unbound variable is represented by the symbol ξ .

2. if there exists a FK I and a conjunctive query $q \in Q$ containing an atom g such that I is applicable to g , then the algorithm adds to Q the query obtained from q by replacing g

Algorithm FKrewrite(Ψ, Σ_I, Q)
Input: global schema \mathcal{G} with relational schema Ψ , and foreign key dependencies Σ_I , UCQ Q
Output: perfect rewriting of Q
 $Q' \leftarrow Q$;
repeat
 $Q_{aux} \leftarrow Q'$;
 for each $q \in Q_{aux}$ **do**
 (a) **for each** $g_1, g_2 \in \text{body}(q)$ **do**
 if g_1 and g_2 unify
 then $Q' \leftarrow Q' \cup \{\text{reduce}(q, g_1, g_2)\}$;
 (b) **for each** $g \in \text{body}(q)$ **do**
 for each $I \in \Sigma_I$ **do**
 if I is applicable to g
 then $Q' \leftarrow Q' \cup \{\text{atom-rewrite}(q, g, I)\}$
until $Q_{aux} = Q'$;
return Q'

Figure 1: Algorithm FKrewrite

with $gr(g, I)$ and by applying the substitution $\sigma_{g, I}$ to q . Namely, this step adds new queries obtained by applying FKs as rewriting rules (applied from right to left), through the following algorithm atom-rewrite:

Algorithm atom-rewrite(q, g, I)
Input: conjunctive query q , atom $g \in \text{body}(q)$, FK I such that I is applicable to g
Output: rewritten conjunctive query q'
 $q' = q$;
 $\text{body}(q') \leftarrow \text{body}(q) - \{g\}$;
 $q' \leftarrow \sigma_{g, I}(q')$;
 $\text{body}(q') \leftarrow \text{body}(q') \cup \{gr(g, I)\}$;
return q'

The above two rules correspond respectively to steps (a) and (b) of the algorithm.

Termination of the algorithm is immediately implied by the fact that the number of conjunctive queries that can be generated by the algorithm is finite, since the maximum length (i.e., the number of atoms) of a generated query is equal to the maximum length of the queries in Q , and the number of different atoms that can be generated by the algorithm is finite, since the alphabet of relation symbols used is finite (and corresponds to the relation symbols occurring in Q and in Σ_I), as well as the set of terms used (corresponding to the set of variable names occurring in the query Q plus the symbol ξ).

4.3 Soundness and completeness

The correctness (soundness and completeness of the algorithm FKrewrite is sanctioned by the following theorem.

Theorem 10 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system of an SINode, let Q be a query (union of conjunctive queries) over \mathcal{G} , and let Ψ the relational schema of \mathcal{G} , and Σ_I the foreign key constraints of \mathcal{G} . Then FKrewrite(Ψ, Σ_I, Q) is a perfect rewriting Q wrt \mathcal{I} .*

What the above theorem says is that evaluating the original query over the retrieved global database, taking into account the foreign key dependencies in the global schema, is equivalent to evaluating the expanded query over the same database without considering the foreign key dependencies.

4.4 Example

We present an example of query expansion. We first describe the global schema in the relational form. Such a schema results from the translation of a corresponding object-oriented schema. The relational schema is the following (in the expression of a foreign key constraint, numbers indicate the positions of the attributes):

```

Category(CatCode,Description,Sector)
Enterprise(Name,Description,Address,PhoneNo,Email,Web,Contact)
BusinessOrganization(Name,Province,BuiltYear,NoEmployees,Turnover)
BusinessOrganizationCat(Name,CatCode)
Manufacturer(Name,Owners,WorkingHours)

KeyConstraint(Category,1)
KeyConstraint(Enterprise,1)
KeyConstraint(BusinessOrganization,1)
ForeingKey(BusinessOrganization,1,Enterprise,1)
KeyConstraint(BusinessOrganizationCat,1)
ForeignKey(BusinessOrganizationCat,1,BusinessOrganization,1)
ForeignKey(BusinessOrganizationCat,2,Category,1)
KeyConstraint(Manufacturer,1)
ForeignKeyInclusion(Manufacturer,1,BusinessOrganizationCat,1)

```

The input query aims at retrieving name and addresses of enterprises whose category has a sector named “IT” (information technology). The SQL version of the query is:

```

SELECT e.Name, e.Address
FROM Enterprise e, BusinessOrganization b, BusinessOrganizationCat c, Category d
WHERE e.Name=b.Name and e.Name=c.Name and b.Name=c.Name and c.CatCode=d.CatCode
and d.Sector='IT';

```

The Datalog version is:

```

Q(X13,X3):-BusinessOrganization(X13,X9,X10,X11,X12),
           BusinessOrganizationCat(X13,X15),
           Category(X15,X16,"IT"),Enterprise(X13,X2,X3,X4,X5,X6,X7).

