Adding Conflict Resolution Features to a Query Language for Database Federations*

Kai-Uwe Sattler[1], Stefan Conrad[2], and Gunter Saake[1]

[1]Department of Computer Science, University of Magdeburg
P.O. Box 4120, D-39016 Magdeburg, Germany
[2]Department of Computer Science, University of Munich
Oettingenstr. 67, D-80538 München, Germany

Abstract. A main problem of data integration is the treatment of conflicts caused by different modeling of real-world entities, different data models or simply by different representations of one and the same object. During the integration phase these conflicts have to be identified and resolved as part of the mapping between local and global schemata. Therefore, conflict resolution affects the definition of the integrated view as well as query transformation and evaluation. In this paper we present a SQL extension for defining and querying database federations. This language addresses in particular the resolution of integration conflicts by providing mechanisms for mapping attributes, restructuring relations as well as extended integration operations. Finally, the application of these resolution strategies is briefly explained by presenting a simple conflict resolution method.

1 Introduction

Nowadays integrating heterogeneous data sources is a significant challenge to the database community. The availability of numerous sources, ranging from legacy systems and enterprise databases to public Internet sources, increases the demand for tools and techniques integrating, condensing and abstracting data. Recently, several integration approaches were developed, particularly multidatabase systems [LA86,BHP92], mediators [Wie92] and federated database systems [SL90]. More or less these approaches are based on the idea of providing an integrated view on the sources. Defining this view is subject of schema integration. During the integration process the individual schemata are analyzed, the global schema is defined and finally the mapping between local and global schema is described. The mapping information provides the base for query processing. A global query is decomposed according to the mapping and translated into sub-queries for the individual sources. After local evaluation of the sub-queries, the sub-results have to be combined to the global result.

Independent from the direction of the schema integration process (bottom-up – integration of all the relevant data vs. top-down – integrating data for a given goal [Has99]) the designer has to resolve conflicts resulting from the heterogeneity of the participating data sources. Examples of conflicts are among others different identifiers for the

* This work was supported in part by the German Research Council (DFG): FOR 345/1-1.

same fact (e.g. entities or attributes), using different modeling concepts for representing real-world entities or conflicts, arising from overlapping of data. Resolving these conflicts is an important step in defining the mapping between local and global schema and therefore affects the query processing.

In this paper we present the query language FRAQL, a lightweight SQL extension for defining integrated object-relational schemata as well as formulating queries on them. This language is implemented as part of a query system for federated databases. The main contribution is the treatment of integration conflicts. The paper is organized as follows: After a brief survey on related work in Section 2, we introduce the query language and the underlying software architecture in Section 3. The resolution of conflicts with the help of the FRAQL language features is discussed in Section 4. In Section 5 we sketch basic principles of a method which we are currently developing for resolving conflicts as part of data integation. Finally, Section 6 concludes the paper and outlines future work.

2 Related Work

The more general problem of schema integration is addressed by several approaches [BLN86,PBE95]. For describing conflicts arising in the integration phase various classifications were developed, e.g. in [KS91,SCG93,SPD92].

Data models and query languages supporting the integration of heterogeneous sources are particularly multidatabase languages like MSQL [GLRS93], SQL/M [KGK+95] and SchemaSQL [LSS96]. Examples of system implementations are federated database systems like IRO-DB [GGF+96], Pegasus [ASD+91] or IBM DataJoiner [VZ98] as well as mediator-based systems like TSIMMIS [GPQ+97] or Information Manifold [LRO96]. MSQL provides basic features for accessing schema labels and converting them into data values. SQL/M addresses mainly description conflicts by providing mechanisms for scaling and unit transformation. More advanced conflict resolution is addressed for example by the restructuring techniques proposed in SchemaSQL, which support the specification of relations with data dependent output schemata.

Pegasus uses a functional object-oriented data manipulation language called HOSQL with non-procedural features, DataJoiner is based on DB2 and therefore provides essentially standard SQL features for conflict resolution. In mediator systems such as TSIMMIS the mediator is specified by a set of rules. Each rule maps a set of source objects into a virtual mediator object. In this way, conflicts are resolved by defining appropriated rules. The special problem of combining objects from different sources (object fusion) in mediators is addressed in [PAGM96].

