August 2004

Living with inconsistencies in a multidatabase system

J. R. Getta
University of Wollongong, jrg@uow.edu.au

Living with inconsistencies in a multidatabase system

Abstract

Integration of autonomous sources of information is one of the most important problems in implementation of the global information systems. This paper considers multidatabase systems as one of the typical architectures of global information services and addresses a problem of storing and processing inconsistent information in such systems. A new data model proposed in the paper separates sure from inconsistent information and introduces a system of elementary operations on the containers with sure and inconsistent information. A review of the implementation aspects in an environment of a typical relational database management system concludes the paper.

Disciplines

Physical Sciences and Mathematics

Publication Details

This article was originally published as: Getta, JR, Living with inconsistencies in a multidatabase system, Proceedings 15th International Workshop on Database and Expert Systems Applications, 30 August-3 September 2004, 905-909. Copyright IEEE 2004.

Living with Inconsistencies in a Multidatabase System

Janusz R. Getta
School of Information Technology and Computer Science
University of Wollongong
Wollongong, NSW 2522, Australia
jrg@uow.edu.au

Abstract

Integration of autonomous sources of information is one of the most important problems in implementation of the global information systems. This work considers multidatabase systems as one of the typical architectures of global information services and addresses a problem of storing and processing inconsistent information in such systems. A new data model proposed in the paper separates sure from inconsistent information and introduces a system of elementary operations on the containers with sure and inconsistent information. A review of the implementation aspects in an environment of a typical relational database management system concludes the paper.

1. Introduction

The integration of a large number of independent and heterogeneous sources of information requires efficient handling of inconsistent information. A global information service based on a network of distributed and heterogeneous database systems, also called as a *multidatabase system*, is a typical environment where the inconsistencies between the contents of local databases are unavoidable[16]. A large number of heterogeneous systems connected to the service and their full autonomy makes preservation of global data consistency too time consuming and too expensive. A lack of global consistency control is a source of discrepancies in the contents of common database domains epresented in the remote systems. This work assumes, that in a general case, it is too expensive to eliminate the inconsistencies detected at a data integration stage and because of that we have to deal with this phenomenon in the ordinary day to day processing of a multidatabase. In particular, we consider representation of inconsistent information in a multidatabase system with a relational view of the integrated and remote databases and we investigate efficient query processing in such a system.

Since the advances in network technologies made the integration of distributed and heterogeneous database systems the implementable reality a number of research works has been performed on both practical and theoretical aspects of handling the inconsistencies detected during data integration. These works included the extensions of relational, object-oriented, and semi-structured data models towards storing and processing uncertain and inconsistent data. The works [12, 13] introduced the models of *i-tables* and *m-tables* that partitioned information into three classes: *sure* or *definite information*, *indefinite information*, or *maybe information*. One of the first practical approaches [7] proposed the extensions of relational algebra operations on the relational tables with incompatible attributes. The model of flexible relations [1] considered the data inconsistencies obtained from integration the multiple autonomous relational databases. A paraconsisted relational model described in [4, 17] and later on extended on paraconsistent object-oriented and semi-structured database models [15, 14] defined a paraconsistent relation as a pair of two relational tables. A *positive table* represents all facts know to be true and *negative table* represents all facts known to be false. A nonempty intersection of the positive and negative tables represents the inconsistencies.

The other group of solutions was based on the formal logic and logic programming. [6] investigated a problem of finding the consistent answers from an inconsistent database. The same problem caddressed in [5, 2], and [3] used a concept of *residue* to derive the consistent answers. Merging of inconsistent databases is investigated in [11, 9], and [10]. In [8] the integrity constraints are expressed as disjunctive DATALOG programs. These programs are used to remove the inconsistencies from a database and to generate the consistent answers.

The practical and implementation aspects of dealing with inconsistencies in a typical commercial database management system remain an open problem. Detection of inconsistencies and derivation of consistent answers from inconsistent data and logical consistency constraints is hard because the constraints are usually implemented as small pieces of code embedded either in the client applications or kept on a database server as stored procedures or database triggers. Representation of the inconsistencies as different variants of the same rows in a relational table has a negative impact on performance of query processing when the computations are performed over all combinations of the variants. It is also impossible to extract the negative facts from the commercial database system due to the Closed World Assumption maintained by these systems. Derivation of correct answers from the large amounts integrated data with the logic programming techniques is too time consuming. This work is an early attempt to address a problem of handling inconsistent information in a typical "of-the-shelf" database management system. We consider a central site of a hypothetical multidatabase system that stores and maintains data extracted from a number of local database sites. Our approach clearly identifies the inconsistencies, separates the inconsistencies from clean data and searches for the effective query procesing techniques in a multidatabase system.