```

The result of the expansion of the query is the following Datalog query:

```

Q(X13,X3):-BusinessOrganization(X13,_,_,_,_),BusinessOrganizationCat(X13,X15),
           Category(X15,_, "IT"),Enterprise(X13,_,X3,_,_,_,_).
Q(X13,X3):-BusinessOrganizationCat(X13,X15),BusinessOrganizationCat(X13,_,_),
           Category(X15,_, "IT"),Enterprise(X13,_,X3,_,_,_,_).
Q(X13,X3):-BusinessOrganizationCat(X13,X15),Category(X15,_, "IT"),
           Enterprise(X13,_,X3,_,_,_,_),Manufacturer(X13,_,_).
Q(X13,X3):-BusinessOrganizationCat(X13,X15),Category(X15,_, "IT"),
           Enterprise(X13,_,X3,_,_,_,_).

```

It should be noted how the query expansion step has “compiled” all the foreign key constraints in the new query. As we said before, the expanded query can now be processed by ignoring the constraints in the schema. The result on the correctness of the expansion process ensures us that we do not loose any information by doing so.

5 Global class materialization

As we said in Section 3, the expanded query is processed in order to single out the global classes that need to be materialized in order to answer the query. This task is simple, and we do not discuss it in any detail. We simply assume in the following that the list of global classes to be materialized is available, and we illustrate the process in the case of the example reported in the previous section. In such example, the global classes to be materialized, together with the query to be issued on such classes for the materialization, is shown in the following (each query is given a name to be used in the final process of query evaluation):

Global Class Query OurRelation_00:

```
SELECT R.Name, R.Province, R.BuiltYear, R.NoEmployees, R.Turnover
FROM BusinessOrganization R;
```

Global Class Query OurRelation_01:

```
SELECT R.Name, R.CatCode
FROM BusinessOrganizationCat R;
```

Global Class Query OurRelation_02:

```
SELECT R.CatCode, R.Description, R.Sector
FROM Category R
WHERE (R.Sector = "IT");
```

Global Class Query OurRelation_03:

```
SELECT R.Name, R.Description, R.Address, R.PhoneNo, R.Email, R.Web, R.Contact
FROM Enterprise R;
```

Global Class Query OurRelation_20:

```
SELECT R.Name, R.Owners, R.WorkingHours
FROM Manufacturer R;
```

In the rest of this section, we concentrate on the issue of materializing the global classes of interest. In particular, we illustrate how we materialize the global classes that are necessary for answering the query. The global class materialization is a sort of mediator whose goal is to retrieve the data satisfying the global class, based on the data stored in the local class. Such mediator assumes that the relevant local classes are already materialized (see the section on the local materializer).

We denote with \mathcal{G} a global class with schema $S(\mathcal{G})$, with \mathcal{L} the set of local classes of \mathcal{G} and with $L \in \mathcal{L}$ a local class. Each local class schema, denoted by $S(L)$, is the set of information available in the class L , represented as a set of attributes. Finally, we introduce \mathcal{M} as the mapping between the global schema and local schemata: $\mathcal{M}[A, L]$ denotes the mapping of information $A \in S(\mathcal{G})$ in the local class L with value *null* if A is not mapped in L ¹. The mapping satisfies the GAV approach: for each $A \in S(\mathcal{G})$ exists at least one $L \in \mathcal{L}$ such that $\mathcal{M}[A, L] \neq \text{null}$. Given the local class $L \in \mathcal{L}$, we define the schema of L w.r.t. \mathcal{G} , denoted by $S^{\mathcal{G}}(L)$, as $S^{\mathcal{G}}(L) = \{A \in S(\mathcal{G}) \mid \mathcal{M}[A, L] \text{ is not null}\}$.

An SINode represents the available knowledge about the real world objects instantiated in the local sources. We denote with \mathcal{O} the set of real world objects (*objects* in the following) available at mediator level and we introduce a function \mathcal{E} , named *extension* ($\mathcal{E} : \mathcal{L} \rightarrow 2^{\mathcal{O}}$), associating each local class L in \mathcal{L} with the subset of \mathcal{O} represented in L .

