Database semantic integrity for a network data manager*

by ELIZABETH FONG and STEPHEN R. KIMBLETON

National Bureau of Standards
Washington, D.C.

1. INTRODUCTION

The goal of semantic integrity is assuring that data within a database is logically correct. Logical correctness is evaluated by showing that the data represents a valid abstraction or model of a 'real world' application. This is required for effective use of the database.

The starting point in assuring database semantic integrity is a clean (semantically correct, i.e. logically correct) database and the objective is ensuring that subsequent updates also result in a clean database. Since individual updates may involve operations across data structures, may require several statements of the data manipulation language being used, and may involve logically interrelated data, assuring semantic integrity is a difficult problem which currently lacks a complete solution.

This paper has two major objectives. The primary objective is establishing a collection of capabilities appropriate for a semantic integrity system. This assumes an environment providing a uniform mode of access for the network user to multiple, remote heterogeneous DBMSs. The secondary objective is providing a brief survey of the existing semantic integrity literature. Thus, the following two sections explore the existing approaches to semantic integrity. Thereafter, the environment supporting remote database access is described. This is followed by a discussion of the design of an Experimental Semantic Integrity System (XSIS) currently being constructed at the National Bureau of Standards.

2. WHAT IS SEMANTIC INTEGRITY?

Data within a DBMS can be in error for one of three reasons: incorrect entry, system failure, or conflict with the intended meaning of the data. Incorrect entry occurs when a wrong value gets keyed-in during the data entry process. For example, due to a keypunch error, a person's weight might be entered as 87 rather than 78 kgs.

System errors reflect a variety of failures of design or implementation including: security violations, hardware failures, and failures in system or DBMS software such as the concurrency control mechanism.

Conflicts with the intended meaning of data occur through violations of perceived interrelationship among the data within DBMS. For instance, the sum of division capital expenditure budgets, after an update, may no longer equal the organizational capital expenditure budget.

Semantic integrity is concerned with assuring that such violations cannot occur. Since such violations represent a conflict with the understood meaning of data, and since this understood meaning is usually not formally expressed, the goal of achieving semantic integrity has proved elusive.

Because semantic integrity is concerned with the logical meaning of data, it can also provide limited assistance in detecting data entry errors or system failures. For instance, through value-range specification, the semantic integrity system could detect the error in entering a person's weight as 570 kg, rather than 70 kg. It could not detect the difference between 78 and 87 kgs. Similarly, system failures resulting in major data errors can be detected when the data is examined.

2.1 Approaches to semantic integrity

There are two primary approaches to semantic integrity. The assertion based approach permits users or Database Administrators to specify those semantic integrity rules which seem important. This approach facilitates incremental incorporation of a semantic integrity capability for a DBMS.

The disadvantages of the assertion based approach are potentially significant and include: i) the inability to determine the completeness of a collection of rules since there is no basic point of reference for their assessment, ii) the possibility that adding a new rule will show that a database previously thought to be semantically correct now suffers from integrity violations, and iii) the need for explicit incorporation of consistency checks on the collection of rules, which might lead to significant overhead.

The major alternative to the assertion based approach depends on a thorough development and application of the
concept of type [BRODM 78]. Strong data types are constructed from system data types. Moreover, through exploiting the concepts of aggregation and generalization, type interrelationships can be formally specified. Such specification has the advantage of permitting semantic integrity assertions to be checked statically at compile-time rather than run-time. Additionally, the completeness and consistency of data type specifications can be shown.

The disadvantage of this approach is the explicit requirement for complete specification. For databases of significant size, completion of the specification process requires a major investment of effort. Another drawback is that not all semantic integrity requirements can be specified using the data type concept because certain validity criteria are "value" rather than "type" oriented. A combined approach may prove best in an operational environment.

2.2 Semantic integrity system

A combined approach to supporting semantic integrity seems appropriate. Implementing such a system requires specification of three major components: i) the rules specifying the semantic integrity constraints which may be either type or value constraints, ii) the process for checking conformance with these rules, and iii) the actions which occur upon detection of an integrity violation.

2.2.1 Rules specifications

The semantic integrity rules or constraints need to be stated prior to database use. These constraints specify all the required information to be used during rules enforcement time. Strong data type specifications effectively augment the traditional concept of schema and are stated through denotational or declarative methods at data definition time. Assertion based approaches require a means for specifying the assertion and are evaluated while a request is being processed.

Designing a semantic integrity specification language requires considering two main issues: i) the style of the specification language used by users and database administrators, and ii) the volume of information required by the system to check conformance with the rules.

