A technique for information system integration

Sergio Greco, **Luigi Pontieri** and **Ester Zumpano**
DEIS - University of Calabria
ISI-CNR -- National Research Council
87030 Rende, Italy
{greco,pontieri,zumpano}@si.deis.unical.it

Abstract: Nowadays, a central topic in database science is the need of an integrated access to large amounts of data provided by various information sources whose contents are strictly related. Often information sources have been designed independently for autonomous applications, so they may present several kinds of heterogeneity. Particularly hard to manage is the semantic heterogeneity, which is due to schema and value inconsistencies. In this paper, we focus our attention mainly on the inconsistency which arises when conflicting instances related to the same concept and possibly coming from different sources are integrated. First, we introduce an operator, called *Merge Operator*, which allows us to combine data coming from different sources, preserving the information contained in each of them. Then, we present a variant of this operator, the *Extended Merge Operator*, which associates the integrated data with some information about the process by which they have been obtained. Finally, in order to manage conflicts among integrated data, we briefly present a technique for computing consistent answers over inconsistent databases.

1. Introduction

The problem of integrating heterogeneous sources has been deeply investigated in the fields of multidatabase systems [Br90], federated databases [Wi92] and, more recently, mediated systems [Wi92] and data warehousing [CD97,In97]. In this paper we deal with the problem of integrating heterogeneous sources using a mediator-based approach. Integrating data from multiple heterogeneous sources determines two main different kinds of inconsistency: schema conflicts, which occur when different sources use different schemas to model the same concept, and data conflicts, which arise when different sources record different values for the same object [Hu97,YO99]. In this paper, we focus our attention on the integration of conflicting instances [Ar95,ABC99,Du96] related to the same concept and possibly coming from different sources. Typically, databases contain intentional knowledge expressed by means of integrity constraints, that give information on the form the data must have. Contraints (such as functional dependencies, inclusion dependencies, etc.) are mainly introduced to prevent incorrect database states. In the approach we propose, a mediator M integrates the information provided by a set of sources $D_1,,...,D_m$ preserving, as much as possible, the set of constraints defined on each source and on the global view furnished by the mediator. In particular, we define an operator, called *Merge Operator,* which allows us to complete data contained in each source preserving the integrity constraints defined on it. A variant of the merge operator, especially useful within the materialized view approach, is the *Extended Merge Operator*, which extends any integrated tuple by storing information relative to the process by which it has been obtained. Value inconsistencies, i.e. violations of integrity constraints, may be present in the integrated view provided by a mediator. In this paper we just consider inconsistencies related to the violation of functional dependencies. In order to manage this kind of inconsistencies we employ a technique defined in [GZ00], where a general bgic framework for computing repairs and consistent answers over inconsistent databases has been proposed. The technique, based on disjunctive programs and stable model semantics, can be used to produce consistent answers over inconsistent database, i.e. maximal set of atoms which do not violate the constraints. The rest of the paper is organized as follows. In Section 2 we present the system architecture used to perform integration process. Section 3 presents the *Merge Operator*, while the *Extended Merge Operator* is described in Section 4. Finally, in Section 5 we briefly illustrate the technique introduced in [GZ00], which we use to compute consistent answers over inconsistent mediator views.

2. System Architecture

In order to perform the integration of multiple heterogeneous sources, we use a common architecture based on mediators, shown in Figure 1.