We investigate our techniques by adopting as reference data model for schema and query specification the relational model. Thus, each local class L is a relation name with schema $S(L)$

¹We introduce no limitation about the global class mapping mechanism. We only require that it is always possible to state if $\mathcal{M}[A, L]$ is *null* or not.

and ℓ denotes a local relation. Given an extension \mathcal{E} , we say that ℓ is *legal* w.r.t. \mathcal{E} iff it exists a bijective function $t_L: \mathcal{E}(L) \rightarrow \ell$. Thus, each object o instantiated in L is represented by the tuple $t_L(o)$ and ℓ corresponds to $\{t_L(o) \mid o \in \mathcal{E}(L)\}$.

For each local class L , we consider an unary operator $(r)_L^{\mathcal{G}}$, that, given a relation ℓ with schema $S(L)$ produces a relation $\ell^{\mathcal{G}} = (r)_L^{\mathcal{G}}$ with schema $S^{\mathcal{G}}(L)$ obtained by²

1. (*Schema Translation*) renaming the attributes of r into attributes of \mathcal{G} by means of the mapping \mathcal{M} (for example, in Figure 2: `firstn` and `lastn` to `Name`);
2. (*Data conversion*) converting the tuples of r into tuple of $r^{\mathcal{G}}$ by suitable functions such as *string concatenation* (for example, in Figure 2: `'Rita' + 'Verde'` to `'Rita Verde'`).

A tuple of $\ell^{\mathcal{G}}$ will be denote with $t_L^{\mathcal{G}}(o)$. Then, given the local class set $\mathcal{L} = \{L_1, \dots, L_n\}$, we introduce the notion of *sources database* legal w.r.t. \mathcal{E} as $db = \langle \ell_1^{\mathcal{G}}, \dots, \ell_n^{\mathcal{G}} \rangle$.

Example 11 As an example, let us consider two relation L_1, L_2 with schema

$$\begin{array}{l} S(L_1) = (\text{firstn}, \text{lastn}, \text{year}, \text{e_mail}) \\ S(L_2) = (\text{name}, \text{e_mail}, \text{dept_code}, \text{s_code}) \end{array}$$

and the global class G with the following mapping table:

	Name	E_mail	Section	Year	Dept
L_1	firstn and lastn	e_mail	null	year	null
L_2	name	e_mail	s_code	null	dept_code

$S(G) = (\text{Name}, \text{E_mail}, \text{Section}, \text{Year}, \text{Dept})$

$S^{\mathcal{G}}(L_1) = (\text{Name}, \text{E_mail}, \text{Year})$

$S^{\mathcal{G}}(L_2) = (\text{Name}, \text{E_mail}, \text{Dept}, \text{Section})$

Figure 2 shows an extension \mathcal{E} , $\mathcal{E}(L_1) = \{o, o_1\}$, $\mathcal{E}(L_2) = \{o, o_2\}$, and the related instances ℓ_1 and ℓ_2 . By applying the operator $(r)_L^{\mathcal{G}}$ to these instances, we obtain $\ell_1^{\mathcal{G}}$ and $\ell_2^{\mathcal{G}}$.

As far as different instantiations of the same object is concerned, we do not address the problem of resolving conflicts that arise when grouping together information about the same real world object [14, 9, 12]. We suppose local classes to satisfy the *semantic homogeneity property* stating that tuples referring to the same object instantiated in different local classes never mismatch. More formally, given an extension \mathcal{E} , we say that a sources database $db = \langle \ell_1^{\mathcal{G}}, \dots, \ell_n^{\mathcal{G}} \rangle$ legal w.r.t. \mathcal{E} is *semantically homogeneous* if for all class pairs (L_1, L_2) , for each $o \in \mathcal{E}(L_1) \cap \mathcal{E}(L_2)$ and for each $A \in S^{\mathcal{G}}(L_1) \cap S^{\mathcal{G}}(L_2)$, it follows that

$$t_{L_1}^{\mathcal{G}}(o)[A] = t_{L_2}^{\mathcal{G}}(o)[A]$$

As to the global class \mathcal{G} , we consider \mathcal{G} as a relation name with schema $S(G)$. Given an extension \mathcal{E} and a semantically homogeneous sources database $db = \langle \ell_1^{\mathcal{G}}, \dots, \ell_n^{\mathcal{G}} \rangle$ legal w.r.t. \mathcal{E} , we derive the *global relation* g in the following way: $g = \{t_G(o) \mid o \in \mathcal{O}\}$ where $t_G(o)$ is obtained by applying the semantic homogeneity property:

$$\forall A \in S(G) : t_G(o)[A] = \begin{cases} t_L^{\mathcal{G}}(o)[A] & \text{if } \exists L \in \mathcal{L} : A \in S^{\mathcal{G}}(L), o \in \mathcal{E}(L) \\ \text{null} & \text{otherwise} \end{cases} \quad (1)$$

Such a global relation g will be called legal w.r.t. \mathcal{E} . For instance, the global relation g related to the example of Figure2, is shown in Figure 3.