2.2.2 Rules enforcement

Evaluating conformance with semantic integrity rules can be thought of as being performed by an abstract observer or daemon monitoring database operations. The times at which the daemon can observe the database are defined; there may be times during an update when the daemon is precluded from judging whether semantic integrity is being maintained.

Implementation of the daemon requires considering three major issues: i) the types of tests which the daemon may perform, ii) the information required to support these tests, and iii) the cost of performing them. This last item is very important from a practical point of view.

2.2.3 Violation actions

Detecting a semantic integrity violation requires flagging the error and, probably, rejecting the update. The precise rule(s) responsible for the violation action should be identified. The options may be a call to a user-specified error routine, reporting an error message to the user, or semi-automated error elimination by the system.

2.3 Related works on semantic integrity

Much of the existing semantic integrity literature is concerned with isolated aspects of the global problem—placement of responsibility, data semantics, specification languages, invocation techniques, and supporting system environments. For instance, [FERNE 76], [GRAV 75], and [MACHC 76] all describe high level semantic integrity specification languages. In [WEBEH 76] semantic integrity is viewed in the context of state transitions; therefore, constraints are expressed upon database operations. Buneman and Morgan [BUNEO 77] developed "alerting" mechanisms for supporting semantic integrity. Several selected semantic integrity systems are reviewed below.

2.3.1 Brodie

Brodie’s approach [BRODM 78] views semantic integrity as a (semantic integrity) system rather than user responsibility. A specification language together with a verification methodology is developed. The specification language is based on a denotational approach and the emphasis is on proving consistency and completeness.

Since Brodie’s approach is not currently implemented, support system requirements have not been identified. However, the extremely systematic approach which is presented requires complete specification at database design time. Operational use would require investigation of the issues of sizing and flexibility.

2.3.2 McLeod

McLeod’s approach [MCLED 76, HAMMM 75] is assertion based, views semantic integrity as a system responsibility, and also describes a semantic integrity system. Special emphasis is placed on the design of a constraint specification language for a relational data model. The specification language is based on using a high level, non-procedural language permitting specification of: i) constraints, ii) times at which the constraints are to hold, and iii) actions to be taken on occurrence of a constraint violation.
2.3.3 System R

System R, a relational database management system developed at IBM, provides semantic integrity facilities as part of the SEQUEL language [ESWAK 75]. The approach uses the assertion based method. An assertion can be any SEQUEL predicate evaluating to a Boolean. System R provides a very extensive collection of supporting capabilities. Various types of semantic integrity features are provided, such as: tuple and set assertions, state and transition assertions, and immediate and delayed assertions. Semantic integrity enforcement is implemented using system triggers and, if the result of a transaction is proven to violate an assertion, the transaction is rejected and a predefined procedure or failure action is invoked.

2.3.4 INGRES

INGRES is a relational database management system developed at University of California, Berkeley. Stonebraker [STONM 75] introduced semantic integrity assertion statements in the INGRES system as one or more range statements plus an integrity qualification. These assertion statements in the form of predicates are appended to all user interactions with a database. Thus certain types of update errors are prevented. This implementation technique has been referred to as query modification, and has been characterized as easy to implement. The INGRES approach of incorporating semantic integrity via query modification minimizes the requirement for support. However, it also limits the set of semantic integrity features which are provided.

2.3.5 CODASYL

The CODASYL Data Description Language Committee Journal of Development specification [CODAS 79] provides a user-written procedure invocation mechanism which allows integrity requirements to be programmed in procedural code by the user. The keyword CHECK with user specified parameters provides the triggering mechanism when a data item is changed.

CODASYL does not use the system approach but, rather, places the burden of specification and enforcement of semantic integrity upon users who must write application programs.

The CODASYL specification supports an underlying network data model with stringent structural requirements; therefore, facilities for duplicate checks, member record insertion conditions, and unique keys checks are supported.

[MELOF 79] proposes semantic integrity facilities to be incorporated in a CODASYL-like DBMS. These integrity constraints are meant to enhance those integrity capabilities that are provided by the CODASYL specification and those that are inherent in the network data model.

3. SEMANTIC INTEGRITY FEATURE LIST

The above brief survey reveals that, currently, there is no unified, practical, comprehensive and complete approach to database semantic integrity. Semantic integrity features offered within a DBMS range from very simple data type checking to complex assertion processing. We present a gross feature list with the following categorizations:

3.1 Strong data type constraints

Semantic integrity constraints based on an extension of the data type concept can be specified at data definition time. Such constraints are related to the concept of strong typing developed for program languages.

