Data Fusion

JENS BLEIHOLDER and FELIX NAUMANN

Hasso-Plattner-Institut

The development of the Internet in recent years has made it possible and useful to access many different information systems anywhere in the world to obtain information. While there is much research on the integration of heterogeneous information systems, most commercial systems stop short of the actual integration of available data. Data fusion is the process of fusing multiple records representing the same real-world object into a single, consistent, and clean representation.

This article places data fusion into the greater context of data integration, precisely defines the goals of data fusion, namely, complete, concise, and consistent data, and highlights the challenges of data fusion, namely, uncertain and conflicting data values. We give an overview and classification of different ways of fusing data and present several techniques based on standard and advanced operators of the relational algebra and SQL. Finally, the article features a comprehensive survey of data integration systems from academia and industry, showing if and how data fusion is performed in each.

Categories and Subject Descriptors: H.2.5 [**Database Management**]: Heterogeneous Databases

General Terms: Algorithms, Languages

Additional Key Words and Phrases: Data cleansing, data conflicts, data consolidation, data integration, data merging, data quality

ACM Reference Format:

Bleiholder, J. and Naumann, F. 2008. Data fusion. ACM Comput. Surv., 41, 1, Article 1 (December 2008), 41 pages DOI = 10.1145/1456650.1456651 http://doi.acm.org/10.1145/1456650.1456651

1. INTRODUCTION

With more and more information sources available via inexpensive network connections, either over the Internet or in company intranets, the desire to access all these sources through a consistent interface has been the driving force behind much research in the field of information integration. During the last three decades many systems that try to accomplish this goal have been developed, with varying degrees of success. One of the advantages of information integration systems is that the user of such a system obtains a complete yet concise overview of all existing data without needing to access all data sources separately: complete because no object is forgotten in the result; concise because no object is represented twice and the data presented to the user is without

This research was supported by the German Research Society (DFG grant no. NA 432).

Authors' addresses: J. Bleiholder and F. Naumann, Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany; email: {Jens.Bleiholder,Felix.Naumann}@hpi.uni-potsdam.de.

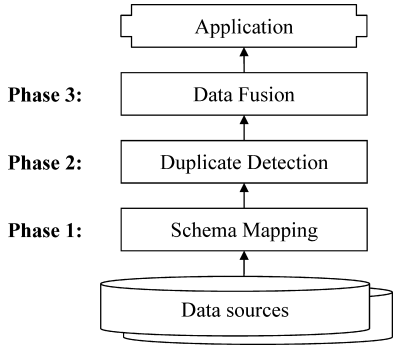

Fig. 1. A data integration process.

contradiction. The latter is difficult because information about entities is stored in more than one source.

After major technical problems of connecting different data sources on different machines are solved, the biggest challenge remains: overcoming semantic heterogeneity, that is, overcoming the effect of the same information being stored in different ways. The main problems are the detection of equivalent schema elements in different sources (schema matching) and the detection of equivalent object descriptions (duplicate detection) in different sources to integrate data into one single and consistent representation. However, the problem of actually integrating, or fusing, the data and coping with the existing data inconsistencies is often ignored.

The problem of contradictory attribute values when integrating data from different sources is first mentioned by Dayal [1983]. Since then, the problem has often been ignored, yet a few approaches and techniques have emerged. Many of them try to avoid the data conflicts by resolving only the uncertainty of missing values, but quite a few of them use different kinds of resolution techniques to resolve conflicts.

With this survey we introduce the inclined reader to the process of *data fusion* in the wider context of information integration. This process is also referred to in the literature as *data merging*, *data consolidation*, *entity resolution*, or *finding representations / survivors*. We also present and compare existing approaches to implement such a data fusion step as part of an information integration process and enable users to choose the most suitable among them for the current integration task at hand.

The survey begins in Section 2 with a brief introduction into information integration and the different tasks that need to be carried out, before describing the process of data fusion in more detail. Section 3 presents, describes, and classifies relational techniques for data fusion, before Section 4 gives a detailed overview and classifies integrated information systems and their data fusion capabilities.

2. DATA FUSION

Integrated (relational) information systems provide users with a unified view of multiple heterogeneous data sources. Querying the underlying data sources, combining the results, and presenting them to the user is performed by the integration system.

In this article we assume an integration scenario with a three-step data integration process, as shown in Figure 1 [Naumann et al. 2006]. First, we need to identify corresponding attributes that are used to describe the information items in the source. The result of this step is a *schema mapping*, which is used to transform the data present in the sources into a common representation (renaming, restructuring). Second, the

different objects that are described in the data sources need to be identified and aligned. In this way, using *duplicate detection* techniques, multiple, possibly inconsistent representations of the same real-world objects are found. Third, in a last step, the duplicate representations are combined and fused into a single representation while inconsistencies in the data are resolved. This last step is referred to as *data fusion*[1] and is the main focus in this article. We briefly discuss the two other steps as prerequisites to data fusion.

In the remainder of this section we present common solutions to the first two steps and then go into more detail in the field of data fusion. Apart from completeness and conciseness in data integration, we regard the different kinds of conflicts that may occur during integration and different possible semantics that can be used in implementing the final step of data fusion. We give a motivating example at the end of this section, which is used throughout the article to show the differences between the techniques and systems.

The following section is used to present relational techniques for data fusion (Section 3) and examines to what degree they accomplish it. The next section (Section 4) surveys various data integration systems and specifically identifies their data fusion capabilities.

2.1. Data Transformation

Integrated information systems must usually deal with heterogeneous schemata. In order to present to the user query results in a single unified schema, the schematic heterogeneities must be bridged. Data from the data sources must be transformed to conform to the global schema of the integrated information system. Two approaches are common to bridge heterogeneity and thus specify data transformation: schema integration and schema mapping. The former approach is driven by the desire to integrate a known set of data sources. Schema integration regards the individual schemata and tries to generate a new schema that is complete and correct with respect to the source schemata, that is minimal, and that is understandable. Batini et al. [1986] give an overview of the difficult problems involved and the techniques to solve them.

The latter approach, schema mapping, assumes a given target schema; that is, it is driven by the need to include a set of sources in a given integrated information system. A set of correspondences between elements of a source schema and elements of the global schema are generated to specify how data is to be transformed [Popa et al. 2002; Melnik et al. 2005]. A particularly interesting addition to schema mapping are *schema matching* techniques, which semi-automatically find correspondences between two schemata. Rahm and Bernstein have classified these techniques based on what input information the methods use [Rahm and Bernstein 2001]. There is much ongoing research in both the areas of schema matching and schema mapping—comprehensive surveys are yet missing.

The goal of both approaches, schema integration and schema mapping, is the same: transform data of the sources so that it conforms to a common global schema. Given a schema mapping, either to an integrated or to a new schema, finding such a complete and correct transformation is a considerable problem [Fagin et al. 2005]. The data

[1]There are two other fields in computer science that also use the term *(data) fusion*. In information retrieval it means the combination of search results of different search engines into one single ranking, therefore it is also called *rank merging*. In networking it means the combination of data from a network of sensors to infer high-level knowledge, therefore also called *sensor fusion*. Beyond computer science, in market research, the term data fusion is used when referring to the process of combining two datasets on different, similar, but not identical objects that overlap in their descriptions.

transformation itself, once found, can be performed offline, for instance, as an ETL process for data warehouses; or online, for instance, in virtually integrated federated databases. After this step in the data integration process all objects of a certain type are represented homogeneously.

2.2. Duplicate Detection

The next step of the data integration process is that of duplicate detection (also known as record linkage, object identification, reference reconciliation, and many others). The goal of this step is to identify multiple representations of the same real-world object: the basic input to data fusion.

In principle, duplicate detection is simple: Compare each pair of objects using a similarity measure and apply a threshold. If a pair is more similar than the given threshold it is declared a duplicate. In reality there are two main difficulties to be solved: *effectiveness* and *efficiency*.

Effectiveness is mostly affected by the quality of the similarity measure and the choice of a similarity threshold. A similarity measure is a function determining the similarity of two objects. Usually the similarity measure is domain-specific, for instance, designed to find duplicate customer entries. Domain-independent similarity measures usually rely on string-distance measures, such as the Levenshtein-distance [Levenshtein 1965]. The similarity threshold determines when two objects are duplicates. A too-low threshold will produce a high recall (all duplicates are found) but a low precision (many nonduplicate pairs are declared duplicates). A too-high threshold results in high precision, but low recall. Tuning the threshold is difficult and very domain- and even dataset-specific.

Efficiency is an issue because datasets are often very large, so even calculating and storing all pairs of objects can become an obstacle. Another obstacle of efficient duplicate detection is the complexity of the similarity measure itself, for instance, because the expensive Levenshtein-distance (or edit-distance) is part of the similarity function. The first obstacle is overcome by an intelligent partitioning of the objects and comparison of pairs of objects only within a partition. A prominent example of this technique is the sorted neighborhood method [Hernández and Stolfo 1998]. The second obstacle can be alleviated somewhat by efficiently computing upper bounds of the similarity and computing the actual distance only for pairs whose bounds are higher than the upper bound [Weis and Naumann 2004].

The result of the duplicate detection step is the assignment of an object-ID to each representation. Two representations with the same object-ID indicate duplicates. Note that more than two representations can share the same object-ID, thus forming *duplicate clusters*. It is the goal of data fusion to fuse these multiple representations into a single one.

2.3. Complete and Concise Data Integration

Data integration has two broad goals: increasing the completeness and increasing the conciseness of data that is available to users and applications. An increase in completeness is achieved by adding more data sources (more objects, more attributes describing objects) to the system. An increase in conciseness is achieved by removing redundant data, by fusing duplicate entries and merging common attributes into one. This distinction (completeness versus conciseness) is along the lines of related work such as Motro [1986], Naumann et al. [2004], and Scannapieco et al. [2004].

In analogy to the well-known precision/recall measure from information retrieval, we can similarly define a measure of conciseness/completeness when regarding unique

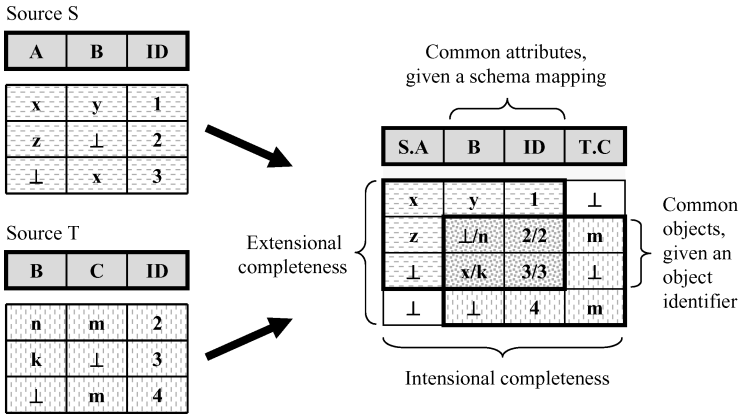

Fig. 2. Visualizing extensional and intensional completeness when integrating two data sources (source S and T with their respective schemas (A, B, ID) and (B, C, ID)) into one integrated result, using information from schema matching (common attributes, here identified by same name) and duplicate detection (common objects, here identified by same values in the ID column).

and additional object representations in the considered universe and the concrete data set that is being measured (see Figure 3).

Completeness. In analogy to recall, completeness of a dataset, such as a query result or a source table, measures the amount of data in that set, both in terms of the number of tuples (extensional, data level) and the number of attributes (intensional, schema level).

Extensional completeness is the number of unique object representations in a dataset in relation to the overall number of unique objects in the real world, such as, in all the sources of an integrated system. It measures the percentage of real-world objects covered by that dataset. We assume we are able to identify same real-world objects, for example, by an identifier created during duplicate detection.

$$\text{extensional completeness} = \frac{\| \text{ unique objects in data set } \|}{\| \text{ all unique objects in universe } \|} = \frac{a}{a+c} \tag{1}$$

The example in Figure 2 shows the combination of two sources. Given the real-world object identifier in the ID column, there are in total four objects in the real world (four distinct values in column ID). The combination of the two sources as given in Figure 2 has an extensional completeness of 1 (= 4/4) and is therefore maximal in the example, whereas both sources S and T are extensionally incomplete (extensional completeness: 3/4). In general, an increase in extensional completeness is achieved by adding more unique objects.

Intensional completeness is the number of unique attributes in a dataset in relation to the overall number of unique attributes available. An increase is achieved by integrating sources that supply additional attributes to the relation; that is, additional attributes that could not be included in one of the schema mappings between the sources considered so far. The result in Figure 2 has an intensional completeness of 1 (= 4/4) because it contains four attributes (S.A, B, ID, T.C) out of all four unique attributes (S.A, S.B, S.ID, T.B, T.ID, T.C, with mappings S.B $\leftrightarrow$ T.B and S.ID $\leftrightarrow$ T.ID). Likewise, sources S and T have an intensional completeness of 3/4.

Conciseness. In analogy to precision, conciseness measures the uniqueness of object representations in a dataset. In the terms of Figure 3, *extensional conciseness* is the

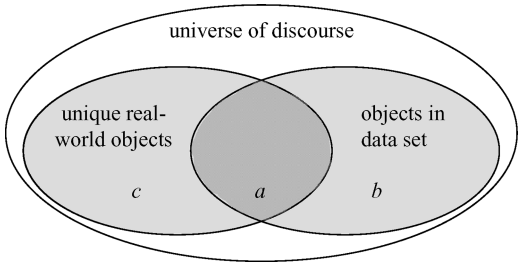

Fig. 3. Measuring completeness and conciseness in analogy to precision and recall, letters *a*–*c* denoting the number of objects in that region, for example, *b* being the total number of objects in the considered data set, *c* being the number of all unique real-world objects, and *a* being the number of unique real world objects that are in the considered data set.

number of unique objects in a dataset in relation to the overall number of object representations in the dataset.

$$\text{extensional conciseness} = \frac{\| \text{ unique objects in data set } \|}{\| \text{ all objects in data set } \|} = \frac{a}{a+b} \tag{2}$$

The example result in Figure 2 results in an extensional conciseness of $1(= 4/4)$, containing four different objects (as given by object-identifier-ID) out of four different object representations (tuples) in the result.

Similarly, *intensional conciseness* measures the number of unique attributes of a dataset in relation to the overall number of attributes. Again, intensional conciseness follows as $1\ (= 4/4)$, as all four attributes in the result are different.

For these measures to be useful, the definition of objects in the universe and the considered dataset needs to be the same. Schema-altering operations, such as joins, may cause problems here. In the description of relational operators and techniques in Section 3 we will focus on extensional completeness and conciseness.

Four degrees of integration. To further illustrate the notions of completeness and conciseness when integrating data sources, Figure 4 depicts four different ways of combining the example tables (sources S and T) from Figure 2 into one result. The generalization to more than two sources is trivial but difficult to visualize.

Without schema mapping information and the knowledge of object identifiers, the best that an integrating system can do is produce a result as seen in Figure 4(a). While this result has high completeness, it is not concise. Knowledge about common attributes, given by the schema mapping, allows results of the shape as seen in Figure 4(b). Incidentally, this shape is the result of an outer union operation on the two source relations (mapping attributes with same names). We call such results intensionally concise: No real-world property is represented by more than one attribute, the full schema mapping is used.

Knowledge about common objects (e.g., using a globally consistent ID) such as the ISBN for books, or using a globally consistent object identifier created by duplicate detection methods, allows results as seen in Figure 4(c). Here, the only known common attribute is ID (mapping inferred from ID being object identifier in both sources S and T). We will see further on how such a result can be formed using the full outer join operation on the IDs of the two source relations. We call such results extensionally concise: No real-world object is represented by more than one tuple.