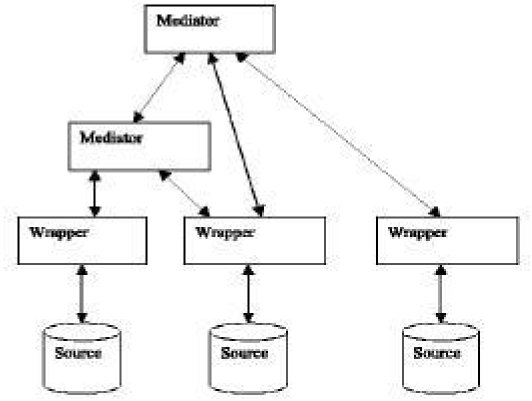

Fig. 1. System Architecture

Mediator-based architectures are characterized by the presence of two types of components: *wrappers*, which translate local languages, models and concepts of the data sources into the global ones, and *mediators*, which take in input information from one or more components below them and provide an integrated view of it [LRO96,Ga97]. Views, managed by mediators, may be virtual or materialized. When a mediator receives a query, it dispatches subqueries to components below it (wrappers and/or mediators), collects the results and merges them in order to construct the global answer. Mediators provide an integrated view of a set of information sources. Each of these sources may be a source database or a database view (virtual or materialized) furnished by another mediator. We denote the former kind of source by *basic source* and the latter one by *derived source*. Similarly, a relation provided by a basic source is said to be a *basic relation* otherwise it is said to be a *derived relation*. A mediator has its own schema, that

we call *mediator schema*, and a set of integrity constraints whose satisfaction means that data are consistent. Integrity constraints, which can also be associated with a source schema, are first order formulas that must always be true. Although in this paper we only consider functional dependencies, our approach to manage inconsistent data is more general. The mediator schema represents, in an integrated way, some relevant concepts that may be modelled differently within the schemas of different sources.
At each level of the integration system, the information provided by different sources and related to the same concept is combined. The necessity of completing the information regarding a concept is due to the fact that some information may not be available at a source because it is not modeled within the schema of the source or simply because some data instances contain undefined values for some attributes. A mediator integrates different sources trying to preserve the available constraints by using one of the merging operators defined in Sections 3 and 4, and manages conflicting values by applying the technique described in Section 5.

3. Data Integration

A mediator has to define the content of any global relation as an integrated view of the information provided by the relations it integrates. We assume that relations corresponding to the same concept and furnished by different sources of a mediator are homogenized with respect to a common ontology, so that attributes denoting the same property have the same name [YO99]. Once the logical conflicts due to the schema heterogeneity have been resolved, conflicts may arise, during the integration process, among instances provided by different sources. In particular, the same real-world object may correspond to many tuples, that may have the same value for the key attributes but different values for some non-key attribute.

Let us introduce some simple definitions in order to simplify the description of our approach. We adopt the relational model for referring to schemas and instances pertaining to the mediator and the sources it integrates.

Let *R* be a relation name, then we denote by: *attr(R)* the set of attributes of *R*; *key(R)* the set of attributes in the primary key of *R*; *inst(R)* the instance of *R* (set of tuples). Moreover, given a tuple *t* *inst(R), key(t)* denotes the values of the key attributes of *t*. The absence of information for an attribute is indicated by the symbol .

Definition 1. A relation *R* is said to be *full* if each tuple *t* in *R* *t(a)* *a* a*ttr(R).*

We say that two homogenized relations R and *S*, associated to the same concept, are *overlapping* if key(R) = key(S).

Definition 2 . Given two relations *R* and *S* s.t. *att(R)* *att(S*) and two tuples *r* *R* and *s* *S,* we say that r $\sqsubseteq$ s if for each attribute A in att(R), is r[A] = s[A] or r[A]= . Moreover we say R $\sqsubseteq$ S if t_1 inst(R) t_2 inst(S) s.t. $t_1 \sqsubseteq t_2$.

Definition 3. Let $R_1,...,R_n$ be a set of overlapping relations. A relation R is a *super relation of* $R_1,...,R_n$ if the following conditions hold:

$$\mathrm{attr}(R) = {}_{i=1}^{n}\ \mathrm{attr}(R_i)$$

$R_i \sqsubseteq| R$,

key(R) = key(R_i)   i=1..n.

Moreover, if R is a super relation of $R_1,...,R_n$, then we say that R_i is a *sub-relation* of R for i=1..n.

A set of tuples with the same value for the key attributes is called *c-tuple* (cluster of tuples) [Ar95]. A relation may be seen as a set of c-tuples. An important feature of the integration process is related to the way conflicting tuples provided by overlapping relations are combined. In the following section we define an operator which allows us to integrate a set of relations preserving the original information.

3.1 The Merge Operator

Given two overlapping relations S_1 and S_2, the *merge operator*, denoted by ⊠, integrates the information provided by S_1 and S_2. Let S= S_1 ⊠ S_2, then the schema of S contains both the attributes in S_1 and S_2, and its instance is obtained by completing the information coming from each input relation with that coming from the other one.

