DEGREE OF DOCTOR OF PHILOSOPHY IN
COMPUTER ENGINEERING AND SCIENCE

INTERNATIONAL DOCTORATE SCHOOL IN
INFORMATION AND COMMUNICATION TECHNOLOGIES

XXII Cycle

UNIVERSITY OF MODENA AND REGGIO EMILIA

INFORMATION ENGINEERING DEPARTMENT

Ph.D. DISSERTATION

Query Optimization and Quality-Driven Query Processing for Integration Systems

Candidate:
Ing. Rodrigue Carlos Nana Mbinkeu
Advisor:
Prof. Domenico Beneventano
Co-Advisor:
Prof. Francesco Guerra
The Director of the School:
Prof. Sonia Bergamaschi

DOTTORATO DI RICERCA IN
COMPUTER ENGINEERING AND SCIENCE

SCUOLA DI DOTTORATO IN
INFORMATION AND COMMUNICATION TECHNOLOGIES

XXII Ciclo

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

TESI PER IL CONSEGUIMENTO DEL TITOLO DI DOTTORE DI RICERCA

Query Optimization and Quality-Driven Query Processing for Integration Systems

Tesi di:
Ing. Rodrigue Carlos Nana Mbinkeu
Relatore:
Prof. Domenico Beneventano
Co-Relatore:
Prof. Francesco Guerra
Il Direttore:
Prof. Sonia Bergamaschi

Acknowledgement

I would like to express my sincere gratitude to Professor Domenico Beneventano for his supervision and guidance.

Also I would to thank Professor Sonia Bergamaschi in helping me to broaden my view and knowledge.

My deepest gratitude to my parents, sisters, brothers and my wife in supporting me.

And I dedicate this thesis to the God of the Holy Bible.

To my daughter Shalonne, I love you.

Abstract

This thesis focuses on Query Optimization and Quality-Driven Query Processing for Integration Systems. My research activity is based on the MOMIS (Mediator Environment for Multiple Information Sources) Data Integration system, developed by the DBGroup of the University of Modena and Reggio Emilia.

Data integration is the process of extracting and merging data from multiple heterogeneous sources to be loaded into a unified view. At the same time, each autonomous source deals with its own quality properties on information, such as accuracy, consistency, completeness, timeliness and cost of information access.

One of the topics of this thesis is to propose new techniques that consider the optimization of full outerjoin operation in MOMIS. Full outerjoin is used in data integration systems for merging multiple records representing the same real-world object into a single, consistent, and clean representation. Our optimization is mainly defined as a substitution of full outerjoin operator by the left(right) outerjoin or inner join operators. A query manager that merge information using full outerjoin can benefit easily from this optimization technique.

Secondly, starting from a MOMIS framework extended with data quality dimensions, this thesis gives the two following original contributions: Data Quality Aware Queries (data quality dimensions are used in the specification of a query to express the quality of the retrieved data) and Quality-Driven Query Processing (quality constraints specified in the query are used to perform query optimization).

Finally, this thesis shows how the MOMIS mediator system can deal with all the queries of the THALIA benchmark, a public available testbed and benchmark for information integration systems.

Sommario

L'argomento principale di questa tesi è l'ottimizzazione e la qualità dei dati nell'elaborazione delle interrogazioni nel sistema MOMIS. La mia attività di ricerca si è basata sul sistema a mediatore MOMIS (Mediator Environment for Multiple Information Sources), sviluppato dal DBGroup dell'Università di Modena e Reggio Emilia.

L'integrazione dei dati è il problema di combinare dati memorizzati da diverse fonti autonome, e di fornire all'utente una visione unificata dei dati. I dati di ogni sorgente sono dati pre-esistenti e possono essere caratterizzati da vari livelli di eterogeneità e qualità.

Il primo argomento di questa tesi è l'ottimizzazione delle interrogazioni nel sistema MOMIS. In MOMIS, la fusione dei dati è effettuata attraverso l'operatore di *full outerjoin*. Essendo questo operatore costoso in termini di tempi di elaborazione, l'obiettivo delle techniche di ottimizzazione proposte nella tesi è quello di sostituire tale operatore con operatori meno costosi quali il left(right) outerjoin oppure l'inner join.

Il secondo argomento della tesi riguarda il sistema MOMIS esteso con metadati di qualità, ovvero le sorgenti dati locali che devono essere integrate sono annotate anche con metadati relativi alla qualità dei dati. In questo framework esteso, l'utente può specificare in fase di interrogazione del sistema, anche i criteri di qualità che definiscono il livello di qualità desiderata. È stato quindi mostrato come questi criteri di qualità possano avere un impatto positivo sull'ottimizzazione dell'elaborazione delle interrogazioni.

Infine, questa tesi illustra come il sistema a mediatore MOMIS è in grado di soddisfare tutte le query del benchmark THALIA, benchmark reso disponibile per valutare e confrontare i sistemi di integrazioni dati.

Indice

Elenco delle figure

Introduction

During the last decade many data integration systems characterized by a classical wrapper/mediator architecture [91] based on a Global Virtual Schema (Global Virtual View - GVV) have been proposed. The data sources store the real data, while the GVV provides a reconciled, integrated, and virtual view of the data sources. Modelling the mappings among sources and the GVV is a crucial aspect. Two basic approaches for specifying the mappings in a Data Integration System have been proposed in the literature: Local-As-View (LAV), and Global-As-View (GAV), respectively [40, 85].

The LAV approach is based on the assumption that a global schema representing the conceptualization of a domain exists and the contents of each local source must be described in terms of the global schema. This assumption holds only in the case that the GVV is stable and well-established in the organization. This constitutes the main limitation of the LAV approach. Another negative issue is the complexity of query processing which needs reasoning techniques. On the other hand, as a positive aspect, the LAV approach facilitates the extensibility of the system: adding a new source simply means enriching the mapping with a new assertion, without other changes [47].

In the GAV approach the contents of the elements of the GVV is not predefined and is described in terms of a view of the local sources. GAV favours the system in carrying out query processing, because it tells the system how to use the sources to retrieve data (unfolding). However, extending the system with a new source is now more difficult: the new source may indeed have an impact on the definition of various classes of the GVV, whose associated views need to be redefined.

Data Integration aims to combine distributed information conforming to different data models and provide interfaces for accessing such information in a unfied view. In chapter 1, we present the Mediator Environment for

Multiple Information Sources (MOMIS), developed by the database research group at the University of Modena and Reggio Emilia, aims to construct synthesized, integrated descriptions of information coming from multiple heterogeneous sources. The goal is to provide users with a global virtual view (GVV) of information sources, independent of their location or their data's heterogeneity. Such a view conceptualizes the underlying domain; you can think of it as an ontology describing the sources involved.

In chapter 2, we show how the MOMIS mediator system can deal with all the twelve queries of the THALIA benchmark, a public available testbed and benchmark for information integration systems [44], by simply extending and combining the declarative translation functions already available in MOMIS and without any overhead of new code. This is a remarkable result, in fact, as far as we know, no system has provided a complete answer to the benchmark [10].

The goal of chapter 3 is to propose new techniques that consider the optimization of full outerjoin operation in MOMIS. Full outerjoin is used in data integration systems for merging multiple records representing the same real-world object into a single, consistent, and clean representation. Our optimization is mainly defined as a substitution of full outerjoin operator by the left(right) outerjoin or inner join operators [51, 14]. A query manager that merge information using full outerjoin can benefit easily from this optimization technique.

Chapter 4 is a review of the literature on the data quality related to data integration systems. This study presents the common dimensions between the different information quality frameworks revealed in [65]. We have mentioned various data anomalies which affect the data quality and we proposed a table that summarizes relationships between data anomalies and data quality dimensions. We made a study of different data integration systems which implement the notion of data quality. Specially, we focus on quality-driven query processing.

Data Integration is a complex process and Data Integration quality is difficult to assess. Chapter 5 aims to improve and extend the MOMIS framework to allow data at sources to be annotated with data quality metadata [13]. Then we will consider queries with quality criteria, i.e. data quality metadata are used in the specification of a query on the Global Schema, to express the quality of the retrieved data, by means of threshold of acceptance which users can ensure minimal performance of the data [55]. Finally, we

discuss the optimization of such Global Queries with Quality [13].

Capitolo 1

Data Integration Systems

Data Integration aims to combine distributed information conforming to different data models and provide interfaces for accessing such information in an unfied view. The Mediator Environment for Multiple Information Sources (MOMIS), developed by the database research group at the University of Modena and Reggio Emilia, aims to construct synthesized, integrated descriptions of information coming from multiple heterogeneous sources. The goal is to provide users with a global virtual view (GVV) of information sources, independent of their location or their data's heterogeneity. Such a view conceptualizes the underlying domain; you can think of it as an ontology describing the sources involved.

1.1 Data Integration Systems

Integration Systems are usually characterized by a classical wrapper-mediator architecture [90] used to build a mediated schema of a set of data sources. The data sources store the real data, while the mediated schema provides a reconciled, integrated and virtual view of the underlying sources.

Many research efforts have been devoted to modeling the mappings between the sources and the mediated schema. Two different approaches have been proposed for specifying these mappings: the Local-as-View approach (LAV) describes the data sources as views over the mediated schema; in contrast, the Global-as-View (GAV) approach describes the mediated schema as a set of view definitions over the source relations [48, 41, 86]. The two approaches have been fused in a unique model, called Global-Local-as-View (GLAV), which synthesizes the characteristics

of the two approaches [34]. A general description is included in the following.

In this thesis I will focus on the MOMIS[1] system (Mediator EnvirOment for Multiple Information Sources), developed by the DBGroup of the University of Modena and Reggio Emilia. MOMIS is an Intelligent Data Integration framework designed for the integration of heterogeneous data sources that adopts a GAV approach. A general description of MOMIS is provided in this chapter. An open source version of the MOMIS system is delivered and maintained by the academic spin-off DataRiver[2].

1.2 The MOMIS Data Integration System

Definition of the MOMIS Integration System

Definition 1 *Given a set $\mathcal{N}$ of data sources to be integrated, we can define a MOMIS Integration System IS = (GS, $\mathcal{N}$, $\mathcal{M}$) as constituted by:*

- *A Global Schema (GS), which is a schema expressed in the ODL_{I^3} language*
- *A set $\mathcal{N}$ of data sources; each source has a schema also expressed in ODL_{I^3}*
- *A set $\mathcal{M}$ of GAV mapping assertions between the GS and $\mathcal{N}$, where each assertion associates to an element G in GS a query $q_{\mathcal{N}}$ over the schemas of the set $\mathcal{N}$ of local sources. More precisely, for each global class $G \in GS$ we define:*
  1. *a (possibly empty) set of classes, denoted by $\mathcal{L}(G)$, belonging to the local sources in $\mathcal{N}$*
  2. *a query $\mathcal{MQ}^G$ over the $\mathcal{L}(G)$ classes*

Intuitively, the GS is the intensional representation of the information provided by the Integration System, whereas the mapping assertions specify how such an intensional representation relates to the local sources managed by the Integration System. The semantics of an Integration System is defined in [24, 12].

MOMIS performs information extraction and integration from both structured and semi-structured data sources. An object-oriented language,

[1] See `http://www.dbgroup.unimore.it` for references about the MOMIS project.
[2] `http://www.datariver.it`

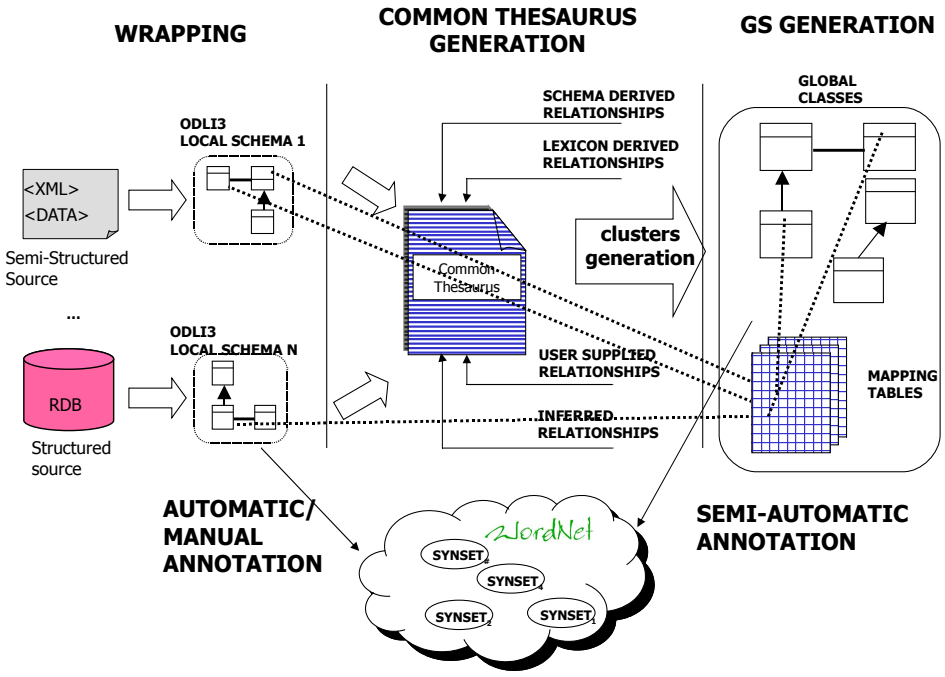

Figura 1.1: Global Schema generation process with the MOMIS system

with an underlying Description Logic, called ODL_{I^3}, described in Section 1.3 (see Appendix A for its syntax), is introduced for information extraction. Information integration is then performed in a semi-automatic way, by exploiting the knowledge in a Common Thesaurus (defined by the framework) and ODL_{I^3} descriptions of source schemas with a combination of clustering techniques and Description Logics. This integration process gives rise to a virtual integrated view (the Global virtual Schema GS) of the underlying sources for which mapping rules and integrity constraints are specified to handle heterogeneity. Given a set of data sources related to a domain, it is possible to semi-automatically synthesize a GS that conceptualizes a domain and thus might be thought as a basic domain ontology for the integrated sources.

1.3 The ODL$_{I^3}$ Language

The ODL$_{I^3}$ language used in the MOMIS system to represent data is an extension of the *Object Definition Language* (ODL), an object-oriented language developed by ODMG[3][4]. ODL$_{I^3}$ is transparently translated into a Description Logic [8, 15]. ODL$_{I^3}$ allows different kinds of data sources and the view resulting from the integration process to be represented in a common data model. In ODL$_{I^3}$ all the data sources are represented as a set of classes and attributes. Moreover, some constructors and rules are present in the language to handle the heterogeneity:

Union constructor. The union constructor is introduced to express alternative data structures in the definition of an ODL$_{I^3}$ class, thus capturing requirements of semistructured data.

Optional constructor. The optional constructor is introduced for class attributes to specify that an attribute is optional for an instance (i.e., it could be not specified in the instance).

Integrity constraint rules. This kind of rule is introduced in ODL$_{I^3}$ in order to express, in a declarative way, *if then* integrity constraint rules at both intra- and inter-source level.

Intensional relationships. They are *terminological relationships* expressing intra- and inter-schema knowledge for the source schemas. Intensional relationships are defined between classes and attributes, and are specified by considering class/attribute names, called terms. The following relationships can be specified in ODL$_{I^3}$:

- SYN (Synonym-of), defined between two terms t_i and t_j, with $t_i \neq t_j$, that are considered synonyms in every considered source (i.e., t_i and t_j can be indifferently used in every source to denote a certain concept).
- BT (Broader Terms), or hypernymy, defined between two terms t_i and t_j such as t_i has a broader, more general meaning than t_j. BT relationship is not symmetric. The opposite of BT is NT (Narrower Terms), or hyponymy.
- RT (Related Terms), or positive association, defined between two terms t_i and t_j that are generally used together in the same context in the considered sources.

[3] `http://www.odmg.org/`

[4] `http://www.service-architecture.com/database/articles/odmg_3_0.html`

An intensional relationship is only a terminological relationship, with no implications on the extension or compatibility of the structure (domain) of the two involved classes (attributes).

Extensional relationships. Intensional relationships SYN, BT and NT between two classes C_1 and C_2 may be "strengthened" by establishing that they are also *extensional* relationships.

Mapping Rules. This kind of rule is introduced in ODL$_{I^3}$ in order to express relationships holding between the integrated ODL$_{I^3}$ schema description of the information sources and the ODL$_{I^3}$ schema description of the original sources.

Using ODL$_{I^3}$ for representing sources and ontologies is not a limitation: the interoperability of the ODL$_{I^3}$ descriptions is guaranteed by a software module able to translate the descriptions into the Web Ontology Language OWL [68].

1.4 Global Schema Generation with MOMIS

The integrated global schema generated by the MOMIS system is composed of a set of global classes that represent the information contained in the underlying sources and the mappings that establish the connections among global class attributes and the source schemata.

The process of creating the GS and defining the mappings, shown in figure 1.1, can be summarized in the following steps:

Local source schemas extraction

The first phase of the integration process is the choice of the data sources and their translation into the ODL$_{I^3}$ format. The translation process is performed by the MOMIS wrappers, which logically converts the source data structure into the ODL$_{I^3}$ model. The wrapper architecture and interfaces are crucial, because wrappers are the focal point for managing the diversity of data sources.

Lexical annotation of local sources

The goal of the annotation phase is to semantically annotate terms

denoting schema elements in data sources according to a common lexical reference, which means to assign a shared meaning to each of them. The WordNet lexical database is usually referred to as lexical reference [33], but other lexical references might be used to cope with specific domains. The annotation step of the schema elements, pre-processed by the system applying stop-words and stemming techniques, can be performed manually by the integration designer, or automatically by the MOMIS system. The manual annotation is composed of two different steps: in the Word Form choice step, the WordNet morphological processor suggests a word form corresponding to the given term; in the Meaning choice step, the designer can choose to map an element to zero, one or more senses. In MOMIS the annotation step can also be performed automatically combining a set of Word Sense Disambiguation algorithms [16]: SD (Structural Disambiguation) [17], WND (WordNet Domains Disambiguation) [17], WordNet first sense heuristic, Gloss Similarity [11] and Iterative Gloss Similarity [11]. For further details about the annotation techniques in MOMIS see [70].

Common thesaurus generation

Starting from the annotated local schemata, MOMIS constructs a Common Thesaurus describing intra and inter-schema knowledge in the form of SYN(synonyms), BT/NT(broader terms/narrower terms), and RT(meronymy/holonymy) relationships. The Common Thesaurus is constructed through an incremental process in which the following relationships are added:

- schema-derived relationships: relationships holding at intra-schema level are automatically extracted by analyzing each schema separately. Some heuristic can be defined for specific kind of sources. For example, MOMIS extracts intra-schema RT relationships from foreign keys in relational source schemas. When a foreign key is also a primary key, in both the original and referenced relation, MOMIS extracts BT and NT relationships, which are derived from inheritance relationships in object-oriented schemas.

- lexicon-derived relationships: the annotation phase is exploited to translate relationships holding at the lexical level into relationships to be added to the Common Thesaurus. These relationships may be inferred from lexical knowledge (e.g. by querying WordNet for relationships between senses).

- designer-supplied relationships: new relationships can be supplied directly by the designer, to capture specific domain knowledge.
- inferred relationships: Description Logics (DL) techniques of ODB-Tools [18], are exploited to infer new relationships.

Global Schema Generation

The Global virtual Schema (GS) consists of a set of classes (called Global Classes), plus mappings to connect the global attributes of each Global Class and the local source attributes. The MOMIS methodology allows identifying similar ODL_{I^3} classes (i.e. classes that describe the same or semantically related concept in different sources) and mappings to connect the global attributes of each global class to the local source attributes. To this end, affinity coefficients are evaluated for all possible pairs of ODL_{I^3} classes, based on the relationships in the Common Thesaurus properly strengthened. Affinity coefficients determine the degree of matching of two classes based on their names (*Name Affinity coefficient*) and their attributes (*Structural Affinity coefficient*) and are fused into the *Global Affinity coefficient*, calculated by means of the linear combination of the two coefficients [26]. Global affinity coefficients are then used by a hierarchical clustering algorithm, to cluster ODL_{I^3} classes according to their degree of affinity. For each cluster C, a Global Class G, with a set of Global Attributes $GA_1, \ldots GA_{\mathcal{N}}$, and a Mapping Table MT, expressing mappings between local and global attributes, are defined. The Mapping Table is a table whose columns represent the local classes which belong to the Global Class and whose rows represent the global attributes. An element $MT[GA][L]$ is a function which represents how local attribute of the Local Class L is mapped into the global attribute GA:

$$MT[GA][L] = LA$$

where LA is the local attribute of L.

GS lexical annotation

To annotate a GS means to assign a name and a set (eventually empty) of meanings to each global element (class or attribute). MOMIS automatically annotates each global element proposing the broadest meaning extracted from the annotations of the local sources, based on the relationships included in the Common Thesaurus. Names and meanings have then to be confirmed

by the ontology designer. This annotation step is a significant result, since these metadata may be exploited by external users and applications.

1.4.1 Mapping Query

After the GS generation process, each global class G is associated to a Mapping Table. Starting from the Mapping Table of G, the integration designer can implicitly define the mapping query $\mathcal{MQ}^G$ associated to the global class G by:

1. extending the Mapping Table with
   - Data Conversion Functions from local to global attributes
   - Join Conditions among pairs of local classes belonging to G
   - Resolution Functions for global attributes to solve data conflicts of local attribute values.
2. using and extending the *full outerjoin-merge* operator, proposed in [60, 22], to solve data conflicts of common local attribute values and merge common attributes into one.

Then, the Integration Designer defines, for each global class, the associated mapping query $\mathcal{MQ}^G$; this is performed with the following two steps:

1. *Data Transformation:* To transform local attribute values into a global attribute value by means of the *Data Conversion Functions* from local to global attributes. (see section 1.4.1.1).
2. *Data Fusion:* A Global Class performs *Data Fusion* among its local class instances when multiple records coming from local classes and representing the same real-world object are fused into a single and consistent record of the global class [22]. The integration designer must decide if he wants that the system perform data fusion on a Global Class or not. If yes, the query $\mathcal{MQ}^G$ associated to a Global Class is automatically composed by the system on the basis of the process described in the **Data Fusion** section below. If not, the Integration Designer must explicitly compose the mapping query

1.4.1.1 Data Conversion Functions

The Ontology Designer can define, for each not null element $MT[GA][L]$, a Data Conversion Function, denoted by $MTF[GA][L]$, which represents the

mapping of local attribute of L into the global attribute GA. $MTF[GA][L]$ is a function that must be executable/supported by the class L local source. For example, for relational sources, $MTF[GA][L]$ is an SQL value expression. $T(L)$ denotes L transformed by the Data Conversion Functions. Further details about MOMIS data conversion functions can be found in Chapter 2.

1.4.1.2 Data Fusion

The mapping query $\mathcal{MQ}^G$ is defined to make a global class perform *Data Fusion* among its local class instances [22]: multiple *local tuples* coming from local classes and representing the same real-world object are fused into a single and consistent *global tuple* of the global class. To identify multiple local tuples coming from local classes and representing the same real-world object, we assume that error-free and shared object identifiers exist among different sources: two local tuples with the same object identifier indicate the same object in different sources.

As an example, in Figure 1.2 we show the Mapping Table of a global class G with schema $G(ID, A, B, C)$ and with local classes $\mathcal{L}(G) = \{L_1, L_2, L_3\}$; for the local classes the schema is, respectively, $L_1(ID, A, C)$, $L_2(ID, B, C)$ and $L_3(ID, C)$. In our example, we assume ID as an object identifier.

G	L_1	L_2	L_3
ID	ID	ID	ID
A	A	-	-
B	-	B	-
C	C	C	C

Figura 1.2: Mapping Table of Global Class G

Data Reconciliation, i.e. to solve conflicts among instantiations of the same object in different sources, is performed by *Resolution Functions* [59]: for each GA such that there are more than one non empty element $MT[GA][L]$, a *Resolution Function* (RF) is defined to obtain, starting from the transformed local classes, a *merged value* for GA. In our example an `AVG` resolution function is defined on `C`. In order to carry on the *Data Fusion* operations described so far, the mapping query $\mathcal{MQ}^G$ is defined by means of the *full outerjoin-merge operator* proposed in [60] and adapted to the MOMIS framework in [7]; this operator is based on the concept of *Resolution Functions*, which are introduced in next section.

1.4.1.3 Resolution Functions

In MOMIS, the approach proposed in [61] has been adopted: a Resolution Function for solving data conflicts may be defined for each global attribute mapping onto local attributes coming from more than one local source; in this way we can define what value should appear in the result. A global attribute with no data conflicts (i.e. the instances of the same real object in different local classes having the same value for this common attribute), is called Homogeneous Attribute. Of course, for homogeneous attributes, resolution functions are not necessary (a global attribute mapped onto only one source is a particular case of an homogeneous attribute). In MOMIS we use conflict resolution strategies based on Resolution Functions as introduced in [61, 21]: for global attributes, mapped in more than one local attributes, the designer defines, in the Mapping Table, Resolution Functions to solve data conflicts of local attribute values. For example, if $L_1.C$, $L_2.C$ and $L_3.C$ are numerical attributes, we can define $G.C = AVG(L_1.C, L_2.C, L_3.C)$. The MOMIS system provides some standard kinds of resolution functions (see the Figure 1.3).

Function	Definitions
MAX / MIN	Returns the maximum/minimum value of the conflicting data values
FIRST / LAST	Takes the first/last values of all values, even if it is a NULL value
VOTE	Returns the value that appears most often among the present values
GROUP	Returns a set of all conflicting values and leaves resolution to the user
COALESCE	Takes the first non-null value appearing
CHOOSE	Returns the value supplied by the specific source
CONCAT	Returns the concatenation of all input values
AVG	Returns the arithmetic mean of all input values
SUM	Returns the sum of all input values
RANDOM	Returns the random non-null input value

Figura 1.3: Resolution Functions in MOMIS

Definition 2 (Resolution function) *Let D be an attribute domain and $D^+ := D \cup \perp$ where $\perp$ represents the null value. A resolution function f is*

an associative function $f : D^+ \times D^+ \rightarrow D^+$ *with*

$$f(x,y) \;:=\; \begin{cases} \perp & \text{if } x = \perp \text{and } y = \perp \\ x & \text{if } y = \perp \text{and } x \neq \perp \\ y & \text{if } x = \perp \text{and } y \neq \perp \\ g(x,y) & \text{else} \end{cases}$$

where $g : D \times D \rightarrow D$. *Function* g *is an internal associative resolution function.*

Communtativity is a prerequisite for resolution functions, because sources and tables in SQL queries are unordered. All presented resolution functions are commutative. Associativity on the other hand is not required but useful.

1.4.1.4 Full Outer join-merge operator

Intuitively, defining $\mathcal{MQ}^G$ by means of a full outerjoin-merge corresponds to the following two operations:

(1) Computation of the *Full Outerjoin*, on the basis of the shared object identifiers, of the *local classes* of G;

(2) Application of the resolution functions.