Finally, Figure 4(d) shows the result after identifying common attributes (using a schema mapping) *and* common objects (using an object identifier). The main feature of this result is that it contains only one tuple per represented real-world object *and*

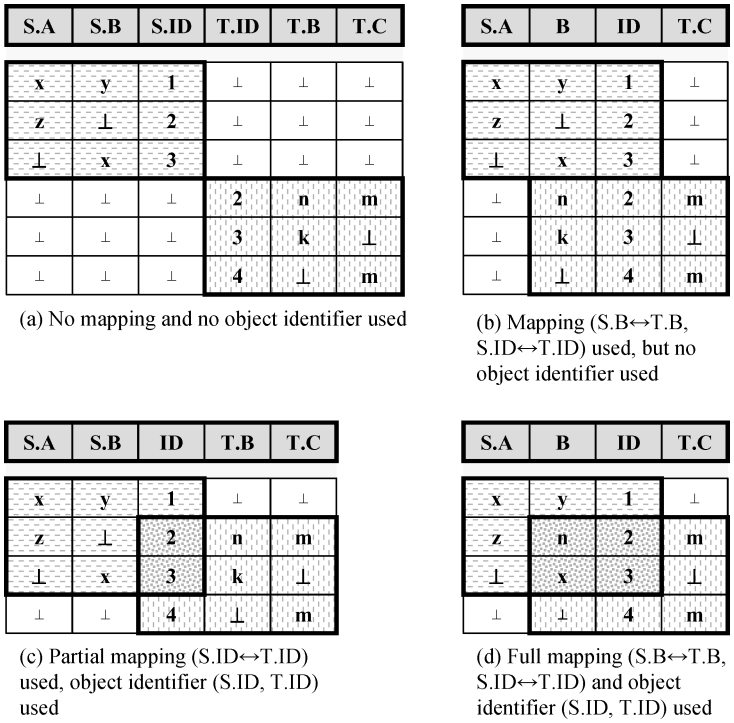

Fig. 4. Four degrees of integration, using to various degrees the information on schema mappings and object identifiers provided by schema matching and duplicate detection.

each row has only one value per represented attribute. Such a result (intensionally and extensionally concise) is the ultimate goal of data fusion. As could be noted by the watchful reader, the result in Figure 4(d) contains only the value x as the value for attribute B and object with ID = 3, although k could also be a valid value (contained in source T). Dealing with these data conflicts in the overlapping region of the two sources (marked by the checked pattern in the results) is an integral part of data fusion and will be considered in the next sections.

As we also see further on, most integration techniques and integrated information systems increase extensional as well as intensional completeness. We can regard this behavior as the primary feature of most integrating information systems. However, only a few also consider the problem of conciseness.

2.4. Conflict Classification

Different ways of representing same real-world objects in the sources results in three types of conflict. First, there are *schematic conflicts*, such as, different attribute names or differently structured data sources. Second, there are *identity conflicts*, as the way of identifying a real-world object may be different in the data sources. These two kinds of conflict are resolved during the first two phases of data integration. As a third kind of conflict, *data conflicts* remain, which are not resolved until data fusion and are caused by the remaining multiple representations of same real-world objects. A data conflict is present if, for the same real-world object (e.g., a student), semantically equivalent attributes, from one or more sources, do not agree on its attribute value (e.g., source 1 reporting “23” as the student’s age, source 2 reporting “25”). An overview on reasons

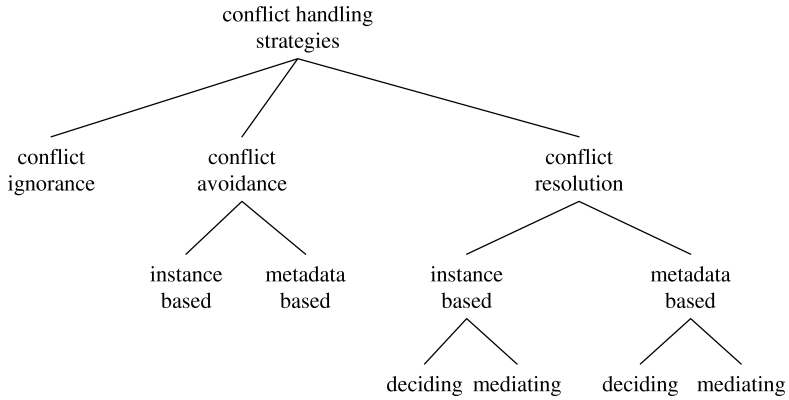

Fig. 5. A classification of strategies to handle inconsistent data [Bleiholder and Naumann 2006].

for dirty data and some more examples can be found in Rahm and Do [2000] and Kim et al. [2003].

We distinguish two kinds of data conflict: (a) uncertainty about the attribute value, caused by missing information; and (b) contradictions, caused by different attribute values.

Uncertainties. An uncertainty is a conflict between a non-null value and one or more null values that are all used to describe the same property of an object. In our scenario, this is caused by missing information, for example, null values in the sources, or an attribute completely missing in one source. The result in Figure 2 shows uncertainties in attributes $S.A$, $T.C$ (missing attribute) and B (missing information in source S) for the object with ID $= 2$.

The reason for considering uncertainties as a special case of conflict is that they are generally easier to cope with than contradictions. We deliberately choose to assume most null values in a data integration scenario being unknown values. But even with considering null values as being inapplicable or withheld, as are the three most common semantics of null values (refer to Ullman et al. [2001]), the assessment of the different techniques and systems remains valid.

Contradictions. A contradiction is a conflict between two or more different non-null values that are all used to describe the same property of the same object. In our data integration scenario, this is the case if two or more data sources provide two or more different values for the same attribute on the same object, sameness as given by the schema matching and duplicate detection steps before. The example in Figure 2 shows a contradiction in attribute B of object with ID $= 3$, between x and k.

2.5. Data Fusion Strategies and Answers

In fusing data from different sources into one consistent representation, we distinguish between and apply different high-level strategies. In Bleiholder and Naumann [2006] we describe and classify such strategies; Figure 5 shows the classification, and Table I briefly summarizes the strategies. The different classes of this classification can be seen as defining different data fusion semantics. We use these strategies further on in Sections 3 and 4 to characterize the different techniques and systems.

Data fusion strategies. 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. Applying this strategy to our example (see

Table I. Possible Conflict Handling Strategies from Bleiholder and Naumann [2006]

Strategy	Classification	Short Description
PASS IT ON	ignoring	escalates conflicts to user or application
CONSIDER ALL POSSIBILITIES	ignoring	creates all possible value combinations
TAKE THE INFORMATION	avoiding, instance based	prefers values over null values
NO GOSSIPING	avoiding, instance based	returns only consistent tuples
TRUST YOUR FRIENDS	avoiding, metadata based	takes the value of a preferred source
CRY WITH THE WOLVES	resolution, instance based, deciding	takes the most often occurring value
ROLL THE DICE	resolution, instance based, deciding	takes a random value
MEET IN THE MIDDLE	resolution, instance based, mediating	takes an average value
KEEP UP TO DATE	resolution, metadata based, deciding	takes the most recent value

Figure 2) would result in what we can see there: presenting both values in the overlap are, for example, x and k for attribute B of object with ID $= 3$.

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. In the example (Figure 2) only data from one source would be included, for example, only $\perp$ and x for the two objects in the overlap when preferring values from S over T.

In contrast to the previous classes, conflict *resolution* strategies do regard 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). Assuming that timestamp information is given for all values, applying strategy Keep Up To Date, which takes the most recent value, could result in taking the values n and x for objects with ID $= 2$ and ID $= 3$ in the example from Figure 2.

Different systems and techniques may use the same high-level strategy, but differ in the realization, as some strategies can be realized in more than one way. Bleiholder and Naumann [2006] include more details on fusion strategies and their realization in an integrated information system.

Data Fusion Answers. The result of a query to an integrated information system is called an *answer*. Such an answer could be characterized, among others, as being one of the following.

—*Complete*. A complete answer contains all the objects (extensionally complete) and also all attributes (intensionally complete) that have been present in the sources. A complete answer is not necessarily concise, as it may contain objects or attributes more than just once.

—*Concise*. An answer is concise if all real-world objects (extensionally concise) and all semantically equivalent attributes (intensionally concise) present are described only once.

—*Consistent*. A consistent answer contains all tuples from the sources that are consistent with respect to a specified set of integrity constraints (inclusion or functional dependencies) [Arenas et al. 1999; Fuxman et al. 2005a]. In this sense, such an answer is not necessarily complete, as all inconsistent object representations are left out of the result. However, given that one of the integrity constraints is a key constraint on some real-world identifier, a consistent answer is extensionally concise for all included object representations.

—*Complete and Consistent*. A complete and consistent answer combines the advantages of completeness and conciseness and consists of all real world object descriptions

Table II. Example Data Source, Representing Students from the First University (U_1) Showing Extensional Completeness of 6/7 and Extensional Conciseness of 6/7 With Regard to the Considered Universe, all Seven Students in the Two Example Tables

Name	Age	Status	Address	Field	Library
Peter	⊥	0	TUB	Computer Science	P201
Alice	22	1	Berlin	Mechanical Engineering	A709
Bob	⊥	1	Boston	⊥	B321
Charly	25	1	⊥	Psychology	⊥
Paul	⊥	1	TUB	Architecture	⊥
Paul	26	1	Berlin	Arch.	P233
Frank	23	0	TUB	Mech. Engineering	F205

Data from data source U_1.

Table III. Example Data Source, Representing Students from a Second University (U_2), Showing Extensional Completeness of 5/7 and Extensional Conciseness of 5/7 With Regard to the Considered Universe, All Seven Students in the Two Example Tables

Name	Age	Status	Address	Field	Phone
Alice	⊥	0	⊥	ME	030/12345
Bob	27	⊥	HUB	CS	030/54321
Charly	25	1	⊥	⊥	⊥
Alice	21	1	HUB	Mech. Eng.	030/98765
Eve	24	1	Berlin	CS/EE	030/55544
Eve	24	1	Berlin	CS/EE	030/55544
Frank	23	0	TUB	Mech. Engineering	⊥

Data from data source U_2.

from the sources, additionally fulfilling a key constraint on some real-world ID. Such a result contains all attributes from the sources and combines semantically equivalent ones into only one attribute. That way, intensional as well as extensional complete- and conciseness is given.

The data fusion step in data integration aims at producing a complete and consistent answer.

2.6. Motivating Example

As motivating example and to compare the different techniques and systems, we want to introduce a small example from the university domain. Tables II and III show two tables from universities (abbreviated U_1 and U_2) that represent data from two data sources. Both tables contain data about students that are currently enrolled in the university the data sources are provided by. As these universities are located in the same town, it is possible to study at more than one university. It is also possible to combine several fields of study and some are offered at more than one university, so there is a certain overlap in topics as well as in students.

As we assume the data to be in the relational model, we show the data as tables, column headers giving the names of the attributes of the two relations. Although different universities organize their data in a different way (heterogeneous sources), we assume in the examples that schemata already have been matched and semantically equivalent attributes have the same name in the examples. To more easily understand the example we further assume that all students are globally and uniquely identifiable by their first name. This way the role of the attribute NAME is that of a *real-world identifier*. Please note that this real-world identifier should not be confused with a database primary key. In the example we do not assume such a database primary key.

In general, data stored in data sources may be dirty. Errors and contradictions, missing values, and especially duplicate representations are commonly found when inserting data, creating entries, or during use, especially when updating data. Therefore, we assume the presence of (possibly contradicting) intrasource duplicates (e.g., Paul in U_1 or Eve in U_2) as well as (possibly contradicting) intersource duplicates (e.g., Bob). Null values in the examples are marked by the symbol $\perp$ and we consider them to be *unknown* values in contrast to be *not applicable* or *withheld*.

Where completeness and conciseness are measured, we only give numbers for extensional completeness and extensional conciseness and refer the reader to the corresponding formulas in Section 2.3. The total number of unique students in the example is 7.

3. RELATIONAL OPERATORS AND TECHNIQUES FOR DATA FUSION

This section introduces standard and advanced relational operators and examines their abilities in fusing data from different data sources. As we specifically regard relational techniques for the remainder of this section, data is integrated from source tables (possibly coming from different data sources) into one integrated table. We first consider standard operators, such as union and join. Join-based techniques generally combine tuples from several tables while evaluating some predicates on some of their columns. Union-based techniques generally build a common schema first and then append the different tuple sets from the source tables. We then regard more advanced relational techniques which combine standard operators, invent a new one, or use a different approach. The descriptions of operators and techniques assume binary operators, operating on two tables only. The extension to more than two tables is straightforward in all cases; however, associativity is not always given. The example of Section 2.6 is used to show and compare the results of the different approaches. Where tables show the measures for extensional completeness and extensional conciseness, the formulas from Section 2.3 are used.

3.1. Properties and Characteristics of Operators and Techniques

We describe the following operators and techniques using one or more of the following characteristics.

Value preservation. When combining tables in order to increase extensional completeness, not necessarily all values from the sources are included in the result (e.g., a regular join that only contains tuples from one table, if it has a join partner in the other table). However, for an operator to be fully *value preserving*, exactly all values for all attributes of all described objects need to remain in the result. We denote an operator that does not lose any such value or creates or duplicates values value preserving. Please note that the sole existence of a value is not sufficient and that we allow duplicate values. Thus, bag union would be an example of a value preserving operator, set union and Cartesian product examples for *nonvalue-preserving* operators. Value preservation becomes important when applying (duplicate-sensitive) functions afterwards. Value preservation should not be mistaken with the property of restorability, meaning that we can invert the operation and infer the values in the sources from the result (e.g., not possible for union). A simpler case is *object preservation*: Actual values of attributes describing objects may get lost, but at least all objects that have been described in the sources are also included in the result. Objects are thereby identified by a real-world identifier. Please note that we do not consider information preservation in terms of intensional completeness (at schema level) [Miller et al. 1993].

Uniqueness. An operator has the *uniqueness preservation* property if corresponding attributes (according to the schema mapping) that contain unique values in the sources also contain unique values in the result. This is, for example, true for an equality join on these attributes, but not for a union. *Uniqueness-preserving* operations are useful in scenarios where multiple representations of one real-world object only exist in different data sources (no intrasource duplicates) and the unique attribute is a real-world identifier. In contrast, an operator that creates a result that contains an attribute that is unique (regardless if there have been unique attributes in the sources) is said to have the *uniqueness-enforcing* property. In order to produce a complete and concise answer, as is the goal in data fusion, an operator needs to be uniqueness enforcing in an attribute holding a real-world identifier.

As the first property covers the requirement of completeness, the second is concerned with conciseness. The ideal data fusion operator, creating complete and consistent answers, should preserve as many values and objects as possible while at the same time enforcing or preserving uniqueness and resolving conflicts. Ideally, it is part of the relational algebra or at least should not be too complex when extending it.

3.2. Join Approaches

Join approaches in general increase intensional completeness, as attributes from different sources are seperately included in the result. However, this comes at the cost of not guaranteeing extensionally completeness, except for the case of a full outer join. Concerning extensional conciseness, join approaches perform very well, but only if we can rely on globally unique identifiers (e.g., created by duplicate detection), and no intrasource duplicates are present.

3.2.1. Standard Joins. An equi-join ($\bowtie_=$) combines tuples from two relations if the join condition consists of equality conditions between columns and if it evaluates to true [Ullman et al. 2001]. The result also includes all other attributes from the sources in the result. In an integration scenario, where the goal is to relate tuples which describe same objects, using an equi-join on some attributes building a real-world identifier (e.g., using `U1.Name=U2.Name`) seems the natural choice. We refer to this as key join ($\bowtie_{id=id}$). However, in general when combining two tables by a join, other conditions are possible as well, as are the use of attributes that do not form a real-world identifier.

The key-join is uniqueness preserving but not necessarily value/object preserving, as only objects that are present in both tables (according to the join condition) are included in the result. Using SQL, the two example tables are combined as follows.

```
SELECT U1.Name, U2.Name, U1.Age, U2.Age, U1.Status, U2.Status,
       U1.Address, U2.Address, U1.Field, U2.Field, U1.Library, U2.Phone
FROM U1 JOIN U2 ON U1.Name=U2.Name
```

The result of a key-join operation on the example tables using the real-world identifier NAME is shown in Table IV.