Definition 4. Let S_1 and S_2 be two overlapping relations and let

$$S_{1,2} = S_1 ⟕_{key(S_1)=key(S_2)} S_2 \text{ (resp. } S_{2,1} = S_1 ⟖_{key(S_1)=key(S_2)} S_2$$

be the result of the left (resp. right) outer join of S_1 and S_2 with join condition key(S_1)=key(S_2).

The *merge operator* is a binary operator defined as follows:

$$S_1 \boxtimes S_2 = \Theta(S_{1,2} \bowtie S_1, S_2) \cup \Theta(S_{2,1} \bowtie S_2, S_1)$$

where:

$$\Theta(R,S) = \{\ t \in R \mid \nexists t_1 \in S \ s.t.\ key(t) = key(t_1)\ \} \cup$$
$$\left\{ t \mid \exists t_1 \in R, \exists t_2 \in S \ s.t. \forall a \in attr(R) \ \left(t[a] = \begin{cases} t_2[a] & if\ (a \in attr(S) \wedge t_1[a] = \bot) \\ t_1[a] & otherwise \end{cases} \right) \right\}$$

Observe that given two relations *R* and *S* such that *attr(R)*   *attr(S)*, the binary operator   replaces null values occurring in *R* with values taken from *S*. Moreover, in the above definition, each tuple *t* in (R,S) is derived from some tuple t_1 of *R* (resp. t_2 of *S*) by replacing null values of t_1 (resp. t_2) with the values of the corresponding attributes of t_2 (resp. t_1). Thus, the merged relation S_1 ⊠ S_2 is defined so that if S_2 does not contain any tuple t_2 such that key(t_1)=key(t_2) the resulting tuple will have null values for the attributes not present in S_1; otherwise, for each tuple t_2 in S_2 such that key(t_1)=key(t_2), the operator produces a tuple completing t_1. In a similar way the operator extends the content of S_2.

Proposition 1. Let S_1 and S_2 be two overlapping relations, then:

attr(S_1 ⊠ S_2)= attr(S_1)   attr(S_2)

key(S_1 ⊠ S_2)= key(S_1) = key(S_2),

$S_1 \sqsubseteq! S_1 \boxtimes S_2$ and $S_2 \sqsubseteq! S_1 \boxtimes S_2$

Example 1. Consider the relations S_1 and S_2 reported in Fig. 2 and storing information about some employees. In each of them *Name* is the key attribute and the functional dependency *Office*   *City* holds. The integrated relation T is obtained by merging S_1 and S_2, i.e. $T = S_1 \boxtimes S_2$.

S_1

Name	Office	City
Greg	Research	NY
Red	Sales	WA
Smith	Admin	NY

S_2

Name	Office	City	BirthYear
Smith	Sales	WA	1965
Taylor	Admin	SF	1971
Lan	Sales	NY	1980

T

Name	Office	City	BirthYear
Greg	Research	NY	
Red	Sales	WA	
Smith	Admin	NY	1965
Smith	Sales	WA	1965
Taylor	Admin	SF	1971
Lan	Sales	NY	1980

Fig.2

Note that in the integrated relation T the cluster associated to *Smith* contains two tuples so the key dependency is violated; on the contrary the functional dependency *Office*   *City* holds.

Let S_1 and S_2 be two overlapping relations, let $K = key(S_1) = key(S_2)$, $A=\{a_1,...,a_n\} = attr(S_1) \quad attr(S_2)-K$, $B=\{b_1,...,b_m\}= attr(S_1)-attr(S_2)$ and $C=\{c_1,...,c_q\} =attr(S_2)-attr(S_1)$. The merge operator introduced in Definition 4 can be easily expressed by means of the following SQL statement (where, given a relation R and a set of attributes $X=\{X_1,...,X_t\}$, the notation R.X stands for $R.X_1,...,R.X_t$:

```
SELECT S1.K, S1.B, COALESCE(S1.a1 , S2.a1),...,COALESCE(S1.an , S2.an), S2.C
FROM S1 LEFT OUTER JOIN S2 ON S1.K = S2.K
UNION
SELECT S2.K, S1.B, COALESCE(S2.a1 , S1.a1),..., COALESCE(S2.a n , S1.an), S2.C
FROM S1  RIGHT OUTER JOIN S2 ON S1.K = S2.K
```

where the standard operator COALESCE(a_1,...,an) returns the first not null value in the sequence.

