LABORATORY FOR
COMPUTER SCIENCE
(formerly Project MAC)

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-165

HIGH LEVEL EXPRESSION OF SEMANTIC INTEGRITY SPECIFICATIONS IN A RELATIONAL DATA BASE SYSTEM

Dennis J. McLeod

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract no. N00014-75-C-0661

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

This blank page was inserted to preserve pagination.

MIT/LCS/TR-165

High Level Expression of Semantic Integrity Specifications in a Relational Data Base System

Dennis J. McLeod

September 1976

Massachusetts Institute of Technology

Laboratory for Computer Science
(formerly Project MAC)

Cambridge Massachusetts 02139

This empty page was substituted for a blank page in the original document.

ABSTRACT

The "semantic integrity" of a data base is said to be violated when the data base ceases to represent a legitimate configuration of the application environment it is intended to model. In the context of the relational data model, it is possible to identify multiple levels of semantic integrity information: (1) the description of the domains of the data base, as abstract sets of atomic data values (domain definition), (2) the specification of the fundamental structure of the relations of the data base (relation structure specification), (3) the definition of the abstract operations which are meaningful in terms of the application environment (structured operations), and (4) the expression of additional semantic information not contained in the structure of the relations nor in the identities of their underlying domains (relation constraints).

A high level, nonprocedural domain definition language facilitates the description of domains. Such a language allows the specification of the properties of the values constituting a domain, and the action that is to occur if an attempt is made to update a column entry such that it does not belong to the underlying domain of that column. The specification of relation structure and structured operations can also be accomplished by means of high level integrity (sub)languages.

A relation constraint has three components: (1) the assertion (a predicate on the state of the data base or on transitions between data base states), (2) the validity requirement (the occasion(s) at which the assertion must hold), and (3) the violation-action (the action that is to occur if the assertion does not hold at a time when it should). Relation constraint specification can be related to an expression framework (classification scheme) which is useful for the construction of a relation constraint language and specification methodology. Assertions are more than expressions of some relationships among different values in a data base; an assertion singles out the data that is constrained, and states the properties that this data must possess. A classification is provided of the various predicate types used to identify constrained data and to state the properties that they are to possess.

A semantic integrity subsystem (of a generalized relational data base management system) can support the generation and maintenance of integrity specifications, verify that these specifications are met by the data base, and take appropriate action if violations are detected.

This empty page was substituted for a blank page in the original document.

ACKNOWLEDGEMENTS

The author is most grateful to Professor Michael Hammer of MIT for his enthusiastic suppport and guidance, and for his many and varied contributions to the contents of this thesis. Many others have helped greatly, providing ideas, comments, and criticisms, including: Jack Aiello, Sheldon Borkin, Daniel Carnese, Arvola Chan, Marvin Essrig, Richard Grossman, Professor Barbara Liskov, Professor William Martin, Professor David Redell, Arnold Schiemann, and Sunil Sarin (all of MIT); Dr. Donald Chamberlin, Dr. Edgar Codd, Dr. Kapali Eswaran, Dr. Frank King, Dr. James Gray, and Dr. Bradford Wade (all of IBM San Jose Research); Professor Michael Stonebraker (of the University of California, Berkeley). Although many of the ideas in this thesis belong to these persons, all of the mistakes belong to the author. Finally, the author would like to thank Mary Rykowski, for her moral support, for polishing the prose of earlier drafts of this document, and for being an inspired and unending critic.

This research was sponsored by the Advanced Research Projects Agency of the Department of Defense and was monitored by the office of Naval Research under contract number N00014-75-C-0661.

This report is a slightly revised version of a thesis submitted to the Department of Electrical Engineering and Computer Science in June 1976, in partial fulfillment of the degree of Master of Science.

This empty page was substituted for a blank page in the original document.

TABLE OF CONTENTS

ABSTRACT 2

ACKNOWLEDGEMENTS 3

TABLE OF CONTENTS 4

LIST OF FIGURES 6

1. INTRODUCTION 7

1.1. Semantic Integrity 9

1.2. The Data Model 11

1.3. The Relational Data Model 13

2. SEMANTIC INTEGRITY 16

2.1. Background 17

2.2. An Approach to Semantic Integrity Specification 20

3. DOMAIN DEFINITION 23

3.1. Describing Sets of Atomic Data Values 24

3.2. A Domain Definition Language 25

3.2.1. Language Details and Examples 28

3.3. Implementation Considerations 32

3.4. Extensions 35

4. RELATION STRUCTURE 37

4.1. Additional Column Information 38

4.2. Comparability 38

4.2.1. Domain Conversions 39

5. STRUCTURED OPERATIONS 44

5.1. Semantic Integrity Information in Structured Operations 44

This empty page was substituted for a blank page in the original document.