Furthermore, structural conflicts and resolution strategies are discussed in detail in [KCGS95]. Techniques for managing schematic heterogeneity (meta conflicts) based on SchemaSQL features are presented in [Mil98]. Resolving description conflicts by using a rule-based data conversion language is described in [CDSS98], [MZ98] presents a schema-based data translation solution. In [Ken91] solving domain and schema mismatch problems with an object-oriented database language is discussed.

In [LCC99] an approach is proposed, where the origin of integrated data is included as an additional tuple attribute in order to improve the interpretation of global data. Another approach, presented in [SSR94], introduces the notion of semantic values enabling the interoperability of heterogeneous sources by representing context information.

3 FRAQL: An Overview

The objective of the FRAQL development is the investigation of techniques for query processing in loosely-coupled database federations. In this context several problems arise, which make query processing and optimization more difficult. These include: heterogeneity of data, missing or uncertain statistical information about data distribution and access paths, the limited query capabilities as well as non-predictable responses of the sources [IFF+99]. In the following we will focus on the aspect of resolving conflicts caused by heterogeneity of data.

FRAQL is a query language for object-relational database federations. It extends SQL by features for defining federations, accessing meta-data in queries, restructuring query results, and resolving integration conflicts. This is comparable with other multidatabase languages like MSQL or SchemaSQL, but in contrast to these FRAQL is extensible by user-defined data types and functions and it supports dynamic integration of new sources. With this features FRAQL could form the base for advanced data integration and fusion tasks [SS99]. In this context, FRAQL is not intended as an end user language, but an intermediate language for specifying the integrated views. Therefore, users can query the global integrated relations by means of usual SQL operations without knowledge of the FRAQL extensions.

In FRAQL a federation is a set of databases consisting of relations. A database can be provided by a full-featured DBMS or even by a Web source encapsulated by a wrapper [RS97,SH99]. This wrapper has to implement the query mechanisms which are not supported directly by the source. FRAQL is based on an object-relational data model: it supports the definition of object types and object tables derived from types. Using object-relational features simplifies the integration of post-relational data sources (e.g. ODBMS-based sources or XML datastores) and provides more advanced modeling concepts for schema definition.

Object types describe the structure of objects as sets of attributes and their domains. Types can be organized in a specialization hierarchy. *Object tables* represent global virtual relations of the federation, i.e. data from the sources are not materialized, except for caching purposes in order to speed up query evaluation. Here we distinguish between import and integration relations. An *import relation* is a projection of a local relation of a data source. The import relation is defined by specifying the origin (the identifier of the source) and, if required, a mapping between local and global attributes.

```
create type type_name under type_names ] (
  attrib_definition );

create table global_name of type_name
  as import from source.local_name
  [ mapping_definitions ];
```

An *integration relation* is a view on other global relations combined by using operators like union, θ-join and outer join. In addition, the standard SQL operations selection and projection are provided, too. An integration relation is defined as follows, where the term `table_expression` denotes a SQL view definition with extensions explained later.

```
create table global_name of type_name as table_expression;
```

Furthermore, FRAQL supports user-defined functions, which are stored in the database of the federation layer (i.e., in the query processing server) and are callable in queries. These functions are implemented in Java and registered in the query system. In order to be able to rewrite queries during optimization, two functions can be specified as inverse to each other.

Similar to SQL, the union and join operations can be refined by an `on` clause specifying the comparison attributes (for the `union` operator) resp. the comparison expression (for `join` operators), e.g.:

```
table1 union table2 on attr1, attr2
table1 join table2 on table1.attr1 = table2.attr1
```

Both the `union` and `join` operators can be applied with an additional `reconciled by` clause which denotes a user-defined function for conflict resolution (see Section 4):

```
table1 join table2 on attr1 = attr2 reconciled by func
```

Restructuring of relations is implemented in a way inspired by SchemaSQL. Variables of a query can not only be bound to relations as tuple variables, but also to meta-data, like the set of attributes of a relation or the set of relations of a schema. But in contrast to SchemaSQL, where meta-data access in queries is implemented as a language extension, in our approach the schema catalog is used. So, the catalog relation `catalog.columns` contains information about attributes of all global relations, whereas the relation `catalog.tables` describes the global relations. Naturally, any global user relation with information about other relations can be used as meta-data source, too.