Thus $\mathcal{MQ}^G$ can be formulated by standard SQL, with the exception of (some) resolution functions; As an example, for the global class G of Figure 1.2, we have the following $\mathcal{MQ}^G$:

```
SELECT COALESCE(L1.ID,L2.ID,L3.ID) AS ID,
       L1.A AS A, L2.B AS B,
       AVG(L1.C,L2.C,L3.C) AS C
FROM L1 FJ L2 ON (L2.ID=L1.ID)
         FJ L3 ON (L3.ID=L1.ID OR L3.ID=L2.ID )
```

where `FJ` is an abbreviation for the SQL full outerjoin operator, `COALESCE` is the standard SQL function which returns its first non-null parameter value and `AVG` is a (non standard SQL) function to compute the average value. As an example, in Figure 1.4, we show the instances of local classes[5] and the corresponding instance of the global class computed by means of $\mathcal{MQ}^G$. We use $\perp$ to denote the *null value* in their instances.

In the case of data fusion with a generic number n of local classes, the computation of the **full outerjoin** expression (FOJ expression) in $\mathcal{MQ}^G$ is defined as:

[5]Let C denote either a local class L or a global class G. An istance of C is a set of tuples conforming with the schema of C; we conflate the notation and use the same symbol C for both the class C and an instance of C.

L_1

ID	A	C
1	3	24
2	⊥	20
3	9	⊥
4	8	25
5	⊥	20

L_2

ID	B	C
1	3	24
2	⊥	30
3	⊥	20
5	⊥	30

L_3

ID	C
5	25

G

ID	A	B	C
1	3	3	24
2	⊥	⊥	25
3	9	⊥	20
4	8	⊥	25
5	⊥	⊥	25

Figura 1.4: Instances of the local class L_1, L_2 and L_3 and of the global class G computed with the full outerjoin-merge operator

```
L1 FJ L2 ON (L1.ID = L2.ID)
    FJ L3 ON (L3.ID = L1.ID OR L3.ID = L2.ID)
      ...
       FJ Ln ON (Ln.ID = L1.ID OR ... OR Ln.ID = Ln-1.ID)
```

As shown in [7] if we use a join condition based on the shared object identifier ID, the order of local classes in the full outer join evaluation is not relevant.

For sake of simplicity, we represent the FOJ operation like as a FOJ-sequence where join conditions are omitted:

`< L1, FJ, L2, FJ, L3,...., FJ, Ln >` .

More formally, the full outerjoin-merge operator is defined as follows [60].

Definition 3 (Join-merge operator) *Let G be a global class with schema $S(G)$ and let L_i and L_j two local classes of G, with schema $S(L_i)$ and $S(L_j)$.*

$$\begin{aligned} L_i \sqcap L_j \;=\; & \{\textit{tuple } t[A] \mid \exists r \in L_i, \exists s \in L_j \textit{ with} \\ & t[ID] = r[ID] = s[ID], \\ & t[A] = s[A], \forall A \in S(L_i) \setminus S(L_j), \\ & t[A] = r[A], \forall A \in S(L_j) \setminus S(L_i), \\ & t[A] = f(r[A], s[A]), \\ & \qquad \forall A \in S(L_i) \cap S(L_j), A \neq ID, \\ & t[A] = \perp, \forall A \in S(G) \setminus (S(L_i) \cup S(L_j))\} \end{aligned}$$

where f is a resolution function as defined before.

The definition of the left outerjoin-merge operator is based on the outer-union operator $\biguplus$, which performs a union over relations with differing attribute sets [73]. The attribute set of the result is the union of the attribute sets of the two relations. In our case this is the entire attribute set $S(G)$ because the result of a join-merge operation has $S(G)$ as attribute set.

Definition 4 (Left outerjoin-merge operator) *Let G be a global class with schema $S(G)$ and let L_i and L_j two local classes of G, with schema $S(L_i)$ and $S(L_j)$.*

$$L_i \sqsupset L_j \quad = \quad L_i \sqcap L_j \uplus \Big(L_i \setminus \Pi_{S(L_i)}(L_i \sqcap L_j)\Big)$$

The left outerjoin-merge corresponds to the classical left outerjoin applying the same restrictions as for the join-merge operator. The left outerjoin-merge $L_i \sqsupset L_j$ guarantees that all tuples from L_i appear in the result. Wherever possible, they are joined with tuples from the other source. If not possible, the missing values are padded with null. Since the right outerjoin-merge operator is basically the same as the left outerjoin-merge, we continue the discussion only with the latter.

Definition 5 (Full outerjoin-merge operator) *Let G be a global class with schema $S(G)$ and let L_i and L_j two local classes of G, with schema $S(L_i)$ and $S(L_j)$.*

$$\begin{aligned} L_i \sqcup L_j \quad = \quad & L_i \sqcap L_j \uplus \Big(L_i \setminus \Pi_{S(L_i)}(L_i \sqcap L_j)\Big) \\ & \uplus \Big(L_j \setminus \Pi_{S(L_j)}(L_i \sqcap L_j)\Big) \end{aligned}$$

The full outerjoin-merge operator guarantees that every tuple from both sources enters the result. Missing values in attributes of tuples that do not have a matching tuple in the other source are padded with null values.

For a global class G with two local classes, the mapping query $\mathcal{MQ}^G$ associated to G is defined on the basis of the *full outerjoin-merge* operator, using ID as join attribute. This definition can be extended to a global class G with more than two local classes: the $\sqcup$ operator is an associative operator and the evaluation order is not relevant [7]:

$$(L_1^{\mathsf{T}} \sqcup L_2^{\mathsf{T}}) \sqcup L_3^{\mathsf{T}} = L_1^{\mathsf{T}} \sqcup (L_2^{\mathsf{T}} \sqcup L_3^{\mathsf{T}})$$

for each L_1, L_2 and L_3 belonging to G.

Then:

Definition 6 (Mapping query $\mathcal{MQ}^G$) *Let G be a global class and let $\mathcal{L}(G)$ be the set of its local classes. The mapping query $\mathcal{MQ}^G$ associated to G is defined as follows:*

$$\mathcal{MQ}^G \quad = \quad \bigsqcup \{L_i^T \mid L_i \in \mathcal{L}(G)\}$$

1.5 Query Execution

The MOMIS Query Manager allows the user to pose a query expressed in OQL [8, 69] over the ontology and to obtain a unified answer from all the data sources integrated in the GS. When the MOMIS Query Manager receives a query, it rewrites the global query as an equivalent set of queries expressed on the local schemas (local queries); this query translation is carried out by considering the mapping between the GS and the local schemata. The query translation is thus performed by means of query unfolding, i.e. by expanding a global query on a global class G of the GS according to the definition of the mapping query $\mathcal{MQ}^G$. The local queries are then executed on the sources, and MOMIS performs the fusion of the local answers into a consistent and concise unified answer, and present the global answer to the user. In order to assure full usability of the system even to users with low information technology skills, that are often the target of many information integration application, a graphical user interface to compose queries over the MOMIS GS was developed. This interface was initially developed as part of a MOMIS application presented in [77]. This interface was then made available as a MOMIS component and can thus be automatically used with any MOMIS schema.

1.5.1 Query Unfolding

The query unfolding process is performed for a Global Query Q over a global class G of the GVV :

```
Q = SELECT <Q_SELECT-list>
    FROM G
    WHERE <Q_condition>
```

where `<Q_condition>` is a Boolean expression of positive atomic constraints: ($GA1$ *op value*) or ($GA1$ *op* $GA2$), with GA1 and GA2 attributes of G.

The query unfolding process is made up of the following three steps:

- Generation of Local Queries:

  ```
  Q_L  = SELECT <SELECT-list>
         FROM L
         WHERE <condition>
  ```

  where L is a local class related to G. The `<SELECT-list>` is computed by considering the union of:

1. the global attributes in `<Q_SELECT-list>` with a not null mapping in L,

2. the global attributes used to express the join conditions for L,

3. the global attributes in `<Q_condition>` with a not null mapping in L.

The set of global attributes is transformed in the corresponding set of local attributes on the basis of the Mapping Table. The `<condition>` is computed by performing an atomic constraint mapping: each atomic constraint of `<condition>` is rewritten into one that is supported by the local source. The atomic constraint mapping is performed on the basis of the MDTFs and Resolution Functions defined in the Mapping Table. For example, if the numerical global attribute GA is mapped onto L1 and L2, and we define AVG as resolution function, the constraint ($GA = value$) cannot be pushed at the local sources, because AVG has to be calculated at a global level. In this case, the constraint is mapped as true in both the local sources. On the other hand, if GA is an homogeneous attribute the constraint can be pushed at the local sources. For example, an atomic constraint ($MT[GA][L]$ *op* *value*) is mapped onto the local class L only if:

(a) $MT[GA][L]$ is not null **and** the op operator is supported into L

(b) true **otherwise**

An atomic constraint ($GA1$ *op* $GA2$) is mapped in a similar way.

- Generation of the sequence `< L1, FJ, L2,...., FJ, Ln >` which computes the Full Outerjoin operation of the local query answers Q_L

- Generation of the final query (application of Resolution Functions)

We will discuss the Query Execution in more details in chapter 3, in the context of Query Optimization in MOMIS.

1.6 MOMIS Architecture

In this section we briefly describe the architecture of the MOMIS system, shown in Figure 1.5. Further details can be found in [69].

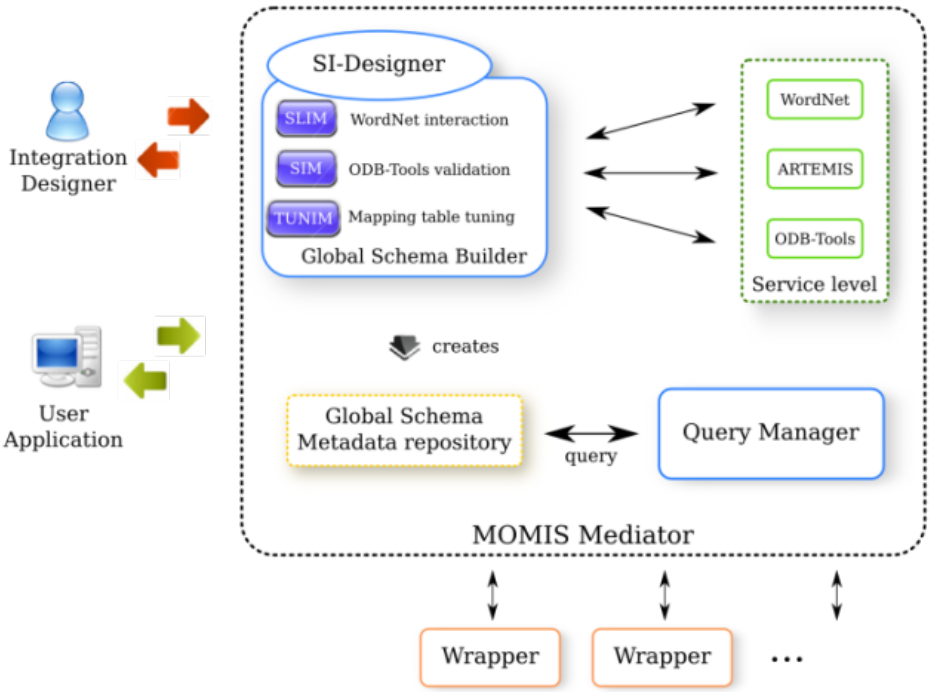

Figura 1.5: MOMIS Architecture

The main components of MOMIS are:

- **Wrappers:** Wrappers are software modules with the role of managing the interactions with the data sources. The MOMIS wrappers translate the source data structures into ODL_{I^3}. Their role is to deal with the diversity of the data sources thus allowing MOMIS to pay no attention to the language details of the different data sources. Wrappers are available for different kind of data sources, ranging from different database management systems to semi-structured data like XML, RDF, OWL formats. Wrappers logically guarantee two main operations:
  - *getschema()* translates the schema from the original format into ODL_{I^3}, dealing with the necessary data type conversions
  - *runquery()* executes a query on the local source. The MOMIS Query Manager translates a query on the GS (a global query) into a set of local queries to be locally executed by means of wrappers.

- **Global Schema Builder:** The Global Schema Builder is the main module that interacts with the wrappers and the Service tools to generate the Global Schema. It is composed of different components that implements the different steps of the integration process described in Section 1.4.

- **Service tools:** the Service tools are important modules exploited by the Global Schema Builder and the Query Manager. These include the software module in charge of managing the WordNet lexical database used for the annotation phase, the Artemis tool implementing the clustering algorithm, and the ODB-Tools reasoner used to calculated inferred relationships.

- **Query manager:** the Query Manager is the component in charge of solving a user query: it generates the local queries queries for wrappers, starting from a global query formulated by the user on the global schema. The Query Manager provides a graphical Query Composer to graphically formulate queries on a Global Schema described in [77].

1.7 Related Work

In the area of heterogeneous information integration, many projects based on mediator architectures have been developed. The mediator-based TSIMMIS project [28] follows a structural approach and uses a self-describing model (OEM) to represent heterogeneous data sources and the MSL (Mediator Specification Language) rule to enforce source integration. In TSIMMIS, by means of MSL, arbitrary views (in particular, recursive views) can be defined at the mediator layer. The MOMIS system made a different choice: starting from the semi-automatic generated mappings between global and local attributes stored in the mapping tables, views (global classes) are defined by means of a predefined operator, i.e. the full disjunction, that has been recognized as providing a natural semantics for data merging queries. In particular, in the view definition resolution functions are defined to take into account data conflicts.

SIMS [2] proposes the creation of a global schema by exploiting the use of Description Logics (i.e., the LOOM language) for the description of information sources. The use of a global schema allows both GARLIC and SIMS projects to support every possible user queries on the schema instead of a predefined subset of them.
The Information Manifold system [49] provides a source independent and

query independent mediator. The input schema of Information Manifold is a set of descriptions of the sources and the integrated schema is mainly defined manually by the designer, while in our approach it is tool-supported.

The goal of Clio [53] is to develop a tool for semi-automatically creating mappings between two data representations (i.e., with user input). First of all, in the Clio framework the focus is on the schema mapping problem in which a source is mapped onto a different, but fixed, target schema, while the focus of our proposal is the semi-automatic generation of a target schema, i.e. the Global Virtual View, starting from the sources. Moreover, the semi-automatic tool for creating schema mappings, developed in Clio, employs a mapping-by-example paradigm that relies on the use of value mappings describing how a value of a target attribute can be created from a set of values of source attributes. Our proposal for creating schema mappings can be considered orthogonal with respect to this paradigm. In fact, the main techniques of mapping construction rely on the meanings of the class and attribute names selected by the designer in the annotation phase and by considering the semantic relationships between meanings coming from the common lexical ontology. On the other hand, MOMIS and CLIO share a common mapping semantics among a (target) global schema and a set of source schemata expressed by the full-disjunction operator.

Infomaster [38] provides integrated access to multiple distributed heterogeneous information sources giving the illusion of a centralized, homogeneous information system. The main difference of this project w.r.t. our approach is the lack of a tool aid-support for the designer in the integration process.

Capitolo 2

Getting Through the THALIA Benchmark with MOMIS

In this chapter we show how the MOMIS data integration system can deal with all the twelve queries of the THALIA benchmark, a public available testbed and benchmark for information integration systems [44], by simply extending and combining the declarative translation functions already available in MOMIS and without any overhead of new code. This is a remarkable result, in fact, as far as we know, no system has provided a complete answer to the benchmark [10].

2.1 Introduction

As said in the previous chapter, during the last decade many data integration systems characterized by a classical wrapper/mediator architecture [91] based on a Global Virtual Schema (Global Virtual View - GVV) have been proposed. The data sources store the real data, while the GVV provides a reconciled, integrated, and virtual view of the data sources. Modelling the mappings among sources and the GVV is a crucial aspect.

Each mediator system proposed tried to solve as much as possible the integration problem, focusing on different aspects to provide a (partial) answer to one or many challenges of the problem, ranging from system-level heterogeneities, to structural syntax level heterogeneities at the semantic level. The approaches still rely on human interventions, requiring customization for data reconciliation and writing specific not reusable programming code. The specialization of the proposed mediator system makes the comparison among the systems difficult. Therefore, the last Lowell Report

[1] has provided the guidelines for the definition of a public benchmark for the information integration problem. The proposal is called THALIA (Test Harness for the Assessment of Legacy information Integration Approaches) [44], and it provides researchers with a collection of downloadable data sources representing University course catalogues, a set of twelve benchmark queries, as well as a scoring function for ranking the performance of an integration system. THALIA benchmark focuses on syntactic and semantic heterogeneities in order to pose the greatest technical challenges to the research community.

In this chapter we show how the MOMIS mediator system can deal with all the twelve queries of the THALIA benchmark, a public available testbed and benchmark for information integration systems [44], by simply extending and combining the declarative translation functions available in MOMIS and without any overhead of new code. This is a remarkable result, in fact, as far as we know, no system has provided a complete answer to the benchmark. More precisely, we developed an extension of the system devoted to the support of syntactic and semantic data heterogeneities by means of declarative Mapping Data Transformation Functions (MDTFs) that avoids the overhead due to write ad-hoc hard-coded transformation functions. MDTFs allow MOMIS to deal with all the twelve queries of the THALIA benchmark.

This Chapter is structured as follows. Section 2.2 presents the THALIA benchmark and Section 2.3 the MOMIS integration methodology applied to THALIA. Section 2.4 defines the mapping refinements that are used for the global query translation. Section 2.5 describes the query rewriting with MDTFs, Section 2.6 reports the experimental results with THALIA and, finally, Section 2.7 concludes our work.

2.2 The THALIA Benchmark

THALIA is a public available testbed and benchmark for information integration systems [44]. It provides over 40 downloadable sources representing University course catalog from computer science around the world. The goal of the benchmark is a systematic classification of the different types of syntactic and semantic heterogeneities that are described by the twelve queries provided.

For each case, a benchmark query is formulated and it is applied (ea-

sily) to a target schema as well as a challenge schema which provides the heterogeneity solved by the integration system. The heterogeneities are divided into three different categories: Attribute Heterogeneities (Query 1, 2, 3, 4 and 5), Missing Data (Query 6, 7, 8) and Structural Heterogeneities (Query 9, 10, 11, 12).

Attribute Heterogeneities: inconsistencies that exist between two single attributes in different schemata.

The simplest is attribute synonyms, where the same information is stored in attributes with different names. In `Query 1` 'instructor' could be thought as a synonyms of 'lecturer'.

A simple mapping heterogeneity is referred to attributes that differ by a mathematical transformation. In `Query 2` the two attributes contain a time value in 12 and 24 hours, respectively.

Union types: in many cases attributes in different schemas use different data types to store the same information: in `Query 3` the target schema uses a string attribute to indicate the course name while the challenge schema uses a string description including the name and the link of the course.

Complex mapping: it is the case where related attributes differ by a complex transformation of their values: for example `Query 4` contains a number of credit hour in the target schema and a string description of the expected work in the challenge schema.

A typical real case involves two sources where the same information is denoted in different languages, giving rise to a language expression heterogeneity. `Query 5` proposes a target schema where the course name is expressed in English and a challenge schema in German.

Missing data: heterogeneities due to missing information (value or structure) in one of the schemas.

Nulls treatment and Semantic incompatibility: distinguishes the cases where an attribute does not exist in a source from the one where the attribute has a null value in a particular record. In `Query 6`, course book is not present in the challenge schema and it is present only for some records in the target schema. `Query 8` proposes a target schema with a

student classification that is not present in the challenge schema.

Virtual column: the information could be explicit in a source and only implicitly available in the others. `Query 7` gives an example where the course prerequisites are present together with the course description in the challenge schema.

Structural Heterogeneities: heterogeneities due to missing information (value or structure) in one of the schemas.

Structural heterogeneity of an attribute: the same attribute may be located in different position in different schemas. `Query 9` propose a target schema with the room attribute in the course relation and a challenge schema where the room information is an element of section that is a part of a course.

Handling sets: a single attribute contains a string that describes a set of values in a schema, while the same information is split into single attributes in an other schema. `Query 10` contains a target schema with a single lecturer attribute of course and a challenge schema where a professor is defined for each section of the course.

Attribute name does not define semantics: a typical case where the attribute name does not refer to the semantics of the contained information. `Query 11` contains a challenge schema where the lecturer are spread in three different attributes whose names are the teaching periods.

Attribute composition: complex data can be represented either as a single string or a set of attributes. `Query 12` contains a challenge schema where the title, day and time of a course are contained in a single attribute rather then in three different attributes.

2.3 MOMIS Integration Methodology applied to THALIA

In the following we describe the MOMIS Ontology generation process by means of three different THALIAs queries (query 4, 7 and 12), each one referring to a different heterogeneity category. In this section, we apply the process of GVV generation (described in 1.4) to schemata and data provided by THALIA benchmark.

2.3.1 GVV Generation applied to THALIA

In this section we apply the MOMIS integration methodology applied to THALIA. The GVV Generation process applied to THALIA can be outlined as follows (see Figure 1.1):

1. Extraction of Local Source Schemata: All schemata and data provided by THALIA benchmark (10 schemas and 701 records) are automatically wrapped into a relational database.

2. Local Source Annotation: Terms denoting schema elements in data sources are semantically annotated according to a common lexical reference in order to provide a shared meaning to each of them. We choose the WordNet (wordnet.princeton.edu) database as lexical reference. Considering the THALIA schemata, the recall rate of the terms automatically annotated in the sources is 80%, with a precision of 82%.

3. Common Thesaurus Generation: MOMIS builds a Common Thesaurus that describes intra and inter-schema knowledge in the form of: synonyms (SYN), broader terms/narrower terms (BT/NT), meronymy/holonymy (RT) relationships. The Common Thesaurus is constructed by extracting new relationship between the local schemas.

4. GVV generation: Starting from the Common Thesaurus and the local sources schemata, MOMIS generates a GVV consisting of a set of global classes, plus mappings to connect the global attributes of each global class and the local sources attributes. The mappings which have been automatically created by the system can be fine-tuned by means of the MDTFs, as will be discussed in next section. For each GVV involving a THALIAs query, MOMIS automatically detect the right basic relations and attributes mapping. For example, the GVV referred to Query 1 contains the mapping between the Instructor and Lecturer attributes, i.e. the THALIA foreseen challenge. For Query 12, the system recognizes the mapping between CourseTitle and Title of the two schemata, while the challenge, i.e. information contained in a unique attribute rather than separate attributes, is obtained by specifying a Data Transformation Function.

5. GVV annotation: The GVV is automatically annotated, i.e. each of its elements is associated to the broadest meanings extracted from the annotated sources. The annotation of a GVV is a significant result, since these metadata may be exploited for external users and applications interoperability. As an example, we report a fragment of

annotated GVV (Query 12) in the figure 2.4.

Global class	Local Classes	Meaning from WordNet
Course	Cmu.Course, Brown.Course	Course#1
Instructor	Cmu.Lecturer, Brown.Instructor	Instructor#1
Title	Cmu.CourseTitle, Brown.Title	Title#1

Figura 2.1: GVV annotation applied to the GVV of Query 12 of THALIA

2.4 Mapping Refinement

As explained in section 1.4.1, after the GS generation process, each global class G is associated to a Mapping Table and starting from the Mapping Table of G, the integration designer can define the mapping query $\mathcal{MQ}^G$ associated to the global class G. The first step of the mapping query definition is *Data Transformation*, to transform local attribute values into a global attribute. In the following, we introduce declarative *Mapping Data Transformation Functions* (*MDTFs*) which allow MOMIS to deal with all the twelve queries of the THALIA benchmark.

2.4.1 Mapping Data Transformation Functions

The Integration Designer may define, or refine, for each element $MT[GA][L]$, a Mapping Data Transformation Function, denoted by $MDTF[GA][L]$, which represents the mapping of local attributes of L into the global attribute GA. $MDTF[GA][L]$ is a function that has to be executable/supported at the local source by the local source wrapper. In fact, we want to push as much as possible function execution at the sources, as suggested in [27] for constraint mappings.

MDTFs are obtained by combining SQL-92 functions, classic string manipulation functions available in MOMIS.

The following SQL-92 like functions are accepted:

CHAR_LENGTH: returns the length of a string

POSITION: searches a pattern in a string

SUBSTRING: returns a part of a string

CAST: converts a value from a type to another

CASE WHEN THEN: transforms a record on the basis of a specific data value

In addition, the classic RIGHT and LEFT string functions, i.e. the first (or the last) n characters of a string, are available in the system.

The above functions are executed at the wrapper level by the right translation of the particular SQL-dialect for relational database systems and are built-in for other wrapper (like XML).

We added a specific function for datetime type conversion:

TIME 12-24: transforms a string to a time value expressed in 12 or 24 hours format.

Example Query 4: a complex transformation function is required to convert the string that describes the expected scope of the course in ethz into a credit unit. At the ETH's Computer Science site is present the conversion formula:

```
#KE = #V + #U + 1
```

that the designer inserts in the mapping table by specifying the following MDTF:

```
MDTF[Unit][ethz.Unterricht] =

CAST(SUBSTRING(Umfang, POSITION('V' IN Umfang)- 1, 1) AS int)
+
CAST(SUBSTRING(Umfang, POSITION('U' IN Umfang)- 1, 1) AS int)+ 1
```

Example Query 7: the following MDTF infers that the course has a prerequisite course by extracting the text following the Prerequisite term.

```
MDTF[prerequisite][asu.Course] =
CASE POSITION('%Prerequisite%' IN Description)
WHEN 0 THEN 'None'
            ELSE RIGHT(Description,
               CHAR_LENGTH(Description) -
                  POSITION('%Prerequisite%' IN Description)+ 1)
END
```

Example Query 12: the challenge is to extract the correct title, day and time values from the title column in the catalog of the Brown University that contains all the above information in a string. By using a combination of SUBSTRING and POSITION functions it is possible to obtain what requested.

```
MDTF[Title][brown.Course] =

SUBSTRING(Title FROM POSITION('/"' IN Title) + 3 FOR
        POSITION ('hr.' IN SUBSTRING(Title FROM
                 POSITION('/"' IN Title) + 3 FOR 100)) - 1)



MDTF[Day][brown.Course] =

SUBSTRING(Title FROM POSITION('hr.' IN Title) + 4 FOR
POSITION(' ' IN SUBSTRING(Title FROM
                      POSITION('hr.' IN Title) + 4 FOR 10)))

MDTF[Time][brown.Course] =

      SUBSTRING(Title IN POSITION(' ' FROM SUBSTRING(Title
                FROM POSITION('hr.' IN Title) + 4 FOR 10)) +
                          POSITION('hr.' IN Title) + 4 FOR 15)
```

The transformation of a local class L obtained by applying all the Mapping Data Transformation Functions $MDTF[GA][L]$ is denoted with $T(L)$.

2.4.2 Mapping Query

After the definition of Mapping Data Transformation Functions, we need to define Join Conditions and Resolution Functions. as already explained in section 1.4.1 . In this section, we just discuss the definition of Join Conditions and Resolution Functions for an example related to THALIA.

We consider the global class `Course` related to Query 1 (see section 2.6); the mapping table is shown in Figure .