An important task of a mediator is to perform *object fusion*, that is grouping together information about the same real- object stored in different data sources. We assume that there exists a

²From a functional point of view, the transformation $(r)_L^{\mathcal{G}}$ is implemented in the wrapper related to the source L .

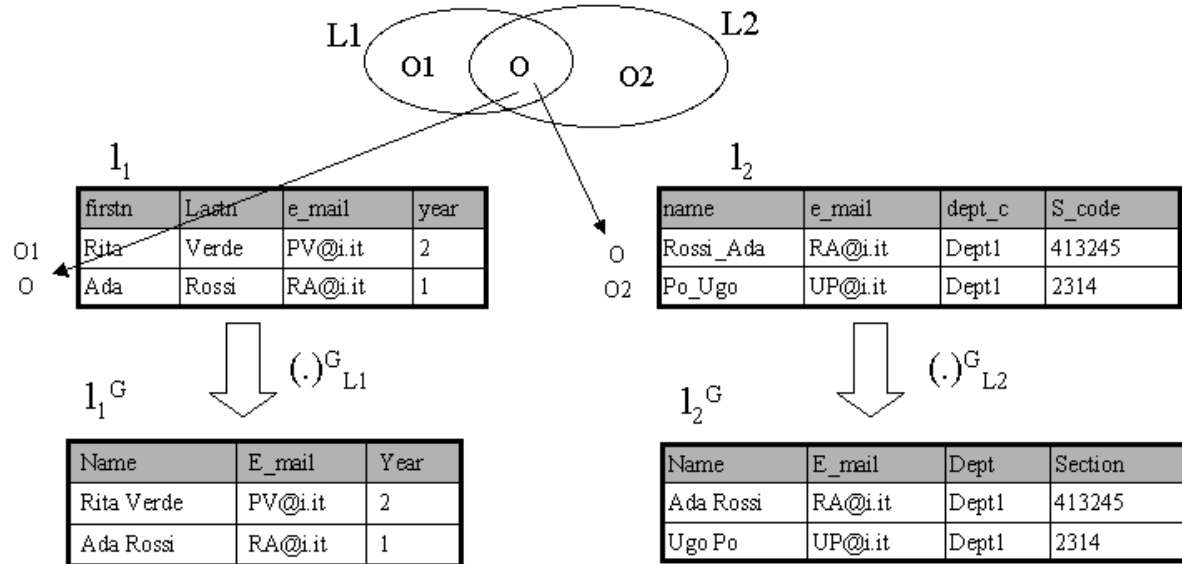


Figure 2: An example of sources database

Name	E_mail	Year	Dept	Section
Rita Verde	pv@i.it	2	<i>null</i>	<i>null</i>
Ada Rossi	ra@i.it	1	Dept1	12345
Ugo Po	up@i.it	<i>null</i>	Dept1	2345

Figure 3: An example of global relation

join condition between each pair of overlapping relations in order to identify tuples corresponding to the same object and fuse them. The join condition is defined on $S(\mathcal{G})$ by the designer. As an example, let us suppose that the *Join Map* between the classes L_1 and L_2 is $L_1.Name=L_2.Name$; given $o_i \in \mathcal{E}(L_1)$ and $o_j \in \mathcal{E}(L_1)$ we have $t_{L_1}^{\mathcal{G}}(o_i)[Name] = t_{L_2}^{\mathcal{G}}(o_j)[Name]$ if and only if o_i and o_j are the same real-object: $o_i \equiv o_j$.

Definition 12 [Join Condition] Let \mathcal{G} a global class and \mathcal{L} its set of local classes. A *join condition* between $L_1, L_2 \in \mathcal{L}$ is a condition ϕ expressed on $S^{\mathcal{G}}(L_1) \cup S^{\mathcal{G}}(L_2)$ such that $\forall \mathcal{E}, \forall t_1 \in \ell_1^{\mathcal{G}}, \forall t_2 \in \ell_2^{\mathcal{G}}, \phi(t_1 t_2)$ is true iff $\exists o \in \mathcal{E}(L_1) \cap \mathcal{E}(L_2)$ such that $t_1 = t_{L_1}(o)$ and $t_2 = t_{L_2}(o)$.

The set of join conditions for all pairs of local classes of the global class \mathcal{G} , called *Join Map*, is denoted with JM ; a *join condition* between $L_1, L_2 \in \mathcal{L}$ will be denoted with $JM(L_1, L_2)$. Given a local class L , the set of attributes in $S^{\mathcal{G}}(L)$ used to express the join conditions for L is called *Join Attribute Set* and it is denoted with $JA(L)$.