As an extension to standard SQL, attributes of tuple variables in queries can be obtained during evaluation. This means, while in SQL names of attributes and relations are constants, in FRAQL they can be constructed from current values of other tuple attributes. This *variable substitution* is written in the notation `$var` and can appear everywhere in a query, where names of attributes or relations are expected. For example, the expression `tbl1.$(tbl2.col)` means the attribute of the current tuple of relation `tbl1`, whose name is obtained from the current value of `tbl2.col`. In the same way, a relation in the `from` clause or a query could be dynamically determined. The following query selects ISBN and title information from all relations implementing the object type `book`. So it is equivalent to a union of all these relations.

```
select t2.isbn, t2.title
from catalog.tables t1, $(t1.name) t2
where t1.type_name = 'book';
```

This technique enables a flexible transformation of schemata by view definition. But we have to take into consideration the effects on query optimization. Therefore, static optimization techniques, which create a complete query plan before beginning the evaluation, are inappropriated for queries containing variable substitutions. Better approaches should support runtime re-optimization at certain points of query processing [GW89,KD98].

FRAQL is implemented as part of a federated query system. This system consists of the following main components: the query parser, the decomposer and the global optimizer, the query evaluator, the Java VM for evaluating user-defined functions, and the catalog. The adapter layer contains the management component as well as the individual adapters providing a uniform access interface to the data sources. The interface to the adapters and the query processor itself are implemented using CORBA. Therefore, adapters can be plugged into the system at runtime. On top of the query interface we have developed a JDBC driver and an interactive query tool.

4 Conflict Resolution in FRAQL

In FRAQL an integrated schema is defined only by global relations which are views on local relations. Therefore, data integration and conflict resolution are parts of query processing, particularly view decomposition, query transformation and result composition. So the main issues for conflict resolution in FRAQL are:

– renaming of attributes as well as transformation of attribute values,
– integration operations like union and joins, and
– restructuring of relations by combining data and meta-data.

In the following we discuss the application of these techniques for description conflicts, semantic conflicts, and structural conflicts. These are three of the four conflict classes introduced in [SPD92]. We close this section by briefly considering the remaining class of heterogeneity conflicts.

Description Conflicts. First, we consider description conflicts. This kind of conflicts occurs, when the same real-word entity is modeled with different properties. In FRAQL we eliminate these differences by defining an import relation. Beginning with the object type describing the desired global properties, we specify how the local relation implements this type. Here the following rules apply to this mapping:

1. Each local attribute corresponding to an attribute defined by the global object type in terms of identifier and type becomes an attribute of the global relation.
2. The notation *g_name* **`is`** *l_name* means renaming the local attribute to *g_name*. This requires type compatibility.
3. The notation *g_name* **`is`** *func(l_name)* defines that the global attribute value is calculated by using the user-defined function *func* on the local attribute value.
4. The definition *g_name* **`is @`***tbl (l_name, src, dest, default)* means that the database table *tbl* is used for mapping the values from the local attribute *l_name*. This value of the global attribute is obtained by looking for the value of attribute *l_name* in column *src* and retrieving the corresponding value of column *dest*. The field *default* denotes a default value, either as literal or as local attribute, which is assigned to the global attribute, if the value of *l_name* is not found in the table.

5. The remaining local attributes are suppressed.
6. To all attributes of the global relation without a mapping the `NULL` value is assigned.

The next example demonstrates these concepts. First we define an object type `book`:

```
create type book (
  isbn varchar(20), title varchar(100), price float);
```

Furthermore we assume a local relation `buch` from the source `src`, which differs from the object type by the attribute names and the currency of the price attribute (DM instead of Dollar). Therefore, a function `dm2dollar` for converting the value is required which could be implemented by accessing a database table. Now we can define the global relation `german_books` as follows:

```
create table german_books of book as import from src.buch (
  title is titel, price is dm2dollar (preis));
```

Based on these definitions the global query

```
select title, isbn from german_books where price < 50;
```

is transformable into a local query. Beside renaming attributes this requires a transformation of the selection expression. Because user-defined functions are available only at the federation layer and not in the source itself, the selection operation has to be performed at global level or – for a constant expression – the expression has to be precomputed by applying the inverse function. For the given example this results in the following transformed query:

```
select titel, isbn from buch where preis < 93.60;
```

Of course, this requires that a function, e.g. `dollar2dm`, is being registered in the global query system as the inverse of `dm2dollar`.