The **Join Conditions** among pairs of local classes belonging to the same global class are introduced in order to identify instances of the same object and fuse them. In the case of the global class `Course` we have the two local class `L1=gatech.Course` and `L2=cmu.Course`; the join condition is performed on the transformed values related to the `CourseXListed` global attribute, i.e.

```
JC(L1,L2) = MDTF[CourseXListed][L1] = MDTF[CourseXListed][L1]
```

Regarding **Resolution Functions**, for solving data conflicts may be defined for each global attribute mapping onto local attributes coming from more than one local source, in the case of the global class `Course` we have that the global attribute `Lecture` is either mapped into the local attribute `L1.Instructor` and `L2.Lecture`; in this case we may define a precedence function for the global attribute `Lecture`:

`L1.Instructor` has a higher precedence than `L2.Lecture`

2.5 Query Rewriting with MDTFs

To answer a query expressed on the GVV (global query) the query must be rewritten as an equivalent set of queries expressed on the local schemata (local queries); this query translation is performed by considering the mappings among the GVV and the local schemata. In a GAV approach query translation is performed by means of query unfolding, i.e., by expanding a global query on a global class G of the GVV according to the definition of the mapping query $\mathcal{MQ}^G$ as defined in section 1.4.1 which describe the MOMIS query unfolding method including $MDTFs$ (more details also in section 1.5.1).

2.5.1 Multilingual Query Condition

In an information integration scenario, data are frequently expressed in different languages, for example a source contains english data and another german or italian data.
MOMIS supports multilingual query conditions expressed by means of a multilingual constraint in the form:

GA op TRANSLATE(term,Language)

where GA is a global attribute and term is a word of a given Language. The condition of a Global Query is then a Boolean expression of atomic and multilingual constraints.
The rewriting of a multilingual constraint is performed by a TRANSLATION function that translate from the language specified in TRANSLATE(term,Language) into the different languages of the local sources. The translation is obtained by exploiting the open dictionary of the Gutenberg Project (www.gutenberg.org) and determining for each term, the translation

in the language of the local source, that is a metadata associated to each source during the integration phase.
More precisely, the rewriting of a multilingual constraint works as follows. Given two languages, Language1 and Language2, and a term of Language1, we consider a function:

TRANSLATION(term,Language1,Language2)

whose result is a set of terms of Language2: $term\_1, ..., term\_n$, with $n \geq 1$. To perform the constraint mapping of a multilingual constraint:

AG LIKE TRANSLATE(term,Language)

w.r.t. a local class L we consider a preliminary step where the multilingual constraint is transformed into a disjunction of atomic constraints:

GA LIKE term_1 OR .... OR GA LIKE term_n

where term_i, $1 \leq I \geq n$, is a term obtained by:

TRANSLATION(term,Language,Language2)

and Language2 is the language of the local class L. Then the disjunction of atomic constraints is mapped into the local class L attributes as discussed before.

Example Query 5: the challenge is related to the expression of the attribute values in different languages. For example, in the target schema the course name is expressed in english and in the challenge schema in german. The global query submitted to the MOMIS Query Manager is the following:

```
SELECT Name
FROM Course
WHERE Name LIKE TRANSLATE(%Database%, en)
```

For the target schema (umd), which is in English, no translation is required, so the local query is:

```
SELECT CourseName
FROM Course
WHERE CourseName LIKE %Database%
```

For the challenge schema (ethz) where the language is german, the global query is translated in the following local query:

```
SELECT Titel
FROM Unterricht
WHERE Titel LIKE '%Datenbank%'
OR Titel like '%Datei%'
OR Titel like '%Datenbasis%'
```

A complete example of Query rewriting will be shown in section 2.6.2.

2.6 THALIA Benchmark: Experimental Results

This section describes the results of MOMIS applied to the THALIA benchmark. MOMIS is fully written in Java and is available on-line via Java Web Start (http://www.dbgrou.unimo.it/momisjws).
The THALIA benchmark provides the twelve queries in XML format, while MOMIS provides an SQL-like syntax as a query language; we transformed the queries in SPJ query by a straightforward operation, with no effect on the benchmark result. In addition, our XML-Schema wrapper generates a relational view of a schema and automatically load the xml data file into a relation database, thus each data set provided by THALIA is loaded by the wrapper in a specific database.

2.6.1 Integration Phase

During this phase, for each pair of target and challenge schema related to a single query, the designer semi-automatically creates a specific GVV, i.e. twelve GVVs has been deployed for the benchmark.

Since the reference schemas are very simple, the GVVs creation was simple and completely automatic, while the designer had to carefully work on mapping refinements. As an example, we describe the GVV creation related to Query 1.

The reference schemas are of Georgia Tech University and of Carnegie Mellon University. Georgia Tech University schema contains only Course class with many attributes such as:

Title, Section, Instructor, Room, Description,...

Also Carnegie Mellon University contains only Course class with attributes like:

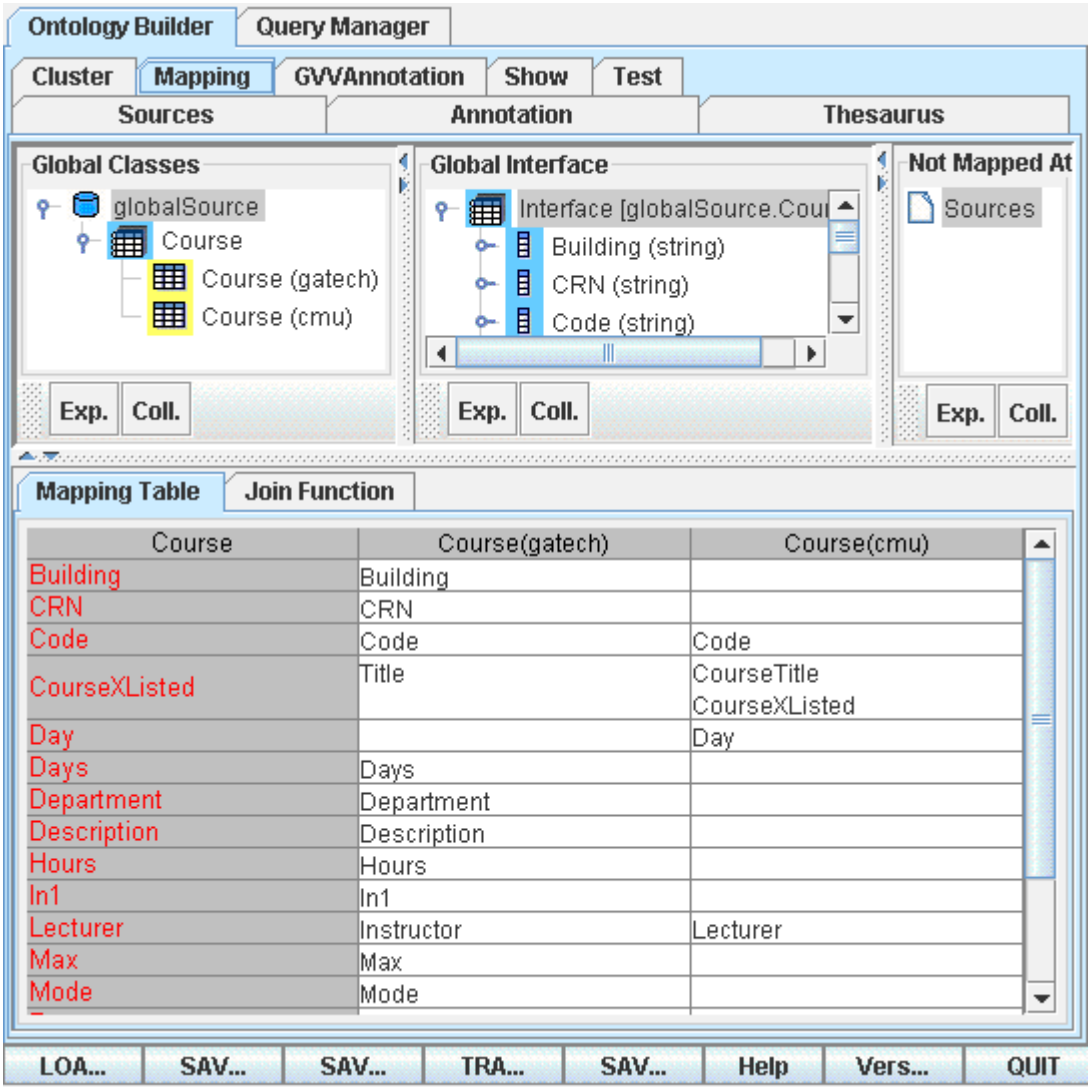

Figura 2.2: MOMIS Schema Mapping example

CourseTitle, Room, Lecturer, Time, Unit,...

The automatic Global Class obtained is shown in Figure 2.2, where it is possible to note that the Lecturer and Instructor attributes of the sources are mapped into a single global attribute.

During the mapping refinement phase, for each query, a specific composition of mapping functions has been inserted to overcome the challenge. Appendix B reports the mapping refinement for each query, while the

following table summarize the MDTF functions used for each query challenge:

Attribute Heterogeneities	
Query 1	Only attributes mapping
Query 2	TIME 12-24, SUBSTRING
Query 3	SUBSTRING, POSITION
Query 4	CAST, SUBSTRING, POSITION
Query 5	TRANSLATE
Missing Data	
Query 6	Attributes mapping, NULL treatment
Query 7	CASE WHEN ... THEN, CHAR_LENGTH, RIGHT, POSITION
Query 8	Attributes mapping, NULL treatment
Structural Heterogeneities	
Query 9	SUBSTRING, POSITION
Query 10	SUBSTRING, POSITION
Query 11	CASE WHEN ... THEN, CHAR_LENGTH
Query 12	SUBSTRING, POSITION

Figura 2.3: MDTF functions used for each query

2.6.2 Query Phase

During this phase, the writes the queries over the GVVs (see appendix B). MOMIS provides a command line interface for queries and a grid interface for the data answer.

To show a complete example of query processing we consider the following query:

Example Query 4: the benchmark query List all database courses that carry more than 10 credit hours is formulated as follows:

```
Select Title, Units
from Course
where Title LIKE TRANSLATE('%Database%', en)
and Units > 10
```

The (portion of) the Mapping Table of the class Course involved in the query is the following: This global query is automatically rewritten for the target schema (cmu) and for the challenge schema (ethz) as:

Course	ethz.Unterricht	Cmu.Course
Title	Titel	CourseTitle
Units	MDTF[Unit][ethz.Unterricht]	Units

Figura 2.4: Portion of Mapping table of The Global Class Course

```
Local Query 1 - Source "cmu"  (LQ1)

SELECT Course.CourseTitle, Course.Units
FROM Course
WHERE (CourseTitle) like ('%Database%')
AND Units > 10

Local Query 2 - Source "ethz" (LQ2)

SELECT Unterricht.Titel,
(CAST(SUBSTRING(Umfang,CHARINDEX('V',Umfang) - 1, 1) AS int)
+
CAST(SUBSTRING(Umfang,CHARINDEX('U',Umfang) - 1, 1) AS int) + 1)
AS Umfang
FROM Unterricht
WHERE ((Titel) like ('%Datenbank%') or (Titel) like ('%Datei%')
or
(Titel) like ('%Datenbasis%')) AND (CAST(SUBSTRING(Umfang,
CHARINDEX('V', Umfang) - 1, 1) AS int) +
CAST(SUBSTRING(Umfang,
CHARINDEX('U',Umfang) - 1, 1) AS int) + 1) > 10
```

The above local queries are executed at the local sources and then the Full Outerjoin (FOJ) operation of their results is computed as follows (see 1.4.1.4 for details):

```
Full Outerjoin computation (FOJ)

    Select LQ1.CourseTitle AS Title_1, LQ2.Titel AS Title_2,
           LQ1.Units as Units_1, LQ2.Umfang AS Units_2
    from LQ1 full outerjoin LQ2 on LQ1.CourseTitle = LQ2.Titel
```

The result of the global query is obtained by applying Resolution Functions to the above FOJ query:

```
     Select resolution(Title_1, Title_2) AS Title,
```

```
            resolution(Units_1, Units_2) as Units
from FOJ
```

The records obtained after the query execution are shows in a grid, where, for each attribute of a single record, the user can visualize the local source that has provided the data.

2.6.3 Experimental Comparison

Three different integration systems have reported the THALIA benchmark results: Cohera, Integration Wizard (IWIZ) [44] and a 'Keyword Join' system [94]. In the following table we compare the results, with the specification of extra effort for query answer.

Query	**Cohera**	**Integration Wizard**	**'Keyword join**
Query 1	Yes	Yes, SMALL	Yes
Query 2	Yes, SMALL	Yes, SMALL	NO
Query 3	Yes, MODERATE	Yes, MODERATE	Yes
Query 4	NO	NO	Yes, difficult
Query 5	NO	NO	Yes, difficult
Query 6	Yes	Yes, MODERATE	Yes
Query 7	Yes, MODERATE	Yes, MODERATE	NO
Query 8	NO	NO	NO
Query 9	Yes	Yes, SMALL	Yes, need semantic metadata
Query 10	Yes	Yes, SMALL	Yes, need semantic metadata
Query 11	Yes, MODERATE	Yes, MODERATE	Yes
Query 12	Yes, MODERATE	Yes, MODERATE	Yes

Figura 2.5: Experimental Result

SMALL: small amount of code
MODERATE: moderate amount of code

Summarizing, Cohera and IWIZ can solve 9 queries, some of these by adding a significant amount of code, while the system presented in [50] could deal with 5 queries easily, and another 2 queries with a small amount of metadata, without any custom code.

In MOMIS, by means of declarative functions, it is very easy to deal with all the 12 queries: we estimate that an integration designer might

define the requested customization (reported in Appendix B) within 4-6 hours.

2.7 Conclusion

In this chapter we presented MOMIS, extended with MDTFs and proved it satisfies all the challenges provided by the THALIA benchmark. Future work will be devoted to enhance the TRANSLATION function by exploiting altavista babelfish translator (http://babelfish.altavista.com), by means of the API provided by Jonathan Feinberg (http://babel.mrfeinberg.com).

Capitolo 3

Query Optimization in MOMIS

The goal of chapter 3 is to propose new techniques that consider the optimization of full outerjoin operation in MOMIS. Full outerjoin is used in data integration systems for merging multiple records representing the same real-world object into a single, consistent, and clean representation. Our optimization is mainly defined as a substitution of full outerjoin operator by the left(right) outerjoin or inner join operators [51, 14]. A query manager that merge information using full outerjoin can benefit easily from this optimization technique.

3.1 Introduction

The research community has been developing techniques for integrating heterogeneous data sources for the last twenty years. One of the main motivations is that data often reside in different, heterogeneous and distributed data sources that users need to access in a single and unified way to gather a complete view of specific domain. A common approach for integrating information sources is to build a mediated schema as synthesis of them. By managing all the collected data in a common way, a global schema allows the user to pose a query according to a global perception of the handled information. A query over the mediated schema is translated into a set of sub-queries for the involved sources by means of automatic unfolding-rewriting operation taking into account the mediated and the source schema. Results from sub-queries are finally unified by data reconcialiation techniques.

Integration systems relying on mediator-based architectures have been developed, and most of them are able to integrate at the same time

heterogeneous data sources, i.e sources that represent their contents with relational, object-oriented and semi-structured (XML and its derived standards) models.

We remember that more data integrations systems use the full outerjoin to fuse data coming from different local data sources [72, 57, 28]. Since the full outerjoin combines local data sources to form the global query's answer, it is too expensive to process. The goal of this chapter is to present algorithms that consider the optimization of full outerjoin expression. Especially, we focus on conjunctive global query that consist of selection condition of terms connected with AND's. Our optimization is defined as a substitution of full outerjoin operator by the left(right) outerjoin or inner Join operators [51, 14]. To obtain the set of optimized query plan execution, we modified the algorithm defined in [37].

In the literature, most authors proposed different algorithms to optimize full outerjoin expresion in data integration or multidatabase systems [29, 35, 37, 57, 76, 96, 73, 30]. Most of these algorithms are not easy to be applicable on MOMIS. So our work consist to study the optimization of full outerjoin in the context of MOMIS/SEWASIE Integration System.

The Chapter is structured as follows. Section 3.2 presents the related works, section 3.3 is an introductory example. Section 3.4 presents the optimization of full outerjoin expression in the data fusion process and section 3.5 is the generalization of simplification techniques for full outerjoin. Finally, section 3.6 concludes our work.

3.2 Related Works

Outerjoin Optimization in multidatabase systems was presented in [29]. The author describes query processing strategies that can be applied to the multidatabase systems. An approach to optimize outerjoin processing is characterized by some important aspects:

1. The use of target attributes which are select attributes and joining attributes;

2. The outerjoin is used only when the target attributes span more than one local relation in either the same database or different databases;

3. The outerjoins may be changed to a one-side outerjoin or inner join.

Outerjoin is used for integrating local schema to provide views of the multidatabase. After the query modification process, a query could contain outerjoin, join, select, and projection operations. Based on the structure of the query and the definition of the schemas, the modified queries are optimized.

In [37], the author proposed the **Outerjoin Simplification and Reordering** techniques which are more useful to generate alternative evaluation orders for the optimization of queries containing both joins and outerjoins. The algorithm developed in this work allows to perform the following tasks:

1. Generation of all alternative plans for queries containing both outerjoin and inner join;
2. The use of associative identities to justify changing the order of evaluation for join operators;
3. Introduction of new operator called a Generalized Outerjoin operator (GOJ)
4. A simplification allows the replacement of outerjoins operator by left-outerjoin(right-outerjoin) or inner join in some cases.

Integration of Maximun Information Using Outerjoins was developed in [57]. The author presents an algorithm for creating an optimized query plan to retrieve maximum information from multiple relations using outerjoin operator. Particularly, he focuses on conjunctive queries in the presence of predicates and foreign functions. The algorithm proposed by the author outlined the following points:

1. It deals with only conjunctive queries which contain conditions composed of terms connected with AND's;
2. The execution order of relation in the tree is constrained by the fact to merge all relations indispensables with inner join.
3. Replacement of outerjoin by left outerjoin (right outerjoin) or inner join;
4. It is applicable only to γ-acyclic hypergraphs.

The proof of the correctness of algorithm is an open research issues.

Many other works [35, 57, 76, 96, 73, 30] exist, and each of these works deals with different aspects of full outerjoin optimization but none takes into account the problem that we need to solve.

3.3 An Introductory Example

We consider a conjunctive query on a global class G of the GVV expressed in an SQL syntax as follow:

```
select A_1, ..., A_h
from G
  where pred_1 and ... and pred_k
```

where A_i are global attributes of G, *pred* is a (simple) predicate : $A\ op\ value$

We assume that a tourist wants to know the rates of different hotels including the stars. He seeks especially the rooms that cost 100 euros in hotels with at least 4 stars. He will write the following query Q1 on the global class `Hotel`:

```
Q1 = select Name, Room
     from Hotel
     where Price = 100 and Stars > 3
```

He will also wants to know all rooms with the wifi connection in hotels with at least 4 stars:

```
Q2 = select Name, Room
     from Hotel
     where Wifi = true and Stars > 3
```

In this example, the global Class G is `Hotel` (figure 3.2); the local sources $L1$, $L2$ are respectively `resort` and `hotel`. We use $\perp$ to denote the *null value*. The answer of the Query Q1 is:

Name	Room
Hilton	12, 13
Ibis	6
Garden	24,18

For the query Q2 is:

Name	Room
Kyriad	4

resort

name	stars	amount	rooms
Hilton	5	120	12
Kyriad	5	350	⊥
Sofitel	⊥	210	40
Ibis	4	100	6
Garden	4	100	24
Continental	3	225	122

hotel

name	price	hotelrooms	wifi
Hilton	80	13	false
Kyriad	400	4	true
Garden	100	18	⊥
Sofitel	100	6	true
Madison	100	⊥	false
Dream	200	35	true

Figura 3.1: Instances of local classes `hotel` and `resort`

Hotel

Name	Stars	Price	Room	Wifi
Hilton	5	100	12, 13	false
Kyriad	5	375	4	true
Sofitel	⊥	155	6, 40	true
Ibis	4	100	6	⊥
Garden	4	100	18, 24	⊥
Continental	3	225	122	⊥
Madison	⊥	100	⊥	false
Dream	⊥	200	35	true

Figura 3.2: Instances of global class Hotel computed as the Full outerjoin-merge between `resort` and `hotel`

To answer a global query on `Hotel`, the query must be rewritten as an equivalent set of queries (*local queries*) expressed on the local classes `resort` , `hotel` belonging to `Hotel`.

This query rewriting is performed by considering the mapping between the GVV and the local schemata; in a GAV approach the query rewriting is performed by means of *query unfolding*, i.e., by expanding the global query

on G according to the definition of its mapping query $\mathcal{MQ}^G$.
The query unfolding of a global query Q produces, for each local class L belonging to G, a local query `Q_L`

```
Q_L = select S_L
      from L
      Where C_L
```

where the select list `S_L` is a list of local attributes of L and the condition `C_L` is a conjunction of (simple) predicates on L. These local queries are executed

Hotel	**resort**	**hotel**
Name	name	name
Room	rooms	hotelrooms
Price	amount	price
Stars	stars	-
Wifi	-	wifi

Figura 3.3: Mapping Table of global Class Hotel

on the local sources and local queries answers are then fused by means of the mapping query $\mathcal{MQ}^G$ to obtain the answer of the global query.
We defined this query execution process in several papers [6, 9]. The novelty contribution of this work is to describe how conjunctive queries are used in the query unfolding process. In particular, we will show how selection condition in conjunctive form are useful to perform query optimization.
The process for executing the global query consists of the following steps:

1. **Computation of Local Query predicates:**
   In this step is evaluated whether a predicate of the global query may be pushed down to the local level. Given $pred_i$, let A_i be the global attribute used in $pred_i$. A predicate $pred_i$ is pushed down on the local class L only if:

   (1) $MT[A_i][L]$ is not null, i.e, exist LA in L with $LA = MT[A_i][L]$ **and**

   (2) A_i is an homogeneous attribute

   Condition (1) and (2) are straightforward.
   For the query Q1, since the global attribute `Stars` is mapped only in `resort`, and then it is homogeneous, the predicate `(Stars > 3)` is pushed down for the local class `resort`. On the other hand, the global

attribute `Price` is not homogeneous (it is defined by means of the `AVG` function) and then the predicate `(Price = 100)` cannot be pushed at the local level: since the `AVG` function is calculated at a global level, the predicate may be globally true but locally false. Thus, for the query Q1 we obtain the following local query conditions:

`C_hotel: true` (i.e. the local query does not have a where condition)
`C_resort: stars > 3`

For query Q2, the translation of the predicate `(Stars > 3)` on local class `resort` is the same process observed for query Q1. The predicate `(Wifi = true)` is pushed down only in `hotel` because the global attribute `Wifi` is mapped only in `hotel`, and then it is homogeneous. In this way, for query Q2 we obtain the following local query conditions:

```
C_hotel:  wifi = true
C_resort:  stars > 3
```

2. **Computation of Residual Conditions:** A predicate on a global attribute A_i that in step (1) is pushed down on all local classes, is already solved, i.e. in the full join of the local queries this predicate is already satisfied; otherwise the predicate is considered as residual and have to be solved at the global level.

   For the query Q1, residual conditions are:

   ```
   Price = 100 and Stars > 3
   ```

   while for the query Q2, residual conditions are:

   ```
   Wifi = true and Stars > 3
   ```

3. **Generation and execution of local queries:** The select list S_L is obtained as X where

   (a) X = union of the attributes of the global select list, of the Join Condition and of the Residual Condition; these attributes are transformed into the corresponding ones at the local level on the basis of the Mapping Table.

With the step 3, query unfolding ends and local queries are generated; for the query Q1 we obtain

```
Q_hotel_1  =  select name, price, hotelrooms from hotel

Q_resort_1 =  select name, stars, amount, rooms from resort
              where stars > 3
```

and for the query Q2 we obtain

```
Q_hotel_2  =  select name, hotelrooms, wifi from hotel
              where wifi = true

Q_resort_2 =  select name, rooms, stars from resort
              where stars > 3
```

Q_resort_1

name	stars	amount	rooms
Hilton	5	120	12
Kyriad	5	350	⊥
Ibis	4	100	6
Garden	4	100	24

Q_hotel_1

name	price	hotelrooms
Hilton	80	13
Kyriad	400	4
Garden	100	18
Sofitel	100	6
Madison	100	⊥
Dream	200	35

Figura 3.4: Instances of local query answers for Q1

Q_resort_2

name	stars	rooms
Hilton	5	12
Kyriad	5	⊥
Ibis	4	6
Garden	4	24

Q_hotel_2

name	hotelrooms	wifi
Kyriad	4	true
Sofitel	6	true
Dream	35	true

Figura 3.5: Instances of local query answers for Q2

4. **Fusion of local answers** :

   The local answers are fused into the global answer on the basis of the mapping query $\mathcal{MQ}^G$ defined for G. As intuitively discussed in Section 1.4.1.4, $\mathcal{MQ}^G$ is defined by using a full outerjoin-merge operation which, given the local answers Q_L1 and Q_L2, is substantially performed in two steps:

   (a) Computation of the full outerjoin between the local query answers `Q_L1` and `Q_L2` of `Q`, denoted with `FOJ_Q`, on the basis of the join condition `JC(L1,L2)` defined between `L1` e `L2`[1]:

   ```
   FOJ_Q = Q_L1 full outerjoin Q_L2 on JC(L1,L2)
   ```

   In our example `JC(L1,L2): L1.name = L2.name`.

   The results of full outerjoin between `Q_resort_1` and `Q_hotel_1` for `Q1`, i.e. `FOJ_Q1`, and between `Q_resort_2` and `Q_hotel_2` for Q2, i.e. `FOJ_Q2`, are shown in figure 3.6 and 3.7 respectively.

   (b) **Application of the Resolution Functions :** for each pairs of attributes in FOJ_Q (except for attributes used in the join condition) mapped in the same global attributes the related Resolution Function is applied. For example, in query Q1, we need to apply the resolution function `AVG` and `CONCATENATE` respectively to global attribute `Price` and `Room`.

Name	Stars	Price	Room
Hilton	5	100	12, 13
Kyriad	5	375	4
Sofitel	$\perp$	100	6
Ibis	4	100	6
Garden	4	100	18, 24
Madison	$\perp$	100	$\perp$
Dream	$\perp$	200	35

Figura 3.6: Instance of FOJ_Q1 for Q1

[1] In this introductory example with only two local class, we don't use the representation with the `FOJ` sequence, i.e., `FOJ_Q = ⟨ Q_L1, FJ, Q_L2 ⟩`, but we write the join operator explicitly.

Name	Stars	Room	Wifi
Hilton	5	12	⊥
Kyriad	5	4	true
Sofitel	⊥	6	true
Ibis	4	6	⊥
Garden	4	24	⊥
Dream	⊥	35	true

Figura 3.7: Instance of FOJ_Q2 for Q2

At this point we introduce our main query optimization techniques, i.e. full outerjoin simplification.

A predicate $pred_i$ is supported in the local class L only if:

$MT[A_i][L]$ is not null, i.e, exist LA in L with $LA = MT[A_i][L]$

Let $P = pred_1$ *and* $\ldots$ *and* $pred_k$; we consider a local class L, we say that P is supported in L if:

$pred_i$ is supported in L , for each i.