The natural join ($\bowtie$) combines two tuples from two relations if all common attributes (same name) coincide in their attribute values [Ullman et al. 2001]. Attributes that are not present in all relations are not considered in joining the relations, but are included in the result. Natural joins are also uniqueness preserving, but not necessarily value/object preserving. Data conflicts are generally ignored, as objects are either not included in the results or result in two different result tuples. Integrating the two example tables via a natural join is done using the following SQL statement.

Table IV. Result of Different Joins Applied to Tables U_1 and U_2, using Name as Join Attribute

U1.Name	U2.Name	U1.Age	U2.Age	...	U1.Library	U2.Phone
Alice	Alice	22	⊥	...	A709	030/12345
Alice	Alice	22	21	...	A709	030/98765
Charly	Charly	25	24	...	⊥	⊥
Bob	Bob	⊥	27	...	B321	030/54321
Frank	Frank	23	23	...	F205	⊥
Peter	⊥	⊥	⊥	...	P201	⊥
Paul	⊥	⊥	⊥	...	⊥	⊥
Paul	⊥	26	⊥	...	P233	⊥
⊥	Eve	⊥	24	...	⊥	030/55544
⊥	Eve	⊥	24	...	⊥	030/55544

Result of $\bowtie_{U1.Name=U2.Name}$ and $\mathbb{I}\bowtie\mathbb{I}_{U1.Name=U2.Name}$ on U_1 and U_2.
The key-join (extensional completeness: 4/7, extensional conciseness: 4/5) contains only the first group of tuples (Alice–Frank), whereas the result of the full outer join also covers the Peter, Paul, and Eve tuples (extensional completeness: 7/7, extensional conciseness: 7/10). The measured completeness for the key-join follows from four students (Alice, Bob, Charly, and Frank) being described in the result, out of the total seven students in the scenario. The measured conciseness follows from the same four unique students being described in a total of five tuples in the result. Likewise follow the measures for the full outer join.

```
SELECT U1.Name, U1.Age, U1.Status, U1.Address, U1.Field, U1.Library, U2.Phone
FROM U1 JOIN U2 ON U1.Name=U2.Name AND U1.Age=U2.Age
                   AND U1.Status=U2.Status AND U1.Address=U2.Address
                   AND U1.Field=U2.Field
```

The result of the natural join operation on the two example tables only contains the Frank tuple.

The full outer join ($\mathbb{I}\bowtie\mathbb{I}$) extends the result of a standard join operation by adding tuples that are only included in one of the two source relations. The missing attributes present in only one relation are padded by null values [Ullman et al. 2001]. There exists an outer join variant that is based on the key-join ($\mathbb{I}\bowtie\mathbb{I}_{id=id}$) instead of the natural join, as well as left ($\mathbb{I}\bowtie_{id=id}$) and right outer join ($\bowtie\mathbb{I}_{id=id}$) variants which only include tuples from the left, right (first, second) relation. The operator is value/object preserving only in its full variant based on the key-join. Like key- and natural join, it is uniqueness preserving in all variants. The result of the operation is shown in Table IV, the corresponding SQL statement is shown here.

```
SELECT U1.Name, U2.Name, U1.Age, U2.Age, U1.Status, U2.Status,
       U1.Address, U2.Address, U1.Field, U2.Field, U1.Library, U2.Phone
FROM U1 FULL OUTER JOIN U2 ON U1.Name=U2.Name
```

3.2.2. Full Disjunction. As the outer join operator in general is not associative, combining more than two tables by an outer join may result in different result tables, depending on the order in which the tables are joined. The full disjunction operator is defined in Galindo-Legaria [1994] as the combination of two or more tables, where all matching tuples are combined into one single tuple. It could therefore be seen as a generalization of a full outer join to more than two relations. Full disjunction can generally not be computed by a combination of outer joins alone [Galindo-Legaria 1994]. Rajaraman and Ullman [1996] cover full disjunctions in more detail and recent work on the topic provides a fast implementation [Cohen and Sagiv 2005]. Full disjunctions are uniqueness preserving and value/object preserving, like full outer joins. For the two example tables, the full disjunction on NAME as key results in the same table as the full outer join (see Table IV). Generally, full disjunctions are beneficial in situations with more than two tables where an outer join would have ben used with only two tables.

3.2.3. Match Join and Conflict Tolerant Queries. The match join has been defined together with a conflict-tolerant query model in the context of the AURORA project [Yan and Zsu 1999]. In a first step, corresponding attribute values from all sources are seperately projected out and combined by union, expanded by their corresponding real-world identifier. Then, the real-world identifiers are used to rejoin, resulting in one single large table. The operation to create this table is called match join.

```
WITH OU AS (
( SELECT Name, Age, Status, Address, Field, Library, NULL as Phone
  FROM U1 )
  UNION
( SELECT Name, Age, Status, Address, Field, NULL as Library, Phone
  FROM U2 )
),
AGE_V (Name, Age) AS ( SELECT DISTINCT Name, Age FROM OU ),
STATUS_V (Name, Status) AS ( SELECT DISTINCT Name, Status FROM OU ),
 ...
PHONE_V (Name, Phone) AS ( SELECT DISTINCT Name, Phone FROM OU ),
SELECT Name, Age, Status, Address, Field, Library, Phone
FROM AGE_V FULL OUTER JOIN STATUS_V ...  FULL OUTER JOIN PHONE_V
 ON AGE_V.Name=STATUS_V.Name AND STATUS_V.Name= ...  =PHONE_V.Name
AS MATCH_JOIN_TABLE
```

In a second step, to increase conciseness, tuples are selected from this table according to the accompanying conflict-tolerant query model. When selecting tuples, it uses a selection predicate and observes the evaluation parameter of the operation. This parameter is either *high confidence*, *random evidence*, or *possible at all* and determines which tuples are kept in the result. If it is set to high-confidence, the selection predicate needs to be true for *all* tuples for each real-world identifier. If it is set to possible-at-all, the selection predicate needs to be true for at least one tuple for each real-world identifier. Random-evidence verifies the predicate for a random tuple per real-world identifier.

The whole operation can be seen as first creating a set of all possible results (match join) and then selecting from it as specified by the user, creating a conflict-free result. This means that in the end there remains only one representation per real-world identifier, thus the operator is uniqueness enforcing. Conflicts are resolved by applying resolution functions, such as `sum`, `max`, `min`, `any`, etc. In contrast to other techniques considered so far, execution complexity becomes important, as the worst-case complexity is exponential in the unique values per attribute. In the worst case all possible combinations of attribute values are regarded.

The operator is uniqueness preserving and, depending on the parameterization, value/object preserving. Uncertainties could be removed, contradictions are resolved. The selection of tuples in the second step is realized by fine-grained operations, extending the relational model, which prevents a rewriting of the operator using standard SQL. An example statement in the proposed syntax is as follow.

```
SELECT [ANY] Name, Age, Status, Address, Field, Library, Phone
FROM matchjoin(U1,U2)
WHERE Age>22
WITH PossibleAtAll
```

where *matchjoin(U1,U2)* is a combination of the two data sources. Table V shows the result of the match join operator on the example data, selecting only tuples with AGE > 22 under the possible-at-all setting. Please note that, since in the intermediate step the operator looks at all possible combinations of attribute values, in some cases the result

Table V. Result of a Match Join/Conflict Tolerant Query on Both Example Tables, Selecting Only Tuples with Age $>$ 22 Under the *Possible at All* Setting (Extensional Completeness: 5/7, Extensional Conciseness: 5/5)

Name	Age	Status	Address	Field	Library	Phone
Bob	27	1	HUB	CS	B321	030/54321
Charly	25	1	$\perp$	Psychology	$\perp$	$\perp$
Eve	24	1	Berlin	CS/EE	$\perp$	030/55544
Frank	23	0	TUB	Mech. Engineering	F205	$\perp$
Paul	26	1	Berlin	Arch.	P233	$\perp$

Result of applying match join/conflict tolerant query on U_1 and U_2.

may contain tuples with attribute value combinations that as a whole have not been present in either of the sources. That way, the operator combines existing facts in a new way. If the any function is used or the query is evaluated according to random-evidence, the operation behaves nondeterministically. Otherwise, match join, in combination with the conflict-tolerant query model, is an expressive operation offering good conflict resolution capabilities.

3.2.4. Increasing Conciseness in Join Results. The schema of regular join results normally does not contain only one attribute for all pairs of semantically equivalent attributes. Therefore these operators are complete but not concise, and all existing, possibly contradictory, data values from the sources still exist in the result. One way of resolving these conflicts and increasing intensional conciseness is by combining these attributes into one with an additional operation.

In combination with join operators, the SQL function COALESCE proves useful: coalesce(v_1, v_2,...,v_n) gives back the first value v_i from v_1 to v_n that is not a null value. If all v_i are null, a null is returned. When the function is used in combination with joins, the inputs to COALESCE are usually values from semantically equivalent attributes from different sources, such as, coalesce($U_1.Age$, $U_2.Age$). That way, COALESCE removes uncertainties (if only one age is given and the other is a null value) or could be used (see Section 3.3.2) to prioritize input tables (here, if both sources give an age value, prefer $U_1.Age$). COALESCE therefore serves in implementing strategies such as Take the information or Trust Your Friends (see Section 2.5 for more information on strategies).

In Lim et al. [1995] a variant of the full disjunction operator is used which merges the key columns into one column, using COALESCE. This is possible because the key columns either contain the same or a null value, resulting from the assumption that no intrasource duplicates exist. They additionally define a *generalized attribute derivation* operation (GAD), which uses functions to combine values from several columns of the output of the full disjunction into a single value. Using the GAD operation it is possible to combine semantically equivalent attributes into a single attribute and resolve conflicts, for example, by taking the maximum age (max($U_1.Age$, $U_2.Age$)) or by concatenating fields (concat($U_1.Field$, $U_2.Field$)). GAD translates to the use of user-defined functions in SQL; however, there is no SQL translation of full disjunction.

```
SELECT Name, max(U1.Age, U2.Age), vote(U1.Status, U2.Status),
       U1.Address as Address, concat(U1.Field,U2.Field) Library, Phone
FROM fulldisjunction(U1,U2)
```

Thus, conciseness is increased and conflicts are resolved depending on the functions used. Please keep in mind that there is no SQL rewriting for this statement; however, it could look like the preceding statement. Still, intrasource duplicates or a poor join order (if using joins instead of full disjunction) leave duplicate tuples in the result.

Table VI. Result of Outer Union on the Example Tables U_1 and U_2 (Extensional Completeness: 7/7, Extensional Conciseness: 7/13) Does Not Contain the Last Tuple (Eve) as it is an Exact Duplicate. The Minimum Union on the Example Tables (Extensional Completeness: 7/7, Extensional Conciseness: 7/11) Does not Contain the Last Three Tuples (Charly, Frank, and Eve), Charly and Frank being Removed Because of Subsumption

Name	Age	Status	Address	Field	Library	Phone
Alice	⊥	0	⊥	ME	⊥	030/12345
Alice	22	1	Berlin	Mechanical Engineering	A709	⊥
Alice	21	1	HUB	Mech. Eng.	⊥	030/98765
Bob	⊥	1	Boston	⊥	B321	⊥
Bob	27	⊥	HUB	CS	⊥	030/54321
Charly	25	1	⊥	Psychology	⊥	⊥
Eve	24	1	Berlin	CS/EE	⊥	030/55544
Frank	23	0	TUB	Mech. Engineering	F205	⊥
Paul	⊥	1	TUB	Architecture	⊥	⊥
Paul	26	1	Berlin	Arch.	P233	⊥
Peter	⊥	0	TUB	Computer Science	P201	⊥
Charly	25	1	⊥	⊥	⊥	⊥
Frank	23	0	TUB	Mech. Engineering	⊥	⊥
Eve	24	1	Berlin	CS/EE	⊥	030/55544

Result after applying outer union ($\uplus$) and minimum union ($\oplus$) on U_1 and U_2.

3.3. Union Approaches

Union approaches are the natural way of accomplishing extensional completeness, as the result naturally includes all tuples from both relations. In contrast to join approaches, union approaches do not perform as well concerning conciseness.

3.3.1. Union, Outer Union, and Minimum Union. The union ($\cup$) operator (with set semantics) combines the tuples of two union-compatible relations and removes exact duplicates, that is, tuples that coincide in all attribute values [Ullman et al. 2001]. Two relations are union compatible if they are defined on the same number of attributes and if the data types of the attributes match. Union is not defined on nonunion-compatible relations. As the two example data sources are not unioncompatible (nonmatching data types), they cannot be integrated using union. Union is not uniqueness preserving as only exact duplicates are removed, resulting in contradicting tuples, with the same key value in the source relations appearing multiple times in the result. Uncertainties and contradictions are ignored; the operator is not value- but object preserving.

Outer union ($\uplus$) overcomes one restriction of union and combines two nonunion-compatible relations by first padding attributes that are not in common with null values and then performing a regular union [Galindo-Legaria 1994]. Like union, outer union is not value- but object preserving. It is not uniqueness preserving, and conflicts are ignored. Outer union is not a standard operator but can be rewritten in SQL.

```
( SELECT Name, Age, Status, Address, Field, Library, NULL as Phone
  FROM U1 )
  UNION
( SELECT Name, Age, Status, Address, Field, NULL as Library, Phone
  FROM U2 )
```

The result of the outer union on the example relations is shown in Table VI.

The minimum union operator ($\oplus$) is defined by Galindo-Legaria [1994] as the result of an outer union from which subsumed tuples have been removed. A tuple t_1 subsumes another tuple t_2: (1) if they comply with the same schema; (2) if t_2 contains more null

Table VII. Result of Merge on the Example Tables (Extensional Completeness: 7/7, Extensional Conciseness: 7/12)

Name	Age	Status	Address	Field	Library	Phone
Alice	22	1	Berlin	Mechanical Engineering	A709	030/12345
Alice	22	1	Berlin	Mechanical Engineering	A709	030/98765
Alice	22	0	Berlin	ME	A709	030/12345
Alice	21	1	HUB	Mech. Eng.	A709	030/98765
Bob	27	1	HUB	CS	B321	030/54321
Bob	27	1	Boston	CS	B321	030/54321
Charly	25	1	⊥	Psychology	⊥	⊥
Eve	24	1	Berlin	CS/EE	⊥	030/55544
Frank	23	0	TUB	Mech. Engineering	F205	⊥
Paul	⊥	1	TUB	Architecture	⊥	⊥
Paul	26	1	Berlin	Arch.	P233	⊥
Peter	⊥	0	TUB	Computer Science	P201	⊥

Result after applying Merge ($\boxtimes$) on tables U_1 and U_2.

values than t_1; and (3) if it coincides in all other attribute values with t_1. Minimum union has the same properties as outer union. The operator is uniqueness enforcing only if all data conflicts are uncertainties as a result of subsumed tuples, but it is not uniqueness enforcing in the general case. Contradictions are not resolved. Using the techniques from Rao et al. [2004] the minimum union operator can be expressed in SQL. For lack of space we omit the rather complicated rewriting and only show the result in Table VI.