The paper is organized in the following way. The next section provides the introduction to integration of multiple database systems and informally describes the basic concepts of our model. The formal specification of the model is presented in Section 3. A system of operations on the elementary and complex units of inconsistent information is defined in Section 4. Section 5 considers the implementation aspect of the model. Finally Section 6 concludes the paper.

2. Concepts

We consider a multidatabase system that integrates a number of remote and independent database systems and provides a transparent view of the component databases as a single monolithic relational database. A lack of global consistency control over the local databases causes inconsistencies later on detected at the data integration stages. A perfect example is a collection of medical records integrated from a number of local medical centers. Each time a patient moves from one suburb to another his/her medical record is recreated from scratch at a medical center. In the relational model of data, the inconsistencies manifest themselves as different sets of tuples representing the same real world entities and/or relationships. Detection of the inconsistencies is performed in the following way. Let r_{d_i} and r_{d_j} be the relational tables defined over the same schema A and obtained from the remote databases d_i and d_j respectively. If the domain constraints over the attributes in A and semantics of r_{d_i} and r_{d_j} expressed as the sets of logical consistency constraints C over A (e.g. a set of all functional dependencies over A) are the same then r_{d_i} must be the same as r_{d_j}. To find all inconsistencies we have to find all subsets of $(r_{d_i} - r_{d_j}) \cup (r_{d_j} - r_{d_i})$ that do not violate the semantics expressed through the constraints in C. As a simple example consider a domain of suppliers including John, Peter, and Paul supplying only bolts and nuts. The semantics are expressed through a functional dependency $supplier, part \rightarrow quantity$ and constraint c saying that each supplier performed at least one shipment. The constraint expresses our beliefs that at least one shipment performed by each supplier has been correctly recorded in a database. Assume that the following sets of tuples have been extracted from the databases d_i and d_j.

d_i	supplier	part	quantity
	John	nut	100
	Peter	bolt	200
	Paul	nut	25

d_j	supplier	part	quantity
	John	nut	100
	Peter	bolt	20
	Paul	bolt	25

It is sure that John supplied 100 nuts. Additionally, the following two sets of cases are possible. Peter supplied either 200 or 20 bolts. The third option is eliminated by the functional dependency $supplier, part \rightarrow quantity$. Paul supplied either 25 bolts or 25 nuts or 25 bolts and 25 nuts. The other cases when Peter or Paul supplied nothing are eliminated by the constraint c. All pairs from two sets of cases above represent all inconsistencies detected during a data integration process. At the end of integration, sure information is stored in a separate relational table and inconsistencies are represented as the additional variants. The formal and implementation details are discussed later.

We introduce a concept of a *common context* defined as a set of logical consistency constraints. In the example above a context consists of the functional dependency $supplier, part \rightarrow quantity$, constraint c and domain constraints: con(supplier) = {John, Peter, Paul}, con(part) = {bolt, nut}, and con(quantity) = N^+. A concept of context allows for clear identification of inconsistencies, representation of inconsistencies as the variants of relational tables, and for separation of clean from inconsistent information. As the result it is possible to construct a system where the applications operate on both clean and inconsistent components of a database and return both sure information and the variants of unsure information. In such a system, a decision which variant is correct is no longer an obstacle for an immediate access to integrated information. It is also possible that future computations performed on both sure and inconsistent information may contribute to elimination of the inconsistencies.

3. Data objects

Let A be a set of attribute names later on called as a schema, and let $dom(a)$ denotes a domain of attribute $a{\in}A$. A tuple t over a schema A is a full mapping $t : A \rightarrow \cup_{a\in A} dom(a)$ and such that $\forall a{\in}x$, $t(a){\in}dom(a)$. A relational table constructed over a schema A is a set of tuples over a schema A. A *context* C is a pair $<A, D>$ where A is a set of attribute names and D is a set of domain constraints imposed on the domains of attributes in A. A *domain constraint* $con(a){\in}D$ determines a set of values an attribute a can take in a given context, i.e. $con(a){\subseteq}dom(a)$. In the rest of this paper we consider only the contexts defined over the sets of domain constraints. Domain constraints being the elementary components of contexts are implemented by the majority of database management systems. A *variant v* in a context $<A, D>$ is a pair $<d, r>$ where d is identifier of the source of information and r is a relational table over schema A such that $\forall t{\in}r{,}a{\in}A\ t(a){\in}con(a)$. A *chunk of inconsistent information* (*ichunk*) is a set of variants $v_1, \ldots v_n$ in the contexts $C_1, \ldots, C_n$ respectively. We call *ichunk c* as a *homogeneous ichunk* if all its variants are created in the same context.