The concept of a strong data type arose from developments in abstract data types [LISKB 74; GUTTG 77] which can be informally described as special purpose entities with constrained usage properties. Constraints typically include both the operations which may be performed on a given data element as well as the collection of other data elements which may be involved in binary operations.

Some examples of strong data type constraints are described as follows:

1. Value constraints—specify the range of acceptable values, establish whether a value must be specified, and define whether a data value must be unique. For instance, data values may be required to lie within certain bounds (e.g., age between 1 to 100). Data values may also be constrained to an enumerated set (e.g., colors are red, green, blue). Data values may be specified to be essential or non-missing in which case missing values are considered to be semantically incorrect (e.g., data element EMPNO must not be missing). Data values may be required to be unique (e.g., EMPNO must be unique).

2. Extended format constraints—permit specification of a data format pattern composed from primitive types, such as character, integer or real. For example, the first character of a supplier number may be required to be the letter S.

3. Domain compatibility—supports assurance that cross domain operations, e.g., binary operators, are applied against compatible units and prohibits such operations against incompatible domains. For example, automatic invocation of scaling factors is required when some weights are expressed in kilograms and others in pounds. Some automatic techniques for performing such conversions are given in [KARRM 78]. Invalid comparisons, e.g., comparing weight to time, should be flagged as constraint violations.

4. Legal operation constraints—limit the operations which can be performed on a given domain to those judged semantically correct. Thus, a set of legal operations may be associated with a data item. Attempting to per-
3.3 Transition constraints

Transition constraints specify relation between old and new states of the databases and are invoked when the database changes from one state to another. The two major types of transition constraints are:

1. Old/New transition constraint—During an update operation, there exists an "old" value to be changed to a given "new" value. For example, new salary must be greater than old salary.
2. Nonexistence/existence transition constraint—The insertion operation involves a nonexistence to existence transition, while a deletion involves an existence to nonexistence transition. For example, deletion of an account number may require that the balance be zero.

3.4 User-controlled enforcement

Using one of the constraints specified above requires establishing when the constraint is to be invoked. User controlled enforcement permits the user to state WHEN to enforce the integrity constraints:

1. Deferred/immediate enforcement—for transactions requiring more than one data management request, will permit suspension of integrity constraints until all requests have been issued. The user is responsible for specifying deferred enforcement and the system must be able to back-out the transaction when deferred enforcement results in a constraint violation.
2. ON/OFF enforcement—permits the user to switch integrity checking ON or OFF depending upon the level of integrity needed for the application. It is useful since some integrity checking is costly.

3.5 Integrity specification and maintenance facilities

Some integrity checking, such as simple data type checking, is easily supported. More complex kinds of assertions need to be stated, maintained, and invoked at the appropriate time. Some semantic integrity system facilities might be:

1. User-written Application Program Interface—The DBMS does not provide a centralized semantic integrity subsystem but provides interface mechanisms so that users can code their own integrity enforcement routines as an application program. This application program interface is usually available in most of the commercially available DBMS.
2. Integrity Specification Language—The semantic integrity system permits users to specify integrity constraints in a higher-level language. This language is compiled into procedures which are triggered for the enforcement of the constraints.
3. Integrity Constraint Maintenance—The semantic integrity system permits users to read, modify, create and destroy integrity constraint assertions.

3.6 Feature summarizations

Not all of the features identified above are offered by each of the reviewed systems. Moreover, substantial differences exist in the implementation of features common to two or more systems. Brodie's approach has not yet been implemented. McLeod's has been partially implemented. System R has not been released as a product. INGRES is available and is being used at several sites. CODASYL specifications are still in the process of enhancement. Implemented versions of CODASYL-like systems usually include the CHECK clause plus interface mechanisms for user-written procedures so that necessary integrity features may be coded by the user via application programs.

The semantic integrity feature list contained in Figure 1 together with the various enforcement approaches provides a basis for the design of a prototype Experimental Semantic Integrity System.

4. XSIS SUPPORT ENVIRONMENT

A prototype Experimental Semantic Integrity System (XSIS) is currently under construction at the National Bu-
Database Semantic Integrity for a Data Manager

4.1 Experimental network data manager (XNDM)

The basic structure of XNDM is illustrated in Figure 2. It is implemented in the C language on a PDP-11/45 attached to the Arpanet and running the UNIX operating system. Server modules exist on target systems to provide local support. XNDM provides a uniform, table based virtual view of data maintained by independent, network accessible DBMS. It differs from a distributed database because: i) the view of data provided the network user is virtual, ii) the DML provided the network user differs from that actually employed by the target DBMSs, and iii) different systems can use different DBMSs, data models, and DMLs.