5.2. The Definition of Structured Operations 45

6. RELATION CONSTRAINTS 48

6.1. Whither Assertion Structure? 49

6.2. Relation Constraint Assertions 51

6.2.1. Simple Assertions 51

6.2.2. Identification of the Constrained Collection 53

6.2.3. Tuple Assertions 57

6.2.4. Set Assertions 61

6.2.5. Scope of Assertions 65

6.3. Relation Constraint Validity Requirement 66

6.4. Relation Constraint Violation-Action 69

6.5. Implementation Considerations 70

6.6. Remarks 71

7. ON THE DESIGN OF A SEMANTIC INTEGRITY SUBSYSTEM 72

7.1. Components of a Semantic Integrity Subsystem 72

7.2. The User's View of the Integrity Mechanism 73

7.3. Some Thoughts on Integrity Subsystem Implementation 74

7.3.1. The Use of Inversions in Relation Constraint Checking (An Example) 76

8. REMARKS AND DIRECTIONS 78

REFERENCES AND BIBLIOGRAPHY 80

LIST OF FIGURES

1-1. Relation EMP 106

1-2. Example Data Base 107

1-3. A Possible Set of Relational Primitive Operations 108

3-1. Selected Example Data Base Domain Definitions 109

3-2. Syntax of the Domain Definition Language 111

6-1. Some Simple Assertions (for data base in figure 1-2) 115

6-2. Local Tuple Predicates 116

6-3. Nonlocal Tuple Predicates 117

6-4. Local Set Predicates 118

6-5. Nonlocal Set Predicates 119

1. INTRODUCTION

Rather than just a collection of values, a data base should be a model of some application environment. When a data base ceases to represent a valid configuration of that application environment, the semantic integrity of the data base is violated. The purpose of this thesis is to examine the problem of describing and preserving the semantic integrity of a data base in the context of a generalized data base system. The general goal is to provide a first approximation to a "theory" of semantic integrity (particularly in the context of the relational data model), and to provide a basis for a semantic integrity specification methodology. This includes an overview of the relevant issues as well as a description of a particular approach to the problem, with emphasis on the high level, nonprocedural expression of semantic integrity requirements.

Data base systems (data base management systems) are intended to assume the tasks of facilitating data storage, manipulation, and retrieval. The data base system should also be responsible for maintaining the correctness of the data in a data base, as well as providing users with appropriate abstract views of the data. This is particularly important for large data bases, as *ad hoc* and "hand" checking is impractical.

By way of background, it might be useful to place the notion of semantic integrity in perspective, and to better define the meaning of the term as used in this thesis. There are a number of ways in which the soundness of data in a data base may be compromised:

1. The reliability of data may be compromised by errors due to hardware failure, as well as those due to failure of the operating system and data base system software. Hardware reliability (in the context of data base systems) has been considered elsewhere [Fossum 1974, Wilkes 1972]. Software reliability is a very prominent research concern at present, as exemplified by the work of those concerned with

establishing the correctness of programs. In the area of data base systems, Hawryszkiewycz and Dennis [Hawryszkiewycz 1972, Hawryszkiewycz 1973] have developed a formal semantic model of a relational data base system, defined data base primitive operations in terms of this model, and proven the correctness of the operation definitions (abstract programs). Weber [Weber 1976] has further developed this approach.

2. The concurrent consistency of data may be violated due to the effects of improperly controlled accesses to shared data by multiple concurrent users (processes). It is desirable to provide each user with a consistent view of a data base, shielding this user from interfering effects due to the activities of other users, while at the same time retaining a maximum amount of legitimate concurrent activity. Eswaran, Gray, Lorie, and Traiger [Eswaran 1974] have described a high level scheme for concurrent consistency control in a relational data base system. Hawryszkiewycz and Dennis [Hawryszkiewycz 1972, Hawryszkiewycz 1973] developed a lower level model of concurrent consistency based on a formal semantic model of a relational data base system.

3. Data security may be compromised by a failure to properly (administratively) restrict the manner in which a given user may access and manipulate a data base. A good deal of pioneering effort in the area of security and protection has been accomplished in the context of operating systems. Some of this work has been extended to data base systems, e.g., the work of Chamberlin, Gray, and Traiger [Chamberlin 1975], and Stonebraker and Wong [Stonebraker 1974d].

4. The semantic integrity of data is violated when the data base ceases to represent a legal configuration of the application environment it is intended to model. Semantic integrity errors may be introduced by user error, lack of understanding, malice, etc.

("inadvertent, improper, or malicious update" [Stonebraker 1974c]). In fact, hardware and software reliability errors, concurrent consistency errors, and security errors may cause the semantic integrity specifications of a data base to be violated. For example, some user may, because of a failure of the data base security mechanism, make an unauthorized change, such as raising his own salary from $20,000 to $90,000; this unauthorized change can then cause a semantic integrity constraint to be violated, such as "all employee salaries are less than $25,000".

This thesis deals specifically with the fourth aspect of the soundness of a data base, namely semantic integrity. In what follows, we assume that hardware and software reliability are guaranteed (eg., by the operating system). We also assume that concurrent consistency is assured; it is sufficient to assume, without loss of generality, that a single user is interacting with the system at any given time. Security issues are not further considered in this thesis.

1.1. Semantic Integrity

A data base is meant to serve as a model of some limited universe; at any given time, the values in the data base represent a particular configuration of that application environment. Every such world has its own intrinsic logic: a set of rules specifying what constitutes a legitimate and plausible configuration of that environment [Florentin 1974]. It should be the function of the data base system to insure that these rules are not violated and therefore that the data base is not in a semantically inconsistent state.

A basic premise we will adopt is that, as noted by Minsky [Minsky 1974a]: "the fundamental property of a data base is that it has an intrinsic meaning which is invariant of its interaction with users". The semantic integrity specifications for a data base capture this intrinsic meaning. The data base system should facilitate the precise expression of

these integrity specifications. We assume that some person (or committee of persons), known as the data base administrator, is responsible for stating the semantic integrity specifications for the data base.

It is possible and indeed desirable for the data base system to support multiple abstract logical views of a data base. These views must however be constructed from and consistent with the semantic integrity specifications (i.e., the data base administrator's view of the data base). Even providing a view of the data base which consists of a subset of that data base is difficult, because of the "connections" between the subset and other elements of the data base.

A variety of causes may result in a compromise of the semantic integrity of a data base, including:

1. inaccurate data recording or entry,

2. inadvertent alteration of data during some transmission or transcription process,

3. deliberate falsification of data,

4. loss, omission, or delay of data.