4. Operations

A system of operations on *ichunks* is a reflection of a standard system of relational algebra operations. We start from the operations that change the domain constraints in a context of a variant. Let $v{=}(d, r)$ be a relational variant in context $<A, D>$. A *split* of variant v over an attribute a and constraint $c{=}con_i(a)$, is denoted by $\prec_c(v)$ and it is defined as variants $v_i{=}(d, r_i)$ in context $<A, D_i>$ and $v_j{=}(d, r_j)$ in context $<A, D_j>$ such that:

(i) $r_i{\cap}r_j{=}\emptyset$ and $r_i{\cup}r_j{=}r$,
(ii)$D_i{\cup}D_j{=}D$ and
$con_i(a)\ {\in}D_i$, $con(a){-}con_i(a){\in}\ D_j$

An operation opposite to a *split* is a *merge*. A *merge* of the variants $v_i{=}(d, r_i)$ in a context $<A, D_i>$ and $v_j{=}(d, r_j)$ in a context $<A, D_j>$, D_i-$con_i(a)\ =\ D_j$-$con_j(a)$ and attribute a is denoted as $v_i{\bullet}v_j$ and its results is a variant $v{=}(d, r)$ in a context $<A, D>$ such that:

(i) $r_i{\cup}r_j{=}r$,
(ii) $con_i(a){\cup}con_j(a){=}con(a){\in}D$ all other attributes have the same domain constraints as D_i, D_j.

Let $\pi, \sigma, \times, \cup, \cap, -$ denote the operations of *projection*, *selection*, *Cartesian product*, *union*, *intersection*, and *difference* of the relational algebra. The respective operations on variants are almost the mirror reflections of the relational operations. The only difference are the actions performed by the operations on the contexts of variants. Consider a variant $v{=}(d, r)$ in a context $<A, D>$. Let $X{\subseteq}A$. Projection of a variant v onto a scheme X is denoted by $\mathbf{project}_X(v)$ and it is equal to a variant $v'{=}(d, r')$ in a context $<X, D_X>$ such that:

(i) $D_X{=}\{con(a){:}a\ {\in}X\}$,
(ii) $r'{=}\pi_X(r)$

Let ϕ be a well-formed formula of the prepositional calculus. Selection from a variant $v{=}(d, r)$in a context $<A, D>$ is denoted by $\mathbf{select}_\phi(v)$ and it is equal to a variant $v'{=}(d, r')$ in a context $<X, D_\phi>$ such that:

(i) $D_\phi{=}\{\sigma_\phi(con(a)){:}a{\in}X\}$,
(ii)$r'{=}\sigma_\phi(r)$

The binary operations on variants are designed to combine the variants from different source of information.. These operations include the set operations and relational join. The set operations are defined only for the variants in the same context. Then, a set operation $\alpha{\in}\{\cup, \cap, -\}$ on the variants $v_i{=}(d_i, r_i)$, $v_j{=}(d_j, r_j)$ both in the same context $<A, D>$ computes a variant $v{=}(d_id_j, r)$ in context $<A, D>$ where $r{=}r_i\ \alpha\ r_j$. Signature of the result v is a concatenation of the signatures of its arguments.

Natural join operation on a variant $v_i{=}(d_i, r_i)$ in a context $<A_i, D_i>$ and variant $v_j{=}(d_j, r_j)$ in a context $<A_j, D_j>$ an such that $D_i[A_i \cup A_j]{=}D_j[A_i \cup A_j]$ is denoted as $v_i{\bowtie}v_j$ and its result is a variant $v{=}(d, r)$ in a context $<A, D>$ such that:

(i) $A_i \cup A_j{=}A$ and $D_i \cup D_j{=}D$,
(ii) $r = r_i \bowtie_{A_i \cup A_j} r_j$

Unary operations of projection (Π_X) and selection (Σ_ϕ) on *ichunk c* are performed by application of the respective operations of **project** and **select** to each variant included in *ichunk c*. Binary operations of union, intersection, difference and join of ichunks are performed on all pairs of customized variants selected from the arguments. Customization to the common context is achieved through projection and horizontal split of variants, For example union of *ichunk* $c_i{=}\{v_i\}$ and ichunk $c_j{=}\{v_j\}$ where $v_i{=}(d_i, r_i)$ in context $<\{a, b\}, \{con(a), con(b)\}>$ and $v_j{=}(d_j, r_j)$ in context $<\{a\}, con(a)>$ needs projection of variant v_i into a context $<\{a\}, \{con(a)\}>$. Then, union of v_j and v_{i_1} can be performed in a common context $<\{a\}, \{con(a)\}>$. Finally, union of c_i and c_j consists of a variant v_i and variant v_{ij} obtained from union of

v_j and v_{i_1}. Signature of variant v_{ij} is obtained by concatenation of the signatures of its arguments.