Semantic conflicts. Semantic conflicts arise, when the relations overlap, which have to be integrated, i.e. there are tuples from both relations representing the same real-world object. First of all, this kind of conflicts can be resolved in FRAQL by applying the standard SQL union operation. However, the following problems remain:

– We have to decide, when two tuples from different relations represent the same real-world object (tuple identity).
– How to process tuples representing the same object but containing different values for the same attribute (data conflicts) ?

The problem of tuple identity is solved by specifying the attributes relevant for deciding equivalence, as illustrated in Fig. 1:

```
create table books of book as books1 union books2 on isbn;
```

The attributes have to identify the tuples of each relation uniquely, e.g. by using the primary key, in order to avoid duplicates.

Data conflicts are resolved in FRAQL with the help of user-defined reconciliation functions. A reconciliation function is called for each pair of tuples fulfilling the comparison condition. The affected tuples are passed as arguments to the function, the resulting tuple is inserted into the global relation.

isbn	*author*	*title*
382578	Williams, T.	Otherland
326523	Gibson, W.	Idoru

(a) *books1*

isbn	*author*	*title*
382578	Tad Williams	Otherland
276830	St. Lem	Solaris

(b) *books2*

Fig. 1. Tuple identity conflicts

isbn	*number*
3324524	2
6710767	2

(a) *bstore1*

isbn	*number*
3324524	1
1267894	2

(b) *bstore2*

isbn	*number*
3324524	3
1267894	2
6710767	2

(c) *stores*

Fig. 2. Resolving data conflicts

In the example from Fig. 2 we want to integrate two overlapping relations `bstore1` and `bstore2`. Both relations contain an attribute `number`. In the integrated relation `stores` this attribute should represent the sum of both values. Therefore we define this relation as follows:

```
create table stores of book as
  bstore1 union bstore2 on isbn reconciled by book_resolve;
```

The reconciliation function is implemented as a stored function in Java and registered in the query system:

```
create function book_resolve (book, book)
  returns book external name 'Books.resolver';
```

In this example the FRAQL type `book` is mapped to a Java type. However, in the current implementation we use generic objects for representing tuples.

Structural Conflicts. Representing a real-world aspect by different modeling concepts results in structural conflicts during integration. Depending on the variety of the data models several kinds of conflicts can occur. In the following we focus only on two special kinds: partitioning and meta conflicts. The resolution of other structural conflicts, particularly for the relational data model, is discussed for example in [KCGS95].

Partitioning occurs, when the relations which have to be integrated represent different aspects of the global relation, but still contain semantically equivalent attributes. This kind of conflicts is usually resolved by applying a θ-join or outer join operation. Like for the union operator a reconciliation function can be specified for resolving data conflicts.

Meta conflicts arise, when a concept is represented as data object in one schema, whereas it is modeled as schema object (attribute or relation) in another one.

In the example in Fig. 3 the database `db1` stores books and journals in a single relation `publication`, where each tuple contains a discriminating attribute `type` with

id	*title*	*type*
382660	Databases	book
017062	Computer	journal

(a) *db1.publication*

id	*title*
382660	Databases
361556	Java

(b) *db2.book*

id	*title*
017062	Computer
000102	CACM

(c) *db2.journal*

Fig. 3. Meta conflicts: relation vs. attribute

possible values `"book"` and `"journal"`. In database `db2` books and journals are stored in separate relations. Integrating both databases requires the adaptation of the relations from `db2` to the structure of `publication`. A straightforward solution would be a union of the relations `book` and `journal` with a constant value for the type attribute:

```
create table publication of publ_type as
  select id, title, "book" from db2.book
  union
  select id, title, "journal" from db2.journal;
```

In a more flexible approach the names of the relations in the `from` clause are determined from the schema catalog by using variable substitution:

```
create table publication of publ_type as
  select t2.id, t2.title, t1.name
  from catalog.tables t1, $(t1.name) t2
  where t1.schema = 'db2';
```

In this way no modifications of the global relation are required, when new relations (e.g. representing another publication type like reports) have to be added.