In step 4.a the computation of the full outerjoin operation can be optimized by reducing it to a left/right outerjoin or inner join on the basis of the following rules:

(1) `FOJ_Q = Q_L1 left outerjoin Q_L2 on JC(L1,L2)`
if P is not supported in L2
(2) `FOJ_Q = Q_L1 inner join Q_L2 on JC(L1,L2)`
if P is not supported in L2 and P is not supported in L1

For the query Q1, since the predicate `P = (Price = 100 and Stars > 3)` is not supported in `hotel` because the simple predicate `(Stars > 3)` is not supported in `hotel`. The predicate `P = (Price = 100 and Stars > 3)` is supported in `resort`; finally the rule 1 holds and then FOJ_Q1 can be simplified as

```
FOJ_Q1_s = Q_resort_1 as R1 left outerjoin Q_hotel_1 as H1
            on (R1.name = H1.name)
```

Name	Stars	Price	Room
Hilton	5	100	12, 13
Kyriad	5	375	4
Ibis	4	100	6
Garden	4	100	18, 24

Figura 3.8: Instance of FOJ_Q1_s, the simplified version of FOJ_Q1

For the query Q2, since the predicate `P = (Wifi = true and Stars > 3)` is not supported in `hotel` because the simple predicate `(Stars > 3)` is not supported in `hotel`. For the local source `resort`, the predicate `P = (Price = 100 and Stars > 3)` is not supported in `resort` because the simple predicate `(Wifi = true)` is not supported in `resort` ; finally the rule 2 holds and then FOJ_Q2 can be simplified as

```
FOJ_Q2_s = Q_resort_2 as R2 inner join Q_hotel_2 as H2
            on (R2.name = H2.name)
```

Name	Stars	Room	Wifi
Kyriad	5	4	true

Figura 3.9: Instance of FOJ_Q2_s, the simplified version of FOJ_Q2

The substitution of a full outerjoin with a left/right outerjoin or a join constitutes a relevant query optimization result, since full outerjoin queries are very expensive, especially in a distributed environment as the one of mediator/integration systems. Moreover, while database optimizers take full advantage of associativity and communtativity properties of join to implement efficient and powerful optimizations on join queries, only limited optimization is performed on full outerjoin [37]. The aim of our research is to propose an algorithm that will simplify the FOJ expression for **n** local classes (also called FOJ sequence).

As demonstrated in [37], it is helpful to rewrite outerjoins as joins, whenever possible, for several reasons. First, the rewritten query often has smaller intermediate results. Second, the set of candidate implementations is largest for join, smaller for outerjoin[2] and smallest for full

[2]Outerjoin stands for left outerjoin or right outerjoin.

outerjoin. For example, the optimizer is free to choose the inner and outer relations for joins. In this example, we show how we can perform Outerjoin Simplification for a conjunctive global queries. We outline how this simplification depend by the selection condition of the global query: full outerjoin is substituted by left outerjoin for query Q1 and by inner join for query Q2. Just to give an idea that outerjoin simplification allows to get fewer tuples in the intermediate results: instance in figure 3.6 vs instance in figure 3.8 for Q1 and instance in figure 3.7 vs instance in figure 3.9 for Q2.

5. **Application of the Residual Condition:** the result of the global query is obtained by applying the residual condition to the `FOJ` (`FOJ` considered are instances in figures 3.8 and 3.9 respectively for Q1 and Q2). In our example, for the query Q1 we have `(Price = 100 and Stars > 3)` as residual conditions then its result is obtained as

```
select Name, Room
from FOJ_Q1_s
where Price = 100 and Stars > 3
```

Name	Room
Hilton	12, 13
Ibis	6
Garden	24,18

while for the query Q2, we have `(Wifi = true and Stars > 3)` as residual conditions then its result is obtained as

```
select Name, Room
from FOJ_Q2_s
where Wifi = true and Stars > 3
```

Name	Room
Kyriad	4

We can notice that the results obtained are the same as those obtained without simplification of the FOJ expression. With this intuitive example, we show how we can optimize the query with selection condition composed of terms connected with AND's;

3.4 Optimization of Full Outerjoin in Data Fusion

3.4.1 Preliminary Definitions and Notations

To describe our method, we give some preliminary definitions and notations. Firstly, Let us consider the rewritting of global query Q as

$$Q = \Pi_{\mathcal{A}}[\sigma_P(G)]$$

where $\mathcal{A}$ is the select list, G is the global class, $\mathcal{L}(G)$ a set of local classes with $card(\mathcal{L}(G)) = n$ and P is a conjunction of (simple) predicates

$$P = pred_1 \ and \ \ldots \ and \ pred_k$$

As an example, for Q1, we have $\mathcal{A} = \{Name, Room\}$, $G =$ `Hotel` ,

$\mathcal{L}(Hotel)$=`{resort, hotel}` and `P = (Price = 100 and Stars > 3)`

As intuitively discussed in the section 1.4.1.4, the computation of G is done by computing $\mathcal{MQ}^G$, then we can write:

$$Q = \Pi_{\mathcal{A}}[\sigma_P(\mathcal{MQ}^G)]$$

$\mathcal{MQ}^G$ is formulated as :

$$\mathcal{MQ}^G = \Pi^*_{S(G)}(FOJ)$$

where $\Pi^*_{S(G)}$ represents the select list of $\mathcal{MQ}^G$ (where, in particular, resolution functions are applied)

As an example for Q1, $\mathcal{MQ}^G$ is

```
SELECT COALESCE(L1.Name,L2.Name) as Name,
       CONCATENATE(L1.Room,L2.Room) as Rooms,
       AVG(L1.Price,L2.Price) as Price, L1.Stars as Stars,
       L2.Wifi as Wifi
FROM   resort as L1 FJ hotel as L2 on (L1.Name = L2.Name)
```

then $\Pi^*_{S(G)}$ is:

```
{ COALESCE(L1.Name, L2.Name) as Name,
   CONCATENATE(L1.Room,L2.Room) as Room,
              AVG(L1.Price,L2.Price) as Price,
                L1.Stars as Stars, L2.Wifi as Wifi }
```

and FOJ as:

$$FOJ = \langle\, L1, FJ, L2 \,\rangle$$

With the query unfolding process, the query Q is rewritten as

$$Q = \Pi_{\mathcal{A}}[\sigma_{Pr}(\Pi^{*}_{\mathcal{A}'}(FOJ\_Q))]$$

where

1. Pr is a residual condition,

2. FOJ_Q is the unfolded full outerjoin sequence, i.e is the FOJ-sequence applied to the local query answers:

   $$FOJ\_Q = \langle\, Q\_L1, FJ, Q\_L2, FJ, Q\_L3, ..., FJ, Q\_Ln \,\rangle$$

   where Q_L_i is the local query answer for the local class L_i obtained by translating P on the local class L_i.

3. $\mathcal{A}^{'} = \bigcup_{i=1}^{n} S(Q\_L_i)$

As an example, for the query Q1 we have:

`Pr = (Price = 100 and Stars > 3)` , $\mathcal{A}^{'} = \{Name, Room, Price, Stars\}$

and FOJ_Q as:

$$FOJ\_Q = \langle\, Q\_L1, FJ, Q\_L2 \,\rangle$$

In a general way to simplify a FOJ_Q sequence of **n** local classes means rewriting it in an equivalent form as:

$$FOJ\_Q_{simpl} = \langle\, Q\_L1, \Theta, Q\_L2, \Theta, Q\_L3, ..., \Theta, Q\_Ln \,\rangle$$

where Θ is FJ (for full outerjoin), LJ (for left outerjoin), RJ (for right outerjoin) and IJ (for inner join).

As an example, for Q1, the simplified sequence of FOJ_Q is (see at page 64):

$$FOJ\_Q_{simpl} = \langle\, Q\_resort\_1, LJ, Q\_hotel\_1 \,\rangle$$

3.4.2 Simplification of FOJ-sequences

To obtain the simplified sequence, we make the following definitions.

Definition 7 (supported) *Given $pred_i$, let A_i be the global attribute used in $pred_i$. A predicate $pred_i$ is supported in the local class $L \in \mathcal{L}(G)$ for the query Q if:*

$$MT[A_i][L] \text{ is not null, i.e, exist } LA \text{ in } L \text{ with } LA = MT[A_i][L]$$

Definition 8 (P is supported in L) *Given $P = pred_1$ and ... and $pred_k$; we consider a local class $L \in \mathcal{L}(G)$; we say that P is supported in L for the query Q if:*

$$pred_i \text{ is supported in } L \text{ , for each } i.$$

Definition 9 (P is supported in $\mathcal{S}$) *Given $P = pred_1$ and ... and $pred_k$ and $\mathcal{S} \subseteq \mathcal{L}(G)$; we say that P is supported in $\mathcal{S}$ for the query Q if:*

$$\forall pred_i \ , \ \exists L \in \mathcal{S} \mid \ pred_i \text{ is supported in } L.$$

As an example, we consider:
P=($A > 3$ and $B < 6$), $\mathcal{S} = \{L1(A, C, E), L2(B, D, F)\} \subseteq \mathcal{L}(G)$;
by considering the definition 9, we can affirm that P is supported in $\mathcal{S}$.

Definition 10 ($pred_i$ is pushed down in L) *Given $pred_i$, let A_i be the global attribute used in $pred_i$. A predicate $pred_i$ is pushed down in the local class $L \in \mathcal{L}(G)$ for the query Q if:*

(1) A predicate $pred_i$ is supported in L ***and***

(2) A_i is an homogeneous attribute

At this point, we have all definitions needed to give an algorithm to simply FOJ_Q into FOJ_Q_{simpl} . In this algorithm, the rules to simplify a full outerjoin operation to left/right outerjoin or inner join operation given a FOJ-sequence of only two local classes (see step 4.a at page 64) are generalized for a generic FOJ-sequence, starting the simplification from the left.

Given a query Q, let FOJ_Q its unfolded full outerjoin sequence, i.e is the FOJ-sequence applied to the local query answers:

$$\texttt{FOJ\_Q} = \langle\ \texttt{Q\_L1, FJ, Q\_L2, FJ, Q\_L3,..., FJ, Q\_Ln}\ \rangle$$

We say that P is supported in a set of local query answers

$$\{\ \texttt{Q\_L1, Q\_L2, ..., Q\_Lk}\ \}$$

if P is supported by the set of related local classes:

$$\{\ \texttt{L1, L2, ..., Lk}\ \}$$

To better understand this equivalence, please refer to section 3.4.1.

The simplification algorithm is the following:

ALGOSIM. Simplification of FJ sequence.
Input: FJ sequence ⟨Q_L1, FJ, Q_L2, FJ, Q_L3,..., FJ, Q_Ln⟩
Output: FJ sequence simplified
Procedure.
1- For i=1 to i=n-1
1.1- let SET_LEFT_i={Q_Lj | j≠i+1} and SET_RIGHT_i={Q_Lj | j>i}
⟨...,Q_Li, FJ, Q_Li+1,..., FJ, Q_Ln ⟩ is simplified as follows
1.2- ⟨....,Q_Li, RJ, Q_Li+1,...., FJ, Q_Ln ⟩
if P is not supported in SET_LEFT_i
1.3- ⟨....,Q_Li, LJ, Q_Li+1,...., FJ, Q_Ln ⟩
if P is not supported in SET_RIGHT_i
1.4- ⟨....,Q_Li, IJ, Q_Li+1,...., FJ, Q_Ln ⟩
if P is not supported in SET_LEFT_i and P is not supported
in SET_RIGHT_i
2- Output: FJ sequence simplified

In the following, we show an application of this algorithm to a case of three local classes. We consider a modified version of the global class Hotel with the following three local classes

$$\mathcal{L}(\texttt{Hotel}) = \{\ \texttt{hotel,resort, dbhotel}\ \}$$

where the local classes hotel and resort are invariant with respect to previous examples (see Figure 3.1), the local class dbhotel is shown in Figure 3.11; the Mapping Table is shown in Figure 3.10.

We consider the following query Q2:

Hotel	resort	hotel	dbhotel
Name	name	name	name
Room	rooms	hotelrooms	dbrooms
Price	amount	price	-
Stars	stars	-	-
Wifi	-	wifi	dbwifi

Figura 3.10: Mapping Table of global class Hotel with three local classes

name	dbrooms	dbwifi
Continental	122	true
Ibis	66	false
Garden	18	true

Figura 3.11: Instance of dbhotel

```
Q2 = select Name, Room
     from Hotel
     where Wifi = true and Stars > 3
```

The local queries are:

```
Q_resort =  select name, rooms, stars
            from resort
            where stars > 3

Q_hotel =   select name, hotelrooms, wifi
            from hotel
            where wifi = true

Q_dbhotel = select name, dbrooms, dbwifi
            from dbhotel
            where where dbwifi = true
```

The residual condition is `Wifi = true and Stars > 3` then the `Q2` answer can be computed as:

```
select Name, Room
from FOJ
where Wifi = true and Stars > 3
```

where `FOJ` is an FOJ sequence. The cardinality of the set of possible full outerjoin sequences for **n** local classes is **n!**. In our example, with $n = 3$ local classes, we have the following six full outerjoin sequences:

1. $FOJ_1 = \langle$ `Q_resort, FJ, Q_hotel, FJ, Q_dbhotel` $\rangle$
2. $FOJ_2 = \langle$ `Q_resort, FJ, Q_dbhotel, FJ, Q_hotel` $\rangle$
3. $FOJ_3 = \langle$ `Q_hotel, FJ, Q_resort, FJ, Q_dbhotel` $\rangle$
4. $FOJ_4 = \langle$ `Q_hotel, FJ, Q_dbhotel, FJ, Q_resort` $\rangle$
5. $FOJ_5 = \langle$ `Q_dbhotel, FJ, Q_hotel, FJ, Q_resort` $\rangle$
6. $FOJ_6 = \langle$ `Q_dbhotel, FJ, Q_resort, FJ, Q_hotel` $\rangle$

The simplified version of the six full outerjoin sequences, denoted with FOJ_i^s, obtained by the algoritm **ALGOSIM** are the following:

1. $FOJ_1^s = \langle$ `Q_resort, LJ, Q_hotel, LJ, Q_dbhotel` $\rangle$
2. $FOJ_2^s = \langle$ `Q_resort, LJ, Q_dbhotel, LJ, Q_hotel` $\rangle$
3. $FOJ_3^s = \langle$ `Q_hotel, RJ, Q_resort, LJ, Q_dbhotel` $\rangle$
4. $FOJ_4^s = \langle$ `Q_hotel, FJ, Q_dbhotel, IJ, Q_resort` $\rangle$
5. $FOJ_5^s = \langle$ `Q_dbhotel, FJ, Q_hotel, IJ, Q_resort` $\rangle$
6. $FOJ_6^s = \langle$ `Q_dbhotel, RJ, Q_resort, LJ, Q_hotel` $\rangle$

As an example, let us shown how the algoritm **ALGOSIM** works for FOJ_1; first of all, we verify on an instance the correctness of the algorithm. In Figure 3.12 we shown the instances related to FOJ_1 and FOJ_1^s.

Then, it is easy to verify that the query

```
select Name, Room
from FOJ1
where Wifi = true and Stars > 3
```

is equivalent to

```
select Name, Room
from FOJ1s
where Wifi = true and Stars > 3
```

The result of these queries (and then the result of Q2) is shown in Figure 3.13.

Let us shown how the steps of the algoritm **ALGOSIM** for FOJ_1.

$FOJ_1 = \langle$ Q_resort, FJ, Q_hotel, FJ, Q_dbhotel $\rangle$

Name	Stars	Room	Wifi
Hilton	5	12	⊥
Kyriad	5	4	true
Sofitel	⊥	6	true
Ibis	4	6	⊥
Garden	4	24, 18	true
Dream	⊥	35	true
Continental	⊥	122	true

$FOJ_1^s = \langle$ Q_resort, LJ, Q_hotel, LJ, Q_dbhotel $\rangle$

Name	Stars	Room	Wifi
Hilton	5	12	⊥
Kyriad	5	4	true
Ibis	4	6	⊥
Garden	4	24, 18	true

Figura 3.12: Instances of FOJ_1 and FOJ_1^s

Name	Room
Kyriad	4
Garden	24, 18

Figura 3.13: query answer for Q2

Input: $\langle$ Q_resort, FJ, Q_hotel, FJ, Q_dbhotel $\rangle$

i = 1, we consider the FJ between Q_resort and Q_hotel; we have:
(a) SET_LEFT_1 ={Q_resort, Q_dbhotel} and
SET_RIGHT_1 = {Q_hotel, Q_dbhotel}
(b) P is supported in SET_LEFT_1 and P is not supported in SET_RIGHT_1, we obtain:
$\langle$ Q_resort, LJ, Q_hotel, FJ, Q_dbhotel $\rangle$

i = 2, we consider the FJ between Q_hotel and Q_dbhotel; we have:
(a) SET_LEFT_1 ={Q_resort, Q_hotel} and
SET_RIGHT_1 = {Q_dbhotel}
(b) P is supported in SET_LEFT_1 and P is not supported in SET_RIGHT_1, we obtain:
$\langle$ Q_resort, LJ, Q_hotel, LJ, Q_dbhotel $\rangle$

output: $\langle$ Q_resort, LJ, Q_hotel, LJ, Q_dbhotel $\rangle$

3.4.3 Demonstration

To reason about equivalence and containment of FOJ expression we convert them into the join-disjunctive normal form introduced by Galindo-Legaria [36]. We use the normal form to obtain our rule for optimize the FOJ expression. Our rules consisting to replace the full outerjoin (FJ) by inner join (IJ) or left outerjoin (LJ) or right outerjoin (RJ).

Definition 11 *The minimum union of relation L1, L2 is*

$$L1 \uplus L2 = (L1 \oplus L2)\downarrow$$

Minimun union has lower precedence than join.

Definition 12 *We say that t_1 subsumes t_2 if they are defined on the same schema, t_1 agrees with t_2 on all columns where they both are non-null, and t_1 contains fewer null values than t_2. The operator removal of subsumed tuples of L, denoted by $L\downarrow$, returns the tuples of L that are not subsumed by any other tuple in L.*

We assume that base tables contain no subsumed tuples. this is usually the case in practice because base tables almost always contain a primary key.

Theorem 1 *(Galindo-Legaria)*

Let Q be a join/outerjoin query. Q can be rewritten as a join-disjunctive query Q'. Such Q' is a normal form for Q [36].

The theorem guarantees that two expressions FOJ and FOJ^s (FOJ^s is the simplified version of FOJ) produce the same result for all database instances if and only if their join-disjunctive forms are equivalent. Each term in the join-disjunctive form of a sequence produces tuples with a unique null-extension pattern [96].

The theorem also guarantees that every FOJ sequence can be rewritten as a normal form. The normal form of FOJ sequence give us the possibility to determine what parts in a normal form of FOJ sequence contribute to the global answer. To obtain our simplification rules, we exploits transformation rules provided by [36] in the section 3.4.3.1.

3.4.3.1 Transformation Rules

The following transformation rules are used for converting FOJ sequence to the normal form (join disjunctive form)

$$\langle\, L1,\, \mathrm{FJ},\, L2 \,\rangle \;=\; [(L1 \;\mathrm{IJ}\; L2) \oplus L1 \oplus L2] \tag{3.1}$$

$$\langle\, L1,\, \mathrm{LJ},\, L2 \,\rangle \;=\; [(L1 \;\mathrm{IJ}\; L2) \oplus L1] \tag{3.2}$$

$$\langle\, L1,\, \mathrm{RJ},\, L2 \,\rangle \;=\; [(L1 \;\mathrm{IJ}\; L2) \oplus L2] \tag{3.3}$$

The proof of the correctness of the first two rules of transformations can be found in [36].

3.4.3.2 Simplification Rules

The following simplification rules are used only in data integration to simplify FOJ sequence. These rules are only true with the global class G that are mapped over two local classes. Our local classes of G are: $L1$ and $L2$.
Our simplification rules consider the global selection condition P posed on global class G. P is a selection condition composed of terms connected with AND's:

$$P = pred_1 \;and\; \ldots \;and\; pred_k$$

$$\langle\, L1,\, \mathrm{FJ},\, L2 \,\rangle \;\equiv\; \langle\, L1,\, \mathrm{LJ},\, L2 \,\rangle$$

$$if\;\; P \;is\; not\; supported\; in\; L2 \tag{3.4}$$

$$\langle\, L1,\, \mathrm{FJ},\, L2 \,\rangle \;\equiv\; \langle\, L1,\, \mathrm{RJ},\, L2 \,\rangle$$

$$if\;\; P \;is\; not\; supported\; in\; L1 \tag{3.5}$$

$$\langle\, L1,\, \mathrm{FJ},\, L2 \,\rangle \;\equiv\; \langle\, L1,\, \mathrm{IJ},\, L2 \,\rangle$$

$$if\; P \,is\, not\, supported\, in\, L1\; and\; if\; P \,is\, not\, supported\, in\, L2 \tag{3.6}$$

Proof. To obtain the simplified version; first we rewrite the FOJ sequence in the normal form. To do that, we need to use the transformation rules proposed in the section 3.4.3.1.

We try to apply the transformation rule 3.1 on the sequence, we have:

$$\langle\, L1,\ FJ,\ L2\,\rangle\ =\ [(L1\ IJ\ L2) \oplus\ L1 \oplus\ L2]$$

We assume $L1$ supports P and $L2$ not supports P. This assumption have a consequence in our FOJ sequence.
We observe that $\oplus\, L2$ does not contribute to the final answer because $L2$ not supports P therefore the term $\oplus\, L2$ is discarded. While $(L1\ IJ\ L2)$ and $\oplus\, L1$ are subsets of tuples for global answer query (each contain a subset of tuples which can be a part of global answer) because $L1$ supports P.
Then we can simplify our FOJ sequence as:

$$\langle\, L1,\ FJ,\ L2\,\rangle\ \equiv\ [(L1\ IJ\ L2) \oplus\ L1]$$

Finally applying the transformation rule 3.2, we can obtain:

$$\langle\, L1,\ FJ,\ L2\,\rangle\ \equiv\ \langle\, L1,\ LJ,\ L2\,\rangle$$

with P is not supported in $L2$.

If we have been assumed P is not supported in $L1$ and P is not supported in $L2$, we can have more simplification of FOJ sequence.
Then we can write:

$$\langle\, L1,\ FJ,\ L2\,\rangle\ =\ [(L1\ IJ\ L2) \oplus\ L1 \oplus\ L2]$$

where $(L1\ IJ\ L2)$ is a potential subset of global answer. All terms $\oplus\, L1$ and $\oplus\, L2$ are not subsets of tuples for global answer query therefore they must be discarded. Then , we have the next simplification:

$$\langle\, L1,\ FJ,\ L2\,\rangle\ \equiv\ \langle\, L1,\ IJ,\ L2\,\rangle$$

with P is not supported in $L1$ and P is not supported in $L2$.

In the following we will discuss how the previous simplification rules can be extended to the case of **n** local classes.

The FOJ sequence for n local classes are an associative generalization of binary full outerjoin to any number of local classes (relations) . The computation of the **full outerjoin** FOJ on **n local classes** in MOMIS is defined in section 1.4.1.4 as:

```
L1 FJ L2 ON (L1.ID = L2.ID)
     FJ L3 ON (L3.ID = L1.ID OR L3.ID = L2.ID)
       ...
          FJ Ln ON (Ln.ID = L1.ID OR ... OR Ln.ID = Ln-1.ID)
```

where `FJ` is full outerjoin operator.

We assume **ID** as a share object identifier amond all local classes. For our example, the share object identifier amond our three local classes is the attribute **name** (the mapping table considered is on figure 3.10).

We can consider the FOJ operation for Q_resort, Q_hotel and Q_dbhotel for query Q2 (at page 70) as:

$$\langle \texttt{ Q\_L1, FJ, Q\_L2, FJ, Q\_L3 } \rangle$$

Our goal is to simplify the sequence $\langle$ `Q_L1, FJ, Q_L2, FJ, Q_L3` $\rangle$. It is not necessary taking into consideration the Join Conditions defined between local classes because all local classes are interconnected by a share object identifier (see graph 1 of the figure 3.16): it is the case of data fusion in MOMIS.

The full outerjoin is a binary operator, so we need to start with two local classes, for example we consider `resort (L1)` and `hotel (L2)`:

$$\langle \texttt{ Q\_L1, FJ, Q\_L2 } \rangle$$

P is not supported in `resort`, so we can simplify our sub-sequence as following (see simplification rule 3.5 in section 3.4.3.2):

$$\texttt{FOJ\_1} = \langle \texttt{ Q\_L1, RJ, Q\_L2 } \rangle$$

The FOJ_1 simplified sub-sequence is not correct because the local class `resort` will lost tuples which are potentialy candidates of final result. These tuples may have their matching tuples in the local class `dbhotel`, then we need to control if P is not supported in the union of `resort` and `dbhotel`. First to perform simplification, we need to verify if P is not supported in {resort, dbhotel} (see the definition 9 in section 3.4.1). We resume this new simplification rule for a sub-sequence $\langle$ `Q_L1, FJ, Q_L2` $\rangle$ as:

(a) We need to consider: SET_LEFT_1 = {Q_L1, Q_L3}

(b) We need to verify: P is not supported in SET_LEFT_1

(c) The verification of this condition give the following simplified sub-sequence:

$$\langle \texttt{ Q\_L1, RJ, Q\_L2 } \rangle$$

for `resort`, we can note that no simplification is performed for FOJ_1 therefore the sub-sequence remain the same:

$$\texttt{FOJ\_1 = } \langle \texttt{ Q\_L1, FJ, Q\_L2 } \rangle$$

For `hotel`, we will do the same thing but we must consider the union of $\{hotel, dbhotel\}$. Finally we resume this step as:

(d) We need to consider: SET_RIGHT_1 = {Q_L2, Q_L3}

(e) We need to verify: P is not supported in SET_RIGHT_1

(f) The verification of this condition give the following simplified sub-sequence:

$$\langle \texttt{ Q\_L1, LJ, Q\_L2 } \rangle$$

The simplification is performed for hotel and the FOJ_1 sub-sequence become:

$$\texttt{FOJ\_1 = } \langle \texttt{ Q\_L1, LJ, Q\_L2 } \rangle$$

In conclusion we obtain the algorithm **ALGOSIM** of page 70.

The cardinaly of the set of possible full outerjoin sequences for **n** local classes is **n!**. To generate all possibles sequences, we use a modified version of the algorithm provided in [81].

ALGOENUM. Enumeration of sequences.
Input. A set of local classes $\mathcal{L}(G)$ with $card(\mathcal{L}(G)) = n$.
Ouput. A set of sequences for $\mathcal{L}(G)$.
Procedure.
B-1. For each local class in $\mathcal{L}(G)$ create a 1-leaf tree.
B-2. for $k = 2, \ldots, n$: Choose T_l, T_r such that
– $leaves(T_l) \cap leaves(T_r) = \emptyset$.
– $|leaves(T_l)| = 1$ and $|leaves(T_r)| = k - 1$.
B-2.1 Create a sequence $T_s = \langle T_l,\ FJ,\ T_r \rangle$.
B-3- Output a set of sequences .