The ramifications of permitting incorrect data to permeate a data base may indeed be far reaching. Crucial decisions may be wrongly influenced, user confidence in the system destroyed, and the reliability and performance of the system degraded (including application programs and packages as well as the data base system itself).

It is generally recognized that the problem of bad data in data bases is a serious one. Unfortunately, the state of the art in error checking in data base systems is quite dismal. Most semantic integrity checking is currently accomplished by means of application programs; data checking mechanisms are embedded in these application programs. Special purpose data base "audit" routines are also sometimes used to check data integrity. Existing commercial data base systems perform limited types of integrity checking, if any. This

checking is nearly always limited to simple data format checks. In any case, semantic integrity information and checking is usually unstructured, and is embedded in application programs in an *ad hoc* manner [Gosden 1974]; furthermore, no discipline is imposed on the semantic integrity specification process. This lack of structure and discipline has the following consequences:

1. The mechanism by which semantic integrity specifications are checked is diffuse.
2. Semantic integrity specifications are not readily modifiable.
3. The abstraction defined by the semantic integrity specifications, which is intended to correspond to the set of rules in the application environment, is difficult to understand.
4. Inconsistencies and redundancies can be present in the semantic integrity specifications, which may be difficult to locate.
5. It is difficult to make the semantic integrity checking process efficient, either by means of manual or automatic optimization.

1.2. The Data Model

The data model upon which a data base system is based is defined here to consist of the type(s) of data structures used to represent information in the data base, along with the set of primitive operations which can be used to manipulate these structures. The nature of the data model underlying a data base system has a very significant effect on the manner in which one describes the semantic integrity of a data base in that system. As described below, some semantic integrity information is often in fact embedded in the structures used in the data model [Date 1975, Mommens 1975].

There have been three principal data models proposed for generalized data base systems [Date 1975]:

1. For historical and other reasons, the hierarchical approach is a very popular one. Examples of hierarchical data base systems and data sublanguages (languages for defining and manipulating data bases) include IMS [IBM], HQL [Fehder 1974], Data Language [Marill 1976] and System 2000 [MRI 1972]. In the hierarchic approach, some semantic integrity information is expressed in the form of one-to-many relationships (trees). Thus, one-to-many constraints are expressed by appropriately constructing the data base hierarchy.

2. The network approach is typified by the Codasyl DBTG proposal [Codasyl 1971a] and the work of Bachman [Bachman 1973]. An example of a network data base system is Adabas [Software AG 1974]. In the network data model, some semantic integrity information is expressed via many-to-many relationships; this is done by appropriately constructing the network structures of the data base.

3. The relational approach was introduced by Codd [Codd 1970] [Codd 1974a]. Examples of relational data base systems and data sublanguages include ALPHA [Codd 1971a], INGRES [McDonald 1974a, McDonald 1974b, Held 1975b], MACAIMS [Goldstein 1970], Query by Example [Zloof 1974, Zloof 1975a, Zloof 1975b], RDMS [Steuert 1974], RISS [McLeod 1975], SEQUEL [Boyce 1975a, Chamberlin 1974b, Chamberlin 1975], and SQUARE [Boyce 1975b, Boyce 1975]. In the relational data model, functional dependencies are normally included in the specification of the basic structure of relations. However, as discussed in section I.3, these functional dependencies may be easily separated from the basic structure of the relations of the data base.

Several (higher level) semantic data models have been recently proposed [Chen 1975, Schmidt 1975, Senko 1975, Smith 1976, Tsichritzis 1975]. These higher level models attempt to incorporate more semantic integrity information in the basic

structure of a data base. Structures in these data models are intended to represent objects, attributes of objects, and relationships between objects (in the application environment). Semantic operations on these structures represent legitimate changes in the application environment.

It is not the purpose here to analyze these data models in detail, although many of the ideas developed herein are quite closely related to work on semantic data models. Rather, and for reasons to be explained later, the relational data model will be used herein, as a basis for the discussion of data base semantic integrity. Although the ideas discussed in this thesis are applicable to data base systems in general, the discussion is couched in terms of the relational model of data.

1.3. The Relational Data Model

The relational data model "appears to be the simplest data structure consistent with the semantics of information and which provides a maximum degree of data independence" [Boyce 1975b]. As concisely stated by Codd [Codd 1974a]: "In the relational approach there exists an interface at which the totality of formatted data in a data base can be viewed as a collection of nonhierarchic relations of assorted degrees on a given collection of simple domains (domains whose elements are not decomposable as far as the data base management system is concerned)."

For the purposes of this thesis, a (relational) data base is defined to be a collection of normalized relations (relations in first normal form [Codd 1970]), and a collection of domains. (The relations present in the data base are specifically called base relations.) A domain is an abstract set of atomic data values (objects). Domains are defined independently of relations. A normalized relation may be viewed as a table, wherein each row of the table corresponds to a tuple of the relation, and the

entries in a column belong to the set of values constituting the underlying domain of that column. (An entry is the value in some particular column for a given row of a relation.) The domain underlying a column consists of precisely those objects which can appear as entries in that column; any value in the underlying domain of a column can appear in that column, and every value in the underlying domain is a plausible entry in that column. Note that domain and relation names are unique with respect to a data base, and that a domain and a relation cannot have the same name.

Consider, for example, a data base which contains information about some company. Assume that a relation called EMP contains data on the employees of the company. EMP is shown in figure 1-1, described by its table representation. The rows of the table correspond to tuples of the relation (records), and the columns correspond to instances of particular domains of the data base. (Loosely speaking, a relation corresponds to a "flat" file, a tuple to a record, and a column to a data field.)