Without the language extension for variable substitution the above query could be evaluated by a dynamic SQL program as illustrated in the following pseudo code:

```
exec sql create table publication (proj_list);
exec sql select name from catalog.tables where schema = 'db2';
foreach tuple t
  qstr := 'insert into publication select id, title, ' ||
      t.name || ' from ' || t.name;
  exec sql prepare query from :qstr;
  exec sql execute query;
end
```

In contrast, in FRAQL this query is processed directly by the query evaluator.

In the second example (Fig. 4) the relation of database `db1` contains books with their prices and suppliers. For each supplier a book is represented by a separate tuple. In contrast, the relation of database `db2` contains the supplier prices as separate attributes. By using the schema catalog and variable substitution we are able to transform the relation of `db2` according to the relation of `db1`.

```
select b.isbn, b.$(a.name), a.name
from db2.book b, catalog.attributes a
where a.table = 'book' and a.name <> 'isbn';
```

isbn	*price*	*supplier*
38934237	59.00	meyers
38934237	48.00	jones

(a) *db1.book*

isbn	*meyers*	*jones*
38934237	59.00	48.00
22660513	27.00	29.00

(b) *db2.book*

Fig. 4. Meta conflicts: attributes vs. data value

Heterogeneity Conflicts. This kind of conflicts occurs if the data of the local data sources is represented in different data models. In FRAQL these conflicts are resolved mainly by the adapters which map the modeling concepts and hide system-dependent differences (e.g. SQL dialects).

5 A Method for Conflict Resolution

Based on the features discussed above we can develop a simple method for conflict resolution as part of data integration. It is not intended as a replacement for schema integration, rather it complements existing integration methods. In the following we sketch the basic ideas of such method.

We propose three steps reflecting the main concepts of FRAQL: object types, import relations and integration relations:

1. First, the global types have to be defined, either top-down or bottom-up by integrating the local types.
2. Next, we have to define the import relations. Here we resolve description conflicts by specifying the mappings of attributes.
3. In the last step we try to integrate import relations representing semantically equivalent real-world entities. In this context we have to consider two subtasks:
   – Resolving structural conflicts using standard SQL operations like projection, selection, renaming, the join operation with reconciliation functions as well as the restructuring mechanisms provided by FRAQL.
   – Resolving semantic conflicts by applying the union operator in conjunction with reconciliation functions.

In practice a lot of existing conflicts are not obvious from only considering the schemata of data sources to be integrated. Therefore, classical schema integration techniques must be enhanced by additional means to detect such conflicts. Due to the fact that data from the data sources is available at integration time we can directly use their data to validate decisions made in the integration process. In addition, conflicts can be detected which are only visible on the data layer. Therefore, the stepwise integration of data sources by means of a language like FRAQL seems to be essential for obtaining an integration result of high quality.

In particular, we expect the following kinds of contributions a stepwise integration using FRAQL can make to database integration:

- Following the (preliminary) results of schema integration FRAQL can be used to define corresponding queries (or views) describing the integrated relations. By applying such queries on real data from the data sources the FRAQL system can check whether there are additional conflicts on the data layer, e.g. not identical values for the same property of corresponding data objects in different data sources. By presenting all non-matching values (or, in case the number of conflicting values is too large, by presenting selected examples) the system can give important hints for resolving such conflicts of which we were not aware when integrating the schemata.
- Furthermore, we strive for enhancing the FRAQL system in such a way that the system can already propose possible ways for resolving such conflicts detected when computing a query (or view) describing an integrated relation. If the system proposes a conflict resolution it should directly generate a reconciliation function which can be used for that query (in the `resolved by` clause). Of course, such an automated generating of reconciliation functions is only possible in certain, restricted cases, e.g. if the conflict is due to a different scaling of numerical values in the data sources. Assuming that a large number of conflicts occuring in practice is of a kind for which a conflict detection and automated generation of conflict resolution functions is possible, this is an important and valueable help in carrying out an integration.