In the following we discuss how to compute the global relation g starting from a sources database $db = \langle \ell_1^{\mathcal{G}}, \dots, \ell_n^{\mathcal{G}} \rangle$. This problem is related to that of computing the natural outerjoin of many relations in a way that preserves all possible connections among facts [13]. Such a computation has been termed a “full disjunction” by Galindo Legaria [10].

5.1 Full Disjunction

In this section we consider the definition of Full Disjunction given in [10], sections **1.2 Basic Definitions** and **2.1 Join disjunctive queries**; the final version of this deliverable we will include these sections.

Definition 13 [Full Disjunction] The *full disjunction* of a query graph G is given by the join disjunctive query $Q = \{V' \mid G \upharpoonright_{V'} \text{ is connected}\}$.

A full disjunction delivers all tuples of the base relations, combining them as a single tuple whenever they match; full disjunctions capture the requirements of data merging queries.

Now, we apply this definition in our context. Let \mathcal{G} be a global class with schema $S(\mathcal{G})$, \mathcal{L} the set of local classes of \mathcal{G} and JM a Join Map of \mathcal{G} . Given an extension \mathcal{E} and a semantically homogeneous sources database $db = \langle \ell_1^{\mathcal{G}}, \dots, \ell_n^{\mathcal{G}} \rangle$ legal w.r.t. \mathcal{E} , we define the *query graph* of \mathcal{G} as $Q_{\mathcal{G}} = (V, E)$, where $V = \{\ell_1^{\mathcal{G}}, \dots, \ell_n^{\mathcal{G}}\}$, i.e., the nodes are the relations of the sources database, and $E = JM$, i.e., the edge are the join conditions between local classes. Then, we consider the *full disjunction*, denoted with fd , of $Q_{\mathcal{G}}$.

From the definition of full disjunction, it follows that the schema of the relation fd is $S^{\mathcal{G}}(L_1) \times S^{\mathcal{G}}(L_2) \times \dots \times S^{\mathcal{G}}(L_n)$. On the basis of the semantic homogeneity property we have that:

$$\forall t \in fd, \forall A \in S^{\mathcal{G}}(L_i) \cap S^{\mathcal{G}}(L_j) \text{ if } t[L_i.A] \neq null \wedge t[L_j.A] \neq null \text{ then } t[L_i.A] = t[L_j.A].$$

It can be shown that this property holds for each Join Map JM of \mathcal{G} .

Then we derive from fd a relation with schema $S(\mathcal{G})$, which will be called *global full disjunction* and will be denote by $GFD(\ell_1^{\mathcal{G}}, \dots, \ell_n^{\mathcal{G}})$ as follows:

$\forall t \in fd$, a tuple t' of $GFD(\ell_1^{\mathcal{G}}, \dots, \ell_n^{\mathcal{G}})$ is defined by:

$$\forall A \in S(\mathcal{G}), \quad t'[A] = \begin{cases} t[L.A] & \text{if } \exists L \in \mathcal{L} : A \in S^{\mathcal{G}}(L) \wedge t[L.A] \neq null \\ null & \text{otherwise} \end{cases} \quad (2)$$

It can be shown that

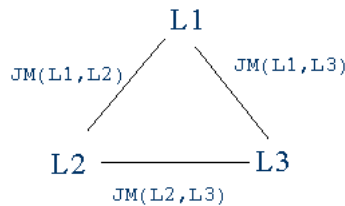
$$g = GFD(\ell_1^{\mathcal{G}}, \dots, \ell_n^{\mathcal{G}})$$

where g is the global relation legal w.r.t. \mathcal{E} as defined by equation 1.

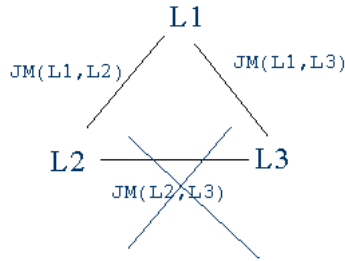
In [13, 10] is discussed when a full disjunction can be computed by some sequence of outerjoins. The answer to this question is: there is a outerjoin sequence producing the full disjunction if and only if the set of relation schemas forms a connected, γ -acyclic hypergraph [8].

It can be shown [11] that, in the general case, a global class with n local classes, $n > 2$ corresponds to a γ -cyclic hypergraph; then, we can't directly apply the methods proposed in [13, 10] in order to compute the Full Disjunction. In the following, we show these results by some examples.