3.3.2. Merge, Prioritized Merge. Based on the match join operator, Greco et al. [2001] define the merge operator ($\boxtimes$). This operator can be described as the union of two outer joins and can be rewritten in SQL.

```
( SELECT U1.Name, coalesce(U1.Age, U2.Age) AS Age,
        COALESCE(U1.Status, U2.Status) AS Status,
        COALESCE(U1.Address, U2.Address) as Address,
        COALESCE(U1.Field, U2.Field) as Field,
        U1.Library, U2.Phone
  FROM U1 LEFT OUTER JOIN U2 ON U1.Name = U2.Name )
  UNION
( SELECT U2.Name, COALESCE(U2.Age, U1.Age) AS Age,
        COALESCE(U2.Status, U1.Status) AS Status,
        COALESCE(U2.Address, U1.Address) as Address,
        COALESCE(U2.Field, U1.Field) as Field,
        U1.Library, U2.Phone
  FROM U1 RIGHT OUTER JOIN U2 ON U1.Name = U2.Name )
```

Applying the merge operator on the example relations results in the relation shown in Table VII. The merge operator shows the use of the COALESCE function to remove uncertainties, as could be seen when looking at the two Bob tuples: the missing AGE, FIELD, and STATUS values are replaced by the corresponding value from the other tuple. Unfortunately, this does not work for corresponding tuples in the same relation, (refer to the Paul tuples), although an additional benefit from this behavior is the removal of intersource subsumed tuples, as is the case with the Frank tuples. However, contradictions are not resolved. There exists a variant, namely prioritized merge ($\triangleleft$), that is used to prioritize among the sources and give preference values from one source. Merge is well suited in scenarios where many null values could be complemented, such as when data sources are sparsely filled or only slightly overlap in their schemata.

3.3.3. Increasing Conciseness in Union Results. As semantically equivalent attributes have already been matched, different representations reside in different rows of a union result. Therefore, the obvious way of removing possible data conflicts is by grouping representations of a single real-world object together and aggregating the values of the attributes.

The grouping part of this operation is uniqueness enforcing in the grouping attribute(s). In general, the result is, after grouping, similar to the union result, extensionally complete, and also does not require any "shaping," as is the case for join results.

Increasing conciseness in this way requires one or more attributes that serve as ID to identify same real-world objects. The aggregation part of this operation then takes the tuples of one group and merges them into one resulting tuple. This is done by applying aggregation functions to the attribute values, as illustrrated in this small example.

```
SELECT Name, max(Age), vote(Status), most_recent(Address), concat(Field),
       coalesce(Library), coalesce(Phone)
FROM union(U1,U2)
GROUP BY Name
```

The aggregation functions (`max`, `min`, `sum`, `count`, etc.) of the SQL standard allow only for limited possibilities in resolving conflicts. Therefore, the technique used by FraQL [Sattler et al. 2000, 2004; Schallehn and Sattler 2003] resolves uncertainties and contradictions by applying nonstandard user-defined aggregation functions to the attribute values in each group. They define four additional 2-parameter aggregation functions, each aggregating one column depending on the values of another column. In general, an arbitrarily complex, user-defined function could be used here in order to combine the values, remove uncertainties, and resolve conflicts. Unfortunately, such user-defined aggregation functions are supported by database management systems only in a very limited way, if at all. Therefore, the AXL framework [Wang and Zaniolo 2000] supports the user with the possibility to enable user-defined aggregation based on SQL in standard object-relational DBMSs for use in data integration.

User-defined aggregation is similar to the approach taken by the data-cleansing system Ajax [Galhardas et al. 2000a]; see Section 4.3.5 for more details.

3.4. Other Techniques

The following techniques are neither join, nor union-based, many incorporating additional information, extending the relational model or existing relational operators, or combining operators in order to fuse data.

3.4.1. Considering All Possibilities. Another possible way of dealing with uncertain data is to explicitly represent uncertainty in relations and use it to answer queries. The approaches described in Lim et al. [1994] and DeMichiel [1989] add an additional column to each table, its value helping to decide whether a tuple should be included in the query answer. Whereas DeMichiel [1989] marks a tuple with a discrete value (the value is one of yes, no, or maybe), the approach in Lim et al. [1994] uses probabilities (the value is $\in [0 \ldots 1]$). The operators of the relational algebra are extended to cope with this additional data and to use it. In the end, all tuples are presented to the user, together with the additional information as to whether it belongs to the result. The two main problems are that uncertainty is modeled at tuple level only and that it is generally difficult to determine the probabilities and where to take them from.

In Lim et al. [1997] relations are extended by an additional attribute, holding information on the data source the tuple is coming from. The relational operators are

extended to use this information and therefore this approach is an integrated implementation of a Trust Your Friend strategy (see Section 2.5).

Another way of representing multiple values and their attached probabilities is in allowing multiple-valued attribute types in a relation [Tseng et al. 1993]. In Tsai and Chen [2000], the authors present the partial natural outer join, which extends the full disjunction operation by allowing multiple values in an attribute together with their probability of being the correct value. Such a result is as complete as a regular full disjunction, but more concise. However, it is not as concise as the result in Lim et al. [1995] because conflicts are not resolved by a GAD operation. Instead, all values are preserved, with an attached probability of being the correct value.

In an OLAP scenario Burdick et al. [2005] use probability distribution functions to model both uncertainty on the value, as well as imprecision according to a dimension hierarchy. However, in contrast to the other approaches that include probabilities in the computation of results, in the end only one value is presented to the user; therefore the approach is uniqueness enforcing. The value given to the user is the one with highest probability of all the possible values. However, the approach does not allow any user interaction beyond giving the probabilities and it especially does not allow to influence the conflict resolution.

All these approaches implement the Consider All Possibilities strategy, using the additional information to allow the user to choose consciously among all possibilities or presenting the most probable value.

3.4.2. Considering Only Consistent Possibilities. A new line of research was initiated by work started in Arenas et al. [1999], which introduced the idea of returning only consistent information (not containing conflicts, given some constraints) from a database to a user when issuing a query. An easy example would be the consistent answer of `SELECT * FROM U2` under the constraint that the NAME column be unique: the tuples for Bob, Charly, and Frank.

Consistent query answering has developed into a large subfield of its own and can therefore only roughly addressed in this survey. The notion of a consistent answer is based on the idea of a repair, a new, consistent database created by only inserting or deleting tuples into/from an inconsistent database. Inconsistency is given by constraints on the data (e.g., functional or inclusion dependencies) that are not fulfilled. A possible repair for the aforementioned query could be created by deleting the first Alice and Eve tuple, respectively. In the relational world, a consistent answer to a query is the set of tuples that are contained in the answer to the query in *all* possible repairs. (In contrast, a possible answer would be a set of tuples contained in *any* possible repair.)

The overall goal in this line of research is to answer a query by giving a consistent answer. This generally requires the computation of all possible repairs and therefore has high computational costs. In contrast, most other techniques mentioned so far are fast and scalable, as only one specific solution is computed, or the requirements on constraints that need to be fulfilled are lowered. Research in this area usually limits the class of queries that are considered so as to be able to deal with tractable cases. It remains an open issue to find and describe more tractable and scalable classes of queries for consistent query answering.

Work in this line of research includes systems for answering queries in the relational world, such as, ConQuer (see Section 4.4.5) or HIPPO (see Section 4.4.4), the use of preference relationships among tuples, such as, Staworko et al. [2006], including aggregate constraints [Flesca et al. 2005; Bertossi et al. 2005], and using update operations instead of insert/delete operations as means to compute repairs [Wijsen 2003]. Recent work also includes modeling the computation of consistent answers as an optimization problem

and solving it heuristically [Bohannon et al. 2005]. Finally, Bertossi and Chomicki [2003] survey the field, giving an overview and additional literature.

3.5. Summary

Among the relational techniques, union approaches are in principle well suited for data fusion, as all information from the source tables are retained and extensional as well as intensional completeness are easily reached. Only unnecessary information (i.e., duplicate tuples or subsumed tuples) is deleted, increasing conciseness. As attributes describing the same semantic concept can be mapped, intensional conciseness is also reached in an intuitive manner. However, in most cases, too many tuples and therefore too much information is kept, requiring an additional step towards a more satisfying level of conciseness. This finally results in one single tuple per real-world object. The intuitive way of increasing conciseness together with the union approach is to apply grouping and aggregation.

On the other hand, join approaches generally retain information on same real-world objects in different columns (not in different rows). Columns with equal semantic attributes do not automatically coincide as in the union approaches. They need to be combined explicitly. Moreover, completeness comes more or less without cost, but conciseness needs to be achieved by an additional processing of the join result. If there are no intrasource duplicates in the tables, a simple combination of two rows using a user-defined function is sufficient. However, if there are intrasource duplicates, again, grouping and aggregation or some advanced join operator is needed.

In general, achieving the goal of completeness is the easy part in data fusion and is generally performed well by all approaches. The difficult part is achieving a concise result. Therefore, it makes sense to consider data fusion as a composition of two distinct operations: one that achieves the goal of completeness and one that achieves the goal of conciseness. The aforementioned techniques cannot be composed in arbitrary fashion, but only in the combinations mentioned in Sections 3.2.4 and 3.3.3. However, under the assumption that there are no intrasource duplicates and only considering two sources (binary operators), the following two combinations of operators are equivalent: (1) a full outer key- join of two tables followed by a GAD operation where semantically quivalent columns are combined by a function; and (2) an outer union (mapping semantically equivalent columns) of the same tables, followed by a grouping by the real-world ID and aggregating the remaining columns using the same functions that were used in the GAD operator. Similar but more general equivalences between these four operators and others, such as full disjunction, are still subject to research.

The other approaches mentioned in Section 3.4 do not fulfill the requirement of completeness (only consistent answers) or conciseness (all possible answers) to fit perfectly.

Data fusion is a highly domain-dependent task. Thus, one approach may work extremely well in one domain and fail in another. The life sciences, for example, comprise a domain where we deal with much contradictory information on same objects (e.g., genes, proteins). The consistent query answering approach would not work very well, as it would often produce an empty answer. Here it would be more beneficial to merge contradictory information into one consistent representation. On the other hand, in domains with a high percentage of exact data (e.g., financial data in companies), consistent query answering is more appropriate because it easily identifies or removes the few remaining errors in the data.

Concerning the different types of data conflicts, almost all approaches do not have any major problem in dealing with null values and therefore in resolving uncertainties. As sometimes (e.g., all approaches based on outer union) additional null values are introduced, techniques such as removing subsumed tuples or using the COALESCE

function are needed. Resolving inconsistencies could not be accomplished without applying functions. In the presence of both intrasource and intersource duplicates only, grouping and aggregation allows for assuring that a real-world object is only described by exactly one tuple. Join and union approaches alone are not able to guarantee this.

Using a modern relational DBMS, all approaches using standard relational operators are scalable. Problems with scalability only occur in systems that do not include an advanced query optimizer, as they are usually not able to cope with complicated rewritings (e.g., outer union or merge) or with user-defined functions or aggregation functions.

When deciding on what technique to use, we can distinguish three basic cases depending on the functional abilities of the system that is used, the nature of the data concerning duplicates, and the requirements on the result. These are as follows.

(1) Using only the functionality offered by SQL (SQL99, basic operations, and merge), we can fuse data but are only able to cope with uncertainties.
(2) If the use of user-defined functions is possible, a join approach in combination with UDFs is furthermore able to resolve inconsistencies, as long as no intrasource duplicates are present.
(3) Using grouping together with user-defined aggregation functions, enables us to also cope with intrasource duplicates and with inconsistencies.

Together with use of state-of-the-art relational DBMSs, which offer functionality as well as scalability, carrying out data fusion operations is possible.

4. INFORMATION SYSTEMS FOR DATA FUSION

In contrast to the previous section which covered relational techniques, in this section we present an overview of different information systems, out of the literature, that integrate data from different sources. We specifically observe their abilities to fuse data and handle conflicts. A few of these systems have already been mentioned because they have been used to demonstrate one of the relational techniques. However, we also include systems that use other data models or have only limited or no data fusion capabilities. Therefore, this is a sound but not necessarily complete list of data fusion/integration systems.

We first categorize data fusion systems into four classes and explain the different criteria we use to describe them. Second, we report on the single systems, categorized by their data fusion abilities.

4.1. Classification of Information Systems for Data Fusion

The following categorization of data fusion systems considers their data fusion capabilities. As stated in Section 2.5, we can distinguish different fusion strategies. As systems can implement one or more of these strategies, we present the systems classified by the most powerful strategy they are able to use. Existing systems can be categorized into the following four groups.

(1) The most advanced systems concerning data fusion are able to perform conflict resolution (Section 4.3).
(2) The next class of systems acknowledges data conflicts and handles them by conflict avoidance (Section 4.4).
(3) The most simple way of handling conflicts is conflict ignorance (Section 4.5).
(4) Finally, we consider all remaining systems that do not handle conflicts when integrating data from different sources (Section 4.6).

Table VIII. Main Architecture of Integration Systems and the Data Models Used in the Mediator/Sources (as well as the systems integration model)

System	Architecture	Internal Data Model (Mediator)	Source Data Model	Integration Model
Multibase	MDBMS	Daplex	Daplex	GaV
Hermes	MW	REL/OO	REL, OO, ...	GaV
Fusionplex	MW	REL	REL, OO, ...	GLaV
HumMer	MW	REL	REL, OO, ...	GaV
Ajax	APP	REL	REL	n/a
TSIMMIS	MW	OEM	OEM, REL, ...	GaV
SIMS/Ariadne	MW/MAS	Loom (OO)	REL, OO, SEMI, ...	LaV
Infomix	MW	SEMI	REL, XML, ...	GaV
Hippo	APP	REL	REL	n/a
ConQuer	APP	REL	REL	n/a
Rainbow	APP	REL	REL	n/a
Pegasus	DBMS	OO	OO, REL, ...	GaV
Nimble	MW	XML	XML	unknown
Carnot	MAS	Cyc	REL, SEMI, ...	LaV
InfoSleuth	MAS	Cyc	REL, SEMI, ...	LaV
Potter's Wheel	APP	REL	REL	n/a

Table IX. Query-Processing Abilities of Integration Systems and Categorization into Materialized/Virtual Integration System

System	Materialization	Manipulation	Access Method
Multibase	no	read-only	Daplex
Hermes	no	read-only	free text, logic based, graphical language
Fusionplex	no	read-only	ext. SQL
HumMer	no	read-only	ext. SQL
Ajax	yes	read-write	own language
TSIMMIS	no	read-only	MSL
SIMS/Ariadne	no	read-only	Loom query language
Infomix	no	read-only	SQL/Datalog
Hippo	no	read-only	SQL
ConQuer	no	read-only	SQL
Rainbow	no	read-only	SQL
Pegasus	no	read-write	HOSQL
Nimble	unknown	unknown	unknown
Carnot	no	read-only	SQL
InfoSleuth	no	read-only	SQL
Potter's Wheel	yes	read-write	GUI

The characterization is reflected in the subdivision of Tables VIII to XII describing the systems details. Systems of the fourth group are not included in the tables.

There does not exist a correlation between the architecture of the system and its fusion capabilities. However, database systems had first been used to integrate information starting in the late seventies, early eighties. They were followed by systems implementing the mediator-wrapper paradigm, as published in Wiederhold [1992]. With the boom of the multiagent system idea in the field of artificial intelligence this paradigm was also used in information integration, only recently replaced by peer-to-peer system architectures. Despite this evolution of system architectures, there have always been specifically designed stand-alone systems to handle data conflicts.

4.2. Properties of Information Systems for Data Fusion

The following list of criteria is used to describe the integration systems in this section. Specifically we give the questions that we want to answer with this criteria.