4.1.1 XNDM data structures

The XNDM data structures are assumed to be constructed by a committee of Data Base Administrators who, collectively, identify the data whose access is to be supported and specify the access paths used to access this data. Although the virtual view of data presented by these structures is relational, i.e., table based, the data models used by the target systems can be relational, hierarchical, or CODASYL.

4.1.2 XNDM data language

The data language for XNDM is termed the Experimental Network Data Language (XNDL). It consists of three major components: a data definition language defining the tables and their data attributes, a data control language providing nondiscretionary access controls and semantic integrity specifications, and a data manipulation language for issuing queries and updates. The DML is based on a subset of SEQUEL (redundant predicates, sort operations, and a programming language interface are excluded) which has been extended to provide primitives appropriate to specifying and controlling concurrent access to multiple databases. A more complete specification of XNDL is contained in [KIMBS 79].

4.1.3 Interfacing to the target systems

Using XNDM requires the DML statements (XNDL queries or updates) to be translated to the DML employed by the

Figure 1—Semantic integrity feature list.
target system(s). Since this translation process is dependent upon both the preestablished source data structures as well as the locally determined target data structures, it proves substantially harder than that required to support database front ends. Substantial work has, however, been done on XNDM query translation [KIMBS 79], [WANGP 80].

Currently, XNDM does not support updates. A major technical problem in their support is related to the recognized problem of updating views. A general solution to this problem appears difficult if not impossible [DAYAU 78] because of hidden data. Although this is valid for a general, and hence arbitrary view, it may be resolvable in the context of XNDM applications by suitably defining the data structures presented to the user. Such definition would require classification of DBMS data into independent groups coupled with a requirement that a user be able to update all data elements of the group if the user can update any data element of the group. Since remote users often have specialized interests, this approach may be commensurate with implicit organizational requirements.

Deferral of XNDM update capabilities reflects the desire to complete implementation of query support since a thorough understanding of this problem is required to provide the basic addressability information required to support updates. In view of the strong interest which has been expressed to the authors about the impact of the network user on the quality of data maintained by a DBMS, it seemed appropriate to establish the types of guarantees which can be provided together with the problems implicit in their support. This is the objective of this paper.

4.1.4 Need for semantic integrity in XNDM

The preceding suggests that local acceptance of a remote updating capability may well depend upon the extent to which suitable correctness guarantees can be provided. Two basic support mechanisms for providing such correctness can be developed. The first is an appropriate collection of discretionary access controls for assuring that individual remote users can only access data appropriate to their access rights. The second is a semantic integrity support capability. Issues related to the first have been discussed in [WOODH 79]; the remainder of this paper is concerned with the second topic.

4.2 Semantic integrity in the networking environment

Semantic integrity in the XNDM environment differs in several essential ways from that usually associated with centralized DBMSs.

1. Global and local integrity constraint conflicts—reflecting the possibility of a conflict between globally established constraints and those established by local DBMS management.
2. Conflicting assertions—constraints appropriate to different target DBMSs may also be in conflict.
3. Partial data problem—local semantic integrity assertions may involve data not accessible to the network user. Thus, it may prove impossible to provide global
checking which is complete from the local point of view.

5. XSIS DESIGN AND IMPLEMENTATION ISSUES

XSIS is also being implemented in C on the same system on which XNDM is being implemented. Its design and implementation are intended to support exploration of both the assertion based and strong data type approaches to semantic integrity. Because of the preliminary nature of XSIS work, the following comments are provisional; based upon accrued experience, the basic XSIS design goals may be substantially modified in the future.

5.1 XSIS design

The major goal of XSIS is to provide a filter for checking remote database operations expressed via the XNDL. XSIS views semantic integrity maintenance as a system rather than user responsibility.

The offloading of XSIS from both target systems and their DBMSs permits a flexible, iterative approach to its design, implementation and modification. The design of XSIS is guided by several principles:

1. Offloading of semantic integrity enforcement—the burden of enforcing semantic integrity constraints on access by the network user should be offloaded, as much as possible, onto XSIS. This reflects the perception that local management may prove unwilling to incur an extra processing burden just to support remote users.
2. Emphasis on global checking—because of the belief that remote users are less knowledgeable users, and the desire to minimize local system overhead in supporting semantic integrity.
3. Modular design and implementation—to permit an iterative design, implementation, operation and modification cycle without impacting local systems or other XNDM components.