Each data base relation is created by naming the relation and each constituent column, and specifying the name of the underlying domain of each column. More than one column in a relation may have the same underlying domain. Column names are unique within a relation. Specifying the name of the underlying domain of each column defines the set of values from which entries in that column may be selected; that is, the set of entries in a column is always a subset of the underlying domain of that column.

Figure 1-2 contains a description of an example data base. The name of each domain and relation of the data base is listed therein (in upper case characters). For each relation, the name of each of its constituent columns is specified (by one upper case character followed by lower case characters), as is the underlying domain of each column. Relation EMP contains information on the employees of the company,

SALES records information on the supplies of items for the company, ORDERS records order information, and BUDGET contains the salary budget for each department of the company.

Figure 1-3 contains a list of some example primitive operations which may be used to interact with a relational data base. It is assumed that in addition to these operations, a high level, nonprocedural query language is provided (e.g., SEQUEL [Chamberlin 1974b], QUEL [Held 1975b], or Query by Example [Zloof 1975a]).

The advantages of the relational data model have been previously elucidated [Codd 1974c, Codd 1975b, Date 1974], and will not be repeated here in detail. For our purposes, the following attributes of the relational model of data are most significant:

1. Access paths are not apparent in the logical view of data.

2. The data model is conducive to (relatively) nonprocedural data selection, query, and manipulation languages.

3. It is possible to cleanly isolate the different levels of semantic integrity in the relational data model, as discussed in chapter 2. For example, in the hierarchical and network data models, certain types of integrity constraints are deliberately built into the data structure itself (e.g., the owner-coupled set construct in the network model). The data base administrator is thus faced with problem of separating the semantic integrity requirements from the complexities of the data structure. However, in the relational data model, "the data base administrator has only one type of structure to consider, and a very simple coordinate system (identification of relations and columns by name and rows by content) by which he may refer to any individual item or portion of that structure." [Date 1975].

2. SEMANTIC INTEGRITY

In the context of the relational data model, it is possible to identify four principal levels of semantic integrity:

1. Domain definition is the description of abstract sets of atomic data values, which are to be used to specify the set of values from which entries in columns of relations can be selected. This can be accomplished by means of a high level domain definition language [McLeod 1976a, McLeod 1976b]. For example, the domain SALARY may be defined as consisting of positive integers less than 100,000.

2. Relation structure specification is the description of the fundamental structure of the base relations. This includes naming each constituent column of a relation, and stating the underlying domain of that column.

3. Structured operations are abstract operations, which are meaningful in terms of the application environment. Structured operations describe data base transactions, and are used to capture the conceptual types of manipulations that are meaningful for a data base (such as, for the example data base of figure 1-2, an operation HIRE-EMPLOYEE).

4. The relation constraints level is concerned with relationships among data base components. Relation constraints are used to define all additional semantic properties of and relationships between the relations of a data base. For example, primary key [Codd 1970] (and third normal form [Codd 1971b, Codd 1971c]) specification is accomplished by appropriate relation constraints. However, relation constraints go far beyond merely supporting functional dependencies; they provide the capability to define a very rich variety of types of data properties. For example, relation constraints may disallow inconsistencies between column entries of a single tuple or

between a tuple and other tuples in the same or other relation(s). They may also preclude some global patterns in some set of tuples in a relation or the data base as a whole, or may disallow certain types of missing data (such as missing tuples, obsolete values, etc.)

Before further describing the approach to semantic integrity which is taken in this thesis, we briefly examine other work that has been done in the area of semantic integrity in data base systems.

2.1. Background

In general, there are two major approaches to the specification of the semantic integrity of a data base:

1. In a state snapshot approach, rules are stated that specify which data base states are permissible (valid states). The data base system is responsible for insuring that the data base is always in a valid state. (As discussed in a later chapter, it may be necessary to allow the data base to temporarily pass through one or more invalid states.)

2. In a state transition approach, the set of legal data base operations is specified. Depending on the data base state, only certain operations (valid operations) are allowed to be performed on that state. These operations are guaranteed to preserve the integrity of the data base.

A state snapshot approach to describing the semantic integrity specifications for a data base involves the expression of logical constraints, which can be viewed as predicates on the state of the data base. These constraints limit the states of a data base to those that conform to some expressed limitations. Several authors [Boyce 1975a, Eswaran 1975, Stonebraker 1974c, Stonebraker 1975c, Zloof 1975b] have discussed semantic integrity

assertions in the context of the relational data model. Graves [Graves 1975] has also considered the problem of semantic integrity.

More specifically, Boyce and Chamberlin [Boyce 1973a] introduced the use of SEQUEL predicates for expressing integrity assertions. For an operation which makes a data base change to be allowed, the predicates must hold on the data base state which results as a consequence of the execution of that operation. Eswaran and Chamberlin [Eswaran 1975] have discussed the functional requirements of a semantic integrity subsystem and have examined semantic integrity in the context of SEQUEL and System R [Chamberlin 1975, Eswaran 1975]. Stonebraker and Wong have considered semantic integrity in terms of the INGRES system and the language QUEL [Stonebraker 1974c], and introduced the concept of query modification as a tool for the implementation of a semantic integrity subsystem [Stonebraker 1975c]. Consider the following example of query modification: a data base operation is attempted which states "increase the salary of each employee in the sales department by 10%"; assuming the existence of an integrity assertion which states that "each employee salary is less than $30,000", query modification would transform the operation into one which specifies "increase the salary of each employee in the sales department by 10%, if that increase results in his salary being less than $30,000". Zloof has studied the problem of semantic integrity with respect to the expression of semantic integrity specifications in Query by Example [Zloof 1975b].