With respect to the original algorithm provided in [81], we changed the following two point:

1. $|leaves(T_l)| = 1$ and $|leaves(T_r)| = k - 1$. instead of $|leaves(T_l)| + |leaves(T_r)| = k$.
2. we removed the condition $leaves(T_l) \cup leaves(T_r)$ is connected since we are considering the case of local classes that are interconnected by Join Conditions as a clique graph (see definition 16 in section 3.5.1) and then this condition is always true.

ALGOENUM describe how to generate sequences for full outerjoin expression. Sequences are build incrementally. Each step combines two sub-sequence, using a full outerjoin as operator.

To compare FOJ sequences from a computational point of view, we introduce a simple *cost model* to evaluate the execution cost of a FOJ sequence: this cost model only consider the type of join and does not consider other parameters such as the cardinality of intermediate results and the presence of indexes. On the basis of this cost model, given two FOJ sequences *homogeneous*, i.e., with the same Q_L_i elements in the same order, FOJ' and FOJ'', with

$$FOJ' = \langle\, Q\_L1, \Theta'_1, Q\_L2, \Theta'_2, Q\_L3, ..., \Theta'_{n-1}, Q\_Ln \,\rangle$$

and

$$FOJ'' = \langle\, Q\_L1, \Theta''_1, Q\_L2, \Theta''_2, Q\_L3, ..., \Theta''_{n-1}, Q\_Ln \,\rangle$$

we say that FOJ' is *more simplified* than FOJ'', denoted by $FOJ' \leq FOJ''$, if $\Theta'_i \preceq \Theta''_i$, for $1 \leq i \leq n-1$, where $\prec$ is defined as follows:

$$\prec \; = \Big\{(IJ, RJ), (IJ, LJ), (IJ, FJ), (LJ, FJ), (RJ, FJ)\Big\}$$

Given a FOJ expression

$$FOJ = \langle\, Q\_L1, \Theta_1, Q\_L2, \Theta_2, Q\_L3, ..., \Theta_{n-1}, Q\_Ln \,\rangle$$

with $\Theta_i = FJ$, for $1 \leq i \leq n-1$, **ALGOSIM** returns FOJ^s that is the **more simplified** expression equivalent to the input sequence FOJ, i.e.

$$FOJ^s \leq FOJ$$

and does not exists an homogeneous sequence FOJ'

$$FOJ' = \langle\, Q\_L1, \Theta'_1, Q\_L2, \Theta'_2, Q\_L3, ..., \Theta'_{n-1}, Q\_Ln \,\rangle$$

equivalent to FOJ with $FOJ' < FOJ^s$. As an example, given

$$FOJ_1 = \langle\, \texttt{Q\_resort, FJ, Q\_hotel, FJ, Q\_dbhotel} \,\rangle$$

we have the following equivalent FOJ expressions:

1. $FOJ_1^a = \langle$ `Q_resort, LJ, Q_hotel, FJ, Q_dbhotel` $\rangle$
2. $FOJ_1^b = \langle$ `Q_resort, FJ, Q_hotel, LJ, Q_dbhotel` $\rangle$
3. $FOJ_1^c = \langle$ `Q_resort, LJ, Q_hotel, LJ, Q_dbhotel` $\rangle$

with $FOJ_1^a \leq FOJ_1$, $FOJ_1^b \leq FOJ_1$, $FOJ_1^c \leq FOJ_1^a$ and $FOJ_1^c \leq FOJ_1^a$.

For FOJ_1, **ALGOSIM** returns FOJ_1^c.

ALGOSIM is applied to all the possible sequences and for each of these sequences it returns the most simplified. The choice of which of these sequences will be made by the optimizer module of the DBMS that must be performed on the basis of other parameters such as the cardinality of intermediate results and the presence of indexes.

3.5 Generalization of Simplification Techniques for Full Outerjoin

So far we have considered the case of Data Fusion (see section 1.4.1.2) that is the "default" case of the MOMIS system. In this case we assume a Global Class G with a shared object identifiers among its local classes, i.e. we assume a global class with a mapping table like the one shown in Figure 1.2.

In the case of data fusion, that in the following we will call **Case 1**, the computation of the **full outerjoin** in $\mathcal{MQ}^G$ is automatically defined by the system as:

```
L1 FJ L2 ON (L1.ID = L2.ID)
       FJ L3 ON (L3.ID = L1.ID OR L3.ID = L2.ID)
          ...
             FJ Ln ON (Ln.ID = L1.ID OR ... OR Ln.ID = Ln-1.ID)
```

For example, for the global class `Hotel` (the mapping table is in Figure 3.10) the related FOJ expression for (`L1=resort`, `L2=hotel` and `L3=dbhotel`) is:

```
FOJ = L1 FJ L2 ON (L1.name = L2.name)
            FJ L3 ON (L3.name = L2.name OR L3.name = L1.name)
```

In this case we have the following three *Join Conditions* among local classes: `JC(L1,L2) = (L1.name=L2.name)`, `JC(L3,L2) = (L3.name=L2.name)`, and `JC(L3,L2) = (L3.name=L2.name)`.

Now we want to consider a case different from the data fusion case, where we have not a shared object identifiers among local classes that in the following we will call **Case 2**; In this case the **full outerjoin operation** is explicitly defined from the Integration Designer. As an example, let us consider a modified version of the local sources `resort`, `hotel` and `dbhotel`, shown in Figure 3.14.

L1 = resort

name	stars	amount	rooms
Hilton	5	120	12
Kyriad	5	350	⊥
Sofitel	⊥	210	40
Ibis	4	100	6
Garden	4	100	24
Continental	3	225	122

L3 = dbhotel

dbhotelcode	dbrooms	dbwifi
015	122	true
019	66	false
003	18	true

L2 = hotel

code	name	price	hotelrooms	wifi
001	Hilton	80	13	false
002	Kyriad	400	4	true
003	Garden	100	18	⊥
004	Sofitel	100	6	true
005	Madison	100	⊥	false
006	Dream	200	35	true

Figura 3.14: Instances of local classes `resort`, `hotel` and `dbhotel`

Moreover, we consider the global class `Hotel` with a different mapping table, shown in Figure 3.15.

Hotel	**resort**	**hotel**	**dbhotel**
Name	name	name	-
Code	-	code	dbhotelcode
Room	rooms	hotelrooms	dbrooms
Price	amount	price	-
Stars	stars	-	-
Wifi	-	wifi	dbwifi

Figura 3.15: Mapping Table of global class Hotel with three local classes(Case 2)

Let us consider the following *FOJ* expression, (we denote such FOJ expression by `FOJ_expl` since it is explicitly defined by the Integration Designer) with `L1=resort`, `L2=hotel` and `L3=dbhotel`:

```
FOJ_expl = L1 FJ L2 ON (L1.name = L2.name)
               FJ L3 ON (L3.dbhotelcode = L2.code)
```

In this case we have the following two *Join Conditions* among local classes: `JC(L1,L2) = (L1.name=L2.name)`, `JC(L3,L2) = (L3.dbhotelcode=L2.code)` and `JC(L1,L3) = null` .
Given a FOJ expression, to represent its Join Conditions between local classes, we use a *undirected graph* where *nodes* represent local classes and *undirected edges* represent *not null* Join Conditions between these local classes, i.e., if `JC(L,L') = null` then there is no edge between `L` and `L'`.

For `FOJ` of Case 1: since $JC(L1, L2) \neq null$, $JC(L1, L3) \neq null$, and $JC(L2, L3) \neq null$, the Join Conditions between local classes are represented by **Graph 1** of the figure 3.16). The Graph 1 is a clique (see definition 16).

For `FOJ_expl` of Case 2: since $JC(L1, L2) \neq null$, $JC(L1, L3) = null$, and $JC(L2, L3) \neq null$, the Join Conditions between local classes are represented by **Graph 2** of the figure 3.16). In this case the Join Conditions between local classes are represented by **graph 2** of the figure 3.16): the graph 2 is a tree (see definition 17).

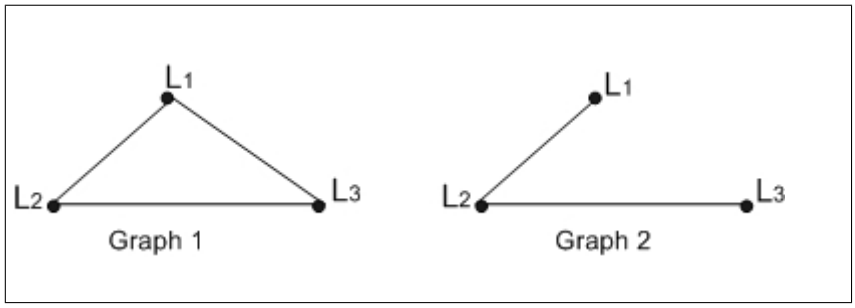

Figura 3.16: Graphs to represent join conditions between local classes

Formal definitions are given in the next section.

3.5.1 Join Condition Graph

In this section we represent the Join Conditions (relationships) between local classes using graphs: each node represents one of the local classes while the edges represent Join Conditions between local classes. More precisely, The set of local classes of G are represented as nodes $\mathbf{N}$ of the graph $\mathcal{G}$ and a set of join conditions $\mathbf{E}$ as edges of the graph $\mathcal{G}$ (these definitions about graph theory are from [42]).

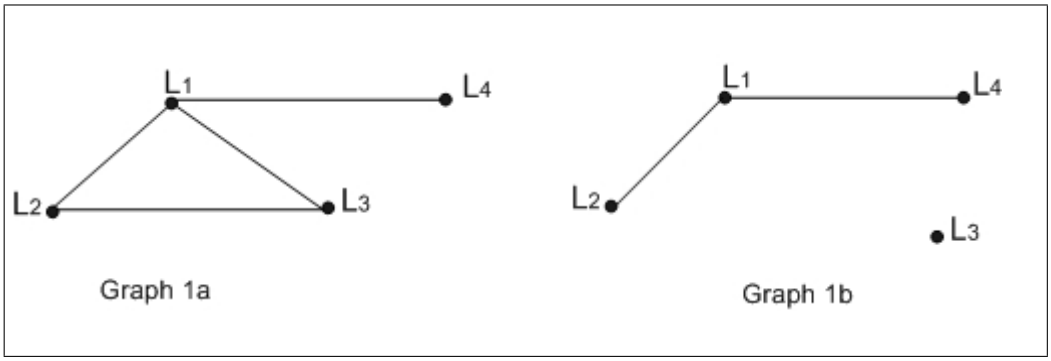

Figura 3.17: Graphs connected and disconnected

Definition 13 (Graph) *A pair $\mathcal{G} = (N, E)$ with $E \subseteq E(N)$ is called a **graph (on N)**. The elements of N are the **nodes** of $\mathcal{G}$, and those of E the **edges** of $\mathcal{G}$. The node set of a graph $\mathcal{G}$ is denoted by $N_{\mathcal{G}}$ and its edges set by $E_{\mathcal{G}}$. Therefore $\mathcal{G} = (N_{\mathcal{G}}, E_{\mathcal{G}})$*

In literature, graphs are also called simple graphs; vertices are called nodes or points; edges are called lines or links.

Typically, a graph is depicted in diagrammatic form as a set of circles for the nodes, joined by lines for the edges (See figure 3.17).

Definition 14 (Graph connected) *Graph $\mathcal{G}$ is **connected** if there exists a walk (and hence a path) from u to v in $\mathcal{G}$ for all $u, v \in \mathcal{G}$. A graph that is not connected is said to be disconneted.*

For example, the graph 1a of the figure 3.17 is connected while the graph 1b is not connected.

Definition 15 (Connected Component) *The maximal connected subgraphs of $\mathcal{G}$ are its connected components. Denote*

$$c(\mathcal{G}) = the\ number\ of\ connected\ components\ of\ \mathcal{G}$$

If $c(\mathcal{G}) = 1$, then $\mathcal{G}$, of course, connected.

The maximality condition means that a subgraph $\mathcal{H} \subseteq \mathcal{G}$ is a connected component if and only if $\mathcal{H}$ is connected and there are no edges leaving, i.e., for every node $v \notin \mathcal{H}$, the subgraph $\mathcal{H}'[N_{\mathcal{H}} \cup \{v\}]$ is disconnected.

By convention, $V^*_{\mathcal{G}}(u)$ is the connected component of $\mathcal{G}$ that contains $u \in \mathcal{G}$. In particular, the connected components form a partition of $\mathcal{G}$. $\mathcal{L}(V^*_{\mathcal{G}}(u))$ is the nodes set of a connected component $V^*_{\mathcal{G}}(u)$.

There are two connected components of the graph 1b in figure 3.17: one with the nodes $\mathcal{L}(V^*_{\mathcal{G}}(L1)) = \{L1, L2, L4\}$ and the other with only the node $\mathcal{L}(V^*_{\mathcal{G}}(L3)) = \{L3\}$. While for a graph 1a there is one connected component in figure 3.17: it is always the graph 1a.

For the sake of simplicity, we also represent a connected component with only its nodes (which are the local classes).

The set of connected components of graph 1a is :

$$S_{1a} = \{\{L1, L2, L3, L4\}\}$$

while for the graph 1b is:

$$S_{1b} = \{\{L1, L2, L4\}, \{L3\}\}$$

Definition 16 (Clique) *A clique in a graph $\mathcal{G} = (N, E)$ is a subset of the nodes set $C \subseteq N$, such that for every two nodes $u, v \in C$, there exists an edge $e \in E(C)$ connecting the two.*

For example, see the graph 1 of figure 3.16.

Definition 17 (Tree) *A graph $\mathcal{G} = (N, E)$ is called acyclic, if it has no cycles. An acyclic graph is also called a forest. A tree is a connected acyclic graph.*

For example, see the graph 2 of figure 3.16.

3.5.2 A generalized simplification algorithm

In this section we will present some preliminar idea related to a generalization of the ALGOSIM algorithm in order to consider the generic **Case 2**.

We consider the global class `Hotel` defined by the Mapping Table of Figure 3.15. with the following `FOJ_expl` expression:

```
FOJ_expl = resort FJ hotel ON (resort.Name = hotel.Name)
                  FJ dbhotel ON (dbhotel.Code = hotel.Code)
```

We start by considering the query Q2:

```
Q2 = select Name, Room
     from Hotel
     where Wifi = true and Stars > 3
```

In this case, the local queries of Q2 are:

```
Q_resort =  select name, rooms, stars
            from resort
            where stars > 3

Q_hotel =   select name, code, hotelrooms, wifi
            from hotel
            where wifi = true

Q_dbhotel = select dbhotelcode, dbrooms, dbwifi
            from dbhotel
            where where dbwifi = true
```

The residual condition is `Wifi = true and Stars > 3` then the `Q2` answer can be computed as:

```
select Name, Room
from FOJ_expl
where Wifi = true and Stars > 3
```

and the result is shown in Figure 3.18

Name	Room
Kyriad	4

Figura 3.18: Query answer for Q2 in the Case 2

If we consider the computation of **ALGOSIM** for the `FOJ_expl` expression we obtain the following simplified sequence (see figures 3.19 and 3.20):

$$\langle \texttt{ Q\_resort, LJ, Q\_hotel, LJ, Q\_dbhotel } \rangle$$

But in this case, in the expression `(Q_resort, LJ, Q_hotel)`, tuples that are only in `Q_resort` cannot match tuples of `L3=dbhotel`, since there is no

Join Condition between `L1=resort` and `L3=dbhotel`; as a consequence the first Left Join in the simplified sequence above can be *further* simplified in an inner join, and thus we obtain (see figure 3.19 and 3.20)

$$\langle \texttt{ Q\_resort, IJ, Q\_hotel, LJ, Q\_dbhotel } \rangle$$

Intuitively, the simplifications carried out by **ALGOSIM** are based on the hypothesis that all local classes are interconnected as a clique graph (see definition 16); in case this hypothesis is not verified, as in the example of `FOJ_expl` (represented by **Graph 2** of the figure 3.16)), some further simplifications are not individuated by **ALGOSIM**.

Here we present a preliminar idea to generalize **ALGOSIM** to handle FOJ sequence represented by a disconnected graph: in this case it is not sufficient to verify if `P` is not supported in `SET_LEFT` or in `SET_RIGHT`, but we need to verify if the graph of these sets are connected or disconnected. If the graph is disconnected, we need to consider the set of its connected components and we need to verify if `P` is not supported in such sets.

To this end, we introduce the following definitions.

Definition 18 (P is supported in $V_{\mathcal{G}}^*$) *Given $V_{\mathcal{G}}^*$ a connected component of a graph $\mathcal{G}$, $P = pred_1$ and ... and $pred_k$, and $\mathcal{L}(V_{\mathcal{G}}^*)$ is the set of the local classes in $V_{\mathcal{G}}^*$; we say P is supported in $V_{\mathcal{G}}^*$ for the query Q if:*

$\forall pred_i \ , \ \exists L \in \mathcal{L}(V_{\mathcal{G}}^*) \mid \ pred_i$ *is supported in* L.

Definition 19 (P is supported in $\mathcal{C}$) *Given $\mathcal{C}$ a subset of connected components of a graph $\mathcal{G}$; we say P is supported in $\mathcal{C}$ for the query Q if:*

$\exists V_{\mathcal{G}}^* \in \mathcal{C} \mid \ P$ *is supported in* $V_{\mathcal{G}}^*$.

The simplification algorithm for case 2 is the following:

ALGOSIM 2. Simplification of FJ sequence.
Input: FJ sequence $\langle$ Q_L1, FJ, Q_L2, FJ, Q_L3,..., FJ, Q_Ln $\rangle$ and J a set of Join Conditions
Output: FJ sequence simplified
Procedure.
1- For i=1 to i=n-1
1.1- let SET_LEFT_i={Q_Lj | j$\neq$i+1 } and SET_RIGHT_i={Q_Lj | j>i}
Construct graphs $\mathcal{G}$_L and $\mathcal{G}$_R respectively for SET_LEFT_i and SET_RIGHT_i by using Join Conditions as edges and local classes as nodes.
1.2- Let SL_i={$V^*_{\mathcal{G}}$(Q_Lj)$\in \mathcal{G}$_L | j$\leq$i} and SR_i={$V^*_{\mathcal{G}} \in \mathcal{G}$_R }
$\langle$...,Q_Li, FJ, Q_Li+1,..., FJ, Q_Ln $\rangle$ is simplified as follows
1.3- $\langle$....,Q_Li, RJ, Q_Li+1,...., FJ, Q_Ln $\rangle$
if P is not supported in SL_i
1.4- $\langle$....,Q_Li, LJ, Q_Li+1,...., FJ, Q_Ln $\rangle$
if P is not supported in SR_i
1.5- $\langle$....,Q_Li, IJ, Q_Li+1,...., FJ, Q_Ln $\rangle$
if P is not supported in SL_i and P is not supported in SR_i
2- Output: FJ sequence simplified

In what follows, we will show how **ALGOSIM 2** works on the FOJ-sequence of case 2 to obtain its simplified version.

Input: $\langle$ Q_resort, FJ, Q_hotel, FJ, Q_dbhotel $\rangle$, J a set of Join Conditions

i = 1, we consider the FJ between Q_resort and Q_hotel; we have:
(a) SET_LEFT_1 ={Q_resort, Q_dbhotel} and
SET_RIGHT_1 = {Q_hotel, Q_dbhotel}
(c) we construct graphs for SET_LEFT_1 and for SET_RIGHT_1 (see figure 3.21)
(b) SL_1 = {{Q_resort}} and SR_1 = {{Q_hotel, Q_dbhotel}};
SL_1 is a subset of connected component of graph 1a and
SR_1 is set of connected component of graph 1b (see figure 3.21)
(d) P is not supported in SL_1 and P is not supported in SR_1, we obtain:
$\langle$ Q_resort, IJ, Q_hotel, FJ, Q_dbhotel $\rangle$

i = 2, we consider the FJ between Q_hotel and Q_dbhotel; we have:
(a) SET_LEFT_2 ={Q_resort, Q_hotel} and
SET_RIGHT_2 = {Q_dbhotel}
(c) we construct graphs for SET_LEFT_2 and for SET_RIGHT_2 (see

figure 3.22)
(b) SL_2 = {{Q_resort, Q_hotel}} and SR_2 = {{Q_dbhotel}};
SL_2 is a set of connected component of graph 1c and SR_2 of graph 1d (see figure 3.22)
(d) P is supported in SL_2 and P is not supported in SR_2, we obtain:
⟨ Q_resort, IJ, Q_hotel, LJ, Q_dbhotel ⟩

output: ⟨ Q_resort, IJ, Q_hotel, LJ, Q_dbhotel ⟩

ALGOSIM 2: FOJ_expl = ⟨ Q_resort, IJ, Q_hotel, LJ, Q_dbhotel ⟩

Code	Name	Stars	Room	Wifi
002	Kyriad	5	4	true

ALGOSIM 1: FOJ_expl = ⟨ Q_resort, LJ, Q_hotel, LJ, Q_dbhotel ⟩

Code	Name	Stars	Room	Wifi
⊥	Hilton	5	12	⊥
002	Kyriad	5	4	true
⊥	Ibis	4	6	⊥
⊥	Garden	4	24, 18	⊥

Figura 3.19: Instances of simplified versions of FOJ_expl for Q2 (Case 2)

The original version of FOJ_expl is in figure 3.20.

Original Version: FOJ_expl = ⟨Q_resort, FJ, Q_hotel, FJ, Q_dbhotel⟩

Code	Name	Stars	Room	Wifi
⊥	Hilton	5	12	⊥
002	Kyriad	5	4	true
⊥	Ibis	4	6	⊥
⊥	Garden	4	24, 18	⊥
003	⊥	⊥	18	true
015	⊥	⊥	122	true
004	Sofitel	⊥	6	true
006	Dream	⊥	35	true

Figura 3.20: Instance of FOJ_expl for Q2 (Case 2)

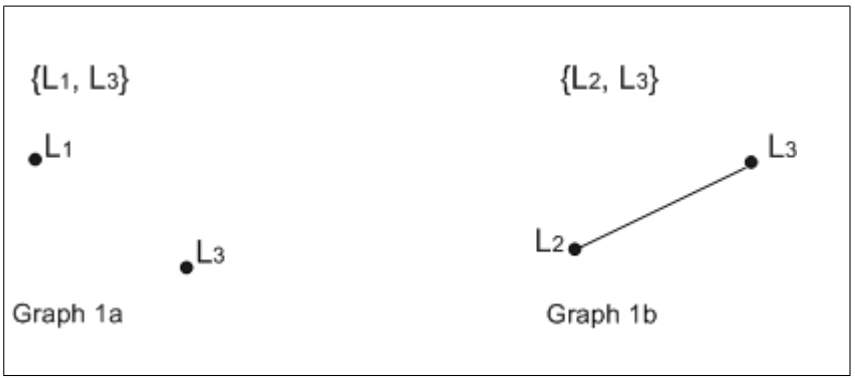

Figura 3.21: Step i = 1

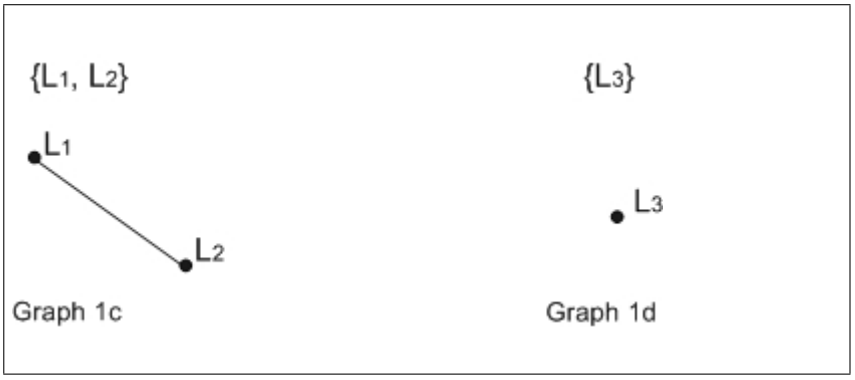

Figura 3.22: Step i = 2

3.6 Conclusion

In this chapter we considered the Query Processing of the MOMIS system and we proposed new Query Optimization techniques based on the simplification of the of full outerjoin operation used in MOMIS to perform data fusion. The results are shown in the context of the MOMIS system; however, the full outerjoin operator is a good candidate to perform data fusion and then the proposed techniques can be generalized to other data integration systems. Full outerjoin is an important SQL2 operation, but is handled poorly by current optimizers.

A major contribution of this work is to optimize conjunctive queries. Our optimization is defined as a substitution of full outerjoin operator by the left(right) outerjoin or inner join operators. In particular, we proposed an algorithm **ALGOSIM** for the simplification when the full outerjoin is computed on the basis of a shared object identifiers among local classes

related to a global class. This is an important case for data fusion. Then we presented some preliminary idea related to a generalization of the **ALGOSIM** algorithm in order to consider the case where we have not a shared object identifiers among local classes (**ALGOSIM 2**) and the Join Condition is explicitly defined by the Integration designer.

We show how **ALGOSIM** (**ALGOSIM 2**) is applied to all the possible sequences and for each of these sequences it returns the most simplified. The choice of which of these sequences will be made by the optimizer module of the DBMS that must be performed on the basis of other parameters such as the cardinality of intermediate results and the presence of indexes.

Capitolo 4

Data Quality Issues in Data Integration Systems

This chapter is a review of the literature on the data quality related to data integration systems. This study presents the common dimensions between the different information quality frameworks revealed in [65]. We have mentioned various data anomalies which affect the data quality and we proposed a table that summarizes relationships between data anomalies and data quality dimensions. We made a study of different data integration systems which implement the notion of data quality. Specially, we focus on quality-driven query processing.

4.1 Introduction

Data quality and data integration technologies have evolved to support organizations grappling with fragmented and defective data. There is no doubt that the two technologies support intrinsically symbiotic activities, but where does one end and the other begin? The answer is that the two functions should work together seamlessly.

The problem with data is that its quality degenerates over time. So just as data integration is an ongoing process, so too is data quality. Data quality encompasses more than finding and fixing missing or inaccurate data. It means delivering comprehensive, consistent, relevant, fit-for-purpose, and timely data to the business regardless of its application, use, or origin.

In Data Integration systems, the heterogeneity among sources may range from the hardware and software platforms to the data model and the schema

used to represent information. Therefore we divide data quality problems in schema level and instance level data problems. Schema level data problems can be avoided with an improved schema design. Instance level data problems are related with the data contents and cannot be avoided with a better schema definition as the schema definition languages are not powerful enough to specify all the requirement data constraints. Then, conflict resolution affects the global answer as well as query transformation and evaluation. More strategies are presented in [23] in order to solve data discrepancies. And the query processing must take into consideration the quality associated with the data. Quality-Driven query processing is the process of integrating and transforming data and content to deliver authoritative, consistent, timely and complete information, and governing its quality throughout its life cycle.

In this chapter, we introduce the notion of data quality (section 4.2), database-related technical solutions for data quality(section 4.3), the data quality problems in data integration systems (section 4.4). Finally we provide an overview of existing proposals to deal with quality-driven query processing (Section 4.5). Some of these contributions are derived from the technical report [74] of the FIRB project NeP4b, where several my research activities were developed.

