Classifying Schematic and Data Heterogeneity in Multidatabase Systems

Won Kim and Jungyun Seo,* UniSQL, Inc.

Schema and data conflicts between component databases are a crucial problem in building multidatabase systems. This article presents a comprehensive framework for classifying these conflicts.

The proliferation of file systems, navigational database systems (hierarchical and network), and relational database systems during the past three decades has created difficult problems arising from the need to access heterogeneous files and databases through a single data definition and query language designed under a single data model. This has been the primary motivation for research into multidatabase systems[1-6] during the past 15 years. An MDBS is a federation of independently developed component database systems (CDBSs). The MDBS provides a homogenizing layer on top of the CDBSs, thus giving users the illusion of a homogeneous system. Because CDBSs operate independently (that is, without central control or distributed coordination), the component databases (CDBs) may include structural and representational discrepancies, or conflicts, called schematic and data heterogeneity. These conflicts must be resolved (homogenized) so MDBS users can access the underlying CDBs with a single uniform database language rather than a different database language for each CDB.

Schematic and data heterogeneity is a crucial problem in building and using an MDBS. Surprisingly then, to the best of our knowledge, no framework exists for comprehensive enumeration and systematic classification of MDBS conflicts. There have been various proposals for MDBS data modeling and even database languages.[1-4,7-9] However, these proposals are based on fairly high-level and intuitive understanding rather than detailed classification. One reason why MDBSs have failed to win marketplace acceptance is the absence of a comprehensive framework for understanding schematic and data heterogeneity among independently created and administered CDBs. Another is the absence of a practical solution for homogenizing this heterogeneity.

Here we develop a complete framework for enumerating and classifying the types of MDBS structural and representational discrepancies. We believe such a framework is prerequisite to developing an MDBS schema definition and query language as well as tools for aiding multidatabase designers. The proposed frame-

*Since collaborating on this article, Jungyun Seo has joined the Computer Science Department at the Korea Advanced Institute of Science and Technology in Daejon.

0018-9162/91/1200-0012$01.00 © 1991 IEEE

Table 1. Library schemas in component databases.

Library	Table name	Attributes	General description
CDB1: Main (Main Library)	item	(i#*, title, author-name, subject, type, language)	Library items
	lc-num	(i#*, c-letter, f-digit, s-digit, cuttering)	Library of Congress number
	publisher	(i#*, name, tel, street, city, zip, state, country)	Publishers
	lend-info	(i#*, lend-period, library-use-only, checked-out)	Lending information
	checkout-info	(i#*, id-num, hour, day, month, year)	Borrower and due date
CDB2: Engineering (Engineering Library)	items	(i#*, title, a-name, type, c-letter, f-digit, s-digit, cuttering)	Library items
	item-subject	(i#*, subject)	Subject of each item
	item-language	(id#*, language)	Language used in each item
	publisher	(i#*, p-name, str-num, str-name, city, zip, state)	Publishers
	lend-info	(i#*, lend-period, library-use-only, checked-out)	Lending information
	checkout-info	(i#*, id-num, hour, day, month, year)	Borrower and due date
CDB3: City (City Public Library)	books	(i#*, lc-num, name, title, subject)	Library items
	publisher	(i#*, p-name, p-address)	Publishers
	lend-info	(i#*, l-period, reference, checked-out)	Lending information
	checkout-info	(i#*, dl-num, day, month, year)	Borrower and due date
CDB4: Comm (Community College Library)	item	(i#*, lc-number, title, a-name)	Library items
	publisher-info	(i#*, p#*, name, tel)	Publishers
	publisher-add	(p#*, st-num, st-name, room-num, city, state, zip)	Publisher address
	checkout-info	(i#*, id, day, month, year)	Borrower and due date
	lc-num	(i#*, category, user-name)	Library card number

*Indicates key attribute.

work is structured according to a relational database schema. It is both practical and complete. It was used to build the UniSQL/M commercial multidatabase system from UniSQL, Inc. This MDBS was built over Structured Query Language-based relational database systems and a unified relational and object-oriented database system named UniSQL/X. Its utility has been verified in this context.