By means of this functionality we aim at supporting an integration method based on an *integration by example* principle. In particular, the third issue, i.e. giving the system examples of conflicting values for automated conflict resolution, will be an essential part in this method.

6 Conclusion

Resolving conflicts is an important step in integrating heterogeneous data sources. Here, differences at schema level as well as problems caused by different data representations have to be eliminated. In this paper we have discussed conflict resolution techniques as part of the query language FRAQL. Integration conflicts are resolved at global level by defining views on the imported relations. In addition to the well-known SQL operations our language provides more advanced mechanisms like renaming, value transformation as well as structural transformation. By using these mechanisms most of the integration conflicts are resolvable. Regarding conflict resolution we see our approach as an extension to DataJoiner and SchemaSQL. DataJoiner's query language SQL permits accessing and querying foreign databases without further resolution mechanisms. SchemaSQL contributes among other things features for restructuring relations and with it for resolving meta conflicts and FRAQL extends these by additional resolution techniques for description and structural conflicts.

FRAQL is currently being implemented in C++ as part of a query processing system for loosely-coupled database federations. Furthermore, we have implemented adaptors to the Oracle and MySQL DBMS. Apart from the JDBC driver and the query tool mentioned before, a graphical design und query workbench for interactive definition of integrated schemata is under development. Moreover, in the future we will examine dynamic optimization techniques for interleaving query planning and execution.

References

[ASD+91] R. Ahmed, P. De Smedt, W. Du, W. Kent, M.A. Ketabchi, W. Litwin, A. Rafii, and M.-C. Shan. The Pegasus Heterogeneous Multidatabase System. *IEEE Computer*, 24(12):19–27, December 1991.

[BHP92] M. W. Bright, A. R. Hurson, and Simin H. Pakzad. A Taxonomy and Curent Issues in Multidatabase Systems. *IEEE Computer*, 25(3):50–60, 1992.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[CDSS98] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your Mediators Need Data Conversion! In L.M. Haas and A. Tiwary, editors, *SIGMOD'98, Seattle, Washington, USA*, pages 177–188. ACM Press, 1998.

[GGF+96] G. Gardarin, S. Gannouni, B. Finance, P. Fankhauser, W. Klas, D. Pastre, R. Legoff, and A. Ramfos. IRO-DB — A Distributed System Federating Object and Relational Databases. In O. A. Bukhres and A. K. Elmagarmid, editors, *Object-Oriented Multidatabase Systems — A Solution for Advanced Applications*, chapter 20, pages 684–712. Prentice Hall, Eaglewoods Cliffs, NJ, 1996.

[GLRS93] J. Grant, W. Litwin, N. Roussopoulos, and T. Sellis. Query Languages for Relational Multidatabases. *VLDB Journal*, 2(2):153–171, 1993.

[GPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, March/April 1997.

[GW89] G. Graefe and K. Ward. Dynamic Query Evaluation Plans. In J. Clifford, B.G. Lindsay, and D. Maier, editors, *SIGMOD'89, Portland, Oregon, USA*, pages 358–366, 1989.

[Has99] W. Hasselbring. Top-Down vs. Bottom-Up Engineering of Federated Information Systems. In S. Conrad, W. Hasselbring, and G. Saake, editors, *EFIS'99, Kühlungsborn, Germany*, pages 131–138. infix-Verlag, Sankt Augustin, 1999.

[IFF+99] Z.G. Ives, D. Florescu, M.A. Friedman, A.Y. Levy, and D.S. Weld. An Adaptive Query Execution System for Data Integration. In A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors, *SIGMOD'99, Philadephia, Pennsylvania, USA*, pages 299–310. ACM Press, 1999.

[KCGS95] W. Kim, I. Choi, S. Gala, and M. Scheevel. On Resolving Schematic Heterogeneity in Multidatabase Systems. In W. Kim, editor, *Modern Database Systems*, chapter 26, pages 521–550. ACM Press, New York, NJ, 1995.

[KD98] N. Kabra and D.J. DeWitt. Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans. In L.M. Haas and A. Tiwary, editors, *SIGMOD'98, Seattle, Washington, USA*, pages 106–117, 1998.