4.2 What is Data Quality?

The quality of the data that is used by a business is a measure of how well its organizational data practices satisfy business, technical, and regulatory standards. Organizations with high data quality use data as a valuable competitive asset to increase efficiency, enhance customer service, and drive profitability. Alternatively, organizations with poor data quality spend time working with conflicting reports and flawed business plans, resulting in erroneous decisions that are made with outdated, inconsistent, and invalid data. Issues surrounding the quality of data and information that cause these difficulties range in nature from the technical (e.g., integration of data from disparate sources) to the nontechnical (e.g., lack of a cohesive strategy across an organization ensuring the right stakeholders have the right information in the right format at the right place and time).

As started in [4], the issue of Data Quality is not new, as, throughout history, people have benefited by, and suffered from, the quality of information made available to them. An obvious example is information made available to military commanders during battles. What is new in the past several

decades is the explosion in the quantity of information and the increasing reliance of most segments of society on that information.

Data quality is a young field with several facets and avenues for research. Many researchs focuses on the dimensions, or measures, of data quality and their formal definitions [82]. The notion of data quality has been widely investigated in the literature. Traditionally, Data Quality is described or characterized via multiple attributes or dimensions which refers to the degree to which data satisfy user requirements (e.g. accuracy, completeness) or are suitable for a specific process(e.g. response time, reliability). Data Quality is described as data that fit-for-use by data consumers [83], which implies that the notion of quality is subjective [63].

Data Quality need to be assessed within the context of its generation [67] and intended use [45]. This is because the dimensions of data quality can vary depending on the context in which the data is to be used. Carlo Batini and Monica Scannapieco in their book Data Quality - Concepts, Methodologies and Techniques provides details of several data quality frameworks and a wide array of data quality concepts so it is a useful resource when building data quality framework.

4.3 Database-Related technical Solutions for Data Quality

Research in this area develops database technologies for assessing, improving, and managing data quality. It also develops techniques for reasoning with data quality and for designing systems that can produce data of high quality.

4.3.1 Data Integration, Data Warehouse

Early data quality research focused mainly on developing techniques for querying multiple data sources and building Data Integration Systems. The work of Wang and Madnick (Composing Answers from disparate information systems, 1989) used a systematic approach to study related data quality concerns. This work identified and addressed entity resolution issues that arose when integrating information from multiple sources with overlapping records. These researchers explored ways to determine whether separate records actually corresponded to the same entity. This issue has become known by terms such as record linkage, record duplication, record matching

and object identification [84, 64, 92].

A main problem for cleaning data from multiple sources is to identify overlapping data, in particular matching records referring to the same real-world entity. This problem is also referred to as the object identity problem [83], duplicate elimination or the merge/purge problem [43]. Frequently, the information is only partially redundant and the sources may complement each other by providing additional information about an entity. Thus duplicate information should be purged out and complementing information should be consolidated and merged in order to achieve a consistent view of real world entities. These techniques are often used to improve completeness, resolve inconsistencies, and eliminate redundancies during data integration processes.

4.3.2 Data Quality Dimensions

Data Quality Dimensions characterize properties that are inherent to data, i.e., depend on the very nature of data; as example, a dimension specifying whether the data about citizen are updated or not. The data quality literature provide many Information Quality (IQ) framework to capture the dimensions of data quality that are important to data consumers.

Traditional quality dimensions are accuracy, consistency, completeness, timeliness, consistency, and accessibility, which represent the dimensions considered by the majority of the authors [52, 65, 25, 95, 75]. Their informal definition is the following [5]:

- *Accuracy*: The degree of correctness and precision with which the real world data of interest to an application domain is represented in an information system.
- *Completeness*: The degree to which all data relevant to an application domain has been recorded in an information system.
- *Timeliness*: The degree to which the recorded data is up-to-date. A closely related concept is *currency* which is interpreted as the time a data item was stored [88].
- *Consistency*: The degree to which the data managed in an information system satisfies specified integrity. constraints
- *Cost*: The time it would take to transmit the information over the network, or the money to be paid for the information, or both.

- *Accessibility*: The degree in which data are available or quickly or easily retrievable.
- *Relevancy*: It is a measure of the appropriateness of the data extracted for the requested task.

Timeleness is usually considered together with other time-related dimensions, typically currency and volatility [3].

In [65], the authors summary 12 widely accepted IQ frameworks collated from the last decade of data quality research. This study reveals the common dimensions between the different IQ frameworks; from this study a more complete list of quality dimension was derived, which provides a summary of the most common dimensions and the frequency with they are included in many frameworks; this list is shown in Figure 4.1.

	Dimension	# of times
1	Accuracy	8
2	Consistency	7
3	Security	7
4	Timeliness	7
5	Completeness	5
6	Concise	5
7	Reliability	5
8	Accessibility	4
9	Availability	4
10	Objectivity	4
11	Relevancy	4
12	Useability	4
13	Understandability	5
14	Amount of data	3
15	Believability	3
16	Navigation	3
17	Reputation	3
18	Useful	3
19	Efficiency	3
20	Value-Added	3

Figura 4.1: The Common Dimensions of IQ/DQ

In [63], the authors have identified three different kinds of classification for data quality dimensions: semantic-oriented, processing oriented, and

goal-oriented classifications.

We call a classification **semantic-oriented** if it is solely based on the meaning of the dimensions. This classification is the most intuitive when dimensions are examined in a most general way, i.e., separated from any information framework. For example, Total Data Quality Management is a project aimed at providing an empirical foundation for data quality. Wang and Strong have empirically identified fifteen IQ dimensions regarded by data consumers as the most important [89].The authors classified their dimensions into the classes intrinsic quality, accessibility, contextual quality, and representational quality. The classification is based on the semantic of the dimension.

A classification is **processing-oriented** if it partitions IQ dimension according to their deployment in different phases of information processing. For example, the criteria of the mediator-based information system (MBIS) of [62] are based on the TDQM dimensions set. However, the dimensions were re-classified to adapt to the query planning processing steps. For the source-selection phase source-specific criteria are employed. For the planning phase where views are combined, view-specific criteria are employed. Finally, when presenting the information, attribute specific criteria are used.

Finally, a classification is **goal-oriented** if it matches goals that are to be reached with the help of quality reasoning. For example, the Data Warehouse Quality (DWQ) project is based on the criteria of TDQM [87]. The authors define operational quality goals for data warehouses and classify the criteria by the goals they describe. These are accessibility, interpretability, usefulness, believability, and validation.

4.3.3 Classification of Data Anomalies

In [56], the authors classify data anomalies into syntactical, semantic, and coverage anomalies. Syntactical anomalies describe characteristics concerning the format and values used for representation of the entities. Semantic anomalies hinder the data collection from being a comprehensive and non-redundant representation the mini-world. Coverage anomalies decrease the amount of entities and entity properties from the mini-world that are represented in the data collection.

1. Syntactical Anomalies

(a) **Lexical errors** name discrepancies between the structure of the data items and the specified format.

(b) **Domain format errors** specify errors where the given value for an attribute A does not conform with the anticipated domain format for A.

2. Semantic Anomalies

(a) **Integrity constraint violations** describe tuples (or sets of tuples) that do not satisfy one or more of the integrity constraints.

(b) **Domain format errors** specify errors where the given value for an attribute A does not conform with the anticipated domain format for A.

(c) **Duplicates** are two or more tuples representing the same entity from the mini-world. The values of these tuples do not need to be complete identical.

(d) **Invalid tuples** represent by far the most complicated class of anomaly found in data collections. By invalid we mean tuples that do not display anomalies of the classes defined above but still do not represent valid entities from the mini-world.

3. Coverage Anomalies

(a) **Missing values** are the result of omissions while collecting the data.

(b) **Domain format errors** specify errors where the given value for an attribute A does not conform with the anticipated domain format for A.

(c) **Missing tuples** result from omissions of complete entities existent in the mini-world that are not represented by tuples in the data collection.

From the table presented in [56], we propose a new table to resume the relationship between Data Anomalies and some Data Quality Dimensions in the figure 4.2.

For more comprehension of the figure 4.2, see the appendix C. Sometimes, the duplicate can introduce contradiction and thus affecting consistency.

Data Anomalies	Completeness	Accuracy	Consistency	Currency
Lexical error		-	-	
Domain Format error		-	-	
Irregularities		-	-	
Constraint Violation		-		
Integrity Violation			-	
Missing Value	-	-		
Missing Tuple	-			
Invalid Tuple		-		
OutDated Value				-
Duplicate			-	

Figura 4.2: Data Anomalies affecting Data Quality Dimensions

4.4 Data Quality Problems in Data Integration Systems

The taxonomy of data problems presented here is based on five relevant works found in the literature [46, 71, 66, 56, 78]. In Data Integration systems, we divide data quality problems in schema level and instance level data problems. Schema level data problems can be avoided with an improved schema design. Instance level data problems are related with the data contents and cannot be avoided with a better schema definition as the schema definition languages are not powerful enough to specify all the requirement data constraints.

4.4.1 Schema Level and Instance Level Data Quality Problems

Schema level data quality problems thus occur because of the lack of appropriate model-specific or application-specific integrity constraints, e.g., due to data model limitations or poor schema design, or because only a few integrity constraints were defined to limit the overhead for integrity control.

The problems present in single sources are aggravated when multiple sources need to be integrated. Each source may contain dirty data and the data in the sources may be represented differently, overlap or contradict. This is because the sources are typically developed, deployed and maintained

independently to serve specific needs. This results in a large degree of heterogeneity w.r.t. data management systems, data models, schema designs and the actual data.

At the schema level, data model and schema design differences are to be addressed by the steps of schema translation and schema integration, respectively. The main problems w.r.t. schema design are naming and structural conflicts [56]. Naming conflicts arise when the same name is used for different objects (homonyms) or different names are used for the same object (synonyms). Structural conflicts occur in many variations and refer to different representations of the same object in different sources, e.g., attribute vs. table representation, different component structure, different data types, different integrity constraints, etc.

Instance level data problems refer to errors and inconsistencies of data that are not visible or avoided at schema level. Note that data instances also reflect schema-level problems (e.g., a record with a null value in a required field). We consider that instance level data problems are divided into single record and multiple records problems:

- Single record data problems concern one or various attributes of a single record. In other words, this kind of problem is related to a single entity and does not depend on other information stored in the database. For example, Missing data in a not null field: An attribute is filled with some dummy value. For instance, a social security number -999999 is an undefined value used to surpass the not null constraint. Other types of errors are: Erroneous data, Misspelled words in database fields, Misfielded value(data is stored in the wrong filed), Ambiguous data(abbreviation).

- Multiple record data problems cannot be detected by considering each record separately as the data problem concerns more than one record. All problems from the single record case can occur with different representations in different sources (e.g.,duplicated records, contradicting records,...). Furthermore, even when there are the same attribute names and data types, there may be different value representations (e.g., for marital status) or different interpretation of the values (e.g., measurement units Dollar vs. Euro) across sources. Moreover, information in the sources may be provided at different aggregation levels (e.g., sales per product vs. sales per product group) or refer to different points in time (e.g. current sales as of yesterday for source 1 vs. as of last week for source 2).

4.4.2 Conflict Resolution and Data Merging

An important issue in the data integration problem is a possibility of conflict among the information sources. The sources may conflict with each other due by different modeling of realworld entities, different data models or simply by different representations of one and the same object. During the integration phase these conflicts have to be identified and resolved as part of the mapping between local and global schemata. Therefore, conflict resolution affects the global answer as well as query transformation and evaluation. More strategies are presented in [23] in order to solve data discrepancies.

In [23], the authors classify and describe existing strategies to approach data conflicts:

- **Conflict-ignoring** strategies do not make a decision as to what to do with conflicting data and sometimes are not even aware of data conflicts. An example for an ignoring strategy is Pass It On, which presents all values and that way defers conflict resolution to the user.
- **Conflict-avoiding** strategies acknowledge the existence of possible conflicts in general, but do not detect and resolve single existing conflicts. Instead, they handle conflicting data by applying a unique decision equally to all data, such as preferring data from a special source with the Trust Your Friends strategy.
- **conflict resolution** strategies take into account all the data and metadata before deciding on how to resolve a conflict. They can further be subdivided into deciding and mediating strategies, depending on whether they choose a value from all the already present values (deciding) or choose a value that does not necessarily exist among the conflicting values (mediating).

These different strategies are applied on the two kinds of data conflicts: uncertainty and contradiction. Incertainty is a conflict between a non-null value and one or more null values that are all used to describe the same property of a real-world entity. Contradiction is a conflict between two or more different non-null values that are used to describe the same property of the same entity.

During the fusion phase of the information integration, multi-instance tuples that are versions of each local sources need to be identified and fused. The problem of tuple identity is solved by specifying the attributes relevant for deciding equivalence, this approach is known as object identification.

Other approach exist like as full outerjoin-merge operation [60, 22] (MOMIS's case).

4.5 Previous Approaches for Quality-Driven Query Processing

In this section we provide an overview of several proposals to perform quality-driven query processing, which returns an answer to a global query, by explicitly taking into account the quality of data provided by local sources; however, several other techniques are present in the literature, e.g., [54, 19, 20, 55, 39, 79, 62].

4.5.1 Data Integration Techniques based on Data Quality aspect

In [39], the autors developed some data integration techniques based on data quality aspects within an object oriented data model, and data quality information stored in a metadata. In the case of data conflicts between semantically equivalent objects, the object with the best data quality must be chosen. However, the quality goals specification limits the possibility of more combinations of priorities from the user, because they are not given in weights or percentages, just the most accurate or the most up to date. Consequently, not just one or two combinations of quality priorities will satisfy users. One result might be good enough for one user under a specific situation, but of poor quality for other.
The authors suggested an extension of a query language, say OQL, by a data quality clause:

```
select < list of attribute >
from G
where < selection condition >
with goal < data quality goal >
```

A simple data quality goal can either be most accurate, most up-to-date, or most compltete. G is the global class.

4.5.2 Fusionplex Query Processing

Fusionplex is data integration system that uses information quality in the evaluation of global queries within the relational data model. Each local

source is annotated with various information quality criteria called feature [55]. The fusion process is multi-step:

- Fusionplex determines the global views that are relevant to the given query. A global view is relevant if its projection attributes intersect with the query's selection predicate does not contradict the query's selection predicate.

- The relevant contributions are materialized and polished: unnecessary attributes are removed, necessary but unavailable attributes are filled with null. The union of these contributions is called a polyinstance.

- The tuples of the polyinstance are clustered in polytuples. The members of each polytuple are different 'version' of the same information. Once data inconsistencies have been detected, they need to be resolved; i.e., every polytuple is reduced to a single tuple. This process is performed in two passes:

  (a) The members of each tuple are ranked and purged according to user-specified preferences.

  (b) In each polytuple, in each attribute, remaining values are purged and then fused in a single value. User preferences and attribut-specific fusion functions necessary for this phase are given in the query.

In [55], the authors propose the extended query language to deal with the query which take into account the quality feature:

```
select[restrict] < list of attribute >
from G
where < selection condition >
using < selection feature >
with  < weight feature >
```

In analogy with the **where** clause, which restricts the answer set with a condition on attributes, the **using** clause restricts the answer set with a condition on features. The **with** clause provides for the specification of the feature weight tuple w used in the inconsistency resolution process. When **restrictive** is present, all comparisons to null values, either in attribute columns (the where clause) or in feature columns (the using clause), evaluate to false; otherwise, they are true.

4.5.3 IQC-DIS: IQ Criteria for Data Integration Systems

Quality-driven Integration of Heterogeneous Information Systems was a project developed by Naumann in [62]. In this section, we describe the approach presented by the authors, which we will refer to as IQC-DIS.

The aims was to identify and to rank high quality plans, which produce high quality results. The query planning finds plans that are correct, but possibly generate different results, while classifical optimization considers plans that all produce same result. The query processing adopted in [62] is performed by considering information quality at different levels of granularity:

- Source-specific criteria, defining the quality of a whole source. Criteria of this category apply to all information of the source, independently from how it is obtained. Examples of such criteria are reputation of the source, based on users personal preferences, and timeliness, measured by the source update frequency.

- QCA-specific criteria, defining the quality of specific query correspondence assertions. An example of such criteria is price, i.e., the price to be paid for the query.

- User-query specific criteria, measuring the quality of the source with respect to the answer provided to a specific user query. Hence the scores for these criteria can only be determined at 'query time'. An example of such criteria is completeness, based on the fullness of source relations.

However, there is no further specification of how to assess quality at different levels of granularity. Data sources are ranked using the 'Data Envelopment Analysis' method. Therefore, there is no consideration of user priorities for this process. Beside, subjective criteria are used for discarding data sources such as reputation and understandability.

4.5.4 DaQuinCIS Query Processing

In [79], the italian DaQuinCIS project was dedicated to cooperative information systems and proposed an integrated framework to improve data quality in cooperative environments. Such a framework started from the

Total Data Quality Management methodology which was extended to suit the cooperative information systems requirements, and supporting data quality improvement. The use of a metadata was required to store the quality score, the meaning of the quality value, and how the measurements were carried out. This approach takes into account the specification of data granularity as the combination of elementary data items that are subject to quality metrics. There is also a difference between computing the quality agregated data and computing an aggregate indicator over a set of items. In the DaQuinCIS environement, the Data Quality Broker performs query processing according to a global-as-view (GAV) approach, by unfolding queries posed over a global schema, i.e., translating the global query into the corresponding view on local data sources. Both the global schema and local schemas exported by cooperating organizations are expressed according to the D^2Q model. The adopted query language is XQuery.

Query processing approach adopted by DaQuinCIS to answer a query with quality constraints are structured as following:

1. Let $\mathcal{Q}$ be a query posed on the global schema G:

2. The query $\mathcal{Q}$ is unfolded according to the static mapping that defines each concept of the global schema in terms of the local sources; such a mapping is defined in order to retrieve all copies of same data that are available in the CIS. Therefore, the query $\mathcal{Q}$ is decomposed in $\mathcal{Q}_1, ..., \mathcal{Q}_k$ queries to be posed over local sources;

3. The execution of the queries $\mathcal{Q}_1, ..., \mathcal{Q}_k$ returns a set of results $\mathcal{R}_1, ..., \mathcal{R}_k$. On such a set an extensional correspondence property is checked, namely: given $\mathcal{R}_i$ and $\mathcal{R}_j$; the items common to both results are identified, by considering the items referring to the same objects. This step is performed by using a record matching algorithm based on quality values exported by organizations;

4. The result to be returned is built as follows:

   (i) if no quality constraint is specified, a best quality default semantics is adopted. This means that the result is constructed by selecting the best quality values;

   (ii) if quality constraints are specified, the result is constructed by checking the satisfiability of the constraints on the whole result. Notice that the quality selection is performed at a property level, on the basis of best (satisfying) quality values.

4.6 Conclusion

Query processing and data quality in data integration systems pose many novel challenges.

Firstly, the data integration systems need to allow data at sources to be annotated with data quality metadata. Therefore we need to identify and define IQ dimensions within the context of its generation. The choice of data quality dimensions must include quality dimensions which are more significant and used (see figure 4.2).

Secondly, to resolve data inconsistency, we need to define some resolution functions which take into account the data quality metadata associated with the local sources. The global queries must include user specific quality considerations into query formulations, in order to address user specific requirements.

Finally, the IQ dimensions must be included to the task of query processing that produce high quality query plans and results. The quality-driven query processing techniques have the purpose of selecting and accessing data of the highest quality [5].

Capitolo 5

Quality-Driven Query Processing in MOMIS

Data Integration is a complex process and Data Integration quality is difficult to assess. This chapter aims to improve and extend the MOMIS framework to allow data at sources to be annotated with data quality metadata [13]. Then we will consider queries with quality criteria, i.e. data quality metadata are used in the specification of a query on the Global Schema, to express the quality of the retrieved data, by means of threshold of acceptance which users can ensure minimal performance of the data [55]. Finally, we will discuss the optimization of such Global Queries with Quality [13].

5.1 Introduction

This research activity was performed in the context of the NeP4b project (http://www.dbgroup.unimo.it/nep4b); in this project, the MOMIS framework was extended to allow data at sources to be annotated with data quality metadata [5]. This extension was motivated by the fact that MOMIS has been conceived to provide an integrated access to distributed and heterogeneous sources. These data sources are independent of each other, which increase the difficulty to obtain sources with the quality desired. In consequence, a query on a global view gives us a result which we are not sure of its quality.

For the sake of completeness we will resume the data quality dimensions defined in MOMIS in section 5.3; More information are available in the technical report D2.1 of the N4PB projet. In this section we also propose an extended mapping table to manage the data quality in MOMIS.

Starting from the MOMIS framework extended with data quality dimensions, my original contribution to this research activity was twofold

1. Data Quality Aware Queries: data quality dimensions are used in the specification of a query on the Global Schema, to express the quality of the retrieved data, by means of threshold of acceptance which users can ensure minimal performance of the data [55]. We present this research result in Section 5.4.

2. Quality-Driven Query Processing : quality constraints specified in the query are useful to perform query optimization. In particular, we will revise and extend the query optimization techniques we presented in the previous chapter in order to consider queries with quality criteria and we will highlight the role of quality constraints in this optimization process. We present this research result in Section 5.4.2.

5.2 Data Quality Dimensions in MOMIS

In this section, we extend the MOMIS framework to allow data at sources to be annotated with *data quality metadata* (see [5]). A Data Quality Dimension is an aspect or feature of information and a way to classify information and data quality needs. Dimensions are used to define, measure, and manage the quality of the data and information. Several data quality dimensions have been proposed in the literature [5].

In [32, 78] accuracy, completeness, consistency and currency are classified as the most important data quality dimensions (see figure 4.1).

The quality of data inside a source can be described according to different levels of *granularity*. It is possible to describe the quality of instances alone, or of aggregations of many instances. In the first case, it is possible to characterize single values inside an instance (*value level*), or entire instances (*instance level*). Quality dimensions at the *attribute level* refer to the set of values a certain attribute assume on an entire relation. Finally, quality dimensions at the source level refer to the entire database.

In the Nep4B project, the MOMIS system was extended with data quality concepts [31]; on the basis of these results, we assume a set QD of quality dimensions with the following hypothesis:

1. As in [55], we consider that each quality dimension $QDim$ in QD is associated with a domain of possible values, and a total order is assumed

to be defined on the domain. All quality dimensions are normalized: each quality dimension value is linearly mapped to a number in the interval [0,1]. The mapping is done so that high quality dimension values are always more desirable than low values; that is, the worst quality dimension value is mapped to 0, and the best to 1.

2. As in [93], quality dimensions are defined at the source attribute level: every local attribute LA has a subset `QD_GA` of $QDim$ associated with it. Starting from the quality dimensions associated with local attributes, for each global attribute GA of a global class G is defined a set of quality dimensions `QD_GA` as the union of the quality dimensions of the local attributes corresponding to GA in the mapping table of G.

5.2.1 Measures for Data Quality Assessment

In this section, we focus only on aggregated metadata, and more specifically on values calculated on all values that single attributes assume over all instances in a relation (i.e. column-level aggregates). We describe next the measures we have defined for the assessment of quality in MOMIS[1]. These measures are defined for the core quality dimensions introduced above[2]. In the following, $r(A_1, \ldots, A_n)$ denotes the union of two relations $r_1(A_1, \ldots, A_n)$ and $r_2(A_1, \ldots, A_n)$ defined over the same set A of attributes, whose definition domains are denoted with $D(A_1), \ldots, D(A_n)$, respectively. Furthermore, we denote with $\Gamma_i = \{\gamma_{i,1}, \ldots, \gamma_{i,|r|}\}$ the multi-set of values that attribute A_i assumes over the tuples in r.

- *(Syntactic) Accuracy* gives a measure of how a certain value stored in a database is close to a correct representation of the read-world property it represents. It can be measured using reference *Dictionaries*, or conformance to specific syntactic rules. Let R_i^a be a unary relationship defined over $D(A_i)$, such that $x \in R_i^a$ only if x is accurate. Let furthermore $\Gamma_i^a = \{x \in \Gamma_i | x \in R_i^a\}$. We define Accuracy of attribute A_i as:

$$Acc(A_i) = \frac{|\Gamma_i^a|}{|r|}$$

[1]More information are avaible in the technical report D2.1 for N4PB projet where i involve

[2]With the exception of *currency*. While our model natively supports the representation of currency values, measuring these values cannot be done when data alone is available, but requires that the data at sources is labelled with timestamps or that statistics about the frequency of updates in the sources are recorded.

Let furthermore $\Phi_i = \{\phi_{i,1}, \ldots, \phi_{i,|\Phi_i|}\}$ be a set of (alternative) syntactic rules defined over attribute A_i (specified, for instance, as regular expressions), and let $\Gamma_{i_j} = \{x \in \Gamma_i | x \text{ satisfies } \phi_{i,j}\}$. We define *Format Consistency* as:

$$Conf(A_i) = \frac{\sum_{j=1}^{|\Phi_i|} |\Gamma_{i_j}|}{|\Phi_i| \cdot |r|}$$

- The term *Completeness* refers to the presence or absence of certain instances or parts of data instances. Formally, we can define column-completeness for an attribute A_i as follows. Let $\hat{\Gamma}_i = \{x \in \Gamma_i | x \neq null\}$, then:

$$Compl(A_i) = \frac{|\hat{\Gamma}_i|}{|r|}$$

- *Internal Consistency* is a propriety of two or more values inside a single tuple. It is a measure of the extent to which the values of such attributes satisfy semantic constraints defined over the containing relation. It can be measured using functional constraints or by verifying compliance to reference dictionaries. Let $R^c_{i,j}$ be a binary relationship defined over $D(A_i) \times D(A_j)$, such that $\langle x, y \rangle \in R^c_{i,j}$ only if $\langle x, y \rangle$ is semantically consistent. Let furthermore $\Gamma^c_{i,j} = \{x \in R^c_{i,j} | x \in \Gamma_i, y \in \Gamma_i\}$. We define Internal Consistency of attributes A_i, A_j as:

$$Cons(A_i, A_j) = \frac{|\Gamma^c_{i,j}|}{|r|}$$

With the term *Consistency*, referred to a single attribute we denote the projection over such attribute of the internal consistency values calculated w.r.t. the entire schema of the containing relation. More formally, we can define consistency for attribute A_i as:

$$Cons(A_i) = \frac{\sum_{j=1}^{|A^c_i|} Cons(A_i, A^c_{i,j})}{|A^c_i|}$$

where $A^c_i = \{A_j \in A \| \exists R^c_{i,j}\} = \{A^c_{i,1}, \ldots, A^c_{i,1}\}$.

5.3 An example of Data Quality Dimensions in MOMIS

In this section, we will extend our example of section 3.3 An Introductory Example with Data Quality Dimensions. We also propose an extended

mapping table to manage the data quality in MOMIS.