We assume that the MDBS common data model is the relational model; that is, each of the CDB schemas is first converted to a semantically equivalent relational schema, and the multidatabase schema is constructed as a view of these relational CDB schemas. However, our results are substantially applicable to heterogeneous database systems that use a nonrelational data model (for example, an object-oriented data model) as the common data model and allow the formulation of queries directly against the CDB schemas (for example, see Litwin et al.[4]).

Example multidatabase scenario

Let us assume that the main library of a university wishes to integrate the databases of various libraries, such as the university's engineering library, a city public library, and the library of a community college. The database of each library has the schema definition shown in Table 1. Each database has a tag name, CDB$_i$($1 \leq i \leq 4$).

Although all these databases are designed for a library, they are independently designed and maintained and therefore differ in many ways. For example, the table name for general information about library items is *item* in CDB1 and CDB4, *items* in CDB2, and *books* in CDB3. Similarly, the table name for information about the publisher is *publisher* in CDB1, CDB2, and CDB3,

Conflicts in a multidatabase system

The schematic and data conflicts in an MDBS environment are mostly due to different symbolic representations for concepts in the component database systems. Since a database is defined by its schema and data, we classify conflicts at the highest level as either schema or data conflicts. Schema conflicts result from the use of different schema definitions in different CDBs. Data conflicts are due to inconsistent data in the absence of schema conflicts. Figure 1 summarizes our framework for the enumeration and classification of MDBS schema and data conflicts.

There are two basic causes of schema conflicts. First is the use of different structures (tables and attributes) for the same information. For example, some CDBs may represent a publisher's address as an attribute, while others represent it in tables. Second is the use of different specifications for the same structure; these include different names, data types, and constraints for semantically equivalent tables and/or attributes. Because the relational model uses either tables or attributes to represent information, we can classify schema conflicts within this model completely by enumerating combinations of different structures used to represent information and all possible specifications of the structures.

Table-versus-attribute conflicts result from the use of tables in some CDBs and attributes in others to represent the same information. Many-to-many table conflicts and many-to-many attribute conflicts are due to different numbers of tables or attributes representing the same information. Table-structure conflicts arise when semantically equivalent tables have different structures (that is, different numbers and/or kinds of attributes). When all CDBs use the same structure for the same information, all user-definable elements within the structure can be enumerated and all conflicts classified as either one-to-one table conflicts or one-to-one attribute conflicts.

Our classification of data conflicts enumerates possible sources of conflict. Broadly, there are two types of data conflicts: (1) wrong data conflicts violate integrity constraints implicit in or explicitly specified in data and (2) conflicts based on different representations for the same data address all conflicts unrelated to integrity constraints. The latter type includes the case of same representation for different data and can be further decomposed by enumerating ways of representing the same data using different scalar values for each data type (for example, integer, real, string, date, and monetary unit).

while CDB4 uses *publisher-info*. Further, the item tables in CDB1 and CDB4 have different arities and different attributes: there are six attributes in CDB1, but four in CDB4. CDB1 stores the Library of Congress number for each item in a separate table, lc-num; in CDB4, it is an attribute, *lc-number*.

Further, some attributes have different names. For example, the attribute for the author's name is called *author-name* in CDB1, *a-name* in CDB2 and CDB4, and *name* in CDB3. The meanings of attributes with the same name also differ. The checkout-info table in CDB4 contains information about the checkout date rather than the due date as in the other CDBs. Therefore, the meaning of the attributes *day*, *month*, and *year* in CDB4 differs from their meanings in the other CDBs.