Table X. General Conflict-Handling Properties of Systems (conflict types and data conflict awareness)

System	Conflict Types	Data Conflict Awareness
Multibase	schematic, data	yes, [Dayal 1983]
Hermes	schematic, data	yes, [Subrahmanian et al. 1995]
Fusionplex	schematic, object, data	yes, [Motro and Anokhin 2006]
HumMer	schematic, object, data	yes, [Bleiholder and Naumann 2006]
Ajax	schematic, object, data	yes
TSIMMIS	schematic, data	yes, [Papakonstantinou et al. 1996]
SIMS/Ariadne	schematic, data	yes, partly, [Ambite et al. 2001]
Infomix	schematic, data	yes, [Leone et al. 2005]
Hippo	schematic, object, data	yes
ConQuer	schematic, object, data	yes
Rainbow	schematic, object, data	yes
Pegasus	schematic, data	no
Nimble	unknown	unknown
Carnot	schematic	no
InfoSleuth	schematic	no
Potter's Wheel	schematic	no

Table XI. Solutions to Schema Matching/Mapping and Duplicate Detection in Integration Systems

System	Schema Matching/Schema Mapping	Duplicate Detection
Multibase	hand crafted	assumes global id
Hermes	hand crafted	assumes global id
Fusionplex	hand crafted, learned in Autoplex	assumes global id
HumMer	semi-automatically	global id generated by duplicate detection
Ajax	hand crafted	global id generated by similarity measure
TSIMMIS	wrapper generation	assumes global id
SIMS/Ariadne	wrapper generation	mapping table
Infomix	hand crafted	assumes global id
Hippo	hand crafted	assumes global id
ConQuer	hand crafted	assumes global id
Rainbow	hand crafted	assumes global id
Pegasus	hand crafted	assumes given mapping table
Nimble	unknown	mapping table, generated
Carnot	hand crafted	assumes global id
InfoSleuth	hand crafted	assumes global id
Potter's Wheel	n/a	n/a

—*Architecture*. What type of architecture does the system implement? Is it a database management system (DBMS), a multidatabase management system (MDBMS), a mediator-wrapper system (MW), a multi-agent system (MAS), or is it a stand-alone application (APP), possibly on top of a DBMS? We can infer the coupling (tight or loose) between systems components from the type of architecture. An MDBMS is a database system with tightly coupled components, storing data in different DBMSs and using a special language to explicitely and directly access the distributed data. On the other hand, a mediator-wrapper system consists of two types of loosely coupled components. A mediator component is queried by the user using a standard language to transparently access the data. Data is stored in distributed sources, and made available through wrappers which translate queries and data between mediator and sources. A multi-agent system is the most loosely coupled system architecture and consists of a collection of agents interacting with each other. That way, it dissolves the strict hierarchical separation of mediator and wrappers. It could be seen as predecessor of peer data management systems (PDMSs). For more information on integrated information systems architecture, see Wiederhold [1992], Domenig and Dittrich [1999], and Sheth and Larson [1990].

Table XII. Data Fusion Capabilities, Possible Strategies and How Fusion is Specified in Integration Systems

System	Fusion Possible	Fusion Strategy	Fusion Specification
Multibase	resolution	Trust your friends, Meet in the middle	manually, in query
Hermes	resolution	Keep up to date, Trust your friends, ...	manually, in mediator
Fusionplex	resolution	Keep up to date	manually, in query
HumMer	resolution	Keep up to date, Trust your friends, Meet in the middle, ...	manually, in query
Ajax	resolution	various	manually, in workflow definition
TSIMMIS	avoidance	Trust your friends	manually, rules in mediator
SIMS/Ariadne	avoidance	Trust your friends	automatically
Infomix	avoidance	No Gossiping	automatically
Hippo	avoidance	No Gossiping	automatically
ConQuer	avoidance	No Gossiping	automatically
Rainbow	avoidance	No Gossiping	automatically
Pegasus	ignorance	Pass it on	manually
Nimble	ignorance	Pass it on	manually
Carnot	ignorance	Pass it on	automatically
InfoSleuth	unknown	Pass it on	unknown
Potter's Wheel	ignorance	Pass it on	manually, transformation

—*Mediator Data Model/Internal Data Model*. What data model does the system use in the mediator? If it is not a mediator-wrapper system, what data model is used internally? Is it based on the relational world (REL), on some object-oriented model (OO), or does it use some semistructured (SEMI) model, such as OEM, XML, or DAPLEX, a functional data model [Shipman 1981], or some other model?

—*Source Data Model*. What data model(s) is/are allowed for use in the data sources (REL, OO, ...)?

—*Integration Model*. Which integration model (GaV, LaV, GLaV) does the system use? The GaV (Global-as-View) integration model expresses a global schema (used in the mediator or internally) as views of local schemata (used in the data sources), whereas LaV (Local-as-View) connects local and global schemata the other way around. It expresses local schemata as views on the global schema. By allowing both possibilities to connect global and local schemata, GLaV (Global-Local-as-View) tries to combine the benefits of both models: the better scalability of LaV and the simpler query execution of GaV.

—*Materialization*. Does the system materialize data of the sources in the main system, or is it a virtual integration system that only provides access to the sources?

—*Manipulation*. Is the system able to manipulate data in the sources, or does it allow only read access?

—*Access Method*. How could the system be accessed or queried (query language, browsing, canned queries, etc.)?

—*Conflict Types*. What types of conflict are considered in the integration process used by the system? What types of conflict could possibly be handled? Schematic- object-, and data-level conflicts?

—*Data Conflict Awareness*. Is the system aware of actual data conflicts? Could they possibly be handled (semi-) automatically?

—*Schema Matching/Mapping*. What are the schema matching/mapping abilities of the system? How are mappings created to solve schematic conflicts?

—*Duplicate Detection*. How does the system detect and/or handle duplicate objects? Does it allow duplicates, does it assume a global ID?

—*Fusion Class*. Is the fusion of same real-world objects possible and what classes of strategy are available (ignorance, resolution, avoidance)?

—*Fusion Strategies*. Which concrete fusion strategies could possibly be used in the system or are implemented in the system? For a list of possible strategies, see Section 2.5.

—*Specification of Fusion*. How is fusion specified: in an automatic, semi-automatic, or manual manner?

The following tables (Table VIII to Table XII) list and overview 16 different integration systems and their properties. Each table orders the systems according to the classification from Section 4.1 into three groups: conflict-resolving systems (Hermes through Ajax), conflict-avoiding systems (TSIMMIS through Rainbow), and conflict-ignoring systems (Pegasus through Potter's Wheel). The many systems of the fourth class are not listed in the tables.

Table VIII lists the architecture of the systems, together with the data models used in the mediator (internally, respectively) and in the sources, as well as the integration model that is used in connecting global and local schemata. Table IX contains information on how data could be accessed and if data manipulation is possible. It also classifies the systems into materialized/virtual integration systems. Both tables together give a first overview on the systems and their integration fundamentals.

Table X considers conflict handling, especially the types of conflict the system is aware of and if the system is aware of data conflicts in particular. If possible, the research paper describing the system's conflict handling is cited therein. The system capabilities concerning schema matching/mapping and duplicate detection are summarized in Table XI. Finally, Table XII reports on the specific data fusion capabilities: what strategies are employed by the systems and how can fusion be specified.

All systems exist at least as a research prototype, although only two are currently downloadable or can be used over the Web: The research system Trio (not in the tables; see Section 4.6.1 for a description) and the conflict-ignoring system Potter's Wheel (see Section 4.5.5). In the remainder of the section we describe some systems in more detail and briefly cover others, emphasizing those systems with better data fusion capabilities.

4.3. Conflict-Resolving Systems

All systems in this group allow for full resolution of conflicting data, using a variety of strategies and following different implementation paradigms. The result of a query to these systems is in most cases a complete and concise answer.

4.3.1. Multibase. Multibase is one of the first multidatabase systems that mentions and considers the integration of data from different sources [Dayal 1983; Dayal and Hwang 1984; Landers and Rosenberg 1982]. Data in the sources is described using the DAPLEX data model [Shipman 1981], allowing also for the definition of global views on this local data.

Multibase is a fully functional database system that takes a global query in DAPLEX,[2] modifies it in terms of the local schema, and sends it to the local sites so that it can be executed there. Global and local query optimization is done by pushing selections down to the local sources.

[2]DAPLEX is used interchangeably to denote the query language and the underlying data model [Shipman 1981].

Data describing the same types of entities in different sources is integrated in the global view using the principle of generalization. A global class always generalizes several local classes. Dayal [1983] is among the first to identify the problem of having data conflicts in the global view when integrating data from the local sources using generalization. The work also describes how to combine instances from the sources if they are not disjoint and how to cope with conflicting data.

The solution implemented in Multibase is the use of an outer join operation, followed by an aggregation where conflicting values are fused by using basic aggregation functions such as min, max, sum, count, average, or choose. As already stated in Section 3.2.1, Multibase cannot resolve conflicts between intrasource duplicates because of the use of outer joins.

4.3.2. Hermes. The Hermes system (HEterogeneous Reasoning and MEdiator System) does not only integrate data sources but also reasoning facilities [Subrahmanian et al. 1995; Adali et al. 1996]. Different sources and reasoning facilities can be combined by a mediator to enable uniform access for an end user. The mediator is specified by an expert in a declarative way and serves to detect and handle several types of schematic conflicts as well as data conflicts. By resolving schematic conflicts, same concepts with different names in different sources could be mapped to a common global concept, following the GaV approach.

The mediator is specified using a variant of an annotated logic, annotations being the time that the concept, object, or value was entered into the system and its reliability. Each content of the database is true with that specified reliability at the given time.

The system includes tools that help the expert in incorporating new domains (during the integration phase of *domain integration*), such as other data models or other types of sources, and then to combine such sources into a mediator specification (during *semantic integration*) which can then be used to access data.

Conflicts on schema- and data level are recognized and partly solved, for example, using reliability and timestamp information. This way it is possible to implement a Trust Your Friends or a Keep Up To Date strategy within the system. The ability to also choose particular values (e.g., the maximum or minimum value) also allows other strategies.

The end user can access the system using a variety of query languages (e.g., free text, logic based, or using a graphical query language). The query is processed and executed by the mediator, query optimization being concerned with accessing the different sources that are being integrated in a cost-based way [Adali et al. 1996].

4.3.3. Fusionplex. The "plex" family of systems (see Motro et al. [2004] for an overview) consists of the three system: Multiplex [Motro 1999], Fusionplex [Motro et al. 2004; Motro and Anokhin 2006], and Autoplex [Berlin and Motro 2006]. Multiplex is a virtual multidatabase system that is capable of integrating information from different heterogeneous sources and serves as the basis for the other two systems. Fusionplex adds inconsistency recognition and reconciliation facilities based on provided quality information of the data source contents as additional *feature attributes*. Autoplex adds support for automatically adding new data sources, based on source discovery using machine learning techniques. The systems use the relational model internally, but are able to import data from other data models as well, as long as the sources are able to export data in a tabular manner.

In all systems, the schemata of the integrated data sources are considered to be different and overlapping, but they do not contain errors, which means that a mapping from the integrated sources to the global schema can be found (GLaV approach).

However, the content of the data sources is considered to contain errors, which means that duplicate objects, as well as different and contradicting object representations, may exist.

Fusionplex allows the user to resolve extensional inconsistencies at tuple level. The system groups tuples representing the same object according to a global key. It then uses quality metadata (timestamp, cost, accuracy, availability, clearance, etc.), represented in additional feature columns, to choose only high-quality tuples to be used in the fusion process. User preferences in the importance of the features, are taken into account. Next, fusion functions (`min`, `max`, `avg`, `any`, ...) are applied attribute-wise per object to come up with one value per attribute and object, the final object representation. Last, feature values are computed for the final representation.

In Autoplex rules are learned per data source, which select-project-join queries represent a good contribution to the global database. The system is then able to classify a new and unknown database on the basis of how good the contribution is, what it can add to the system as a whole and come up with a select-project-join query that incorporates the source into the global schema.

4.3.4. HumMer. The Humboldt-Merger (abbreviated HumMer) system is an integrated information system and allows the semi-automatic virtual integration of several remote and heterogeneous data sources [Bilke et al. 2005; Naumann et al. 2006]. All three steps of data integration (schema matching, duplicate detection and data fusion) are supported and automated as far as possible.

The system is used via a wizard-like interface which requests user input if needed, but uses default settings whenever possible to semi-automatically define a schema mapping between the sources that are to be integrated. This is done to semi-automatically detect duplicate object representations and fuse them into one clean representation. At each step, the user can influence the process. He can alter the proposed mapping, he can additionally classify possible duplicates as real or no duplicates, and he can define how data conflicts are to be resolved when fusing the object representations in the end. The system handles relational data, but is able to access a variety of data sources. HumMer can also be used in a query-driven way, where a fusion query is formulated using an extension to SQL and schema mapping and where duplicate detection is done beforehand using default settings.

The system considers the whole range of conflict types (schematic, identity, and data conflicts). Schematic conflicts are solved using a duplicate-based schema-matching approach [Bilke and Naumann 2005], whereas identity conflicts are solved using the DogmatiX approach [Weis and Naumann 2005] applied to relational data. The result of the duplicate detection step consists of duplicate clusters, where all duplicate object representations of a real-world object are assigned to the same group. Data fusion is accomplished by using this grouping, removing subsumed tuples, and then applying standard and advanced user-defined aggregation functions, such as `min`, `vote`, `most_abstract_concept`, to each group [Bleiholder and Naumann 2005]. Here, additional information (such as a taxonomy for the `most_abstract_concept` function) can be used, as well as functions that relate different columns (`choose_depending_value`), or we can use all values from a column when determining the fused value (e.g., using statistics or histograms). The data fusion operation is implemented as a database operator, which enables the use of algebraic and cost-based optimization techniques within the system.

4.3.5. Ajax. The data cleansing process in the Ajax system [Galhardas et al. 2000a, 2000b, 2001] is defined as a workflow on a logical layer using a declarative language.

Besides relational operators (`select`, `project`, `join`) it uses specialized operators for data transformation (e.g., `format` or `unit conversion`) and two more advanced operators for matching and merging. Duplicate detection (matching) is done by grouping object representations assigning keys, whereas the merging operator resolves existing data conflicts by aggregation. Once a cleansing process is defined on the logical layer, it is translated into Java programs on the physical layer, allowing for the use of different implementations of a logical operator. An exception handling mechanism allows users to influence the specified workflow during execution.

The system accounts for data conflicts and provides a basic mechanism for their resolution. However, conflicts are not classified and the merging operation is only briefly covered. The main focus of the AJAX system is on the matching step and an efficient and effective way of detecting duplicates while at the same time being able to specify, translate, and execute a complex data cleansing workflow.

4.4. Conflict-Avoiding Systems

The systems in this group allow for handling conflicting data by conflict avoidance. The result of these systems is, at the least, a concise answer.

4.4.1. TSIMMIS. The TSIMMIS (The Stanford IBM Manager of Multiple Information Sources) system uses wrappers to extract data from sources and convert it into its own data model OEM (Object Exchange Model) [Hammer et al. 1997; Garcia-Molina et al. 1997]. OEM is a semistructured data model, where data is represented as a quadruple (object ID, label, type, value) and there is no defined schema as in the relational model.

A wrapper converts data in the sources into the central data model. Wrappers are specified using a special language, WSL (Wrapper Specification Language), either by an expert or automatically by machine learning methods. The mediator combines data from the wrappers, and is specified using a similar language, MSL (Mediator Specification Language).

An end user queries the system using MSL. It is based on OQL and uses an object-oriented approach. Query processing follows the GaV model, with the mediator carrying out the necessary query unfolding and the wrappers converting and sending subqueries to sources. The result is converted back and shown to the user according to the mediator specification.

To identify objects, the system uses a skolem function to assign a unique global identifier to real-world objects. The existence of representations of same objects in different sources (intersource duplicates) is recognized, as are the possible data conflicts between them. However, in the presence of conflicting data, the system does not allow for resolution of the conflicts in the mediator, but avoids them using a simple Trust Your Friends strategy by only allowing to specify a preferred source during mediator specification [Papakonstantinou et al. 1996]. The value from that preferred source is then used in the final representation.

Some basic optimization issues in the distributed environment of the TSIMMIS system are considered, especially the execution of queries enhancing objects with information from other sources. To answer such "fusion"[3] queries, the system uses semijoins on distributed sources [Yerneni et al. 1998].

4.4.2. SIMS and Ariadne. The SIMS system (Services and Information Management for decision Systems) is a mediator-wrapper system and serves as implementation platform

[3]Interestingly, "fusion" is used here in the same sense as it is used in market research; see footnote in Section 2.

for integration projects from different domains [Ambite et al. 2001; Knoblock 1995; Arens et al. 1996]. SIMS manages a global schema together with various local schemata and a mapping between them expressed in Loom, a knowledge representation language often used in AI projects. Whereas initial versions of SIMS followed a GaV approach to integration, later versions [Ambite et al. 2001] combine LaV source descriptions with a mechanism to compile GaV mappings out of them.