We consider the following three local classes (relations) of three different local sources:

resort(name, rooms, amount, stars)

hotel(name, hotelrooms, price, wifi)

dbhotel(name, dbrooms, dbwifi)

We use $\perp$ to denote the null value in their instances.

resort

name	stars	amount	rooms
Hilton	5	120	12
Kyriad	5	350	$\perp$
Sofitel	$\perp$	210	40
Ibis	4	100	6
Garden	4	100	24

hotel

name	price	hotelrooms	wifi
Hilton	80	13	false
Kyriad	400	4	true
Garden	100	18	$\perp$
Sofitel	100	6	true

dbhotel

name	dbrooms	dbwifi
Continental	122	true
Ibis	66	false
Garden	18	true

Figura 5.1: Instances of local data sources `hotel`,`resort` and `dbhotel`

To extend the example with Data Quality Dimensions, we will use as quality dimensions the following set:

$$QD = \{\texttt{acc}, \texttt{comp}, \texttt{cons}, \texttt{curr}\}$$

where `acc` stands for accuracy, `comp` for completeness, `cons` for consistency and `curr` for currency; moreover, we consider:

`QD_resort.amount = {acc, comp}` and
`QD_hotel.price = {acc, comp, cons}`

Then for the global attribute `Price` we obtain:

`QD_Price = {acc, comp, cons}`

The quality dimension values for local attributes are represented by means of a *Quality Mapping Table MTQ* associated to a global class G:

MTQ is a table whose columns represent the local classes $\mathcal{L}(G)$ belonging to G and whose rows represent the quality dimensions defined for each global attributes A of G. An element $MTQ[A.QDim][L]$ represents the value of $QDim$ for the local attribute $LA = MT[A][L]$ of L, or is `NULL` if $QDim$ is not associated to LA, i.e. if $QDim \notin QD\_LA$; as an instance, in our example MTQ[`Price.cons`][`resort`] = `NULL`. To give an immediate representation of quality dimensions we represent both the Mapping Table MT and the Quality Mapping Table MTQ in the same table, as in figure 5.2 (when $MT[A][L]$ is NULL, $MTQ[A.QDim][L]$ doesnt exists: in the figure, this is denoted by a empty cell).

Hotel	**resort**	**hotel**	**dbhotel**
Name	name	name	name
Room	rooms	hotelrooms	dbrooms
acc	0.8	0.9	0.8
Price	amount	price	-
acc	0.7	0.8	
comp	0.8	0.9	
cons	NULL	0.8	
Stars	stars	-	-
acc	0.6		
comp	0.5		
cons	0.3		
curr	0.7		
Wifi	-	wifi	dbWifi
cons		0.5	0.7

Figura 5.2: Mapping Table and Quality Mapping Table of the global class `Hotel`

5.4 Data Quality Aware Queries in the MOMIS

In this section, we introduce Data Quality Aware Queries in the MOMIS framework: in the specification of a *global query* on the GVV, quality dimensions can be used to express the quality of the retrieved data.

Hotel

Name	Stars	Price	Room	Wifi
Hilton	5	100	12, 13	false
Kyriad	5	375	4	true
Sofitel	⊥	155	6, 40	true
Ibis	4	100	6, 66	false
Garden	4	100	18, 24, 18	true
Continental	⊥	⊥	122	true

Figura 5.3: Instances of global class Hotel computed as the Full outerjoin-merge between `resort`, `hotel` and `dbhotel`

The term quality-aware queries is introduced in [93] where the authors propose to include user specific quality considerations into query formulations, in order to address user specific requirements. The approach of [93] is based on generic Collaborative Information Systems. A more specific approach in the context of Data Integration Systems is proposed in [55], where queries on the integrated and global schema are able to express the quality of the retrieved data by means of threshold of acceptance with which users can ensure minimal performance of the data. Other mechanisms by which queries can be expressed over data and quality metadata have been proposed [80, 39, 58].

In this Chapter we adopt the proposal of [55]: quality dimensions are used to specify `quality constraints`[3] that express the quality of the retrieved data, by means of threshold of acceptance, excluding from the answer data that do not satisfy the specified quality constraint.

A global query Q with *quality constraints* is a conjunctive query on a global class G of the GVV expressed in an extended SQL syntax as follow:

`select [restrictive]` $A_1, \ldots, A_h$
`from` G
  `where` $pred_1$ *and* ... *and* $pred_k$
  `using` $Qpred_1, \ldots, Qpred_m$

where A_i are global attributes of G, *pred* is a (simple) predicate : A *op value* *Qpred* is a *quality constraint*: $A.QDim > Qvalue$, where $QDim$ is a quality dimension defined for the global attribute A and $Qvalue$ is a value in [0, 1].

Example 1. Let us consider the query Q1 without quality constraints.

[3] In [55] the term *quality predicate* is used

```
Q1 = select Name, Room, Wifi
     from Hotel
     where Price = 100 and Stars > 3
```

By considering the instance of `Hotel` of figure 5.3, the answer of this query is:

Name	Room	Wifi
Hilton	12, 13	false
Ibis	6, 66	true
Garden	18, 24, 18	true

The `using` clause implements a selection predicate for specifying the desired quality criteria of the answer. In analogy with the `where` clause, which restricts the answer set with a condition on attributes, the `using` clause restricts the answer set with a condition on quality dimensions. Let us give an intuitive explanation of the semantics of the `using` clause; a deep discussion about quality constraints is in [55]. Let A be a global attribute, L a local class and $LA = MT[A][L]$ the local attribute of L mapped into A. Since a quality dimension $QDim$ is defined at the local attribute level, the quality constraint $(A.QDim\ op\ Qvalue)$, specifies that the values of LA will be considered in the answer of the query only if the quality dimension $QDim$ associated with LA is greater than the desired $Qvalue$, i.e. if $MTQ[A.QDim][L] > Qvalue$. In this case we will say that the quality constraint is satisfied by LA. As discussed in [55] , when $MTQ[A.QDim][L]$ is `NULL` the quality constraint is satisfied by LA, i.e the values of LA will be considered in the answer of the query, only if **restrictive** is specified.

Example 2. Let Q2 obtained from Q1 by adding some quality constraints:

```
Q2 = select Name, Room, Wifi
     from Hotel
     where Price = 100 and Stars > 3
     using Price.acc > 0.7, Price.cons > 0.6, Room.acc > 0.7
```

The result of the query Q2 is:

Name	Room	Wifi
Garden	18, 24, 18	true

where we can observe that there is no longer (w.r.t the Q1 answer) a Hilton record. Since the quality constraint `Price.acc > 0.7` is not satisfied by the local attribute `amount`, the values of this local attribute are not considered to calculate the value of the `Price` global attribute: then the value of `Price` for Hilton record is equal to 80 and thus the constraint `Price = 100` is false for this record. For the same reason the Ibis record is not in the Q2 result.

5.4.1 Preliminary Definitions

In Chapter 4, we introduced the notion of 'supported' (see at page 69). Now we will try to extend it in order to take into consideration the quality constraints expressed in the query Q (quality-supported).

Definition 20 (quality-supported) *Given $pred_i$, let A_i be the global attribute used in $pred_i$. A predicate $pred_i$ is supported in the local class $L \in \mathcal{L}(G)$ for the query Q if:*

(1) $MT[A_i][L]$ is not null, i.e, exist LA in L with $LA = MT[A_i][L]$ ***and***

(2) All quality constraints expressed on A_i are satisfied by $LA = MT[A_i][L]$

Definition 21 (P is supported in L) *Given $P = pred_1 and \ldots and pred_k$; we consider a local class $L \in \mathcal{L}(G)$; we say that P is supported in L for the query Q if:*

$$pred_i \text{ is quality-supported in } L \text{ , for each } i.$$

Definition 22 (P is supported in $\mathcal{S}$) *Given $P = pred_1 and \ldots and pred_k$ and $\mathcal{S} \subseteq \mathcal{L}(G)$; we say that P is supported in $\mathcal{S}$ for the query Q if:*

$$\forall pred_i \ , \ \exists L \in \mathcal{S} \mid \ pred_i \text{ is quality-supported in } L.$$

Definition 23 ($pred_i$ is pushed down in L) *Given $pred_i$, let A_i be the global attribute used in $pred_i$. A predicate $pred_i$ is pushed down in the local class $L \in \mathcal{L}(G)$ for the query Q if:*

(1) A predicate $pred_i$ is quality-supported in L ***and***

(2) A_i is an homogeneous attribute ***or***

(3) for any other local attribute LA different from $LA = MT[A_i][L]$ where A_i is mapped, there are quality constraints on A_i not satisfied from LA.

5.4.2 Quality-Driven Query Processing

To answer a global query on G, the query must be rewritten as an equivalent set of queries (*local queries*) expressed on the local classes $\mathcal{L}(G)$ belonging to G. This query rewriting is performed by considering the mapping between the GVV and the local schemata; in a GAV approach the query rewriting is performed by means of *query unfolding*, i.e., by expanding the global query

on G according to the definition of its mapping query $\mathcal{MQ}^G$.
The query unfolding of a global query Q produces, for each local class L belonging to G, a local query `Q_L`

```
Q_L = select S_L
      from L
      where C_L
```

where the select list `S_L` is a list of local attributes of L and the condition `C_L` is a conjunction of (simple) predicates on L. These local queries are executed on the local sources and local queries answers are then fused by means of the mapping query $\mathcal{MQ}^G$ to obtain the answer of the global query.
We defined this query execution process in several papers [6, 9]. In the preliminary work [13], we described in an informal way and limited to global classes with two local classes, how quality constraints may be used in the query unfolding process; in particular, we showed how quality constraints may be useful to perform query optimization. The contribution of this Chapter is a generalization and formalization of these results to the general case of global classes with more than two local classes.

We consider a query Q with quality constraints

`select [restrictive]` $A_1, \ldots, A_h$
`from` G
`where` $pred_1$ *and* ... *and* $pred_k$
`using` $Qpred_1$, ..., $Qpred_m$

The process for executing the global query consists of the following steps:

1. **Computation of Local Query predicates:**
   In this step is evaluated whether a predicate of the global query may be pushed down to the local level, i.e. if it may be into the local query condition for a local class; this decision also depends on the presence of quality constraints. A predicate $pred_i = (A_i \, op \, value)$ is translated into the predicate $(MT[A_i][L] \;\; op \;\; value)$ on the local class L only if $pred_i$ satisfy the definition 23 .

   conditions (2) and (3) of definition 23 are discussed in the following by considering queries Q1 and Q2, respectively.
   For the query Q1, since the global attribute `Stars` is mapped only in resort, and then it is homogeneous, the predicate `(Stars > 3)` is pushed down for the local class `resort`. On the other hand, the global attribute `Price` is not homogeneous (it is defined by means of the `AVG` function) and then the predicate `(Price = 100)` cannot be pushed at

the local level: since the AVG function is calculated at a global level, the predicate may be globally true but locally false. The translation of global predicates is not possible on dbhotel because theses (simple) predicates are not quality-supported by dbhotel. Thus, for the query Q1 we obtain the following local query conditions:

C_hotel: true (i.e. the local query does not have a where condition)

C_resort: stars > 3

C_dbhotel: true (i.e. the local query does not have a where condition)

For query Q2 the translation of the predicate (Stars > 3) doesn't change w.r.t. query Q1 since for Stars there are no quality constraints specified. In the translation of the predicate (Price = 100) on the local class hotel; conditions 1, 2 and 3 of definition 23 are true. Specially the condition 3 has the effect of making the local attribute hotel.price homogeneous: since the quality constraint (Price.acc > 0.7) is not satisfied by the other local attribute resort.amount, only values coming from hotel must be considered in the evaluation of predicate (Price = 100), as discussed above; then this predicate can be pushed down on hotel. In this way, for query Q2 we obtain the following local query conditions:

C_hotel: price = 100

C_resort: stars > 3

C_dbhotel: true (i.e. the local query does not have a where condition)

In this way the presence of a quality constraint allows to perform the push down of the predicate (Price = 100) on local class hotel.

2. **Computation of Residual Conditions:** A predicate on a global attribute GA that in step (1) is pushed down on all local classes, is already solved, i.e. in the full join of the local queries this predicate is already satisfied; otherwise the predicate is considered as residual and have to be solved at the global level.

   For the query Q1, the computation of residual conditions gives:

   ```
   Price = 100 and Stars > 3
   ```

   while for query Q2 there are residual conditions:

   ```
   Price = 100 and Stars > 3
   ```

3. **Generation and execution of local queries:** The select list S_L is obtained as $X - Y$ where

   (a) X = union of the attributes of the global select list, of the Join Condition and of the Residual Condition; these attributes are transformed into the corresponding ones at the local level on the basis of the Mapping Table.

   (b) Y = union of local attribute $LA = MT[A][L]$ such that there is a quality constraint on A not satisfied from LA.
   For instance, for the query Q2 since `(Price.acc > 0.7)` is not satisfied by `resort.amount`, this local attribute can be eliminated from the select list of Q_resort. In other words, the presence of quality constraints may allow the elimination of attributes from the select list of a local query: this corresponds to a query optimization technique (called *projection reduction*) since it reduces the size of the local query answer.

With the step 3, query unfolding ends and local queries are generated; for the query Q1 we obtain

```
Q_hotel_1  = select name, price, hotelrooms, wifi
             from hotel

Q_resort_1 = select name, amount, rooms, stars
             from resort
             where stars > 3

Q_dbhotel_1  = select name, dbrooms, dbwifi from dbhotel
```

and for the query Q2 we obtain

```
Q_hotel_2  = select name, hotelrooms, price, wifi
             from hotel
             where price = 100

Q_resort_2 = select name, rooms, stars
             from resort
             where stars > 3

Q_dbhotel_2  = select name, dbrooms, dbwifi from dbhotel
```

Q_resort_1

name	stars	amount	rooms
Hilton	5	120	12
Kyriad	5	350	⊥
Ibis	4	100	6
Garden	4	100	24

Q_hotel_1

name	price	hotelrooms	wifi
Hilton	80	13	false
Kyriad	400	4	true
Garden	100	18	⊥
Sofitel	100	6	true

Q_dbhotel_1

name	dbrooms	dbwifi
Continental	122	true
Ibis	66	false
Garden	18	true

Figura 5.4: Instances of local query answers of Q1 for hotel,resort and dbhotel

Q_resort_2

name	stars	rooms
Hilton	5	12
Kyriad	5	⊥
Ibis	4	6
Garden	4	24

Q_hotel_2

name	price	hotelrooms	wifi
Garden	100	18	⊥
Sofitel	100	6	true

Q_dbhotel_2

name	dbrooms	dbwifi
Continental	122	true
Ibis	66	false
Garden	18	true

Figura 5.5: Instances of local query answers of Q2 for hotel,resort and dbhotel

4. **Fusion of local answers** : The local answers are fused into the global answer on the basis of the mapping query $\mathcal{MQ}^G$ defined for G. As intuitively discussed in Section 1.4.1, $\mathcal{MQ}^G$ is defined by using a Full Outerjoin-merge operation which is substantially performed in two steps:

   (a) Computation of the **full outerjoin** FOJ on the basis of the join conditions $JC(L_i, L_j)$ defined between L_i and L_j.

In more general, the computation of the **full outerjoin** FOJ on **n local classes** is defined as:

```
L1 FJ L2 ON (L1.ID = L2.ID)
    FJ L3 ON (L3.ID = L1.ID OR L3.ID = L2.ID)
    ...
     FJ Ln ON (Ln.ID = L1.ID OR ... OR Ln.ID = Ln-1.ID)
```

where FJ is full outerjoin operator.

We assume **ID** as a share object identifier amond all local classes. For our example, the share object identifier amond our three local classes is the attribute **name**.

For sake of simplicity, we represent the FOJ operation like as a sequence where join condition are omitted:

⟨ L1, FJ, L2, FJ, L3,...., FJ, Ln ⟩

With the query unfolding process for query Q, the FOJ-sequence applied to the local query answers of the related local classes above give:

⟨ Q_L1, FJ, Q_L2, FJ, Q_L3,...., FJ, Q_Ln ⟩

We say that P is supported in a set of local query answers

{ Q_L1, Q_L2, ..., Q_Lk }

if P is supported by the set of related local classes:

{ L1, L2, ..., Lk }

To better understand this equivalence, please refer to section 3.4.1. For the query Q2, the FOJ-sequence applied to the local query answers of `resort`, `hotel` and `dbhotel` give:

⟨ Q_resort_2, FJ, Q_hotel_2, FJ, Q_dbhotel_2 ⟩

(b) **Application of the Resolution Functions :** for each pairs of attributes in FOJ (except for attributes used in the join condition) mapped in the same global attributes the related Resolution Function is applied. For example, in query Q1, we need to apply the resolution function `AVG` and `CONCATENATE` respectively to global attribute `Price` and `Room`.

In step 4.a the computation of the full outerjoin operation can be optimized by reducing it to a left outerjoin (LJ)/right outerjoin (RJ)

or inner join (IJ) on the basis of the following algorithm:

ALGOSIM. Simplification of FJ sequence.
Input: FJ sequence ⟨Q_L1, FJ, Q_L2, FJ, Q_L3,..., FJ, Q_Ln⟩
Output: FJ sequence simplified
Procedure.
1- For i=1 to i=n-1
1.1- let SET_LEFT_i={Q_Lj | j≠i+1} and SET_RIGHT_i={Q_Lj | j>i}
⟨...,Q_Li, FJ, Q_Li+1,..., FJ, Q_Ln ⟩ is simplified as follows
1.2- ⟨....,Q_Li, RJ, Q_Li+1,...., FJ, Q_Ln ⟩
if P is not supported in SET_LEFT_i
1.3- ⟨....,Q_Li, LJ, Q_Li+1,...., FJ, Q_Ln ⟩
if P is not supported in SET_RIGHT_i
1.4- ⟨....,Q_Li, IJ, Q_Li+1,...., FJ, Q_Ln ⟩
if P is not supported in SET_LEFT_i and P is not supported in SET_RIGHT_i
2- Output: FJ sequence simplified

We assume resort as L1, hotel as L2, dbhotel as L3.

The computation of FOJ-sequence for Q1 (FOJ_Q1) and Q2 (FOJ_Q2) give respectively the following instances in figures 5.6 and 5.7.

FOJ_Q1 = ⟨ Q_resort_1, FJ, Q_hotel_1, FJ, Q_dbhotel_1 ⟩

Name	Stars	Price	Room	Wifi
Hilton	5	100	12, 13	false
Kyriad	5	375	4	true
Sofitel	⊥	100	6	true
Ibis	4	100	6, 66	false
Garden	4	100	18, 24, 18	true
Continental	⊥	⊥	122	true

Figura 5.6: Instances of FOJ_Q1 for Q1

For the query Q1, the application of our simplification algorithm to the initial sequence FOJ_Q1 give the next simplified sequence:

⟨ Q_resort_1, LJ, Q_hotel_1, LJ, Q_dbhotel_1 ⟩

FOJ_Q2 = ⟨ Q_resort_2, FJ, Q_hotel_2, FJ, Q_dbhotel_2 ⟩

Name	Stars	Price	Room	Wifi
Hilton	5	⊥	12	⊥
Kyriad	5	⊥	⊥	⊥
Ibis	4	⊥	6, 66	false
Garden	4	100	18, 24, 18	true
Continental	⊥	⊥	122	true

Figura 5.7: Instances of FOJ_Q2 for Q2

The simplified sequence of FOJ_Q1 can be computed as:

```
FOJ_Q1_s = Q_resort_1 R1 left join Q_hotel_1 H1
               on (R1.name=H1.name) left join Q_dbhotel H2
                   on (R1.name=H2.name or H1.name=H2.name)
```

FOJ_Q1_s = ⟨ Q_resort_1, LJ, Q_hotel_1, LJ, Q_dbhotel_1 ⟩

Name	Stars	Price	Room	Wifi
Hilton	5	100	12, 13	false
Kyriad	5	375	4	true
Ibis	4	100	6, 66	false
Garden	4	100	18, 24, 18	true

Figura 5.8: Instances of FOJ_Q1_s is the simplified version of FOJ_Q1

For the Query Q2, the simplied sequence of FOJ_Q2 is:

⟨ Q_resort_2, IJ, Q_hotel_2, LJ, Q_dbhotel_2 ⟩

The simplified sequence of FOJ_Q2 can be computed as:

```
FOJ_Q2_s = Q_resort_2 R1 inner join Q_hotel_2 H1
             on (R1.name = H1.name) left join Q_dbhotel H2
                   on (R1.name=H2.name or H1.name=H2.name)
```

The simplified sequence of Q2 is more optimized than a simplified sequence of Q1 (the first LJ become IJ) because of the presence of data quality constraints in the global query Q2.

FOJ_Q2_s = ⟨ Q_resort_2, IJ, Q_hotel_2, LJ, Q_dbhotel_2 ⟩

Name	Stars	Price	Room	Wifi
Garden	4	100	18, 24, 18	true

Figura 5.9: Instances of FOJ_Q2_s is the simplified version of FOJ_Q2

During the computaton of **ALGOSIM** on the sequence of Q2, we have to control if P (`Price = 100 and Stars > 3`) is not supported by {`resort, dbhotel`}. The criteria condition `Price.acc > 0.7` expressed on the local attribute amount of local source `resort` is not satisfied and the same time all simple predicates are not supported by `dbhotel` ; consequently P is not supported by { `resort, dbhotel`}.

The substitution of a full outer join with a left/right outer join or a join constitutes a relevant query optimization result, since full outer join queries are very expensive, especially in a distributed environment as the one of mediator/integration systems. Moreover, while database optimizers take full advantage of associativity and communtativity properties of join to implement efficient and powerful optimizations on join queries, only limited optimization is performed on full outer join [37]. The presence of quality constraints in the global query may contribute to perform this kind of optimization, as shown for the query Q2.

5. **Application of the Residual Condition:** the result of the global query is obtained by applying the residual condition to the `FOJ`.
In our example, for the query Q1 we have `Price = 100 and Stars > 3` as residual condition then its result is obtained as

```
select Name, Room, Wifi
from FOJ_Q1_s
where Price = 100 and Stars > 3
```

while for the query Q2, the residual condition is the same as for Q1; then its result is obtained as

```
select Name, Room, Wifi
from FOJ_Q2_s
where Price = 100 and Stars > 3
```

5.5 Conclusion

In this Chapter, we presented our current work to extend the MOMIS Data Integration System with Data Quality Information; in particular, we introduced data quality metadata in the specification of queries on the Global Schema, by considering Data Quality Aware Queries. An interesting result is that quality constraints specified in the query are useful to perform some relevant query optimization techniques, such as predicate push down, projection reduction and full outerjoin simplification.

Conclusions and suggestions for future work

This thesis focused on some core aspects in data integration, i.e. Query Processing and Data Quality. First this thesis proposed new techniques that consider the optimization of the full outerjoin operation, which is used in data integration systems for data fusion. Then this thesis demonstrated how to achieve Quality-Driven Query Processing, where quality constraints specified in Data Quality Aware Queries are used to perform query optimization.

Regarding Query Processing and Query Optimization, further developments must focus on the implementation of the proposed optimization techniques in the MOMIS Data Integration System, in order to evaluate their behavior in practice and then to get experimental results. This development should be performed with the open source version of the MOMIS system which is delivered by the academic startup DataRiver (www.datariver.it).

Regarding Quality-Driven Query Processing, further developments could focus on investigate how quality metadata can be used in other important aspects of a Data Integration System. First of all, we can evaluate how quality metadata can be exploited to define quality-based resolution function and how these new resolution functions affect the Query Processing. Moreover, Data Quality Aware Queries can be further investigated, by considering new kind of quality constraints and how the Quality-driven Query Processing need to be extended for these new quality constraints.

Appendice A

The ODL$_{I^3}$ language syntax

The following is the BNF description of the ODL$_{I^3}$ description language. This object-oriented language, with an underlying Description Logic, is introduced for information extraction. The ODL$_{I^3}$ language is presented in [15], in the following I include the syntax fragments which differ from the original ODL grammar, referring to it for the remainder.

⟨interface_dcl⟩ ::= ⟨interface_header⟩
{[⟨ interface_body⟩]**}**;
[**union** ⟨identifier⟩ **{** ⟨interface_body⟩ **};**]
⟨interface_header⟩ ::= **interface** ⟨identifier⟩
[⟨inheritance_spec⟩]
[⟨type_property_list⟩]
⟨inheritance_spec⟩ ::= : ⟨scoped_name⟩
[,⟨inheritance_spec⟩]

Local schema pattern definition: the wrapper must indicate the kind and the name of the source of each pattern.

⟨type_property_list⟩ ::= **(** [⟨source_spec⟩]
[⟨extent_spec⟩]
[⟨key_spec⟩] [⟨f_key_spec⟩] [⟨c_key_spec⟩] **)**
⟨source_spec⟩ ::= **source** ⟨source_type⟩
⟨source_name⟩
⟨source_type⟩ ::= **relational** | **nfrelational**
| **object** | **file**
| **semistructured** | **multimedia**
| **glso**
⟨source_name⟩ ::= ⟨identifier⟩
⟨extent_spec⟩ ::= **extent** ⟨extent_list⟩
⟨extent_list⟩ ::= ⟨string⟩ | ⟨string⟩,⟨extent_list⟩
⟨key_spec⟩ ::= **key**[**s**] ⟨key_list⟩
⟨f_key_spec⟩ ::= **foreign_key** (⟨f_key_list⟩)
references ⟨key_list
⟩ [⟨f_key_spec⟩]
⟨c_key_spec⟩ ::= **candidate_key** ⟨identifier⟩
(⟨key_list⟩**)**

Global pattern definition rule, used to map the attributes between the global definition and the corrisponding ones in the local sources.

⟨attr_dcl⟩ ::= **[readonly] attribute**
[⟨domain_type⟩]
⟨attribute_name⟩ **[*]**
[⟨fixed_array_size⟩]
[⟨mapping_rule_dcl⟩]
⟨mapping_rule_dcl⟩ ::= **mapping_rule** ⟨rule_list⟩
⟨rule_list⟩ ::= ⟨rule⟩ | ⟨rule⟩**,**⟨rule_list⟩
⟨rule⟩ ::= ⟨local_attr_name⟩ |
'⟨identifier⟩'
⟨and_expression⟩ |
⟨union_expression⟩
⟨and_expression⟩ ::= **(** ⟨local_attr_name⟩ **and**
⟨and_list⟩ **)**
⟨and_list⟩ ::= ⟨local_attr_name⟩
| ⟨local_attr_name⟩ **and**
⟨and_list⟩
⟨union_expression⟩ ::= **(** ⟨local_attr_name⟩ **union**
⟨union_list⟩ **on** ⟨identifier⟩ **)**
⟨union_list⟩ ::= ⟨local_attr_name⟩
| ⟨local_attr_name⟩ **union**
⟨union_list⟩
⟨local_attr_name⟩ ::= ⟨source_name⟩**.**⟨class_name⟩**.**
⟨attribute_name⟩
...

Terminological relationships used to define the Common Thesaurus.

⟨relationships_list⟩ ::= ⟨relationship_dcl⟩**;** |
⟨relationship_dcl⟩**;**
⟨relationships_list⟩
⟨relationships_dcl⟩ ::= ⟨local_name⟩
⟨relationship_type⟩
⟨local_name⟩
⟨local_name⟩ ::= ⟨source_name⟩**.**
⟨local_class_name⟩
[**.**⟨local_attr_name⟩]
⟨relationship_type⟩ ::= **SYN** | **BT** | **NT** | **RT**
...

Extended base type definition for multimedia objects.

⟨BaseTypeSpec⟩ ::= ⟨FloatingPtType⟩ |
⟨IntegerType⟩ |
⟨CharType⟩ |
⟨BooleanType⟩ |
⟨OctetType⟩ |
⟨RangeType⟩ |
⟨AnyType⟩ |
⟨MultiMediaType⟩

⟨MultiMediaType⟩ ::= **Text** |
Image

OLCD integrity constraint definition: declaration of rule (using *if then* definition) valid for each instance of the data; mapping rule specification (*or* and *union* specification rule).