As these examples show, CDB users can express the same concept in different ways. Semantically equivalent tables or attributes can have different names, structures, and data types. This makes it impossible for a query language such as SQL, designed for a single database, to manipulate data in different CDBs. Database languages must be extended so that users can define an MDBS schema and formulate one query for *n* databases rather than *n* queries, one for each database. MSQL[4] is an early proposal for such a database language.

I. Schema Conflicts
- A. Table-versus-table conflicts
  - 1. One-to-one table conflicts
    - a. Table name conflicts
      - 1) Different names for equivalent tables
      - 2) Same name for different tables
    - b. Table structure conflicts
      - 1) Missing attributes
      - 2) Missing but implicit attributes
    - c. Table constraint conflicts
  - 2. Many-to-many table conflicts
- B. Attribute-versus-attribute conflicts
  - 1. One-to-one attribute conflicts
    - a. Attribute name conflicts
      - 1) Different names for equivalent attributes
      - 2) Same name for different attributes
    - b. Default value conflicts
    - c. Attribute constraint conflicts
      - 1) Data type conflicts
      - 2) Attribute integrity-constraint conflicts
  - 2. Many-to-many attribute conflicts
- C. Table-versus-attribute conflicts

II. Data Conflicts
- A. Wrong data
  - 1. Incorrect-entry data
  - 2. Obsolete data
- B. Different representations for the same data (Same representation for different data)
  - 1. Different expressions
  - 2. Different units
  - 3. Different precisions

Figure 1. Schema and data conflict classification.

Schema conflicts

Because we assume that all CDBs are in the relational model, we use the data definition facility of the American National Standard Institute SQL syntax[10] to enumerate all items that can be defined differently and are therefore candidates for conflicts. Here, rather than enumerating all possible cases of conflicts, we classify the cases.

The sidebar presents an overview of our classification of MDBS schema and data conflicts, and Figure 1 outlines it. The following sections provide more detailed descriptions of the classification elements.

Table-versus-table conflicts. These conflicts occur when different CDBs use different definitions to represent information in tables (for example, different names, structures, or constraints on the tables). Table-versus-table conflicts can be decomposed into one-to-one table conflicts and many-to-many table conflicts. (We consider one-to-many table conflicts a special case of many-to-many table conflicts.

```
CREATE TABLE item
  (title ...., author-name ...., subject ....,type ...., language ....,
    CHECK (language = "english" OR type = "book")
    CHECK (title is NOT NULL))
```

Figure 2. Table definition with two Check conditions.

One-to-one table conflicts. These conflicts can occur when CDBs represent the same information in single tables using different names, structures, and constraints. We can enumerate all cases of this conflict type by enumerating all user-definable items in the SQL language table definitions. Thus, one-to-one table conflicts are further decomposed into table name conflicts, table structure conflicts, and table constraint conflicts:

- *Table name conflicts.* These conflicts arise due to different names assigned to tables in different CDBs. There are two types: conflicts due to the use of different names for semantically equivalent tables and conflicts due to the use of the same name for semantically different tables. The library scenario included examples of both cases.
- *Table structure conflicts.* These conflicts arise from differences in the number of attributes in CDB tables, that is, when a table in one CDB is missing some attributes in a corresponding table in another CDB. A CDB table is not union-compatible with corresponding tables in other CDBs if it is missing some attributes. There are two interpretations for missing attributes: The attributes are indeed missing, or the missing attributes are implicit and can thus be deduced. For example, since there are different types of library items (such as books, cartographic material, film, and sound recordings), the item tables in CDB1 and CDB2 have an attribute *type* to specify the type of a library item. However, since all items in CDB4 are books, the item table in CDB4 does not need an attribute for the type of items; that is, the implicit attribute *type* has the default value *book*.
- *Table constraint conflicts.* These conflicts arise from differences in the specifications of table constraints. SQL provides four alternatives specifications: *candidate key definition*, *primary key definition*, *foreign key definition*, and *check condition*. Thus, this conflict type includes four subcategories. Unlike other constraints, which cause difficulties in the formulation of queries or in the definition of views involving multiple CDBs, constraint conflicts (including attribute constraint conflicts, which are discussed later) can cause difficulties with simultaneous updates to multiple CDBs. For example, if an attribute is a key attribute in one CDB, but the corresponding attribute in another CDB is not key, it is difficult to impose the key constraint on the attribute at the MDBS level.