The heterogeneous operations **drop** and **append** variant complete a set of operations on variants. An operation $\textbf{drop}_v(c)$ removes a variant v from ichunk c and operation $\textbf{append}_v(c)$ appends a variant v to c.

5. Implementations

This section considers implementation of the model defined above. We use an ordinary "of-the-shelf" relational database management system. Implementation of a variant $v = (d, r)$ in a context $<A, D>$ is a relational table r and a collection of the "lookup" relational tables that represent a set of domain constraint D. An enumerated domain constraint with explicitly provided list of values is implemented as a single column table that contains all values of the domain. An analytically defined domain constraint with or without increment/decrement is implemented as a s `CHECK` constraint on the attribute in r and single column table that contains increment/decrement values. For example, a domain constraint $age > 30 \cup \{25, 26\}$ is implemented as a check constraint `CHECK(AGE>30)` and lookup table with two rows [25], [26]. A set of attributes A and additional constraints for r are implemented in a data dictionary (repository) of a selected DBMS.

An *ichunk* $c = \{v_1, \ldots, v_n\}$ is implemented a set of relational tables $r_1, \ldots, r_n$ where $n \leq m$ and table that contains the names tables $r_1, \ldots, r_n$ and names of "lookup" tables implementing the domain constraints. A number n of relational tables is less than a number of variants m because all homogeneous variants i.e. in the same context are implemented as a single relational table with the columns representing the characteristic functions. Consider a set of homogeneous variants $v_1 = (d_1, r_1) \ldots . v_k = (d_k, r_k)$ all in a common context $<A, D>$. Implementation of the homogeneous variants is a relational table $r = \cup_{i=1,\ldots,k}(r_i)$ over a schema $A \cup \{V_1, \ldots, V_k\}$. The columns $V_i, i = 1, \ldots, k$ implement the characteristics functions that map the tuples in r into $\{0, 1\}$ depending on whether a tuple belongs to a variant v_i or not.

Computation of queries in the database systems with inconsistent information is similar to computation of queries in the ordinary relational database systems. A query expressed in a high-level declarative query language is initially translated into an expression of relational algebra with *ichunks* as the arguments and operations on *ichunks* . Implementations of the operations on variants are systematically applied to all variants in *ichunk* or to all pairs of variants from two *ichunks* . Let $c = \{v_1, \ldots, v_k\}$ be a set of variants in a common context $<A, D>$. Then, projection of *ichunk* , $\Pi_Y(c)$ is implemented as

```
SELECT Y, max($V_1$), ..., max($V_k$)
FROM r
GROUP BY Y;
```

where $r = \cup_{i=1,\ldots,k}(r_i)$. Let $c = \{v_1, \ldots, v_m\}$ and $d = \{w_1, \ldots, w_n\}$ be two sets of variants in a common context $<A, D>$. Union of *ichunks* , $c \cup d$ is implemented as a join of the results R(A,U_i,U_j) over a set of attributes A of the following `SELECT` statements computed for all pairs v_i, w_j of variants in c and d.

```
R(A,$U_i$,$U_j$) :=
  SELECT A, max($U_i$), max($U_j$)
  FROM(
      SELECT A, $V_i$ $U_i$
      FROM R
        UNION
      SELECT A, $V_j$ $U_j$
      FROM R )
  GROUP BY A;
```

Intersection and difference of *ichunks* are also computed as a join of the results obtained from respective intersection or difference performed on all pairs of variants from the arguments. Let c be an *ichunk* in a context $<A, D>$ and d be an *ichunk* in a context $<B, E>$. Natural join of *ichunks* $c \bowtie d$ is implemented as a join of the results R(A $\cup$ B ,U_i,U_j) over a set of attributes a *cup* B of the following `SELECT` statements computed for all pairs v_i, w_j of variants in c and d.

```
R(A ∪ B,$U_i$,$U_j$) :=
  SELECT A,B max($U_i$), max($U_j$)
  FROM(
      SELECT A, $V_i$ $U_i$
      FROM R JOIN S
      ON R.(A∪ B) = S.(A∪ B) )
  GROUP BY A, B;
```

Decomposition of *ichunks* into sure and inconsistent components eliminates redundancies and speeds up query processing. It is justified by a fact that the majority of data belongs to the "sure" components and for all of them there is no need to represent and to process the variants. This part of query computation can be simply done as evaluation of relational algebra expressions. Computation of the operations on the mixtures of "sure" and "imprecise" components can be done faster because of the expected small size of imprecise components. The small "imprecise" components can be kept in a main memory for the computation time. Finally, decomposition allows for unified evaluation of expressions where only some of the arguments contain inconsistent data and the rest are the plain relational tables.