For example, with $n = 3$, the query graph $Q_{\mathcal{G}}$ of the global class \mathcal{G} with $\mathcal{L} = \{L_1, L_2, L_3\}$, is the following:



Intuitively, this $Q_{\mathcal{G}}$ is cyclic. We obtain an acyclic query graph if one or more join condition is false; for example, if $JM(L3, L2)$ is equal to false, we have:



The join condition between two local classes L_i and L_j is false iff the that local classes L_i and L_j are extensionally disjoint, that is they share no real world object. In [7] we discussed how the introduction of extensional knowledge as an *additional mapping knowledge* allowing semantic optimization during the query planning phase.

Then, in the general case, we need to develop new methods to compute Full Disjunction. The proposed method to compute the full disjunction of n local classes L_1, L_2, \dots, L_n of a global class \mathcal{G} , is still based on outerjoin:

```
select *
from (L1 outer join L2 on JM(L1,L2))
     outer join L3 on (JM(L1,L3)) OR JM(L2,L3))
     ...
     outer join Ln on (JM(L1,Ln)) OR JM(L2,Ln) OR ... JM(Ln-1,Ln))
```

This query computes the full disjunction of L_1, L_2, \dots, L_n , independently by the order of the relations used in the `from` clause. For example, with $n = 3$:

```
select *
from (L1 outer join L2 on JM(L1,L2))
     outer join L3 on (JM(L1,L3)) OR JM(L2,L3))
```

The main problem/limitation of this method is the execution cost of the SQL query: in fact, the selectivity factor of the outer join conditions are low since in these condition we have the logical OR of one or more clauses; modifications and extensions to this method will be investigated in the project.

5.2 Single Class Query Rewriting

We assume a global SQL-like query expression over the global schema \mathcal{G} having the form

$$\text{select } AL \text{ from } \mathcal{G} \text{ where } Q \quad (3)$$

where $AL \subseteq S(\mathcal{G})$ is an attribute list and Q is query condition specified as a boolean expression of positive *atomic predicate* having the form $(A \text{ op } value)$ or $(A \text{ op } A')$, with $A, A' \in S(\mathcal{G})$.

For example, by considering the following query:

```
select Name from G
where Name like 'P*' and (Year='1' or Dept='Dept1')
```

we have $AL = \{Name\}$ and $Q = \text{Name like 'P*' and (Year='1' or Dept='Dept1')}$.

Given an extension \mathcal{E} , a semantically homogeneous sources database $db = \langle \ell_1^{\mathcal{G}}, \dots, \ell_n^{\mathcal{G}} \rangle$ legal w.r.t. \mathcal{E} , and a global relation g legal w.r.t. \mathcal{E} , we define the answer to a query expression having the form of 3 w.r.t. a global relation g as

$$\pi_{AL}(\sigma_Q(g))$$

Since $g = GFD(\ell_1^G, \dots, \ell_n^G)$ we can write

$$\pi_{AL}(\sigma_Q(GFD(\ell_1^G, \dots, \ell_n^G)))$$

In our context, the query rewriting problem consists into rewrite this expression into an equivalent form

$$\pi_{AL}(\sigma_{Q_r}(GFD((lq_1)_{L_1}^G, \dots, (lq_n)_{L_n}^G)))$$

where

- $lq_i = \pi_{AL_i}(\sigma_{LQ_i}(\ell_i))$ is the answer to the local query for the local class L_i
- Q_r is the residual condition.

Since a SNode is not the owner of the data, the local queries are sent and execute on local sources; then, in order to reduce the size of the the local query answer lq_i , it is important: (1) to maximize the selectivity of the local query condition LQ_i and (2) to minimize the cardinality of the local query select-list AL_i . For the query rewriting method shown in the following, both these properties hold.

The steps to compute LQ_i , $1 \leq i \leq n$, and Q_r are the following:

1. Query normalization

The first step is to convert Q into a Disjunctive Normal Form (DNF) query Q^d where the predicates are atomic. The DNF query will be of the form $Q^d = C_1 \vee C_2 \vee \dots \vee C_m$, where each conjunction term C_i has the form $P_1 \wedge P_2 \wedge \dots \wedge P_n$, i.e., conjunction of atomic predicates.

In the example, we have:

$$Q^d = (\text{Name like 'P*' and Year='1'}) \text{ or } (\text{Name like 'P*' and Dept='Dept1'})$$

2. Local Query condition LQ_i

In this step, each atomic predicate P in Q^d is rewritten into one that can be supported by the local class L .

An atomic predicate P is *searchable* in the local class L if the global attributes used in P are present in the local class L , i.e., the global attributes are mapped into L with a non null mapping.