Any of the three key definition constraint types (candidate, primary, or foreign) is defined on one or a combination of table attributes. Even if the key constraint is defined on just one table attribute, it applies collectively to all records of the table. Therefore, we simply classify key constraints as table-level conflicts. The check condition can also be specified either as one or a set of attributes. However, we regard check conditions that are defined on a single attribute as attribute constraint conflicts. For example, consider the definition in Figure 2 of the item table with two check conditions. The first check clause, referring to multiple attributes, is a table-level constraint. However, the second check clause, for the attribute *item.title*, applies to only a single attribute.

Many-to-many table conflicts. These conflicts occur when CDBs use different numbers of tables to represent the same information. Since CDB users may define tables in different ways for various reasons — to remove redundant data, to reduce the possibility of inconsistencies when deleting and inserting records, etc. — this type of conflict can occur frequently in an MDBS. For example, in our library scenario, CDB1 uses two tables, item and lc-num, for general information on each library item, while CDB2 uses three tables: items, subject, and language.

Some conflicts may combine many-to-many table conflicts with subcategories of one-to-one table conflicts (that is, table name conflicts, table structure conflicts, and table constraint conflicts). However, separate categories are not required for such compound conflicts because they can be decomposed into basic conflicts and handled with the corresponding tools (see "Compound conflicts" sidebar on next page).

Attribute-versus-attribute conflicts. These conflicts are caused by different definitions for semantically equivalent attributes in different CDBs, including different names, attribute data types, and integrity constraints. Like the table-versus-table conflicts, these conflicts can be decomposed into one-to-one and many-to-many conflicts.

One-to-one attribute conflicts. These are due to different definitions for semantically equivalent attributes in different tables. The attribute definition consists of the attribute name, data type, constraints, and a default value. Since the data-type specification for an attribute is a special case of its constraint definition, we decompose one-to-one attribute conflicts into attribute name conflicts, default value conflicts, and attribute constraint conflicts.

- *Attribute name conflicts.* Attribute name conflicts are similar to the table name conflicts discussed earlier. There are two types: one arising from the use of different names for semantically equivalent attributes in different CDBs and the other arising from the use of the same name for semantically different attributes. (Again, the library schema includes examples of these cases.) The latter type is often caused by the use of incompletely specified names. For example, one CDB uses an attribute name *salary* for gross salary, and another CDB uses the same name for net salary. Similarly, the attribute name *price* may represent the price including tax in one CDB and the price before tax in other CDBs.

• *Default value conflicts.* This conflict type, like constraint conflicts, can manifest itself during update. For example, suppose that the default value of the attribute *subject* in the CDB2 item table is "applied-science," while it is null in CDB1. If "applied science" is chosen as the default in the MDBS view, the addition of an "applied science" book to CDB1, without an explicit specification for the value of the subject attribute, may result in a record with a null value (this conflict can be prevented).

• *Attribute constraint conflicts.* These conflicts are further decomposed into data type and attribute integrity-constraint conflicts. *Data type conflicts* occur when semantically equivalent attributes in different CDBs have different data types. For example, the data type of the attribute *zip* in the CDB1 publisher table is defined as Char(5), while that of the corresponding attribute in CDB2 is defined as Integer. *Attribute integrity-constraint conflicts* are similar to default value conflicts. Any conflict due to different definitions of attribute integrity constraints defined by a Check clause is classified as an attribute constraint conflict. For example, suppose that the attribute *f-digit* of the CDB1 ls-num table is defined as an integer ≤ 9999, while the corresponding attribute in CDB2 is defined as an integer ≤ 999. Then, depending on which constraint is adopted in the MDBS view, an attempt to add a record with an f-digit value of 2120 to both CDBs simultaneously may or may not succeed in CDB2.