In these approaches, facilities are provided to allow the user to state predicates (expressed in SEQUEL, QUEL, or Query by Example) which are to hold on the data base. Assertions must be satisfied by the result of a data base change for that change to be allowed. Several significant problems exist with these approaches:

1. They do not deal with the entire problem of semantic integrity in a relational data base, but rather focus primarily on relation constraints.

2. They are inadequately flexible with regard to when assertions are to be checked.

3. The types of actions possible upon detection of semantic integrity violations are limited.

4. No structure is placed on the semantic integrity specifications; assertions are arbitrary predicates on the state of the data base or on transitions from one data base state to another.

A state transition approach to semantic integrity specification consists of describing the set of legal operations which may be performed on a data base. In this approach, the user is confined to interacting with the data base by means of a limited set of operations. Semantic integrity information is thus procedurally embedded in the operations. This approach has been suggested by Minsky [Minsky 1974a, Minsky 1974b], in the context of data base systems. Related work in the area of the definition of abstract data types (e.g., the work of Liskov and Zilles [Liskov 1974]) has much in common with this operational approach.

Some of the most significant problems with the state transition approach are:

1. Semantic integrity information is embedded in procedures in an unstructured manner, and is consequently hard to modify and potentially redundant, inconsistent, and incomplete.

2. The conceptual semantic model of a data base is difficult to abstract from the procedurally embedded semantic integrity information.

3. It is difficult to verify the correctness of the semantic integrity information, as it is scattered through the operations.

4. It is not always possible to precisely characterize the set of operations which are meaningful for a data base at the time the data base is created. Data is often kept in a data base before uses for it are discovered, or at least before all of its potential uses

are discovered; nevertheless, it is often possible to describe the semantic integrity of this data by means of properties it must satisfy (e.g., assertions which must hold on the data).

5. Different data base "views" (external schemas) may include very different sets of semantically meaningful operations, while still couched in terms of a single data base schema (conceptual schema). It is difficult to insure the consistency and completeness of the semantic integrity checking which is performed by the operations in different views.

6. Some data base operations are not meaningful in terms of the semantic integrity of a data base, but are nonetheless required in practice (e.g., an operation to change a person's date of birth, the value of which was originally incorrectly entered into the system).

2.2. An Approach to Semantic Integrity Specification

The major goal of this thesis is to provide a first approximation to a "theory" of semantic integrity, particularly in the context of the relational data model. In so doing, it is hoped that a basis for a semantic integrity specification methodology will be developed. This methodology should assist in the formulation of the semantic integrity rules of a given application environment, and direct the selection of those rules which will constitute the semantic integrity specifications of a data base (e.g., in the face of implementation cost tradeoffs).

A semantic integrity subsystem must be capable of performing:

1. semantic integrity checking (error detection),

2. semantic integrity violation localization (determining precisely which data values are in error),

3. semantic integrity violation-action (reporting/response).

The semantic integrity specification language(s) must provide the user with the ability to state all information required to perform these tasks. (This includes, of course, a precise specification of the semantic integrity rules themselves.)

Actually, it is desirable not only to encapsulate (in the data base semantic integrity specifications) knowledge about the semantic integrity of a data base, but also knowledge about how users will interact with the data base. The meaning of a data base includes the manner in which users interact with it; semantic integrity and user abstraction are closely related issues.

Some semantic integrity information is best expressed via a state snapshot approach, while other information is best expressed in terms of state transitions. The approach described in this thesis includes both state snapshot and state transition aspects.

Basically, then, the approach to semantic integrity taken here has several major objectives:

1. It should be possible to express semantic integrity specifications:
    a. on a high level,
    b. declaratively, rather than procedurally,
    c. in a structured manner,
    d. abstractly, in a way relevant to the application environment.
2. These specifications should be:
    a. easily modifiable,
    b. nonredundant,
    c. consistent,
    d. complete (as a model of the application environment),
3. Semantic integrity checking should be:

a. the responsibility of the system (but the system may sometimes need to ask for advice from the user),

b. flexible, allowing appropriate specification of when checking is to be done (e.g., after primitive data base change, after conceptual transaction, etc.),

c. acceptably efficient in terms of the overall performance of the data base system.

4. Semantic integrity violation-action should be:

a. flexible, allowing an appropriate violation-action to be specified (e.g., including error reporting, corrective action, etc.),

b. sufficiently "localized" so as not to generate time-consuming, expensive, and potentially destructive "side effects".

The approach to semantic integrity described in this thesis may in fact be viewed as a generalized approach to data base design and/or data definition. That is, we are attempting to provide a framework by which the data in a data base may be described. Additionally, the framework described herein may prove useful as a base language into which specifications in terms of a higher level data model (such as those described in [Chen 1975, Schmidt 1975, Senko 1975, Smith 1975, Tsichritzis 1975]) may be translated.

3. DOMAIN DEFINITION

The purpose of this chapter is to discuss domain definition, one level of semantic integrity in the context of the relational data model. Specifically, the precise definition of domains, viewed as sets of atomic data values, is considered. This includes a review of the functional requirements for dealing with the problem of domain definition, a discussion and evaluation of other work that has been done in the area, and the description of a specific solution to the domain definition problem.

It is important to note that a domain is different from a unary relation. Domains are abstract sets of atomic data values, and may in fact contain an infinite number of elements. A relation, by contrast, must contain a finite number of tuples. Abstractly, relations are subject to change (e.g., by the addition of new tuples), but domains are changed only when the associated abstraction changes. To a crude first approximation, the set of values constituting a domain is fixed at the time the data base is defined ("compile time"), while the set of tuples in a relation is normally changed during the day-to-day operation of the data base system ("run time").