Proposition 2.

$S_1 \boxtimes S_2 = S_2 \boxtimes S_1$ (commutative property)

$(S_1 \boxtimes S_2) \boxtimes S_3 = S_1 \boxtimes (S_2 \boxtimes S_3)$ (associative property)

$S_1 \boxtimes S_1 \sqsubseteq S_1$

Note that if the relation S does not contain null value or it is consistent the idempotent property holds ($S_1 \boxtimes S_1 = S_1$).
Obviously, given a set of overlapping relations S_1 , S_2 ,..., S_n, the associated super-relation S can be obtained as $S = S_1 \boxtimes S_2 \boxtimes ... \boxtimes S_n$, in other words S is the integrated view of S_1 , S_2,..., S_n.

The problem we have considered is similar to the one treated in [YO99], which assumes the source relations, involved in the integration process, have previously been homogenized. In particular, any homogenized source relations is a fragment of the global relation, that is it contains a subset of the attributes of the global relation and has its same key K. The technique proposed in [YO99] makes use of an operator $\bowtie$, called *Match Join*, to manufacture tuples in global relations using fragments. This operator consists of the outer-join of the *ValSet* of each attribute, where the *ValSet* of an attribute *A* is the union of the projections of each fragment on {K,A}. Therefore, the application of the

Match Join operator produces tuples containing associations of values that may not be present in any fragment.

Example 2. We report in Figure 3 the relation obtained by applying the Match Join operator to the relations S_1 and S_2 of Example 1.

Name	Office	City	BirthYear
Greg	Research	NY	
Red	Sales	WA	
Smith	Admin	NY	1965
Smith	Admin	WA	1965
Smith	Sales	NY	1965
Smith	Sales	WA	1965
Taylor	Admin	SF	1971
Lan	Sales	NY	1980

Fig. 3 The merged relation T= $S_1 \boxtimes$ S2

The Match Join operator applied to the source relations of Example 1 produces tuples violating the functional dependency *Office* *City* since it mixes values coming from different tuples with the same key in all possible ways. On the contrary our merge operator only tries to derive unknown values so that number of integrated tuples violating the constraints is reduced (see Example 1).

4. Data Integration in the Materialized Approach

The materialization of integrated views, commonly adopted in data warehousing systems, produces a significant reduction in query response time with respect to the virtual approach. In particular, the advantage of the materialized approach is significant when data are provided by multiple databases, entailing expensive joins to build the integrated view [Hu97]. In the presence of materialized views it can be advantageous to store some information about the way data have been obtained, in order to answer queries containing conditions on data origin. Given a set of overlapping relations $R_1,...,R_n$ the correspondent super relation *R*, obtained by applying the merge operator, does not contain any information on the origin of the data in *R*. In order to maintain information about the process by which the derived data have been obtained, we associate to each tuple a new attribute (*integrating path attribute*) representing the sequence of the sources that have been employed for deriving the tuple. Thus the schemas of the overlapping basic relations have to be extended with the integrating path attribute, denoted by *Path*.

Definition 5. Let *R* be a basic relation. A relation *S* is the *extended basic relation* of *R* if the following conditions hold:

attr(S) = attr(R) Path ,

Path(S)=R,

$R \sqsubseteq S$,

key(R) = key(R_i) i=1..n.

Example 3. The extended basic relation associated with the basic relations S_1 and S_2 in Example 1 are represented in Figure 4 .

Name	**Office**	**City**	**Path**
Greg	Research	NY	S_1
Red	Sales	WA	S_1
Smith	Admin	NY	S_1

S1

Name	**Office**	**City**	**Birth**	**Path**
Smith	Sales	WA	1965	S_2
Taylor	Admin	SF	1971	S_2
Lan	Sales	NY	1980	S_2

S2

Note that from the integration of a set of overlapping extended basic relations we obtain an extended derived relation. In the rest of the paper, if there is no ambiguity, we indicate with extended relation both basic and derived extended relations, that is a generic relation enriched with the integrating path attribute.