[Ken91] W. Kent. A Rigorous Model of Object Reference, Identity, and Existence. *Journal of Object-Oriented Programming*, pages 28–36, June 1991.

[KGK+95] W. Kelley, S. Gala, W. Kim, T. Reyes, and B. Graham. Schema Architecture of the UniSQL/M Multidatabase System. In W. Kim, editor, *Modern Database Systems*, chapter 30, pages 621–648. ACM Press, New York, NJ, 1995.

[KS91] W. Kim and J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer*, 24(12):12–18, December 1991.

[LA86] W. Litwin and A. Abdellatif. Multidatabase interoperability. *IEEE Computer*, 19(12):10–18, December 1986.

[LCC99] E.-P. Lim, R.H.L. Chiang, and Y. Cao. Tuple source relational model: A source-aware data model for multidatabases. *Data & Knowledge Engineering*, 29(1):83–114, January 1999.

[LRO96] A.Y. Levy, A. Rajaraman, and J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In T.M. Vijayaraman, A.P. Buchmann, C. Mohan, and N.L. Sarda, editors, *VLDB'96, Mumbai (Bombay), India*, pages 251–262. Morgan Kaufmann, 1996.

[LSS96] L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. SchemaSQL - A Language for Interoperability in Relational Multi-Database Systems. In T. M. Vijayaraman, A.P. Buchmann, C. Mohan, and N.L. Sarda, editors, *VLDB'96, Mumbai (Bombay), India*, pages 239–250. Morgan Kaufmann, 1996.

[Mil98] R. J. Miller. Using Schematically Heterogeneous Structures. In L.M. Haas and A. Tiwary, editors, *SIGMOD'98, Seattle, Washington, USA*, pages 189–200, 1998.

[MZ98] T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, New York City, New York, USA*, pages 122–133. Morgan Kaufmann, 1998.

[PAGM96] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In T.M. Vijayaraman, A.P. Buchmann, C. Mohan, and N.L. Sarda, editors, *VLDB'96, Mumbai (Bombay), India*, pages 413–424. Morgan Kaufmann, 1996.

[PBE95] E. Pitoura, O. Bukhres, and A. K. Elmagarmid. Object Orientation in Multidatabase Systems. *ACM Computing Surveys*, 27(2):141–195, June 1995.

[RS97] M.T. Roth and P.M. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In M. Jarke, M.J. Carey, K.R. Dittrich, F.H. Lochovsky, P. Loucopoulos, and M.A. Jeusfeld, editors, *VLDB'97, Athens, Greece*, pages 266–275. Morgan Kaufmann, 1997.

[SCG93] F. Saltor, M. Castellanos, and M. Garcia-Solaco. Overcoming Schematic Discrepancies in Interoperable Databases. In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, *Interoperable Database Systems, Proc. of the IFIP WG 2.6 Database Semantics Conf., DS-5, Lorne, Victoria, Australia, November, 1992*, pages 191–205, Amsterdam, 1993. North-Holland.

[SH99] K. Sattler and M. Höding. Adapter Generation for Extraction and Querying Data from Web Sources. In *Proc. of 2nd ACM SIGMOD Workshop WebDB'99*, 1999. http://www-rocq.inria.fr/~cluet/webdb99.html.

[SL90] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SPD92] S. Spaccapietra, C. Parent, and Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemas. *The VLDB Journal*, 1(1):81–126, July 1992.

[SS99] K. Sattler and G. Saake. Supporting Information Fusion with Federated Database Technologies. In S. Conrad, W. Hasselbring, and G. Saake, editors, *EFIS'99, Kühlungsborn, Germany*, pages 179–184. infix-Verlag, Sankt Augustin, 1999.

[SSR94] E. Sciore, M. Siegel, and A. Rosenthal. Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. *ACM Transactions on Database Systems*, 19(2):254–290, June 1994.

[VZ98] S. Venkataraman and T. Zhang. Heterogeneous Database Query Optimization in DB2 Universal DataJoiner. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, New York City, New York, USA*, pages 685–689. Morgan Kaufmann, 1998.

[Wie92] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, March 1992.