Domain semantic integrity errors, i.e., errors which involve the presence of entries in some column of a relation which do not belong to the domain underlying that column, occur frequently enough to justify a facility to handle them. Specific experience with a particular data base application environment has shown that, for an experimental sample of user-data base interactions, a large percentage of errors discovered are domain semantic integrity errors [McLeod 1975].

3.1. Describing Sets of Atomic Data Values

As discussed in chapter 2, several approaches to semantic integrity for relational data bases have been recently presented. As noted in that chapter, all of these approaches essentially deal with relation constraints, i.e., facilities are provided that allow the user to state predicates (expressed in SEQUEL, QUEL, or Query by Example) which are to hold on the data base.

The requirements of domain definition are not adequately supported in these systems. They lack the capability to allow domains to be precisely defined as abstract sets of atomic data values. All of these systems allow the data type of each column of a relation (not each domain of the data base) to be defined, but the possible types are limited and very representation-oriented. It should be possible, for example, to define domains like SOCIAL_SECURITY_NUMBER and GEO_COORDINATE, rather than being limited to such domains as INTEGER and CHARACTER_STRING. It is desirable to be able to describe a conceptual class of data values. This abstract description is quite different from a mere specification of the physical representation of the values in a domain; rather, the semantic properties of the domain are pronounced. The work of Liskov and Zilles [Liskov 1974] concerning abstract data types is related to this notion, in that classes of abstract data objects (values) are being described.

Boyce and Chamberlin [Boyce 1979a] have proposed attaching attributes to each column of a relation ("column descriptors"). One of these attributes is the scope of a column, which specifies the set of permissible values for entries in that column, e.g., salary is a positive integer less than 20000. Similarly, Zloof [Zloof 1975b] has indicated that provisions should be made for facilitating the specification of entry "formats" ("their type, size, etc.").

A detailed scheme is needed to facilitate the precise description of domains, and to

integrate the domain definitions with the structure of the relational data base. Such a scheme should (at least) satisfy the following criteria:

1. facilitate the precise and detailed description of sets of atomic data values, as subsets of one of the natural domains: real number and character string (these "natural" domains are the primitive domains which are used to construct other domains),

2. provide for the proper abstraction of defining domains independent of their use as underlying domains of columns in one or more relations,

3. force a domain definition to be a single module, so that domain semantic integrity information is localized,

4. facilitate automatic domain definition checking and flexible types of action which are to occur upon detection of a domain defintion violation,

5. support specifications that describe when and how domain values can be compared (e.g., when two values being compared are from the same domain, and when the two values are from different domains), and converted (e.g., when it is desired to convert the value in one domain into and "equivalent" value in another domain).

3.2. A Domain Definition Language

A high level, nonprocedural language can be used to express domain definitions. In this language, each domain in a data base is described by a single domain definition (domain definition module). The definition of a domain is "installed" (bound) at the time the domain is created. Domain creation may be viewed as the compilation of the domain definition module. Note that a domain definition specifies an underlying set of atomic values. Domains are not dynamic as are unary relations; rather, they constitute fixed abstract sets of data values. The definition of a domain may be modified, but this occurs

only when the abstraction has changed.

As noted by Hammer and McLeod [Hammer 1975], three types of information are required by the semantic integrity subsystem to deal with domain definitions:

1. a specification of the set of atomic data values constituting the domain,

2. information describing when the domain definition is to be checked,

3. a specification of the action that is to occur if the domain definition is not satisfied.

Since we shall assume that domain definitions are checked whenever an entry in some column of a relation is created or altered (e.g., by an operation which inserts or updates a row), the specification of when a domain definition is to be checked need not be explicit. Thus all that need be explicitly expressed in the statement of a domain definition is the precise description of the set of values comprising the domain, and the action that is to occur if an entry in some column of a relation is created or modified so that it does not belong to the underlying domain of that column.

Each domain definition therefore consists of the following four components, represented as clauses in the domain definition language:

1. Domain name

2. Description

The description clause allows the set of atomic data values constituting a domain to be specified. The set of values constituting a domain is defined as some subset of one of the two natural domains: real number and character string. Every domain is thus defined and represented as a subset of the real numbers or of the set of (varying length) character strings. This specification may be accomplished by:

a. enumerating the domain values,

b. decomposing the domain values by specifying the subunits of which they

are composed,

c. placing restrictions on the set of values by stating predicates that describe a subset of one of the natural domains,

or a combination of the above. The special data value "null" (undefined) is present in each domain. This is to allow missing data to be represented in the data base. (It may sometimes be useful to distinguish an "unknown" value from a value which "does not make sense" [Florentin 1976], but this distinction is not made here.)

3. Ordering

The ordering clause is used to indicate how domain values are ordered with regard to comparisons with other values in the same domain. This information is important in identifying the semantic properties of a domain. One type of ordering specification is that the values in a domain inherit the (total) ordering of the natural domain of which the domain is a subset. Inherited ordering may also be by subunit (e.g., the primary ordering is by one subunit, the secondary ordering by another subunit, etc.). Inherited ordering is numeric for domains which are defined as subsets of the real numbers and lexicographic for domains which are defined as subsets of the character strings. Another type of ordering specification is that no ordering exists, in which case only equality comparisons are meaningful. An external procedure (i.e., a procedure in some programming language other than the domain definition language) can also be used to define the ordering specifications for a domain; this procedure is called whenever two values in the domain are to be compared. Such a procedure accepts two domain values (which are to be compared) and returns the value that is first in the ordering sequence.

4. Violation-action

The violation-action clause specifies the action that is to occur if an entry in some

column of a relation is created or changed in such a way that the entry does not belong to the underlying domain of that column. Types of violation-action include:

a. the change may be refused and an error signaled,

b. a particular value, either constant or calculated from the erroneous value by means of operations (such as substring, concatenate, etc.) may be substituted as the new value of the entry,

c. a call may be made to an external procedure, the erroneous value being passed as the argument to the procedure, and the procedure returning the new value of the entry.