6. Summary

This work addresses a problem of dealing with inconsistent information in a multidatabase system that integrate a number of independent and heterogeneous database systems. A concept of context is used to detect the inconsistencies in the descriptions of identical database domain extracted from the component databases. A model presented in this paper generalizes a concept of relational table to a set of variants (*ichunk*) each valid in a predefined context and defines a system of algebraic operations on *ichunks* . We shown how *ichunks* and operations on *ichunks* can be implemented in a typical relational database management system. The contexts allow for precise identification of the inconsistencies and for separation of clean and inconsistent information in a multidatabase system. This approach also allows for a direct access to the results of integration without the resolutions of detected inconsistencies. It contributes to more effective computation of queries and elimination of inconsistencies from the answers through restrictions on domain contexts.

An area that need further work concerns the impact of the other decomposition methods on efficiency of query processing. For example, decomposition of *ichunks* into ”sure” ”maybe” ”exclusive” and the other more specific sorts of inconsistent information may provide the chances for faster query processing and more precise specifification of the results. A vertical decomposition of the variant parts of*ichunks* is another idea that also needs to be considered.

References

[1] S. S. Agarwal, A. Keller, G. Wiederhold, and S. Saraswat. Flexible relation: An approach for integrating data from multiple, possibly inconsistent databases. In *Proc. of IEEE Data Engineering Conf.*, 1995.

[2] M. Arenas, L. E. Bertosi, and J. Chomicki. Consistent query answers in inconsistent database. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'99)*, pages 68–79, 1999.

[3] M. Arenas, L. E. Bertosi, and M. Kifer. Applications of annotated predicate calculus to querying inconsistent databases. In *Proc. of the First International Conference on Computational Logic (CL 2000)*, 2000.

[4] R. Bagai and R. Sunderraman. A paraconsistent relational data model. *International Journal of Computer Mathematics*, 55(1):39–55, 1995.

[5] P. Barcel and L. Bertossi. Logic programs for querying inconsistent databases. In *Proc of 5th Int. Symp. on Practical Aspects of Declarative Languages, PADL 2003*, pages 208–222, 2003.

[6] F. Bry. Query answering in information systems with integrity constraints. In *IFIP WG 11.5 Working Conf. on Integrity and Control in Information System*. Chapman & Hall, 1997.

[7] L. G. DeMichiel. Resolving database incompatibility: An approach to performing relational operations over mismatched domains. *IEEE Transactions on Knowledge and Data Engineering*, 1(4):485–493, 1989.

[8] G. Greco, S. Greco, and E. Zumpano. A logic programming approach to the integration, repairing, and querying inconsistent databases. In *Proc of the 17th Int. Conf. on Logic Programming, (ICLP'01)*, pages 348–346. Springer, 2001.

[9] D. Lembo, M. Lenzerini, and R. Rosati. Inconsistency and incompleteness in data integration. In *Proc. of the 9th Int. Workshop on Knowledge Representation meets Databases (KRDB 2002)*, 2002.

[10] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002*, pages 233–246, 2002.

[11] J. Lin and A. O. Mendelzon. Merging databases under constraints. *Int. J. of Cooperative Systems*, 7(1):55–76, 1998.

[12] K.-C. Liu and R. Sunderraman. Indefinite and maybe information in relational databases. *ACM Trans. Database Systems*, 15(1):1–39, 1990.

[13] K.-C. Liu and R. Sunderraman. A generalized relational model for indefinite and maybe information. *IEEE Transactions on Knowledge and Data Engineering*, 3(1):65–77, 1991.

[14] L. Mengchi and W. L. Tok. A data model for semistructured data with partial and inconsistent information. In *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology*, pages 317–331. Springer Verlag, 2000.

[15] S. J. K. R. Bagai. Paraconsistency in object-oriented databases. In *Soft-Ware 2002: Computing in an Imperfect World, First International Conference, Soft-Ware 2002, Proceedings*, pages 141–150. Springer, 2002.

[16] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous and autonomous databases. *ACM Computing Surveys*, 22(1):183–236, 1990.

[17] N. Q. Trân and R. Bagai. Efficient representation and algebraic manipulations of inifinite relations in paraconsistent database. *Information Systems*, 25(8):491–502, 2000.