4.1 The Extended Merge Operator

In this section we introduce a variant of the merge operator presented previously, which is particularly useful to mediators maintaining materialized views of the integrated information. The *Extended Merge operator*, denoted by $\boxminus$, receives in input two extended overlapping relations and builds the correspondent super-relation, where the value of the *Path* attribute of each output tuple is obtained by concatenating the *Path* values of the input tuples it derives from.

Definition 6. Let S_1 and S_2 be two extended overlapping relations, the relation

$$S_1 \boxminus S_2 = \Theta(S_{1,2} \bowtie S_1, S_2) \cup \Theta(S_{2,1} \bowtie S_2, S_1)$$

where:

$$\Theta(R, S) = \{\ t \in R \mid \nexists t_1 \in S \text{ s.t. } key(t) = key(t_1)\ \} \cup$$
$$\{\ t \mid \exists t_1 \in R, \exists t_2 \in S \text{ s.t. } t[Path] = t_1[Path] \cdot t_2[Path] \wedge$$
$$\forall a \in attr(R) - \{Path\}\ t[a] = \begin{cases} t_2[a]\ if(a \in attr(S) \wedge t_1[a] = \bot) \\ t_1[a]\ otherwise \end{cases} \}$$

and the symbol . denotes the concatenation operator.

Example4. Consider the extended basic relations S_1 and S_2 in Figure 4. The relation T=S_1 $\boxminus$ S2 is reported in Figure 5.

Name	**Office**	**City**	**BirthYear**	**Path**
Greg	Research	NY		S_1S_2
Red	Sales	WA		S_1S_2
Smith	Admin	NY	1965	S_1S_2
Smith	Sales	WA	1965	S_1S_2
Taylor	Admin	SF	1971	S_1S_2
Lan	Sales	NY	1980	S_1S_2

Fig.5 The extended merged relation T= $S_1 \boxminus S_2$

Obviously, given a set of extended overlapping relations S_1, S_2 ,..., S_n , the associated extended super-relation S can be obtained as S = $S_1 \boxminus S_2 \boxminus ... \boxminus S_n$. In other words S is the integrated view of S_1,, S_2 ,..., S_n. The attribute *Path* of each tuple *t* in *S* corresponds to a permutation of all the extended basic relations S_1 ,S_2 ,..., S_n.

4.2 Querying extended relations

The information provided by the *Path* attribute can be useful for evaluating queries on the materialized views residing at the mediator. In particular, it can be employed to reconstruct the contents of a source relation from the materialized view or to answer queries expressing the satisfaction of user preference criteria about the origin of data.

Origin of data. As previously stated, the *Path* value of each tuple in the extended super relation represents the sequence of the sources that have been employed for deriving the tuple. In particular, the first source in the path attribute is the one from which the integration process has begun, i.e. the source whose values have been preserved in the global relation. Thus, the original instance of each basic relation, R_i, can be reconstructed by selecting from the extended super relation the tuples having R_i as first source in the *Path* sequence, and then by projecting on attr(R_i).

Definition 7. Let R_1,..., R_n be a set of extended overlapping relations and *R* be the corresponding extended super relation, then a tuple *t* *R* is said to be R_i *-derived* if:

$$\exists t_1 \in R_i,\ t_1(K) = t(K)\ (\forall a \in attr(R_i) - Path)\ t_1[a] = t[a]\ \vee\ t_1[a] = \perp$$

Proposition 3. Let *R* be an extended relation and R_i be a sub-relation of R, then a tuple t in *R* is R_i *- derived* if Path(t)= R_i * _, where _ denotes any source sequence.

Proposition 4. Let *R* be an extended relation and R_i be a basic sub-relation of *R*, and $R' = \Pi_{attr(R_i)}(\sigma_{Path=R_i*\_}(R))$, then

- R_i R' if R_i is a full relation
- $R_i \sqsubseteq$ R' otherwise

Note that, in the materialized approach, the content of each full basic relation can be easily reconstructed from the extended super-relation corresponding to it.