System-generated or user-specified messages may be optionally returned to the user or calling program. Note that in cases b and c, it may be necessary to recheck the domain definition after the corrected value of the entry has been determined.

At this point it should be noted that the use of external procedures for ordering and violation-action specification should be minimized, insofar as possible. The capability for such use of external procedures is provided for generality and completeness.

3.2.1. Language Details and Examples

Figure 3-1 contains domain definitions for some of the example data base domains. An indentation-oriented syntax is used in this figure. Examples of values in each domain are listed (in parentheses) to the right of the corresponding domain definition.

Figure 3-2 contains a specification of the syntax of the domain definition language. In figure 3-2, syntactic classes are denoted by lower case strings, while keywords are in upper case; actually, the language should include both upper and lower case keywords. Optional parts are enclosed in "[]", and alternatives are separated by "|".

In figure 3-1, the description clause of the NAME domain definition specifies that it

consists of (character) strings, each of which is composed of a string followed by a ", ", followed by another string. In this description clause, data values are decomposed into subunits; the first and third are variable subunits, while the second is constant. Subunits may be labeled, so that they may be referenced elsewhere in the domain definition. As stated above, external to a domain definition, the data values constituting a domain are either atomic numbers or atomic strings. The rule is, if a description clause of a domain contains only number subunits (variable or constant), then the values in that domain are numbers, otherwise they are strings. Number and string subunits may be mixed, and if so, number subunits are converted to string form to yield the string values constituting the domain. For example, domain MONEY is defined to consist of strings of the form "$25,000". Values in domain MONEY have two subunits, the first of which is the string constant "$", and the second of which is a positive number. Values in domain MONEY are thus represented as strings; the number subunit of any value in domain MONEY is viewed as a number (and can be manipulated as such, e.g., by "+") when the subunit alone is considered, but it is viewed as its string "equivalent" with regard to the domain value as a whole (and can be manipulated by string operations).

The description clause of the domain SEX indicates that it consists of two data values: "female" and "male" (in addition to the ever-present "null"). This is an example of description by enumeration.

For domain MONEY, the subunit labeled "value" must be greater than or equal to zero, as specified by the subunit where restriction. A subunit where restriction contains a predicate that is to be true for the subunit and involves only that subunit; that is, this predicate is a restriction on the set of numbers or strings which values for this subunit may have. It is thereby possible to express properties of number subunits involving comparators (such as "=" and ">") and number constants. It is also possible to state that a number is an

exponential (exponential notation) or an integer (as for domain DATE). For string subunits, a size (length) specification can be made, the set of characters permissible in a string can be defined (as for domain ITEM), and a lexicographic ordering comparison (such as "=" or ">") with constants can be stated.

A global where restriction permits expression of properties involving multiple subunits, as well as those on domain values viewed as a unit. A global where restriction contains a predicate that may involve a domain value, subunit values, operations, and comparators. String operations can be employed to generate substrings, calculate lengths, perform concatenations, etc. Number operations include the usual arithmetic operations and "maximum" and "minimum". For example, in the description of domain MONEY, the global where restriction states that domain values (viewed as strings) must either have two digits to the right of the decimal point or else have no decimal point. Here, "right(⋄, '.' + 1)" evaluates to the right substring of the domain value (which is referenced by "⋄"), starting at the character after the occurrence of ".". (This form of the "right" operation takes two arguments: a string whose right substring is to be calculated, and another string whose index in the first string is calculated to determine at which character of the first string the right substring is to begin.) The operation "present" yields "true" if the first string specified contains an occurrence of each of the following strings, otherwise it yields "false". The global where restriction of domain ITEM illustrates the specification of the number of times some contiguous group of subunits can repeat.

A where restriction may also contain a call of an external boolean procedure (as for domain ITEM). If this procedure call is in a global where restriction, the procedure is invoked with the domain value in question as its argument; the procedure returns "true" if the value is present in the domain, otherwise it returns "false". If the procedure call is in a subunit where restriction, the procedure is invoked with the subunit value in question as its

argument; it returns "true" if the subunit value is legal, otherwise it returns "false".

Boolean combinations of the above types of where restriction are allowed in both subunit and global where restrictions, as are conditionals (as for domain DATE). In addition, an "or" may be used to indicate that the domain contains values that come in more than one form, i.e., that the domain consists of the union of two or more sets of values, each of which is defined separately.

The second clause in a domain definition is the ordering clause. This may specify that no ordering exists on values in the domain ("none"), which means that only equality comparisons are allowed (as for domain SEX). An ordering specification of "atomic" means that values in the domain are ordered by the usual numeric or lexicographic ordering, viewing the domain values as atomic numbers or strings (as for domain QUAN). The ordering clause may also contain an ordered list of labels (subunit names), indicating that domain values are ordered according to the values of the specified subunits. The usual numeric or lexicographic ordering on these subunits is used, and the subunits are taken in sequence: primary ordering, secondary ordering, etc. (as for domains NAME, MONEY, and DATE). Finally, an external procedure can be used to specify the ordering on the values in a domain. This procedure is passed the two values being compared, and returns the value that is first in the ordering sequence (as for domain ITEM).