Queries are processed in a three-step process: First, all relevant sources are selected and the query, expressed in terms of the global schema, is reformulated to be executed at the local sources. Second, the system generates a detailed query execution plan. During the third step the query is reformulated into a semantically equivalent plan of lower execution cost.

The main focus of this project is query planning in an AI setting. However, incompleteness and inconsistency in and between sources is recognized as a problem in such a data integration setting. Handling these problems is deferred to future work.

Ariadne extends the SIMS system to a WWW environment [Knoblock et al. 1998; Ambite et al. 1998, 2001]. It consists of wrappers for Web sources and a central mediator which is responsible for query processing. The main concerns are how to apply AI-fashion query planning to a large-scale environment such as the Web using adequate source descriptions.

The domain model and query language are the same as in SIMS, whereas Ariadne additionally includes binding patterns in the source descriptions. These are needed, as Web sources may have input constraints.

Query planning is broken down into two phases. During the first phase, the system selects an appropriate set of sources to query, and during the second phase an initial plan is subsequently improved by plan rewritings.

Wrappers for Web sources are constructed by automatically learning an extraction grammar from examples given by a user where the required information could be found on the Web page.

Ariadne recognizes the problem of same real-world entities having multiple representations in different sources, although the problem of intrasource duplicates is ignored. It uses manually constructed mapping tables to connect object IDs in different sources. How conflicts in other attributes are handled is not described, although for object IDs a Trust Your Friends strategy is used, in that values for object IDs are taken from a designated source. As future work, there is mention of developing semi-automatic learning techniques for creating the mapping tables.

4.4.3. Infomix. A mediator-wrapper system following the Consistent Query Answering approach (CQA) is the Infomix system [Eiter et al. 2003; Lembo et al. 2002; Leone et al. 2005] (see Section 3.4.2 for more information on CQA). In contrast to all other systems following this approach, very general queries could be answered because of the approach taken here, namely evaluating logic programs. However, the inherent complexity in evaluating logic programs constrains the use to only small databases. The system uses GaV mappings to map from a global schema to sources. It uses a subset of XML Schema and is therefore capable of incorporating not only relational data but also XML data, HTML data or data from other data models. The LixTo wrapper generation technique is used to add new sources. Optimization techniques are used to render the search for repairs feasible, in that it tries to do early selections in the sources. It uses logic programs to specify all repairs of the underlying inconsistent data sources and from thereon the consistent answer to a query. The problem of duplicate detection is not mentioned explicitly, but as data fusion is handled by only including consistent answers in the result, Infomix is aware of both problems and even offers a solution for the latter.

4.4.4. HIPPO. The HIPPO system also follows the Consistent Query Answering (CQA) paradigm [Chomicki et al. 2004a; Chomicki et al. 2004b]. Efficiently returning consistent answers heavily depends on the possibility to efficiently compute repairs, respectively to avoid having to compute all repairs. HIPPO solves this problem by using a special data structure: the conflict hypergraph. HIPPO can be used with select-join-union-difference queries and a special class of dependencies. The system is implemented on top of a relational DBMS, at the same time holding the conflict hypergraphs in main memory. Candidates are computed by the database system, whereas the final consistent answer is computed with the help of the data structure. However, main memory restricts the size of the conflict hypergraph and therefore the size of the databases that could be queried efficiently. Integration and duplicate detection capabilities depend on the underlying DBMS, although the system as a whole offers a solution to the data fusion problem by returning only consistent answers, thus implementing a No Gossiping strategy.

4.4.5. ConQuer. The ConQuer system, in contrast, uses a rewriting technique to compute consistent answers [Fuxman et al. 2005a, 2005b]. Given a set of key constraints the system rewrites an SQL query into an equivalent one that only returns consistent answers. In contrast to other consistent query answering systems, the rewriting works with any underlying relational DBMS and does not need any additional processing. As in all other CQA systems, duplicate detection is reduced to assuming a global key and data conflicts are avoided by only returning consistent answers (No Gossiping).

4.4.6. Rainbow. Another line of research in consistent query answering is followed by the Rainbow system [Caroprese and Zumpano 2006; Caroprese et al. 2006], which is a framework for implementing different possible ways of answering queries to inconsistent integrated databases in the presence of integrity constraints. A relational integration operation, such as prioritized merge (see Section 3.3.2) or the match join (see Section 3.2.3), is combined with a second step of returning consistent answers: The approach taken here is to introduce preference relations among different repairs and choose answers in preferred repairs.

4.5. Conflict-Ignoring Systems

The systems in the third group handle conflicting data by conflict ignorance. The result of these systems often is a complete answer.

4.5.1. Pegasus. The Pegasus system was developed in HP Labs as a multidatabase system, allowing a user to access different heterogeneous distributed data sources [Ahmed et al. 1991; Kent et al. 1992]. It serves as reference for many of the systems developed thereafter.

The system uses an object-oriented data model to represent data and offers a suitable query language, HOSQL, to access the data that either resides at the main site or is distributed among other sites. There, other data models could be used, enabling the system to access relational data as well as object-oriented data or data stored using another data model. Data access is not read only, but also considers updating the data by further distributing updates to the foreign sites.

For each foreign database there exists a mapping between the tables there and the view representing it at the main site. Query processing is done GaV style, translating the queries posed in HOSQL to Pegasus into queries to the foreign databases, and then sending and executing them there in the respective data model and query language. This is accomplished by the Pegasus agent, which resides on the local site and

represents Pegasus there. Global query optimization is done and tries to reduce intersite communication costs. Furthermore, it tries to reduce the number of queries sent to the distributed databases.

Imported schemata and schemata residing at the main site can be integrated into one schema using mappings as well. As an object-oriented model is used, generalization is used as in Multibase to combine two equivalent imported classes into one single global class. Here, the problem of eventually having duplicate representations of one and the same object of that class, and thus conflicts in different object representations, is recognized but not solved. The mechanism to deal with duplicate object representations in different sources is described in Kent et al. [1992] and consists of mapping object identifiers in different sources describing the same object to a global one. This is done using mapping tables or mapping functions. In general Pegasus assumes no extensional and intensional overlap (conflict ignorance). However, if there are duplicate objects, then they only exist in the view of the administrator of the system. The end user using the system should only see one representation of the object. In that way, the administrator is responsible for resolving conflicts by hand. There is no support by the system.

4.5.2. Nimble. Nimble is a commercial integration system [Draper et al. 2001a, 2001b; Halevy et al. 2005]. It is used by several large companies to clean and integrate different heterogeneous data sources. It is also one of the first integration systems that uses XML as data model.

As it is a commercial system, not many details on the techniques are published. However, the existence of duplicates is recognized and the system tries to find them by mining for columns in different tables that could be joined. The assumption is that there are pairs of values in these columns that identify same real-world objects in each table. Such a join index is built semi-automatically by using data mining techniques and exceptions are handled by an expert user, finally resulting in a mapping table. There is no specific information on how conflicts are handled in the system, but hints that some join-style integration mechanism (as detailed in Section 3.2) is implemented together with manual conflict resolution.

The system supports data lineage and some limited capabilities for defining cleansing processes in the Ajax style (see Section 4.3.5 for details on the Ajax system).

4.5.3. Carnot. The Carnot system [Collet et al. 1991; Singh et al. 1997] is a multiagent integration system. It consists of independent software agents that are not necessarily run at the same location but can be arbitrarily distributed. Each agent has its own area of expertise, some mapping source content into the global data model and schema (wrappers), others transforming or combining data (modeling a mediator). As global data model, Carnot uses the Cyc knowledge base [Lenat et al. 1990], an effort to model real-world knowledge and make it available to information systems by building a complex and complete ontology of real-world concepts. Queries to the system can be issued using SQL against one of the existing local schemata. Agents are responsible for mapping these queries to concepts from the Cyc ontology and from thereon to the data sources themselves. From a database perspective the system could be seen as a distributed, object-oriented database with a global schema that is built using very small components.

The mapping of sources into a global schema enables the integration of structured as well as unstructured data. Due to its modular structure, queries could be issued against local schemata or the global schema, with automatic translations between them. These translations use the source mappings and are able to overcome mismatched domains by value transformations (e.g., from dollar to euro).

Mappings are handcrafted by experts and when combining results, duplicates are not detected and data conflicts are ignored. Duplicate results coming from different sources are put together into one single result (Pass It On).

4.5.4. InfoSleuth. The InfoSleuth project [Bayardo, Jr. et al. 1997; Nodine et al. 1999] is the successor of the Carnot project and extends it to a more dynamic WWW scenario. Specifically, this scenario is where resources are added and removed over time, where it is unclear what sources are available at query time, and in short where we have truly autonomous data sources. The essential architecture of InfoSleuth does not differ much from Carnot; here as well are different specialized components (agents) that take care of the different tasks that need to be performed when integrating information. Queries are issued by an user agent and are distributed to the currently available resources that are able to deliver an answer. They are executed there and the results are combined to form the final result shown to the user. Combining the individual results is done by a specialized agent (mediator) using an integration plan. Although the use of statistical functions in an integration plan may lead to assume that some data fusion functionality is used, the problem of duplicate objects and possibly conflicting values is neither explicitly recognized nor solved.

4.5.5. Potter's Wheel. Potter's Wheel is a data cleansing system that operates on one source only [Raman et al. 1999; Raman and Hellerstein 2001]. The main focus is on the interactive definition of cleansing operations by a user. The system is freely available and consists of a GUI to the relational data the user wants to clean. He only sees a sample of the entire data, as the system aims at providing real-time cleansing facilities. Other data models, such as XML, are not supported.

A discrepancy detection process runs in the background and shows to the user data values, or whole columns that supposedly are dirty, for example, that contain outliers. The system does not only detect numerical outliers, but also string values that are formatted in a different way than the other values in that column. Using the discrepancies detected by the system as hints, a user can define transformations on the data. That way, values in columns can be reformatted, changed, copied to another column, or split and divided into two columns. Complex selections combining two columns into one and a folding operation are also available. Complex combinations of transformations could be saved, reloaded, and executed not only on the sample, but also on the whole dataset. The translation of transformations into SQL or XQuery statements is mentioned as an extension to the system, to use already existing query optimization facilities.

The idea of duplicate representations of same real-world objects does not exist as concept in the system. Therefore there is no explicit possibility to detect such duplicate representations and combine them into only one representation. In our three-step integration process, a system like Potter's Wheel could be used before the duplicate detection step, in order to increase the quality of the data to be able to more easily detect duplicates.

4.6. Other Systems

The following systems do not consider duplicate detection or data fusion techniques in detail, but nevertheless should be mentioned in order to give a more complete overview of integrating information systems and to illustrate the relatively small percentage of systems handling conflicting data.

Many of the research systems are described in more detail in surveys such as Sheth and Larson [1990] or Domenig and Dittrich [1999]. However, the main focus of these other surveys is on distinguishing and classifying distributed information systems

mainly by their architecture and other characteristics, but not according to their data fusion capabilities.

4.6.1. Research Systems. The Trio project aims at developing a new DBMS for storing inexact information and revives some of the older research done in the area of probabilistic databases [Widom 2005; Sarma et al. 2006; Agrawal et al. 2006; Benjelloun et al. 2006]. The project tackles data conflicts by explicitly including information on accuracy and lineage into its data model and thereby into the data itself. As with all other probabilistic databases (see Parsons [1996] for a survey on probabilistic and other techniques to handle and model imperfect information), the approach taken here is to keep all data in the database, annotate it with quality metadata and lineage, and provide a powerful query language. That way they do not have a built-in notion of duplicate objects and defer conflict handling to the user.

The Information Manifold system is the first system to mention and implement what is later coined the Local-as-View approach (LaV) to data integration [Levy et al. 1996a; Levy et al. 1996b]. A specialized query planner is used for query answering. It uses coverage and overlap information on the sources, as well as their capabilities (what selections are possible, full relational capabilities, what inputs are possible, etc.) to determine in which order to access and query what sources. Information Manifold assumes that objects are globally uniquely identified by object IDs, using correspondence functions to map object identifiers from one source to another source if not. There is no further discussion on duplicate detection methods and the need for data fusion is not mentioned.

The Garlic project was initiated to manage large amounts of multimedia data [Cody et al. 1995; Josifovski et al. 2002]. All data is stored at different distributed sources and is virtually integrated in a mediator. The main focus in the Garlic project is on distributed storing of large amounts of data and query planning in this setting. Thus, the characteristic integration problems of identifying same real-world objects and how to handle conflicts are not considered deeply, and not even recognized. Garlic was subsequently used for biological data [Haas et al. 2000] and was the basis for IBM's DB2 DiscoveryLink, which is now part of the commercial DB2 database system.

The Disco (Distributed Information Search Component) system [Tomasic et al. 1997, 1998] focuses on and solves three problems when querying heterogeneous data sources: (1) the problem of how to cope with data sources that do not support the full language of the mediator; (2) the problem of cost based wrapper optimization; and (3) the problem of what wrapper to use, given that not all wrappers have equal costs. The problem of duplicate detection is mentioned, as well as the need for data fusion, although no solutions are given.

Among the oldest mediator-wrapper systems are Papyrus [Connors et al. 1991] and Nomenclatur [Ordille and Miller 1993]. In the midst of the nineties, there were many research groups dealing with the topic, building systems such as DIOM [Liu and Pu 1995], Rufus [Shoens et al. 1993], or KOMET [Calmet et al. 1997; Calmet and Kullmann 1999]. A local-as-view approach is used in Infomaster [Genesereth et al. 1997] and Occam [Kwok and Weld 1996], which comes from the AI community like SIMS or the Internet Softbot. Examples for systems from the late nineties are Singapore [Dittrich and Domenig 1999], Magic [Reck and König-Ries 1997], and Observer [Mena et al. 1996]. Lore [McHugh et al. 1997] and Tukwila [Ives et al. 1999] are examples for systems who focus on semistructured data.

Other research database management systems dealing with data integration were developed mostly in the eighties. Research on data integration during that time mostly dealt with issues in resolving syntactic heterogeneity. In the beginning of the eighties,

research revolved around nonfederated or tightly coupled systems such as SIRIUS-DELTA [Litwin et al. 1982], DDTS [Dwyer and Larson 1987], Mermaid [Brill et al. 1984; Templeton et al. 1987], or UNIBASE [Brzezinski et al. 1984]. In the late eighties the focus switched to research on more loosely coupled systems, like MRDSM [Litwin 1985; Litwin and Abdellatif 1987], OMNIBASE [Rusinkiewicz et al. 1989], CALIDA [Rajinikanth et al. 1990; Jakobson et al. 1988], or DQS [Belcastro et al. 1988]. None of these systems specifically considers or solves the problem of duplicate detection or data fusion, despite their ability to integrate several heterogeneous, remote data sources.

4.6.2. Commercial DBMSs, Data Warehouse Systems, and ETL Systems. Commercially available database management systems from vendors such as IBM, Oracle, or Microsoft, to name just a few, can usually be used to integrate data from different sources. They offer the full functionality to use the relational techniques mentioned in Section 3, sometimes extending them with vendor-specific support for user-defined functions, aggregations, or procedures. In addition, some vendors support the connection of other, remote, data sources to be included in standard query processing and basic notions of a distributed database. Connecting other sources is mostly done via a GaV mapping, allowing for easier query processing and use of the built-in optimizer. Practically all commercially available DBMSs support the XML data model, in varying ways, next to the relational model. A few even come shipped with specialized tools as add-on-to detect duplicates or design ETL workflows for data cleansing (see, e.g., Chaudhuri et al. [2005]). Here, various fuzzy duplicate detection methods are either already provided or could be plugged in easily. The subsequent data fusion phase, however, is frequently neglected. In all other DBMSs, duplicate detection is implemented using key constraints.