We make the hypothesis that each atomic predicate P searchable in L is fully expressible/supported in L , i.e., it exists a query condition Q_L^P expressed w.r.t. the local class schema $S(L)$ such that, for each ℓ , if ℓ' is the relation obtained as answer to the evaluation of Q_L^P on ℓ , then

$$(\ell')_L^G = P(\ell^G)$$

An atomic predicate P in Q^d is rewritten into P_L w.r.t. L as follows

- if P is searchable in L : $P_L = Q_L^P$,
- if P is not searchable in L : $P_L = \text{true}$

In the example, we consider the following predicate rewriting:

	Name like 'P*'	Year='1'	Dept='Dept1'
L1	lastn like 'P*'	year='1'	true
L2	name like 'P*'	true	dept_code='Dept1'

The computation of the local query condition LQ_i is obtained by rewriting each atomic predicate P in Q^d into P_{L_i} w.r.t. L_i . In the example, we have:

$$LQ_1 = (\text{lastn like 'P*' and year='1'})$$

$$LQ_2 = (\text{name like 'P*' and dept_code='Dept1'})$$

3. Residual Condition Q_r

The computation of Q_r is performed by the following three steps:

- (a) Transform Q into a Conjunctive Normal Form (CNF) query Q^c , the logical AND of clauses C which are the logical OR of atomic predicate P . In the example, we have:
 $Q^c = \text{Name like 'P*' and (Year='1' or Dept='Dept1')}$
- (b) Any clause C of Q^c containing not searchable (in one or more local classes) atomic predicates is a *residual clause*; in the example, we have:
 - Dept='Dept1' is not searchable in L_1 then $(\text{Year='1' or Dept='Dept1'})$ is a residual condition for L_1
 - Year='1' is not searchable in L_2 then $(\text{Year='1' or Dept='Dept1'})$ is a residual condition for L_2
- (c) Residual Condition Q_r is equal to the logical AND of residual clauses. in the example, we have:
 $Q_r = (\text{Year='1' or Dept='Dept1'})$

4. Select list of a local query LQ

The select list of the query LQ for the local class L , denoted with LAL , is obtained by considering the union of the following sets of attributes:

- (a) attributes of the global select list, AL
- (b) *Join Attribute Set* for the local class L , $JA(L)$
- (c) attributes in the Residual Condition, A_r

and by transforming these attributes on the basis of the Mapping Table:

$$LAL = \{A \in S(L) | \exists AG \in AL \cup JA(L) \cup A_r, A \in M[AG][L]\}$$

In the example, we have:

$$A_r = \{ \text{Year}, \text{Dept} \}$$

$$AL = \{ \text{Name} \}$$

$$JA(L_1) = \{ \text{Name} \}$$

$$JA(L_2) = \{ \text{Name} \}$$

then:

$$LAL_1 = \{ \text{lastn}, \text{firstn}, \text{year} \}$$

$$LAL_2 = \{ \text{name}, \text{dept_code} \}$$

In conclusion, we have the following local queries, LQ_1 and LQ_2 :

```
select lastn,firstn,year
from L1
where (firstn like 'P*' and year='1')
```

```
select name,dept_code
from L2
where (name like 'P*' and dept_code= 'Dept1')
```

The local query LQ_i is sent to the source related to the local class L_i ; its answer, q_i , is transformed by the operator $(.)_G^{L_i}$ and the result is stored in a temporary relation T_i , defined by equation 2 and the residual condition Q_2 . Now, we first compute the full disjunction of the temporary tables T_i and, then, we obtain the global query answer by applying the transformation defined by equation 2.

6 Local class materialization

As we said before, one of the task of the global class materializer is to single out the local classes that are relevant for the evaluation of the expanded query (queries LQ_i in the previous section). Since we are following the Global-as-views approach, this task is straightforward: in order to single out the local classes that are relevant for the evaluation of the expanded query, it is sufficient to look at the mapping from the global classes mentioned in the expanded query to the local classes.

Thus, the mapping tells us which are the relevant local classes for the evaluation of the query: such local classes should now be materialized, by accessing the corresponding sources. We call **local database** the set of instances of the various relevant local classes. As we said before, the local database is used by the global class materializer, according to what stated in the previous section.

We remind the reader that, associated to each local classes, there is a corresponding query over the external sources of the SINode. A notable case is when such an external source is expressed in XML. In this case, the associated query is expressed in an XML-based query language, such as Xquery.

In this part (part A) of the deliverable, we will not delve into the details of the local class materializer. We will simply assume in the following that the relevant data are loaded from the external sources, and the corresponding instances are inserted into the local classes. A detailed description of the local materializer will be presented in the second part (part B) of this deliverable.