Many-to-many attribute conflicts. An example of this conflict type occurs in the library scenario where the street addresses of publishers are represented in CDB1 in one attribute, *street*, in the publisher table, while the same information is represented in two attributes, *str-num* and *str-name*, in the CDB2 publisher table.

As remarked for many-to-many table conflicts, these conflicts may combine many-to-many attribute conflicts with subcategories of one-to-one attribute conflicts (that is, attribute name conflicts, default value conflicts, and attribute constraint conflicts). Again, there is no need for separate categories for such compound conflicts; they can be decomposed into several types of basic conflicts.

Table-versus-attribute conflicts. These conflicts occur if some CDBs use tables and others use attributes to represent the same information. In the library scenario, CDB3 uses an attribute, p-address, in the publisher table to represent the publisher's address, while CDB4 represents the same information in the publisher-add table. This conflict type can be regarded as a combination of many-to-many table conflicts and many-to-many attribute conflicts. It is an example of a table-versus-attribute conflict.

Data conflicts

Thus far, our discussion has focused on schema conflicts. In this section, we examine the two types of data conflicts

Compound conflicts

Interesting compound conflicts occur in practice. We regard them as combinations of different conflict types in our classification schema. This approach makes it possible to classify arbitrarily complex conflicts. For example, suppose that we add to our library scenario another library database as follows:

CDB5: Art (Art-Library)

item(i#, title, author-name, subject, type, language)	Library items
lc-num(i#, category, number, cuttering)	Library of Congress number

Then a query such as "Select the titles, Library of Congress numbers, subjects, and, if available, languages of all library items from CDB2, CDB3, and CDB5" will need to overcome several different conflict types. A table-versus-attribute conflict occurs because CDB3 represents the Library of Congress number as an attribute, lc-num, while CDB5 represents it as a table, lc-num. A many-to-many attribute conflict occurs because CDB5 uses three attributes — category, number, and cuttering — to represent the Library of Congress number; CDB2 uses four attributes — c-letter, f-digit, s-digit, and cuttering; and CDB3 uses one attribute, lc-num.

A many-to-many table conflict occurs because CDB2 uses three tables for library items, while CDB5 uses two tables and CDB3 uses only one. A table name conflict, table structure conflict, and attribute name conflict also occur.

A potentially troublesome situation arises when CDBs contain similar, but not identical, data. For example, consider an MDBS that consists of CDB20 and CDB21 as follows:

CDB20: restaurant(Rest-name, type, city)
CDB21: restaurant(Rest-name, type, county)

Handling a query such as "Select the names of all French restaurants in Travis County from CDB20 and CDB21" is not trivial. A city and a county may have an inclusion relation (that is, city < county), so the answer to the query must include French restaurants in Travis County cities such as Austin, Bastrop, and Westlake. This is different from conflicts related to the semantics of CDB symbols. Clearly, the meanings of data for the attributes city and county are different. Because the restaurant tables have different attributes, this conflict is, in a sense, a table structure conflict between CDB20 and CDB21. On the other hand, to classify this conflict as simply one type of table structure conflict does not take into account information implicit in the related attributes. The problem inherent in this situation is the existence of semantic relationships among attributes in CDBs. However, we do not believe that such relationships should be regarded as MDBS conflicts.

listed in Figure 1: wrong data and different representations.

Wrong data. One type of wrong data conflict is due generally to failures in maintaining a database, such as failing to keep the database up to date and to enforce integrity constraints. We call this type *incorrect-entry data*. It occurs when equivalent attributes in different CDBs, which are expected to have the same value, have different values. For example, if the birth date of an employee in one CDB is different from that of the same person in other CDB, an MDBS query for the person's birth date will return a wrong answer. (We assume that the "same data" in different CDBs are identifiable through user-defined key attributes.)