Answering queries satisfying user Preference. The availability of the *Path* information makes the satisfaction of user preference criteria in queries against materialized views easy. Since the *Path* attribute stores information about the order in which source relations have been integrated, it establishes a priority order on the way relations have been used to build the output tuple.
A *preference constraint* is a rule of the form Si $\ll$ Sj, where Si, Sj are two source relations. Preference constraints imply a partial order on the source relations. We shall write $S_1 \ll S_2 \ll ... \ll S_k$ as shorthand for $S_1 \ll S_2$,$S_2 \ll S_3$,...,$S_{k-1} \ll S_k$. The presence of such constraints requires the satisfaction of preference criteria during the computation of the answer. A priority statement of the form $S_i \ll S_j$ specifies a preference on the tuples provided by the relation *Si* with respect to the ones provided by the relation *Sj.*
The satisfaction of a set of priority statements can be easily performed by selecting the tuples whose *Path* value corresponds to a permu tation of the sources coherent with the partial order imposed by the preference criteria.

5. Managing Inconsistent data

We assume that each mediator component involved in the integration process contains an explicit representation of intentional knowledge, expressed by means of integrity constraints. Integrity constraints express semantic information over data, i.e. relationships that must hold among data. Generally, a database *D* has associated a set of integrity constraints *IC*. A database instance *D* is said to be *consistent* if D satisfies IC, otherwise it is inconsistent. In this paper we concentrate on functional dependencies. Using the technique proposed in [GZ00] we compute consistent answers for possibly inconsistent databases. The technique is based on the generation of a disjunctive program *DP(IC)* derived from the set of integrity constraints *IC*. The computation of the consistent answers of a query *G* can be derived by considering the minimal models of the program *DP(IC)* over the database *D*. For a complete description see [GZ00].

Definition 8. Let *c* be a functional dependency x |= y over P, which can be expressed by a formula of the form

(x,y,z,u,v) [P(x,y,u) P(x,z,v) y = z]

then, dj(c) denotes the extended disjunctive rule

P(x,y,u) P(x,z,v) ¬ P(x,y,u) , P(x,z,v) , y z

Let *IC* be a set of functional dependencies, then *DP(IC) = { dj(c) | c in IC }*.
Thus, *DP(IC)* denotes the set of disjunctive rules derived from the rewriting of *IC*. *MM(DP(IC),D)* denotes the set of minimal models of *DP(IC)* *D.*

Definition 9. Given a database D, a set of integrity constraints IC and a query G, then the consistent answer for G over D consists of the three distinct sets denoting, respectively, true, undefined and false atoms:

- $Ans^+(G,D,\mathcal{IC}) = \{ q(t) \in D \mid \nexists M \in \mathrm{MM}(\mathcal{DP}(\mathcal{IC}),D)$ *s.t.* $\neg q(t) \in M \}$
- $Ans^u(G,D,\mathcal{FD}) = \{ q(t) \in D \mid \exists M_1, M_2 \in \mathrm{MM}(\mathcal{DP}(\mathcal{IC}),D)$ *s.t.* $\neg q(t) \in M_1$ *and* $\neg q(t) \notin M_2 \}$
- $Ans^-(G,D,\mathcal{FD})$ denotes the set of atoms which are neither true nor undefined (false atoms). □

Theorem 1. Let *D* be an integrated database, *FD* a set of functional dependencies and *G* a query, then, the computation of a consistent answer of *G* over *D* can be done in polynomial time.

Example 5. Consider the integrated relation T of Example 1 and the functional dependency *Name (Office,City,BithYear)* stating that *Name* is a key for the relation. The functional dependency can be rewritten as first order formula:

$$(\forall x,y,z,w,y',z',w')[T(x,y,z,w) \wedge T(x,y',z',w') \supseteq y = y' \wedge z = z' \wedge w = w']$$

The associated disjunctive program is

$$\neg T(x,y,z,w) \wedge \neg T(x,y',z',w') \leftarrow T(x,y,z,w) \wedge T(x,y',z',w') \wedge (y \neq y' \vee z \neq z' \vee w \neq w')$$

The above program has two stable models M1 = D { *T(Smith,Admin,NY,1965)* } and M2 = D { *T(Smith,Sales,WA, 1965)* }. Thus, the query answering for the office of employee *Red* is *Sales*, whereas the answer to the query asking for the office of employee *Smith* is unknown since there are two alternative values.