7 Evaluation of the expanded query

The expanded query (union of conjunctive queries) is expressed in SQL, and is then evaluated over the global database computed by the global class materializer. This expanded query is submitted to the SQL Engine, and the resulting tuples are given as result of the overall process of query evaluation within the SINode.

Referring to the example presented in Sections 4 and 5, the final query is the following (note that the names of the global class queries are used in the definition of the query):

```
SELECT R1.Name, R4.Address
FROM OurRelation_00 R1, OurRelation_01 R2, OurRelation_02 R3, OurRelation_03 R4
WHERE (R1.Name = R2.Name) and (R2.Name = R4.Name) and (R2.CatCode = R3.CatCode)
UNION
SELECT R1.Name, R4.Address
FROM OurRelation_01 R1, OurRelation_01 R2, OurRelation_02 R3, OurRelation_03 R4
WHERE (R1.Name = R2.Name) and (R2.Name = R4.Name) and (R2.CatCode = R3.CatCode)
UNION
SELECT R1.Name, R4.Address
FROM OurRelation_20 R1, OurRelation_01 R2, OurRelation_02 R3, OurRelation_03 R4
WHERE (R1.Name = R2.Name) and (R2.Name = R4.Name) and (R2.CatCode = R3.CatCode)
UNION
SELECT R1.Name, R3.Address
FROM OurRelation_01 R1, OurRelation_02 R2, OurRelation_03 R3
WHERE (R1.Name = R3.Name) and (R1.CatCode = R2.CatCode)
```

8 Conclusions

In this part (part A) of the deliverable, we have described the basic steps of the process that answers a query posed to an SINode. We conclude this document by pointing out the aspects that we deliberately left out from the present work, and that will be addressed in the second part

(part B) of the deliverable. Part B of the deliverable will be produced later on in the project, in particular at month 25, when the second release of the prototype will be delivered.

- We remind the reader that, when processing a query, the query agent in Sewasie should be able to:
 1. interact with relevant brokering agents,
 2. ask the various SINodes the right information, according to a specific query plan.

While we have studied item (2) in this document, the problem of designing techniques to be used by the query agent in order to decide which brokering agents to contact (item 1 above), and how to interact with them, will be addressed in the second part of this deliverable.

- In this part of the deliverable, we did not delve into the details of the local class materializer. We have simply assumed in the following that the relevant data are loaded from the external sources, and the corresponding instances are inserted into the local classes. A detailed description of the local materializer will be presented in the second part of this deliverable.
- Currently, data reconciliation is only partially addressed by the global class materializer. In the second part of the deliverable we will deal with this issue in a more comprehensive way.
- Finally, techniques for handling in an ad-hoc way sets of queries posed to the same SINode will be addressed in the second part of the deliverable.

References

- [1] Andrea Cali, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of the 20th Int. Conf. on Conceptual Modeling (ER 2001)*, pages 270–284, 2001.
- [2] Andrea Cali, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. 2003. To appear.
- [3] Andrea Cali, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. 2003. To appear.
- [4] The Sewasie Consortium. Specification of the architecture of the brokering agent. *Sewasie, Deliverable D.4.1.a, Final Version*, Feb. 2003.
- [5] University of Modena e Reggio Emilia. Specification of the general framework for the multilingual semantic enrichment processes and of the semantically enriched data stores. *Sewasie, Deliverable D.2.1, Final Version*, Feb. 2003.
- [6] University of Rome “La Sapienza”. General framework for query reformulation. *Sewasie, Deliverable D.3.1, Final Version*, Feb. 2003.
- [7] D. Beneventano, S. Bergamaschi, and F. Mandreoli. Extensional Knowledge for semantic query optimization in a mediator based system. In *Proc. of FMII*, 2001.
- [8] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30:3, pp 514-550, 1983.
- [9] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative Data Cleaning: Language, Model, and Algorithms. In *Proc. of VLDB*, pages 371–380, 2001.
- [10] C. Glaindo-Legaria. Outerjoins as disjunctions. *CWI*, 1993.

- [11] Konstantin Eugeniev Kostenarov. Elaborazione di interrogazioni in un sistema multi-database. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 2001-2002.
- [12] D. Lembo, M. Lenzerini, and R. Rosati. Source inconsistency and incompleteness in data integration. In *Proc. of KRDB*, 2002.
- [13] Anand Rajaraman and Jeffrey D. Ullman. Integration information by outerjoins and full disjunction. *ACME*, 1997.
- [14] L.-L. Yan and M. T. Özsu. Conflict Tolerant Queries in AURORA. In *Proc. of CoopIS*, pages 279–290, 1999.