⟨rule_list⟩ ::= ⟨rule_dcl⟩**;** | ⟨rule_dcl⟩**;** ⟨rule_list⟩
⟨rule_dcl⟩ ::= **rule** ⟨identifier⟩ ⟨rule_spec⟩
⟨rule_spec⟩ ::= ⟨rule_pre⟩ **then** ⟨rule_post⟩ |
{ ⟨case_dcl⟩ **}**
⟨rule_pre⟩ ::= ⟨forall⟩ ⟨identifier⟩ **in** ⟨identifier⟩ **:**
⟨rule_body_list⟩
⟨rule_post⟩ ::= ⟨rule_body_list⟩
⟨case_dcl⟩ ::= **case of** ⟨identifier⟩ **:** ⟨case_list⟩
⟨case_list⟩ ::= ⟨case_spec⟩ | ⟨case_spec⟩ ⟨case_list⟩
⟨case_spec⟩ ::= ⟨identifier⟩ **:** ⟨identifier⟩ **;**
⟨rule_body_list⟩ ::= **(** ⟨rule_body_list⟩ **)** |
⟨rule_body⟩ |
⟨rule_body_list⟩ **and**
⟨rule_body⟩ |
⟨rule_body_list⟩ **and**
(⟨rule_body_list⟩ **)**
⟨rule_body⟩ ::= ⟨dotted_name⟩
⟨rule_const_op⟩
⟨literal_value⟩ |
⟨dotted_name⟩
⟨rule_const_op⟩
⟨rule_cast⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ **in**
⟨dotted_name⟩ |
⟨forall⟩ ⟨identifier⟩ **in**
⟨dotted_name⟩ **:**
⟨rule_body_list⟩ |
exists ⟨identifier⟩ **in**
⟨dotted_name⟩ **:**
⟨rule_body_list⟩
⟨rule_const_op⟩ ::= = | ≥ | ≤ | > | <
⟨rule_cast⟩ ::= **(**⟨simple_type_spec⟩**)**
⟨dotted_name⟩ ::= ⟨identifier⟩ | ⟨identifier⟩**.**
⟨dotted_name⟩
⟨forall⟩ ::= **for all** | **forall**

Appendice B

A Mapping Refinement

Query 1 Mapping between Instructor attribute of Georgia Tech University and Lecturer attribute of Carnegie Mellon University. No mapping refinement.

Query 2 Mapping between Time attribute of Carnegie Mellon University and Times attribute of University of Massachusetts. Mapping refinement:

```
MDTF[Time][umb.Course] = TIME12-24(Times, 1, 12) +
                                          SUBSTRING(Times, 6, 1) +
                                           TIME12-24(Times, 7, 12)
```

Query 3 Mapping between CourseName attribute of University of Maryland and Title attribute of Brown University. Mapping refinement:

```
MDTF[Title][brown.Course] = SUBSTRING(Title FROM
      POSITION('/"' IN Title) + 3 FOR POSITION ('hr.' IN
             SUBSTRING (Title FROM POSITION ('/"' IN Title)
                                        + 3 FOR 100)) - 1)
```

Query 4 Mapping between Units attribute of Carnegie Mellon University and Umfang attribute of ETH Zurich. Mapping refinement:

```
MDTF[Unit][ethz.Unterricht] =
CAST(SUBSTRING(Umfang, POSITION('V' IN Umfang) - 1, 1) AS int)
+
CAST(SUBSTRING(Umfang,POSITION('U' IN Umfang) - 1,1) AS int) + 1
```

Query 5 Mapping between CourseName attribute of University of Maryland and Title attribute of ETH Zurich. No mapping refinement. **Query 6**

Mapping between title attribute of University of Toronto and no attribute of Carnegie Mellon University. **Query 7** Mapping between prerequisite attribute of University of Michigan and description attribute in Arizona State University. Mapping refinement:

```
MDTF[prerequisite][asu.Course] =
CASE POSITION('%Prerequisite%' IN Description)
WHEN 0 THEN 'None'
           ELSE RIGHT(Description,
               CHAR_LENGTH(Description) -
                 POSITION('%Prerequisite%' IN Description) + 1)
END
```

Query 8 Mapping between Course restricted attribute of Georgia Tech University and no attribute of ETH Zurich. No mapping refinement. **Query 9** Mapping between room attribute of Brown University and time attribute of University of Maryland. Mapping refinement:

```
MDTF[Room][umd.section] =   SUBSTRING(Time FROM
                                 POSITION('%(%' IN Time) FOR 30)
```

Query 10 Mapping between lecturer attribute of Carnegie Mellon University and title attribute of University of Maryland. Mapping refinement:

```
MDTF[Title][umd.section] =   SUBSTRING(Title FROM
POSITION('%.%' IN Title) FOR POSITION('%)%' IN Title) + 2)
FOR POSITION('%.%' IN Title) + 1)
```

Query 11 Mapping between lecturer attribute of Carnegie Mellon University and attributes named Fall2003, Winter2004 and Spring2004 of University of California, San Diego. Mapping refinement:

```
MDTF[Lecturer][ucsd.Course] =
CASE  WHEN   (CHAR_LENGTH (Fall2003) >
CHAR_LENGTH (Winter2004)  AND
CHAR_LENGTH (Fall2003) > CHAR_LENGTH (Spring2004))
THEN Fall2003
WHEN (CHAR_LENGTH (Winter2004) >
CHAR_LENGTH (Fall2003) AND CHAR_LENGTH
               (Winter2004) > CHAR_LENGTH (Spring2004))
THEN Winter2004
WHEN (CHAR_LENGTH (Spring2004) >
                 CHAR_LENGTH (Fall2003) AND
```

```
          CHAR_LENGTH (Spring2004) >
           CHAR_LENGTH (Winter2004)) THEN Spring2004
END
```

Query 12 Mapping between CourseTitle, Day, Time attribute of Carnegie Mellon University and Title attribute of Brown University. Mapping refinement:

```
MDTF[Title][brown.Course] =
SUBSTRING(Title FROM POSITION('/"' IN Title) + 3 FOR
             POSITION ('hr.' IN SUBSTRING(Title FROM
                      POSITION('/"' IN Title) + 3 FOR 100)) - 1)
MDTF[Day][brown.Course] =
SUBSTRING(Title FROM POSITION('hr.' IN Title) + 4 FOR
           POSITION(' ' IN SUBSTRING(Title
                      FROM POSITION('hr.' IN Title) + 4 FOR10)))
MDTF[Time][brown.Course] =
SUBSTRING(Title IN POSITION(' ' FROM SUBSTRING(Title
                FROM POSITION('hr.' IN Title) + 4 FOR 10)) +
                           POSITION('hr.' IN Title) + 4 FOR 15)
```

Appendice C

Data Quality Dimensions / Examples

Data quality is not linear and has many dimensions like Accuracy, Completeness, Consistency, and Timeliness (Currency). Having data quality on one dimension is as good as 'no quality'.

None of the Data Quality dimensions is complete by itself, and many a times dimensions are overlapping.

Data Accuracy dimension of Data Quality

Accuracy of data is the degree to which data correctly reflects the real world object or an event being described.

Examples of Data Accuracy:

- The address of customer in the customer database is the real address.
- The temperature recorded in the thermometer is the real temperature.
- The bank balance in the customer's account is the real value customer deserves from the Bank.

Data Completeness dimension of Data Quality

Completeness of data is the extent to which the expected attributes of data are provided.
For example, a customer data is considered as complete if:

- All customer addresses, contact details and other information are available.

- Data of all customers is available.

Data Completeness definition is the 'expected completeness'. It is possible that data is not available, but it is still considered completed, as it meets the expectations of the user. Every data requirement has 'mandatory' and 'optional' aspects. For example customer's mailing address is mandatory and it is available and because customers office address is optional, it is OK if it is not available. Data can be complete, but inaccurate:

- All the customers' addresses are available, but many of them are not correct.
- The health records of all patients have 'last visit' date, but some of it contains the future dates.

Data Consistency dimension of quality of data

Consistency of Data means that data across the enterprise should be in synch with each other.
Examples of data inconsistency are:

- An agent is inactive, but he still has his disbursement account active.
- A credit card is cancelled, and inactive, but the card billing status shows 'due'.

Data can be accurate (i.e., it will represent what happened in real world), but still inconsistent.

- An Airline promotion campaign closure date is Jan 31, and there is a passenger ticket booked under the campaign on Feb. 2.

Data is inconsistent, when it is in synch in the narrow domain of an organization, but not in synch across the organization. For example:

- Collection management system has the Cheque status as 'cleared', but in the accounting system, the money is not shown being credited to the bank account. Reason for this kind of inconsistency is that system interfaces are synchronized during the end-of-day batch runs.

Data can be complete, but inconsistent

- Data for all the packets dispatched from New York to Chicago are available., but some of the packages are also shown as 'under bar-coding' status.

Data Timeliness: 'Data delayed' is 'Data Denied'

The **timeliness** of data is extremely important. This is reflected in:

- Companies are required to publish their quarterly results with in a given frame of time.
- Customers service providing up-to date information to the customers.
- Credit system checking on the credit card account activity.

The timeliness depends on user expectation. An online availability of data could be required for room allocation system in Hospitality, but an overnight data is fine for a billing system.
Example of Data not being timely (OutDated):

- The courier package status is delivered, but it will be updated in the system only in the night batch run. This means that online status will not be available.
- The financial statements of a company are published one month after the year-end.
- The census data is available two years after the census is done.

Bibliografia

[1] S. Abiteboul, R. Agrawal, P. Bernstein, M. Carey, S. Ceri, B. Croft, D. DeWitt, M. Franklin, H. G. Molina, D. Gawlick, J. Gray, L. Haas, A. Halevy, J. Hellerstein, Y. Ioannidis, M. Kersten, M. Pazzani, M. Lesk, D. Maier, J. Naughton, H. Schek, T. Sellis, A. Silberschatz, M. Stonebraker, R. Snodgrass, J. Ullman, G. Weikum, J. Widom, and S. Zdonik. The lowell database research self-assessment. *Commun. ACM*, 48:111–118, May 2005.

[2] J. L. Ambite and C. A. Knoblock. Flexible and scalable cost-based query planning in mediators: A transformational approach. *Artificial Intelligence Journal*, 118:1–2, 2000.

[3] D. Ballou, R. Wang, H. Pazer, and G. K. Tayi. Modeling information manufacturing systems to determine information product quality. *Manage. Sci.*, 44:462–484, April 1998.

[4] M. S. E. Ballou, Donald and R. Y. Wang. Special section: Assuring information quality. *Journal of Management Information Systems*, 20(3):9–11, 2004.

[5] C. Batini and M. Scannapieco. Data quality: Concepts, methodologies and techniques. *Springer Verlag, 2006.*

[6] D. Beneventano and S. Bergamaschi. Semantic search engines based on data integration systems. In *Semantic Web Services: Theory, Tools and Applications.* Idea Group., 2006. To appear. Available at http://dbgroup.unimo.it/SSE/SSE.pdf.

[7] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Orsini. Data integration. In D. Embley and B. Thalheim, editors, *Handbook of conceptual modelling*. Springer-Verlag, 2010. To appear. Available at http://dbgroup.unimo.it/SSE/SSE.pdf.

[8] D. Beneventano, S. Bergamaschi, and C. Sartori. Description logics for semantic query optimization in object-oriented database systems. *ACM Trans. Database Syst.*, 28:1–50, 2003.

[9] D. Beneventano, S. Bergamaschi, M. Vincini, M. Orsini, and R. C. N. Mbinkeu. Query translation on heterogeneous sources in momis data transformation systems. In *VLDB 2007 - International Workshop on Database Interoperability (InterDB 2007), Vienna, Austria, September 24, 2007.*

[10] D. Beneventano, S. Bergamaschi, M. Vincini, M. Orsini, and R. C. Nana Mbinkeu. Getting through the thalia benchmark with momis. In *Proceedings of the Third International Workshop on Database Interoperability (InterDB 2007) held in conjunction with the 33rd International Conference on Very Large Data Bases, VLDB 2007, Vienna, Austria, September 24*, 2007.

[11] D. Beneventano, F. Guerra, M. Orsini, L. Po, A. Sala, M. D. Gioia, M. Comerio, F. de Paoli, A. Maurino, M. Palmonari, C. Gennaro, F. Sebastiani, A. Turati, D. Cerizza, I. Celino, and F. Corcoglioniti. Detailed design for building semantic peer. *Networked Peers for Business, Deliverable D.2.1, Final Version, available at http : //www.dbgroup.unimo.it/publication/d2_1.pdf*, pages 52–57, apr 2008.

[12] D. Beneventano and M. Lenzerini. Final release of the system prototype for query management. sewasie, deliverable d.3.5, final version. http://www.dbgroup.unimo.it/prototipo/paper/D3.5Final.pdf, April 2005.

[13] D. Beneventano and R. C. N. Mbinkeu. Quality-driven query processing techniques in the momis integration system. In *ADBIS*, pages 46–57, 2010.

[14] D. Beneventano, R. C. N. Mbinkeu, C. Gennaro, and M. Mordacchini. Query processing in a mediator system for data and multimedia. In *workshop of interoperability through Semantic Data and Service Integration Co-located with SEBD,June 25th,Camogli (Genova),Italy*, 2009.

[15] S. Bergamaschi, S. Castano, M. Vincini, and D. Beneventano. Semantic integration of heterogeneous information sources. *Data Knowl. Eng.*, 36(3):215–249, 2001.

[16] S. Bergamaschi, L. Po, and S. Sorrentino. Automatic annotation in data integration systems. In R. Meersman, Z. Tari, and P. Herrero, editors, *OTM Workshops (1)*, volume 4805 of *Lecture Notes in Computer Science*, pages 27–28. Springer, 2007.

[17] S. Bergamaschi, L. Po, and S. Sorrentino. Automatic annotation for mapping discovery in data integration systems. In S. Gaglio, I. Infantino, and D. Saccà, editors, *SEBD*, pages 334–341, 2008.

[18] S. Bergamaschi, C. Sartori, D. Beneventano, and M. Vincini. Odb-tools: A description logics based tool for schema validation and semantic query optimization in object oriented databases. In M. Lenzerini, editor, *AI*IA*, volume 1321 of *Lecture Notes in Computer Science*, pages 435–438. Springer, 1997.

[19] L. Berti-Equille. Integration of biological data and quality-driven source negotiation. In *Proceedings of the 20th International Conference on Conceptual Modeling: Conceptual Modeling*, ER '01, pages 256–269, London, UK, 2001. Springer-Verlag.

[20] L. Berti-Equille. Quality-adaptive query processing over distributed sources. In *Information quality management: theory and applications*, 2007.

[21] J. Bleiholder and F. Naumann. Declarative data fusion - syntax, semantics, and implementation. In *Advances in Databases and Information Systems - ADBIS*, pages 58–73, 2005.

[22] J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, 41(1), 2008.

[23] J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, 41:1:1–1:41, January 2009.

[24] A. Cali, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Data integration under integrity constraints. In *Information Systems*, pages 262–279. Springer, 2002.

[25] C. Cappiello, C. Francalanci, and B. Pernici. Time-related factors of data quality in multichannel information systems. *J. Manage. Inf. Syst.*, 20:71–92, December 2003.

[26] S. Castano, V. D. Antonellis, and S. D. C. di Vimercati. Global viewing of heterogeneous data sources. *IEEE Trans. Knowl. Data Eng.*, 13(2):277–297, 2001.

[27] K. C.-C. Chang and H. Garcia-Molina. Mind your vocabulary: Query mapping across heterogeneous information sources. In A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors, *SIGMOD Conference*, pages 335–346. ACM Press, 1999.

[28] A. L. P. Chen. Outerjoin optimization in multidatabase systems. In *Proceedings of the second international symposium on Databases in parallel and distributed systems table of contents. Dublin, Ireland*, pages 211–218. ACM, 1990.

[29] A. L. P. Chen. Outerjoin optimization in multidatabase systems. In *Proceedings of the second international symposium on Databases in parallel and distributed systems*, DPDS '90, pages 211–218, New York, NY, USA, 1990. ACM.

[30] S. Cohen and B. Kimelfeld. Full disjunctions: Polynomial-delay iterators in action. In *In VLDB*, 2006.

[31] T. N. Consortium. Revised specification for building semantic peer. Nep4b - networked peers for business, deliverable d2.2, Dipartimento di Ingegneria dell'Informazione, 2009. http://dbgroup.unimo.it/TechnicalReport/NeP4BD2.2.pdf.

[32] Y. H. et al. Data quality , information and software technology, 32(8), pp. 559-565.

[33] C. Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[34] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *In Proceedings of the National Conference on Artificial Intelligence (AAAI*, pages 67–73. AAAI Press/The MIT Press, 1999.

[35] P. G. G. Bhargava and B. Iyer. Hypergraph based reordering of outer join queries with complex predicates. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data, San Jose, California, United States*, pages 304 – 315. ACM, 1995.

[36] C. A. Galindo-legaria. Outerjoins as disjunctions. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 348–358. ACM Press, 1994.

[37] C. A. Galindo-Legaria and A. Rosenthal. Outerjoin simplification and reordering for query optimization. *ACM Trans. Database Syst.*, 22(1):43–73, 1997.

[38] M. R. Genesereth, A. M. Keller, and O. Duschka. Infomaster: An Information Integration System. In *Proceedings of 1997 ACM SIGMOD Conference*, 1997.

[39] M. Gertz and I. Schmitt. Data integration techniques based on data quality aspects. In *3rd National Workshop on Federal Databases*, 1998.

[40] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10:270–294, December 2001.

[41] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[42] T. Harju. Graph theory. In *Lectures Notes of Mathematics*, 1994-2011.

[43] M. A. Hernndez and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Min. Knowl. Discov.*, pages 9–37, 1998.

[44] O. T. Joachim Hammer, Mike Stonebraker. Thalia: Test harness for the assessment of legacy information integration approaches. In *In Proceedings of the International Conference on Data Engineering (ICDE*, pages 485–486, 2005.

[45] P. Katerattanakul and K. Siau. Measuring information quality of web sites: development of an instrument. In *Proceedings of the 20th international conference on Information Systems*, ICIS '99, pages 279–285, Atlanta, GA, USA, 1999. Association for Information Systems.

[46] W. Kim, B.-J. Choi, E.-K. Hong, S.-K. Kim, and D. Lee. A taxonomy of dirty data. *Data Min. Knowl. Discov.*, 7:81–99, January 2003.

[47] M. Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 233–246, New York, NY, USA, 2002. ACM.

[48] M. Lenzerini. Data integration: A theoretical perspective. In L. Popa, editor, *PODS*, pages 233–246. ACM, 2002.

[49] A. Y. Levy. The information manifold approach to data integration. *IEEE Intelligent Systems*, 13:12–16, 1998.

[50] C. Li, R. Yerneni, V. Vassalos, H. Garcia-Molina, Y. Papakonstantinou, J. D. Ullman, and M. Valiveti. Capability based mediation in tsimmis. In *SIGMOD Conference*, pages 564–566, 1998.

[51] R. C. N. Mbinkeu. Full outer join optimization techniques in integration information systems. *6th French national conference of PhD student, Majectics08, Marseille France, October 2008.*

[52] M. Mecella, M. Scannapieco, A. Virgillito, R. Baldoni, T. Catarci, C. Batini, and C. N. D. Ricerche. Managing data quality in cooperative information systems (extended abstract). In *In Proc. of the 10th International Conference on Cooperative Information Systems (CoopIS*, pages 486–502. Springer, 2002.

[53] R. J. Miller, M. A. Hernández, L. M. Haas, L. Yan, C. T. Howard Ho, R. Fagin, and L. Popa. The clio project: managing heterogeneity. *SIGMOD Rec.*, 30:78–83, March 2001.

[54] P. Missier, S. Embury, M. Greenwood, A. Preece, and B. Jin. Quality views: capturing and exploiting the user perspective on data quality. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 977–988. VLDB Endowment, 2006.

[55] A. Motro and P. Anokhin. Fusionplex: resolution of data inconsistencies in the integration of heterogeneous information sources. *Information Fusion , Volume 7 Issue 2, june 2007. Elsevier Science Publishers B. V.*

[56] H. Muller and J.-C. Freytag. Problems, methods, and challenges in comprehensive data cleansing. *Technical Report HUB-IB-164*, page 23, 2003.

[57] K. Munakata. Integration of maximum information using outerjoins, predicates and foreign functions. In *IEICE TRANSACTIONS on Information and Systems Vol.E82-D No.1 pp.64-75*, 1999.

[58] F. Naumann. *Quality-driven query answering for integrated information systems.* Springer-Verlag, Berlin, Heidelberg, 2002.

[59] F. Naumann and J. Bleiholder. Conflict handling strategies in an integrated information system. In *Proceedings of the WWW Workshop in Information integration on the Web (IIWEB)*, 2006.

[60] F. Naumann, J. C. Freytag, and U. Leser. Completeness of integrated information sources. *Inf. Syst.*, 29(7):583–615, 2004.

[61] F. Naumann and M. Häussler. Declarative data merging with conflict resolution. In C. Fisher and B. N. Davidson, editors, *IQ*, pages 212–224. MIT, 2002.

[62] F. Naumann, U. Leser, and J.-C. Freytag. Quality-driven integration of heterogeneous information systems. In *In VLDB Conference*, pages 447–458, 1999.

[63] F. Naumann and C. Rolker. Assessment methods for information quality criteria. In *In Proceedings of the International Conference on Information Quality (IQ*, pages 148–162, 2000.

[64] M. T. Neiling and H.-J. Lenz. Data integration by means of object identification in information systems. In *In Proceedings of European Conference on Information Systems*, 2000.

[65] J. night, S.A. Burn. Developing a framework for assessing information quality on the world wide web. In *Proc. Information Science + Information Technology Education Joint Conference (InSITE), June 16-19, Flagstaff, Arizona.*, 2005.

[66] P. Oliveira, F. Rodrigues, P. R. Henriques, and H. Galhardas. A taxonomy of data quality problems. In *2nd Int. Workshop on Data and Information Quality*, pages 219–233, Porto, Portugal, Jun 2005. (em conjunto com a conferencia CAiSE'05).

[67] A. C. On, G. Shanks, and B. Corbitt. Abstract understanding data quality: Social and cultural aspects proc. 10 th australasian conference on information systems, 1999.

[68] M. Orsini. Interoperabilitá tra ontologie eterogenee: i traduttori OWL - ODLI3. Master's thesis, University of Modena and Reggio Emilia, 2003/2004.

[69] M. Orsini. *Query Management in Data Integration Systems: the MOMIS approach.* PhD thesis, International Doctorate School in Information and Communication Technologies of the University of Modena and Reggio Emilia, 2009.

[70] L. Po. *Automatic Lexical Annotation: an effective technique for dynamic data integration.* PhD thesis, International Doctorate School in Information and Communication Technologies of the University of Modena and Reggio Emilia, 2009.

[71] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23:2000, 2000.

[72] Rajaraman and J. D. Ullman. Integrating information by outerjoins and full disjunctions. In *In Proceedings of the 15th ACM Symposium on Principles of Database Systems, Montreal, (Canada), June 1996.*

[73] A. Rajaraman and J. D. Ullman. Integrating information by outerjoins and full disjunctions (extended abstract). In *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '96, pages 238–248, New York, NY, USA, 1996. ACM.

[74] D. research group of Modena University(DBgroup). State of the art of current p2p and ontology languages initiatives. In *Networked Peers for Business*, 2006.

[75] Y. W. L. Richard Y. Wang, Mostapha Ziad. In *Data Quality*. Kluwer Academic Publishers, 2001.

[76] A. Rosenthal and D. Reiner. Extending the algebric framework of query processing to handle outerjoins. In *In Proceedings of the 10th International Conference on Very Large Data Bases, Singapore*, pages 334–343, 1984.

[77] A. Sala. Data and service integration architectures and applications to real domains. *PhD Thesis, International Doctorate School in Information and Communication Technologies of the University of Modena and Reggio Emilia.*, page 147, 2010.

[78] M. Scannapieco, P. Missier, and C. Batini. Data quality at a glance. *Datenbank-Spektrum*, 14:6–14, 2005.

[79] M. Scannapieco, A. Virgillito, C. Marchetti, M. Mecella, and R. Baldoni. The daquincis architecture: a platform for exchanging and improving data quality in cooperative information systems. *Inf. Syst.*, 29:551–582, September 2004.

[80] M. Scannapieco, A. Virgillito, M. Marchetti, M. Mecella, and R. Baldoni. The daquin-cis architecture: a platform for exchanging and improving data quality in cooperative information systems. *Information Systems, 29(7):551582, 2004.*

[81] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, and Lorie. Access path selection in a relation database system. In *In proceedings*

of ACM SIGMOD,International Conference on Management of Data. USA, Boston, 1979.

[82] V. Sessions and M. Valtorta. Towards a method for data accuracy assessment utilizing a bayesian network learning algorithm. *J. Data and Information Quality*, 1:14:1–14:34, December 2009.

[83] D. M. Strong, Y. W. Lee, and R. Y. Wang. Data quality in context. *Commun. ACM*, 40:103–110, May 1997.

[84] S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Inf. Syst.*, 26:607–633, December 2001.

[85] J. D. Ullman. Information integration using logical views. In *Proceedings of the 6th International Conference on Database Theory*, pages 19–40, London, UK, 1997. Springer-Verlag.

[86] J. D. Ullman. Information integration using logical views. In F. N. Afrati and P. G. Kolaitis, editors, *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 1997.

[87] I. V, M. Jarke, M. Jarke, L. F. I. V, Y. Vassiliou, and Y. Vassiliou. Data warehouse quality: A review of the dwq project, 1997.

[88] R. Y. Wang, M. P. Reddy, and H. B. Kon. Toward quality data: an attribute-based approach. In *Special issue on information technologies and systems*, pages 349 – 372. Elsevier Science Publishers, Mar. 1995.

[89] R. Y. Wang and D. M. Strong. Beyond accuracy: what data quality means to data consumers. *J. Manage. Inf. Syst.*, 12:5–33, March 1996.

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

[91] G. Wiederhold. Intelligent integration of information. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, SIGMOD '93, pages 434–437, New York, NY, USA, 1993. ACM.

[92] W. E. Winkler, W. E. Winkler, and N. P. Overview of record linkage and current research directions. Technical report, Bureau of the Census, 2006.

[93] N. K. Yeganeh, S. W. Sadiq, K. Deng, and X. Zhou. Data quality aware queries in collaborative information systems. In *APWeb/WAIM*, pages 39–50, 2009.

[94] O. B. C. T. K. L. Yu Bei, Liu Ling. Keyword join: Realizing keyword search for information integration. *MIT PRESS*, 2006.

[95] T. R. YU Huh, FR Keller and A. Watkins. Data quality. *Information and Software Technology*, 32:559–565, 1990.

[96] P. ke Larson and J. Zhou. View matching for outer-join views. In *Proceedings of the 31st international conference on Very large data bases, August 30-September 02, 2005, Trondheim, Norway*, pages 445–456. Springer-Verlag, 2005.