Typical ETL tools, such as IBM's Information Server (IIS) or Microsoft's SQL Server Integration Services (SSIS), provide some rule-based functionality to specify data fusion semantics. Usually the rules allow only to remove uncertainties, but do not resolve data conflicts. Such functionality is typically offered using the keywords "survivor" and "consolidation".

4.6.3. Peer Data Management Systems. Recently, peer data management systems (PDMSs) have become a popular architecture for data integration. Therein, autonomous peers host a schema and possibly some data. Peers are interconnected via schema mappings in a network fashion. Queries can be posed against any peer schema and are translated along mapping paths so that all participating peers contribute to the final result.

Due to the recency of this architecture, not many PDMSs have tackled the problem of data fusion. Merely a few explicitly recognize inconsistent tuples and remove them from the query result. The system Orchestra maintains conflicting data by allowing multiple viewpoints, each from the view of a specific source [Ives et al. 2005]. Thus conflicts are not resolved, but recognized and hidden. Orchestra also reconciles contradictory updates, relying on the principle of least surprise, that is, choosing the most recent update as the resolved value.

Similarly, the proposed logical extension of the Hyper system is able to recognize and hide local inconsistencies from the global perspective of the integrating system by isolating locally inconsistent peers [Calvanese et al. 2005]. Global inconsistency, that is, conflicts that arise only when integrating data from different peers, is achieved by isolating the minimum amount of data to reach consistency.

4.7. Summary

Research in information integration started with the relatively easy problems of overcoming technical heterogeneity before moving on to the more interesting problems of

structural and schematic heterogeneity. Over time, as more and more of the relevant issues in these fields had been solved, research started tackling semantic heterogeneity and dealing with dirty data. Besides that trend, efficient query processing in integrated information systems had at all times be an issue.

Out of all the research information integration systems that have been built so far, only some are actually able to handle conflicting data. Most integration systems only cope with schematic conflicts, as this has been the field with the longest research history. Some also consider identity conflicts: the existence of duplicate records in different sources. However, only a few of them recognize and deal with all possible varieties of intra- and intersource data conflicts. These systems take data conflict handling seriously and often return a complete and/or concise answer. They can be divided into two groups: The first group recognizes and handles, but does not resolve, conflicting data (conflict avoiding systems). They implement a simple strategy to handle data conflicts, for example, by preferring data from one specific source. The second group detects data conflicts and uses (nearly all) existing information to come up with a good conflict resolution in the end.

Specification of data fusion is different from system to system, at various levels of complexity, increasing complexity resulting in more possibilities to specify data fusion. As the research focus changed from schematic over identity to data conflicts, the systems' architecture also changed. In the beginning, research in this field mainly dealt with DBMSs, multi-DBMSs, or distributed DBMSs, before systems architectures became much more loosely coupled with the advent of mediator-wrapper systems, or lately PDMSs.

In comparison to research systems, commercially available information systems have only limited data fusion capabilities, if at all. The problem of duplicate records has received relatively large attention in companies and therefore there exist, in addition to standard DBMSs, many specialized software products that are able to detect duplicates in data. However, how to fuse duplicate records into one representation in the end often is also an open issue there, and most products use a survivor strategy by choosing one of the existing representations and additionally replace null values with data values from other representations.

5. CONCLUSION AND SUMMARY

In this survey we introduced the problem of data fusion in the larger context of data integration, where data fusion is the last step in a data integration process, schemata have been matched, and duplicate records have been identified. Merging these duplicate records into a single representation and at the same time resolving existing data conflicts is still out of the focus of mainstream research in the field of information integration systems. However, the problem has been addressed by several researchers in the past decades.

In the second part of this survey, we gave an overview and compared common relational techniques for data fusion. We showed how they cope with data conflicts and mention the characteristics of the results that they produce. In the third part, we presented and commented on a list of information integration systems that are capable of fusing data in various ways. We classified the systems according to their abilities of handling conflicts.

Conflict handling can easily be described by the strategy the systems use to handle conflicts. Many of the possible strategies mentioned in the first part have been successfully implemented and tested in a system or research prototype, or can be implemented using relational operators. The sheer amount of existing information integration

systems does not allow to conclude that data fusion is easily accomplished because only a few of the systems actually tackle the problem.

With this survey we intend to introduce the interested reader to the existing possibilities to accomplish data fusion as part of an information integration process and help to choose among them.

REFERENCES

ADALI, S., CANDAN, K. S., PAPAKONSTANTINOU, Y., AND SUBRAHMANIAN, V. S. 1996. Query caching and optimization in distributed mediator systems. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, 137–146.

AGRAWAL, P., BENJELLOUN, O., SARMA, A. D., HAYWORTH, C., NABAR, S. U., SUGIHARA, T., AND WIDOM, J. 2006. Trio: A system for data, uncertainty, and lineage. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1151–1154.

AHMED, R., DE SMEDT, P., DU, W., KENT, W., KETABCHI, M. A., LITWIN, W. A., RAFII, A., AND SHAN, M.-C. 1991. The Pegasus heterogeneous multidatabase system. *IEEE Comput. 24*, 12, 19–27.

AMBITE, J. L., ASHISH, N., BARISH, G., KNOBLOCK, C. A., MINTON, S., MODI, P. J., MUSLEA, I., PHILPOT, A., AND TEJADA, S. 1998. Ariadne: A system for constructing mediators for Internet sources. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, 561–563.

AMBITE, J. L., KNOBLOCK, C. A., MUSLEA, I., AND PHILPOT, A. G. 2001. Compiling source descriptions for efficient and flexible information integration. *J. Intell. Inf. Syst. 16*, 2, 149–187.

ARENAS, M., BERTOSSI, L. E., AND CHOMICKI, J. 1999. Consistent query answers in inconsistent databases. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*. ACM Press, 68–79.

ARENS, Y., KNOBLOCK, C. A., AND SHEN, W.-M. 1996. Query reformulation for dynamic information integration. *J. Intell. Inf. Syst. 6*, 2-3 (June), 99–130.

BATINI, C., LENZERIN, M., AND NAVATHE, S. B. 1986. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv. 18*, 4, 323–364.

BAYARDO, JR., R. J., BOHRER, W., BRICE, R., CICHOCKI, A., FOWLER, J., HELAL, A., KASHYAP, V., KSIEZYK, T., MARTIN, G., NODINE, M., RASHID, M., RUSINKIEWICZ, M., SHEA, R., UNNIKRISHNAN, C., UNRUH, A., AND WOELK, D. 1997. InfoSleuth: Agent-Based semantic integration of information in open and dynamic environments. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, New York, 195–206.

BELCASTRO, V., DUTKOWSKI, A., KAMINSKI, W., KOWALEWSKI, M., MALLAMACI, C. L., MESZYK, S., MOSTARDI, T., SCROCCO, F. P., STANISZKIS, W., AND TURCO, G. 1988. An overview of the distributed query system DQS. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. Springer, 170–189.

BENJELLOUN, O., SARMA, A. D., HAYWORTH, C., AND WIDOM, J. 2006. An introduction to ULDBs and the Trio system. *IEEE Data Eng. Bull. 29*, 1, 5–16.

BERLIN, J. AND MOTRO, A. 2006. Tuplerank: Ranking discovered content in virtual databases. In *Proceedings of the International Workshop on Next Generation Information on Technology and Systems (NGITS)*, 13–25.

BERTOSSI, L. E., BRAVO, L., FRANCONI, E., AND LOPATENKO, A. 2005. Complexity and approximation of fixing numerical attributes in databases under integrity constraints. In *Proceedings of the International Conference on Database Programming Languages (DBPL)*, 262–278.

BERTOSSI, L. E. AND CHOMICKI, J. 2003. Query answering in inconsistent databases. In *Logics for Emerging Applications of Databases*, 43–83.

BILKE, A., BLEIHOLDER, J., BÖHM, C., DRABA, K., NAUMANN, F., AND WEIS, M. 2005. Automatic data fusion with HumMer. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1251–1254.

BILKE, A. AND NAUMANN, F. 2005. Schema matching using duplicates. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 69–80.

BLEIHOLDER, J. AND NAUMANN, F. 2005. Declarative data fusion—Syntax, semantics, and implementation. In *Proceedings of the East European Conference on Advances in Databases and Information Systems (ADBIS)*, 58–73.

BLEIHOLDER, J. AND NAUMANN, F. 2006. Conflict handling strategies in an integrated information system. In *Proceedings of the IJCAI Workshop on Information on the Web (IIWeb)*.

BOHANNON, P., FLASTER, M., FAN, W., AND RASTOGI, R. 2005. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*, 143–154.

BRILL, D., TEMPLETON, M., AND YU, C. T. 1984. Distributed query processing strategies in Mermaid, a frontend to data management systems. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 211–218.

BRZEZINSKI, Z., GETTA, J. R., RYBNIK, J., AND STEPNIEWSKI, W. 1984. Unibase—An integrated access to databases. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. Morgan Kaufmann, San Francisco, CA, 388–396.

BURDICK, D., DESHPANDE, P., JAYRAM, T. S., RAMAKRISHNAN, R., AND VAITHYANATHAN, S. 2005. OLAP over uncertain and imprecise data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 970–981.

CALMET, J., JEKUTSCH, S., AND SCHÜ, J. 1997. A generic query-translation framework for a mediator architecture. In *Proceedings of the International Conference on Data Engineering (ICDE)*, W. A. Gray and P.-Å. Larson, Eds. IEEE Computer Society, 434–443.

CALMET, J. AND KULLMANN, P. 1999. Meta Web search with KOMET. In *Proceedings of the Workshop on Intelligent Information Integration*.

CALVANESE, D., GIACOMO, G. D., LEMBO, D., LENZERINI, M., AND ROSATI, R. 2005. Inconsistency tolerance in P2P data integration: An epistemic logic approach. In *Proceedings of the International Conference on Database Programming Languages (DBPL)*.

CAROPRESE, L., GRECO, S., TRUBITSYNA, I., AND ZUMPANO, E. 2006. Preferred generalized answers for inconsistent databases. In *Proceedings of the Internation Symposium on Methodologies for Information Systems (ISMIS)*, 344–349.

CAROPRESE, L. AND ZUMPANO, E. 2006. A framework for merging, repairing and querying inconsistent databases. In *Proceedings of the East European Conference on Advances in Databases and Information Systems (ADBIS)*, 383–398.

CHAUDHURI, S., GANJAM, K., GANTI, V., KAPOOR, R., NARASAYYA, V., AND VASSILAKIS, T. 2005. Data cleaning in Microsoft SQL Server 2005. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, New York, 918–920.

CHOMICKI, J., MARCINKOWSKI, J., AND STAWORKO, S. 2004a. Computing consistent query answers using conflict hypergraphs. In *Proceedings of the Internation Conference on Information and Knowledge Management (CIKM)*. ACM Press, New York, 417–426.

CHOMICKI, J., MARCINKOWSKI, J., AND STAWORKO, S. 2004b. Hippo: A system for computing consistent answers to a class of SQL queries. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 841–844.

CODY, W. F., HAAS, L. M., NIBLACK, W., ARYA, M., CAREY, M. J., FAGIN, R., FLICKNER, M., LEE, D., PETKOVIC, D., SCHWARZ, P. M., THOMAS, J., ROTH, M. T., WILLIAMS, J. H., AND WIMMERS, E. L. 1995. Querying multimedia data from multiple repositories by content: The Garlic project. In *Proceedings of the IFIP Working Conference on Visual Database Systems (VDB-3)*. Chapman & Hall, Ltd., 17–35.

COHEN, S. AND SAGIV, Y. 2005. An incremental algorithm for computing ranked full disjunctions. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*. ACM Press, New York, 98–107.

COLLET, C., HUHNS, M. N., AND SHEN, W.-M. 1991. Resource integration using a large knowledge base in Carnot. *IEEE Comput. 24*, 12, 55–62.

CONNORS, T., HASAN, W., KOLOVSON, C., NEIMAT, M.-A., SCHNEIDER, D., AND WILKINSON, K. 1991. The Papyrus integrated data server. In *Proceedings of the International Conference on Parallel and Distributed Information Systems*. IEEE Computer Society Press, 139–141.

DAYAL, U. 1983. Processing queries over generalization hierarchies in a multidatabase system. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 342–353.

DAYAL, U. AND HWANG, H.-Y. 1984. View definition and generalization for database system integration in a multidatabase system. *IEEE Trans. Softw. Eng. 10*, 6 (Nov,), 628–645.

DEMICHIEL, L. G. 1989. Resolving database incompatibility: An approach to performing relational operations over mismatched domains. *IEEE Trans. Knowl. Data Eng. 1*, 4, 485–493.

DITTRICH, K. R. AND DOMENIG, R. 1999. Towards exploitation of the data universe: Database technology for comprehensive query services. In *Proceedings of the International Conference on Business Infromation Systems (BIS)*.

DOMENIG, R. AND DITTRICH, K. R. 1999. An overview and classification of mediated query systems. *SIGMOD Rec. 28*, 3, 63–72.

DRAPER, D., HALEVY, A. Y., AND WELD, D. S. 2001a. The Nimble integration engine. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, New York, 567–568.

DRAPER, D., HALEVY, A. Y., AND WELD, D. S. 2001b. The Nimble XML data integration system. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 155–160.

DWYER, P. AND LARSON, J. 1987. Some experiences with a distributed database testbed system. *Proc. IEEE 75*, 5 (May), 633–648.

EITER, T., FINK, M., GRECO, G., AND LEMBO, D. 2003. Efficient evaluation of logic programs for querying data integration systems. In *Proceedings of the International Conference on Logic Programming (ICLP)*, 163–177.

FAGIN, R., KOLAITIS, P. G., AND POPA, L. 2005. Data exchange: Getting to the core. *Trans. Dat. Syst. 30*, 1, 174–210.

FLESCA, S., FURFARO, F., AND PARISI, F. 2005. Consistent query answers on numerical databases under aggregate constraints. In *Proceedings of the International Conference on Database Programming Languages (DBPL)*, 279–294.

FUXMAN, A., FAZLI, E., AND MILLER, R. J. 2005a. ConQuer: Efficient management of inconsistent databases. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, New York, 155–166.

FUXMAN, A., FUXMAN, D., AND MILLER, R. J. 2005b. ConQuer: A system for efficient querying over inconsistent databases. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1354–1357.

GALHARDAS, H., FLORESCU, D., SHASHA, D., AND SIMON, E. 2000a. AJAX: An extensible data cleaning tool. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*, W. Chen et al., 590.

GALHARDAS, H., FLORESCU, D., SHASHA, D., AND SIMON, E. 2000b. An extensible framework for data cleaning. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 312.

GALHARDAS, H., FLORESCU, D., SHASHA, D., SIMON, E., AND SAITA, C.-A. 2001. Declarative data cleaning: Language, model, and algorithms. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 371–380.

GALINDO-LEGARIA, C. A. 1994. Outerjoins as disjunctions. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, 348–358.

GARCIA-MOLINA, H., PAPAKONSTANTINOU, Y., QUASS, D., RAJARAMAN, A., SAGIV, Y., ULLMAN, J., VASSALOS, V., AND WIDOM, J. 1997. The TSIMMIS approach to mediation: Data models and languages. *J. Intell. Inf. Syst. 8*, 2, 117–132.

GENESERETH, M. R., KELLER, A. M., AND DUSCHKA, O. M. 1997. Infomaster: An information integration system. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*, 539–542.

GRECO, S., PONTIERI, L., AND ZUMPANO, E. 2001. Integrating and managing conflicting data. In *Revised Papers from the 4th International Andrei Ershov Memorial Conference on Perspectives of System Informatics*. Springer, 349–362.