6. Conclusions

In this paper, we focused our attention on the integration of conflicting instances [Ar95,ABC99,Du96] related to the same concept and possibly coming from different sources. We have presented an operator, called *Merge operator*, which allows us to combine data coming from different sources, preserving the information contained in each of them and a variant of it, i.e. the *Extended Merge Operator,* which keep track of the process by which the derived data have been obtained.
The problem we have considered is similar to the one treated in [YO99], which defines the *Match Join* operator to manifacture tuples in global relations using fragments. The Match Join operator produces tuples containing associations of values that may be not present in any fragment, while the merge operators we introduced only tries to derive unknown values.

Bibliography

[Ar95] Argaval, S.; Keller, A.M.; Wiederhold, G.; Saraswat, K.: Flexible Relation: an Approach or Integrating Data from Multiple, Possibly Inconsistent Databases. In *Proc. 11th Int. Conf. on Data Engineering*, 1995; pp. 495--504.

[ABC99] Arenas, M.; Bertossi, L.; Chomicki, J.: Consistent Query Answers in Inconsistent Databases. In *Proc. PODS Conference*, 1999; pp. 68--79.

[Br90] Breitbart, Y: Multidatabase interoperability. In *Sigmod Record* 19(3), 1990; pp. 53-60.

[CD97] Chaudhuri, S.; Dayal, U.: An Overview of Data Warehousing and OLAP Technology. In ACM SIGMOD Record, 26(1), 1997; pp. 65--74.

[Du96] Dung, P. M.: Integrating Data from Possibly Inconsistent Databases. In *Proc. 1st Int. Conf. on Cooperative Information Systems,* 1996, pp. 58--65.

[EGM97] Eiter, T.; Gottlob, G.; Mannila, H.:Disjunctive Datalog. In *ACM Transactions on Database Systems*, 22(3), 1997; pp. 364--418.

[Ga97] Garcia-Molina, H.; Papakonstantinou, Y.; Quass, D.; Rajaraman, A.; Sagiv, Y.; Ullman, J.; Vassalos, V.; Widom, J.: The TSIMMIS approach to mediation: Data models and languages. In *J. Intelligent Information Systems*, 8(2),1997; pp. 117--132.

[GS95] Grant, J.; Subrahmanian, V.S.: Reasoning in Inconsistent Knowledge Bases. In IEEE-Trans. on Knowledge and Data Eng., 7(1),1195;pp. 177--189.

[GZ00] Greco, S.; Zumpano, E.:Querying Inconsistent Database.In Proc. 7th Int. Conf. Logic for Programming and Automated Reasoning,2000; pp. 308--325.

[Hu97] Hull, R.:Managing Semantic Heterogeneity in Databases: a Theoretical Perspective.In *Proc. Symposium on Principles of Database Systems*, 1997; pp. 51--61

[In97] Inmon, W.H.:What is a Data Warehouse?. Prism Tech. Topic, 1(4),1997.

[LRO96] Levy, A.; Rajaraman, A.; Ordille, J.:Querying heterogeneous information sources using source descriptions. In *Proc. VLDB.* Conf.,1996; pp. 251--262.

[LM99] Lin, J.; Mendelzon, A.O.:Knowledge Base Merging by Majority.In Pareschi, R.; Fronhoefer, B. (Eds.): *Dynamic Worlds*.Kluwer, 1999.

[YO99] Yan, L.L.; Ozsu, M.T.:Conflict Tolerant Queries in Aurora. In *Proc. 4th Int. Conf. on Cooperative Information System*s,1999; pp. 279--290.

[Wi92] Wiederhold, G.:Mediators in the architecture of future information systems.*IEEE Com*puter, 25(3),1992; p. 38--49.