Another type of wrong data is *obsolete data*. For example, two CDBs of a company might have different salary data for the same employee because one salary has not been updated. Another interpretation of this conflict is that one salary is for one job the employee performs and the other salary is for a different job performed by the same employee. However, we regard this possibility as an attribute name conflict.

Different representations for the same data. There are three different aspects to the representation of data across CDBs: expressions, units, and precision.

Different expressions. Conflicts in *expressions* arise between two CDBs when both use the same type of data or different types of data. The following examples show various expressions for the same data:

- Different words for the same data: Texas, TX, Tx
- Different strings for the same data:
  9390 Research Blvd. Ste. 220, Austin, TX
  9390 Research Boulevard, Suite #220, Austin, Texas
- Different codes for the same data:
  *****, A, Excellent, 1, 5
  ****, B, Good, 2, 4
  ***, C, Fair, 3, 3
  **, D, Poor, 4, 2
  *, E, Bad, 5, 1

The third example illustrates the conflict that occurs when different types of values are used in CDBs for the same data; for example, where some CDBs use strings, while others use integers.

Different expressions, units, and precision result in conflicting representations of the same data across CDBs.

Different units. These conflicts arise when two CDBs use different units for numeric data. Different units give different meanings to numeric values. For example, suppose that CDB1 expresses the lend-period attribute in the lend-info table in terms of days (for example, 1 to 28 days) and CDB2 expresses the same attribute in weeks (for example, 1 to 4 weeks). Then even if the CDBs have the same number, say 2, as the attribute value, the meaning of the numbers is different in each CDB.

In a sense, this conflict type can be regarded as an attribute name conflict. Thus, if a fully qualified name is used for each attribute (such as lend-period-in-days or lend-period-in-weeks), the attributes in different units can be regarded as distinct attributes. However, we regard attributes in different units as carrying semantically equivalent information.

Different precisions. Conflicts in precision occur when two CDBs use values from the domains of different cardinalities for the same data. For example, suppose that the data type of an attribute *weight-of-ship* in CDB10 is defined as {heavy, middle, light, ultra-light}, while that of the same attribute in CDB11 is defined as Integer with tons as the unit. Then, the domain size of the value for weight-of-ship in CDB10 is four; while that of CDB11 may be a million (that is, any integer between 1 and 1,000,000).

We note that when different CDBs use different values from domains with the same cardinalities, they are in expressions conflict rather than precisions conflict, as in the third example under "Different expressions."

This framework provides for comprehensive enumeration and classification of schema and data conflicts among component relational databases organized into a multidatabase system. We derived the classification of schema conflicts from an enumeration of all possible conflict sources based on the ANSI-SQL relational database language. We developed the classification of data conflicts by examining the sources of different representations for the same data.

Our approach to addressing schematic heterogeneity is to define views on the schemas of more than one component database and to formulate queries against the views. The view definition can specify how to homogenize the schematic heterogeneity in CDBS views. Our approach to data heterogeneity is twofold: First, we allow the MDBS query processor to issue warnings when it detects wrong-data conflicts in query results. Second, we allow the MDBS users and/or database administrator to prepare and register lookup tables in the database so that the MDBS query processor can match different representations of the same data.

We note that fully satisfactory solutions to many-to-many table conflicts require significant additional research. It is often possible to reduce many-to-many table conflicts to one-to-one table conflicts by converting multiple tables to a single table in each CDB. However, in general, the conversion of a set of tables in one CDB into a semantically equivalent set in another CDB is a very involved process. It requires a set of standard relational and conflict-homogenizing operations at both the table and attribute level. Even then, it introduces problems of incomplete information and null values in the CDBs.[11]

Our classification framework assumes that the schemas of all underlying component databases have been converted to equivalent schemas in the standard relational data model. A more natural and powerful common model is the object-oriented data model, which includes such concepts as generalization, aggregation, inheritance, and methods. The development of an MDBS based on an object-oriented data model is an exciting new research area. ■

Acknowledgments

We thank the anonymous referees who offered many insightful suggestions that helped us improve the technical accuracy and presentation of this article.