HAAS, L. M., KODALI, P., RICE, J. E., SCHWARZ, P. M., AND SWOPE, W. C. 2000. Integrating life sciences data-with a little Garlic. In *Proceedings of the IEEE International Conference on Bioinformatics and Bio Engineering (BIBE)*. IEEE Computer Society, 5.

HALEVY, A. Y., ASHISH, N., BITTON, D., CAREY, M. J., DRAPER, D., POLLOCK, J., ROSENTHAL, A., AND SIKKA, V. 2005. Enterprise information integration: Successes, challenges and controversies. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, New York, 778–787.

HAMMER, J., MCHUGH, J., AND GARCIA-MOLINA, H. 1997. Semistructured data: The TSIMMIS experience. In *Proceedings of the East European Conference on Advances in Databases and Information Systems (ADBIS)*, 1–8.

HERNÁNDEZ, M. A. AND STOLFO, S. J. 1998. Real-World data is dirty: Data cleansing and the merge/purge problem. *Data Mining Knowl. Discov. 2*, 1, 9–37.

IVES, Z. G., FLORESCU, D., FRIEDMAN, M., LEVY, A. Y., AND WELD, D. S. 1999. An adaptive query execution system for data integration. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*, 299–310.

IVES, Z. G., KHANDELWAL, N., KAPUR, A., AND CAKIR, M. 2005. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 107–118.

JAKOBSON, G., PIATETSKY-SHAPIRO, G., , LAFOND, C., RAJINIKANTH, M., AND HERNANDEZ, J. 1988. CALIDA: A knowledge-based system for integrating multiple heterogeneous databases. In *Proceedings of the 3rd International Conference on Data and Knowledge Bases: Improving Usability and Responsiveness*, 3–18.

JOSIFOVSKI, V., SCHWARZ, P., HAAS, L., AND LIN, E. 2002. Garlic: A new flavor of federated query processing for DB2. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, 524–532.

KENT, W., AHMED, R., ALBERT, J., KETABCHI, M. A., AND SHAN, M.-C. 1992. Object identification in multidatabase systems. In *Proceedings of the IFIP WG 2.6 Database Semantics Conference on Interoperable Database Systems (DS-5)*, 313–330.

KIM, W., CHOI, B.-J., HONG, E.-K., KIM, S.-K., AND LEE, D. 2003. A taxonomy of dirty data. *Data Mining Knowl. Discov. 7*, 1, 81–99.

KNOBLOCK, C. A. 1995. Planning, executing, sensing, and replanning for information gathering. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, C. Mellish, ed. Morgan Kaufmann, San Francisco, CA, 1686–1693.

KNOBLOCK, C. A., MINTON, S., AMBITE, J. L., ASHISH, N., MODI, P. J., MUSLEA, I., PHILPOT, A. G., AND TEJADA, S. 1998. Modeling Web sources for information integration. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. American Association for Artificial Intelligence, Menlo Park, CA, 211–218.

KWOK, C. T. AND WELD, D. S. 1996. Planning to gather information. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. AAAI/MIT Press, Portland, 32–39.

LANDERS, T. AND ROSENBERG, R. L. 1982. An overview of MULTIBASE. In *Proceedings of the 2nd International Symposium on Distributed Data Bases*, H. J. Schneider, ed. North Holland, Berlin.

LEMBO, D., LENZERINI, M., AND ROSATI, R. 2002. Source inconsistency and incompleteness in data integration. In *Proceedings of the International Workshop on Knowledge Representation Meets Databases (KRDB)*.

LENAT, D. B., GUHA, R. V., PITTMAN, K., PRATT, D., AND SHEPHERD, M. 1990. CYC: Toward programs with common sense. *Commun. ACM 33*, 8, 30–49.

LEONE, N., GRECO, G., IANNI, G., LIO, V., TERRACINA, G., EITER, T., FABER, W., FINK, M., GOTTLOB, G., ROSATI, R., LEMBO, D., LENZERINI, M., RUZZI, M., KALKA, E., NOWICKI, B., AND STANISZKIS, W. 2005. The INFOMIX system for advanced integration of incomplete and inconsistent data. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*, 915–917.

LEVENSHTEIN, V. 1965. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems Inf. Transm. 1*, 8–17.

LEVY, A. Y., RAJARAMAN, A., AND ORDILLE, J. J. 1996a. Querying heterogeneous information sources using source descriptions. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. Morgan Kaufmann, 251–262.

LEVY, A. Y., RAJARAMAN, A., AND ORDILLE, J. J. 1996b. The World Wide Web as a collection of views: Query processing in the information manifold. In *Proceedings of the SIGMOD Workshop on Materialized Views: Techniques and Applications (VIEW)*, 43–55.

LIM, E.-P., CAO, Y., AND CHIANG, R. H. L. 1997. Source-Aware multidatabase query processing. In *Proceedings of the Workshop on Engineering Federated Information Database Systems (EFDBS)*, 69–80.

LIM, E.-P., SRIVASTAVA, J., AND HWANG, S.-Y. 1995. An algebraic transformation framework for multidatabase queries. *Distrib. Parallel Databases 3*, 3, 273–307.

LIM, E.-P., SRIVASTAVA, J., AND SHEKHAR, S. 1994. Resolving attribute incompatibility in database integration: An evidential reasoning approach. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 154–163.

LITWIN, W. 1985. An overview of the multidatabase system MRDSM. In *Proceedings of the ACM Annual Conference on the Range of Computing : Mid-80's Perspective*. ACM Press, New York, 524–533.

LITWIN, W. AND ABDELLATIF, A. 1987. An overview of the multi-database manipulation language MDSL. *Proc. IEEE 75*, 5 (May), 621–632.

LITWIN, W., BOUDENANT, J., ESCULIER, C., FERRIER, A., GLORIEUX, A. M., CHIMIA, J. L., KABBAJ, K., MOULINOUX, C., ROLIN, P., AND STANGRET, C. 1982. SIRIUS system for distributed data management. In *Distributed Databases*. North-Holland, Amsterdam, The Netherlands, 311–343.

LIU, L. AND PU, C. 1995. The distributed interoperable object model and its application to large-scale interoperable database systems. In *Proceedings of the Internation Conference on Information and Knowledge Management (CIKM)*. ACM Press, New York, 105–112.

MCHUGH, J., ABITEBOUL, S., GOLDMAN, R., QUASS, D., AND WIDOM, J. 1997. Lore: A database management system for semistructured data. *SIGMOD Rec. 26*, 3, 54–66.

MELNIK, S., BERNSTEIN, P. A., HALEVY, A., AND RAHM, E. 2005. Supporting executable mappings in model management. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*, 167–178.

MENA, E., KASHYAP, V., SHETH, A. P., AND ILLARRAMENDI, A. 1996. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. In *Proceedings of the IFCIS Conference on Cooperative Information Systems (CoopIS)*, 14–25.

MILLER, R. J., IOANNIDIS, Y. E., AND RAMAKRISHNAN, R. 1993. The use of information capacity in schema integration and translation. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, R. Agrawal et al., eds. Morgan Kaufmann, 120–133.

MOTRO, A. 1986. Completeness information and its application to query processing. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 170–178.

MOTRO, A. 1999. Multiplex: A formal model for multidatabases and its implementation. In *Proceedings of the International Workshop on Next Generation Information on Technology and Systems (NGITS)*. Springer, 138.

MOTRO, A. AND ANOKHIN, P. 2006. Fusionplex: Resolution of data inconsistencies in the integration of heterogeneous information sources. *Inf. Fusion 7*, 2, 176–196.

MOTRO, A., ANOKHIN, P., AND ACAR, A. C. 2004. Utility-Based resolution of data inconsistencies. In *Proceedings of the International Workshop on Information Qualities in Information Systems (IQIS)*. ACM Press, 35–43.

MOTRO, A., BERLIN, J., AND ANOKHIN, P. 2004. Multiplex, Fusionplex, and Autoplex—Three generations of information integration. *SIGMOD Rec. 33*, 4, 51–57.

NAUMANN, F., BILKE, A., BLEIHOLDER, J., AND WEIS, M. 2006. Data fusion in three steps: Resolving schema, tuple, and value inconsistencies. *IEEE Data Eng. Bull. 29*, 2, 21–31.

NAUMANN, F., FREYTAG, J.-C., AND LESER, U. 2004. Completeness of integrated information sources. *Inf. Syst. 29*, 7, 583–615.

NODINE, M. H., FOWLER, J., AND PERRY, B. 1999. Active information gathering in InfoSleuth. In *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications (CODAS)*, 15–26.

ORDILLE, J. J. AND MILLER, B. P. 1993. Distributed active catalogs and meta-data caching in descriptive name services. In *Proceedings of the International Conference on Distributed Computing Systems*, 120–129.

PAPAKONSTANTINOU, Y., ABITEBOUL, S., AND GARCIA-MOLINA, H. 1996. Object fusion in mediator systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. Morgan Kaufmann, 413–424.

PARSONS, S. 1996. Current approaches to handling imperfect information in data and knowledge bases. *IEEE Trans. Knowl. Data Eng. 8*, 3 (Jun.), 353–372.

POPA, L., VELEGRAKIS, Y., MILLER, R. J., HERNÁNDEZ, M. A., AND FAGIN, R. 2002. Translating Web data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*.

RAHM, E. AND BERNSTEIN, P. A. 2001. On matching schemas automatically. Tech. Rep. MSR-TR-2001-17, Microsoft Research, Redmond, Washington. February.

RAHM, E. AND DO, H. H. 2000. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull. 23*, 4, 3–13.

RAJARAMAN, A. AND ULLMAN, J. D. 1996. Integrating information by outerjoins and full disjunctions (extended abstract). In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*. ACM Press, 238–248.

RAJINIKANTH, M., JAKOBSON, G., LAFOND, C., PAPP, W., AND PIATETSKY-SHAPIRO, G. 1990. Multiple database integration in CALIDA: Design and implementation. In *Proceedings of the International Conference on Systems Integration (ICSI)*. IEEE Press, 378–384.

RAMAN, V., CHOU, A., AND HELLERSTEIN, J. M. 1999. Scalable spreadsheets for interactive data analysis. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*.

RAMAN, V. AND HELLERSTEIN, J. M. 2001. Potter's Wheel: An interactive data cleaning system. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. Morgan Kaufmann, 381–390.

RAO, J., PIRAHESH, H., AND ZUZARTE, C. 2004. Canonical abstraction for outerjoin optimization. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, 671–682.

RECK, C. AND KÖNIG-RIES, B. 1997. An architecture for transparent access to semantically heterogeneous information sources. In *Proceedings of the International Workshop on Cooperative Information Agents (CIA)*. Springer, 260–271.

RUSINKIEWICZ, M., ELMASRI, R., CZEJDO, B., GEORGAKOPOULOS, D., KARABATIS, G., JAMOUSSI, A., LOA, K., AND LI, Y. 1989. Omnibase: Design and implementation of a multidatabase system. In *Proceedings of the 1st Annual Symposium in Parallel and Distributed Processing*, 162–169.

SARMA, A. D., BENJELLOUN, O., HALEVY, A. Y., AND WIDOM, J. 2006. Working models for uncertain data. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 7.

SATTLER, K., CONRAD, S., AND SAAKE, G. 2000. Adding conflict resolution features to a query language for database federations. In *Proceedings of the Workshop on Engineering Federated Information System (EFIS)*, M. Roantree et al., eds, 41–52.

SCANNAPIECO, M., VIRGILLITO, A., MARCHETTI, C., MECELLA, M., AND BALDONI, R. 2004. The DaQuinCIS architecture: A platform for exchanging and improving data quality in cooperative information systems. *Inf. Syst. 29*, 7, 551–582.

SCHALLEHN, E. AND SATTLER, K.-U. 2003. Using similarity-based operations for resolving data-level conflicts. In *Proceedings of the British National Conference on Databases (BNCOD)*, 172–189.

SCHALLEHN, E., SATTLER, K.-U., AND SAAKE, G. 2004. Efficient similarity-based operations for data integration. *Data Knowl. Eng. 48*, 3, 361–387.

SHETH, A. P. AND LARSON, J. A. 1990. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv. 22*, 3, 183–236.

SHIPMAN, D. W. 1981. The functional data model and the data languages DAPLEX. *Trans. Dat. Syst. 6*, 1, 140–173.

SHOENS, K. A., LUNIEWSKI, A., SCHWARZ, P. M., STAMOS, J. W., AND II, J. T. 1993. The Rufus system: Information organization for semi-structured data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, R. Agrawal et al., eds. Morgan Kaufmann, 97–107.

SINGH, M. P., CANNATA, P., HUHNS, M. N., JACOBS, N., KSIEZYK, T., ONG, K., SHETH, A. P., TOMLINSON, C., AND WOELK, D. 1997. The Carnot heterogeneous database project: Implemented applications. *Distrib. Parallel Databases 5*, 2, 207–225.

STAWORKO, S., CHOMICKI, J., AND MARCINKOWSKI, J. 2006. Preference-Driven querying of inconsistent relational databases. In *Proceedings of the International Workshop on Inconsistency and Incompleteness in Databases (IIDB)*.

SUBRAHMANIAN, V. S., ADALI, S., BRINK, A., EMERY, R., LU, J., RAJPUT, A., ROGERS, T., ROSS, R., AND WARD, C. 1995. Hermes: A heterogeneous reasoning and mediator system. Tech. Rep., University of Maryland.

TEMPLETON, M., BRILL, D., DAO, S., LUND, E., WARD, P., CHEN, A., AND MACGREGOR, R. 1987. Mermaid—A front-end to distributed heterogeneous databases. *Proc. IEEE 75*, 5 (May), 695– 708.

TOMASIC, A., AMOUROUX, R., BONNET, P., KAPITSKAIA, O., NAACKE, H., AND RASCHID, L. 1997. The distributed information search component (Disco) and the World Wide Web. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, 546–548.

TOMASIC, A., RASCHID, L., AND VALDURIEZ, P. 1998. Scaling access to heterogeneous data sources with Disco. *IEEE Trans. Knowl. Data Eng. 10*, 5, 808–823.

TSAI, P. S. M. AND CHEN, A. L. P. 2000. Partial natural outerjoin—An operation for interoperability in a multidatabase environment. *J. Inf. Sci. Eng. 16*, 4 (Jul.), 593–617.

TSENG, F. S.-C., CHEN, A. L. P., AND YANG, W.-P. 1993. Answering heterogeneous database queries with degrees of uncertainty. *Distrib. Parallel Databases 1*, 3, 281–302.

ULLMAN, J. D., GARCIA-MOLINA, H., AND WIDOM, J. 2001. *Database Systems: The Complete Book*. Prentice Hall PTR.

WANG, H. AND ZANIOLO, C. 2000. Using SQL to build new aggregates and extenders for object- relational systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, A. E. Abbadi et al., eds. Morgan Kaufmann, 166–175.

WEIS, M. AND NAUMANN, F. 2004. Detecting duplicate objects in XML documents. In *Proceedings of the International Workshop on Information Quality Informative Systems (IQIS)*.

WEIS, M. AND NAUMANN, F. 2005. DogmatiX tracks down duplicates in XML. In *Proceedings of the ACM International Conference on Management of Data SIGMOD*. ACM Press, New York, 431–442.

WIDOM, J. 2005. Trio: A system for integrated management of data, accuracy, and lineage. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 262–276.

WIEDERHOLD, G. 1992. Mediators in the architecture of future information systems. *Comput. 25*, 3 (Mar.), 38–49.

WIJSEN, J. 2003. Condensed representation of database repairs for consistent query answering. In *Proceedings of the International Conference on Database Theory (ICDT)*, 378–393.

YAN, L. L. AND ZSU, M. T. 1999. Conflict tolerant queries in AURORA. In *Proc. of CoopIS*. IEEE Computer Society, 279.

YERNENI, R., PAPAKONSTANTINOU, Y., ABITEBOUL, S., AND GARCIA-MOLINA, H. 1998. Fusion queries over Internet databases. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 57–71.

Received May 2007; revised September 2007; accepted December 2007