XSIS consists of two major components: One component supports constraint specification and maintenance. The constraint specification processor accepts a constraint specification and stores an intermediate representation in the semantic integrity tables which are linked to the XNDM data dictionary.

The second component supports constraint evaluation and enforcement. It is activated when a user issues a request in XNDL. Enforcement decomposes into those checks which can be performed independently of the target DBMSs, and those requiring retrieval of data from local databases. The former are primarily type checks while the latter test whether the appropriate assertions are satisfied.

The XSIS processing sequence consists of: i) receipt of a parsed tree representation of an XNDL statement by the XNDM translator, ii) performance of type constraint tests, iii) retrieval of dependent data required to process run-time tests, iv) performance of run-time tests, and v) return of a condition code indicating whether a semantic integrity violation was detected and, if so, its type.

5.1.1 XSIS strong type constraints

XSIS supports strong data types whose specification includes:

1. Data name.
2. Data format—a description of the format of the data type as composed from primitive types such as character, integer, or real.
3. Legal operators—permissible for a given domain, e.g., arithmetic, relational, and DML operators which, for binary operators, will be domain dependent.
4. Compatible domain names—a list of domain names which can be involved in binary operations with the given domain.
5. Value restriction—assertions upon value such as legal range or permitted set of values.

This initial collection of capabilities is significantly less encompassing than those reported in [BRODM 78] but does provide a reasonable range of functionality. Moreover, as experience is gained into their utility, additional extensions can be implemented because of the modular nature of XSIS.

5.1.2 XSIS assertion processing

Key issues in supporting XSIS assertion processing are: i) maintenance of the appropriate database of assertions, ii) evaluation of the collection of assertions for consistency, and, iii) utilization of these assertions in performing the appropriate run time checks. The first two issues are currently being investigated. The third, given the nature of the networking environment, is affected by data movement requirements necessary to support run time processing. Such data movement can be reduced by having the local node perform some of the checking. This distributed approach has the obvious disadvantage of requiring local additions and modifications. In an operational environment, however, such modifications may prove highly cost effective.

XSIS assertions may involve interrelated data elements. Since a table based data model is being employed, such assertions can be classified into row, column, intra-table, or inter-table assertions. Evidently, processing of more complex assertions is likely to require retrieval of greater amounts of data from the supported DBMSs.

Consistency constraints can be further classified as: change rules, insertion rules, and deletion rules. Initially, only change rules are being implemented.

The distributed environment provided by XNDM imposes a bandwidth limitation on communications between the requesting process and the target DBMSs. As a result, it is highly desirable that such interactions have a transaction oriented flavor as observed in [GRAYJ 78]. In turn, this re-
quires capability for user specification of when semantic integrity rules are to be enforced. Since XSIS is external to the target DBMS, such temporary extensions are easy to effect. However, backing out an extension may prove rather difficult if the target system does not provide an appropriate collection of backout mechanisms.

5.1.3 XSIS system issues

XSIS semantic integrity specifications are maintained in tables associated with the XNDM data dictionary. Integrity maintenance commands are provided for DISPLAYing, DELETEing, DEFINEing and CHANGEing integrity specifications.

The cost of achieving a high degree of database semantic integrity may be prohibitive [BADAD 79; HAMM 78]. The distributed nature of semantic integrity enforcement provided by XSIS naturally raises the issue of performance. Consequently, timing and frequency statistics are to be maintained with the integrity specifications for assessing invocation rates and performance penalties. Such information should permit estimating the cost and performance of assuring semantic integrity in the network database environment.

6. CONCLUDING REMARKS AND IMPLEMENTATION STATUS

The preceding discussion structured a basic approach to providing semantic integrity while supporting uniform access to multiple, remote, heterogeneous DBMSs. Moreover, the discussion has shown that the implementation framework of XSIS permits a blend of both strong data type and assertion based approaches to semantic integrity. Thus, a flexible vehicle supporting future research in this area has been achieved.

XSIS implementation is currently in a very preliminary state. Some strong data typing capabilities exist and the implementation of assertion checking has begun. Although the early implementation status precludes reporting operational results, the structure has been described to encourage discussion of this important issue.

ACKNOWLEDGMENT

The authors would like to express their appreciation to L. J. Miller and Pearl S.-C. Wang for their helpful comments on earlier versions of this paper.

REFERENCES