References

1. Y. Breitbart, P.L. Olson, and G.R. Thompson, "Database Integration in a Distributed Heterogeneous Database System," *Proc. Second IEEE Data Eng. Conf.*, CS Press, Los Alamitos, Calif., Order No. 655, 1986, pp. 301-310.

2. U. Dayal and H. Hwang, "View Definition and Generalization for Database Integration in a Multidatabase System," *IEEE Trans. Software Eng.*, Vol. SE-10, No. 6, Nov. 1984, pp. 628-645.

3. T. Landers and R.L. Rosenberg, "An Overview of Multibase," in *Distributed Databases*, H.J. Schneider, ed., North Holland, The Netherlands, 1982, pp. 153-184.

4. W. Litwin et al., "MSQL: A Multidatabase Language," *Information Sciences*, Vol. 49, June 1987.

5. W. Litwin, L. Mark, and N. Roussopoulos, "Interoperability of Multiple Autonomous Databases," *ACM Computing Surveys*, Vol. 22, No. 3, Sept. 1990, pp. 267-293.

6. Y.R. Wang and S.E. Madnick, "A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective," *Proc. 16th VLDB Conf.*, Morgan Kaufman, Palo Alto, Calif., 1990, pp. 519-538.

7. C. Batini, M. Lenzerini, and S. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys*, Vol. 18, No. 4, Dec. 1986, pp. 323-364.

8. L. DeMichiel, "Performing Operations over Mismatched Domains," *Proc. Fifth IEEE Data Eng. Conf.*, CS Press, Los Alamitos, Calif., Order No. 1915, 1989, pp. 36-45.

9. S. Hayne and S. Ram. "Multi-User View Integration System (MUVIS): An Expert System for View Integration," *Proc. Sixth IEEE Data Eng. Conf.*, CS Press, Los Alamitos, Calif., Order No. 2025, Feb. 1990, pp. 402-409.

10. American National Standards Institute, "Database Language — SQL with Integrity Enhancement," Tech. Report X3.135-1989, ANSI, New York, 1989.

11. T. Imielinski and W. Lipski, "Incomplete Information in Relational Databases," *J. ACM*, Vol. 31, No. 4, Oct. 1984, pp. 761-791.

Won Kim is the founder and president of UniSQL, Inc., a database corporation in Austin, Texas. Before founding that company, he was director of the Object-Oriented and Distributed Systems Laboratory at Microelectronics and Computer Technology Corp., where he was the chief architect of the Orion series of object-oriented database systems.

Kim received a PhD in computer science from the University of Illinois at Urbana-Champaign. His dissertation was on query processing in relational database systems. He is the author of *Introduction to Object-Oriented Databases* (MIT Press, 1990). He currently chairs ACM SIGMOD and serves on the editorial boards of *ACM Transactions on Database Systems*, *Communications of the ACM*, *IEEE Transactions on Knowledge and Data Engineering*, and *IEEE Data Engineering Bulletin*.

Jungyun Seo is an assistant professor in the Computer Science Department at the Korea Advanced Institute of Science and Technology in Daejon. Previously, he was a member of technical staff at UniSQL, Inc., Austin, Texas. His research interests include natural language processing, artificial intelligence, and knowledge-based semantic analysis in distributed query processing.

Seo obtained a BA in mathematics from Sogang University in Seoul, Korea, and an MS and PhD in computer science from the University of Texas, Austin.

Readers can contact Albert D'Andrea, UniSQL, Inc., 9390 Research Blvd., Austin, TX 78759, e-mail execu!sequoia!unisql!albert@cs.utexas.edu.