The third clause in a domain definition is the violation-action clause. As discussed above, it may specify that an error is to be signaled, indicating that the data base change specified by a user is incorrect and should be rejected. A system-generated or user-specified message may be optionally returned to the user or calling program. This is also true for the other types of violation-action. If the violation-action is specified as "error", then an error is signaled and a system-generated message is returned (as for domains NAME and DATE). Domain SEX has a violation-action clause that specifies error signaling with a user-

specified error message. If a system-generated message were desired the specific message could be replaced by "SYSTEM-GENERATED". A system-generated message can be of the form "the definition of domain SEX is violated", or can bear more information if the system is a bit smarter (e.g., "the definition of domain SEX is violated, it consists of only the two values 'female' and 'male'"). The "substitute" violation-action allows a constant value to be substituted as the new value of the entry being created or changed (as for domain MONEY). A calculated value, obtained via string or number operations, can also be substituted (as for domain ITEM). In the specification of this calculation, "*" represents the value that is being checked to determine if it is in the domain. The calculated value is then checked to make sure that it is in fact a valid domain value; if not, then an error is signaled (to avoid infinite recursion). The definition of domain QUAN offers an example of an external procedure call violation-action.

3.3. Implementation Considerations

The domain definition language processor translates domain definitions into an internal form used in semantic integrity checking. The semantic integrity subsystem has the responsibility of determining what checking is to be done whenever some data base change request is issued by a user. It must also assume the responsibility of performing this necessary checking. Whenever a new entry is created in a column (e.g., by an insert row operation) or an existing entry in some row is changed (e.g., by an update row operation), the system must make sure that this new entry belongs to the underlying domain of the column in which it occurs. The information in the description clause of the underlying domain of the column is used for this purpose. If the domain description is violated, the information in the violation-action clause is used. The ordering information is used when comparing two values in the same domain, as discussed in chapter 4.

A domain definition may be used to obtain the information necessary to construct several internal relations, which are used by the semantic integrity subsystem to facilitate domain definition checking:

1. The domain definition relation contains a single tuple for each domain of the data base; this relation has the following columns (with primary key domain name):

   a. domain name,

   b. description type, which is "simple" if the domain has one nonlabeled subunit with no where restriction, otherwise "complex",

   c. global where restriction,

   d. violation-action type, which is "error", "substitute", or "call",

   e. violation-action modifier, which for violation-action type "substitute" is the value (constant or calculated) to be substituted, for "call" is the name of the external procedure to be called, otherwise "null",

   f. error/warning message, which is either a constant (user-specified message), "system-generated", or "null",

   g. ordering type, which is "atomic", "none", "subunit" (for subunit specified ordering), or "call" (for external procedure call ordering),

   h. ordering procedure name, which is the name of the external ordering procedure if the ordering type is "call", otherwise "null".

2. The subunit definition relation contains a tuple for each subunit of each domain; this relation has the following columns (with primary key domain name, subunit index):

   a. domain name,

   b. subunit index, which is the ordinal number of the subunit in the domain definition,

c. subunit type, which is either "constant" or "variable",

d. label, which for constant subunits is "null",

e. variable subunit class, which is "number", "string", or "oneof", and "null" for constant subunits,

f. subunit where restriction, "null" if none exists,

g. ordering index, which is the ordinal number of the subunit in the ordering clause, and "null" if this subunit is not referenced in the ordering clause.

3. The oneof constant relation contains a tuple for each constant in a "oneof" description of domain values or domain subunit values (for each domain in the data base with such a "oneof" description); this relation has the following columns (with all columns in the relation as primary key):

a. domain name,

b. subunit index,

c. oneof constant, which is a constant in the "oneof" list for the subunit identified by the subunit index (for the domain specified by the domain name).

Domain definitions may be utilized to automatically determine the appropriate physical storage type to be used to represent values in a domain. For strings, a fixed length character string representation can be used when possible, such as when domain values are enumerated (via "oneof"), or when an upper bound is placed on the length of string values in the domain. In other cases, varying length character strings can be used. For numbers, it may be necessary in many cases to make a compromise for efficiency. Integers ("number where integer") may be represented by a fixed binary storage scheme (e.g., single word binary), but it must be clear that this is only an approximation to the domain definition. A similar situation exists for real numbers: a float binary representation may be used for storage.

3.4. Extensions

Important issues to be considered in future research on domain definition include:

1. It is possible to extend the domain definition language so that previously defined domains may be used as subunits in the definition of a new domain. If this hierarchic approach is used, care must be taken by the system to retain domain definitions until they are no longer referenced in any other domain definition.

2. It may be useful to introduce domain operations. In this approach, operations are defined for each domain, and manipulation of values in the domain is restricted to the specified operations. This approach is similar to the notion of abstract data types of Liskov and Zilles [Liskov 1974]. It may be argued that the approach taken in this paper is still too representation-oriented. For example, values in the domain MONEY may be strings or numbers, but this is irrelevant with respect to abstraction. The important properties of the values constituting a domain may be best characterized by specifying the operations that are defined on the values in the domain. Of course, in this case a domain will no longer be defined as a subset of one of the natural domains (string and real number), and the standardized set of domain operations (such as ">", "=", "+", etc.) will probably no longer be appropriate.

3. It may be advantageous, in some cases, to defer the checking of domain definitions, and not report violations at the time the data is actually entered into the system. For example, in the case where a data base is being "bulk loaded" or updates are being "batched", it may be desirable to report all violations of domain definitions at a later time, say to an interactive user or as part of a summary report.

4. The modifiability of domain definitions is a very important issue. It should be possible for the definition of a domain to be changed as the corresponding abstraction changes. If this is allowed, then it is necessary to verify that all entries in

columns having a given underlying domain satisfy the new definition of that domain.

5. It is possible to call an external procedure to verify that a value in question belongs to a domain. An external procedure call may also be used in the ordering and violation-action specifications. However, we have no guarantee that the external procedure is correct. Some reliability is nonetheless guaranteed by the fact that this external procedure must use the normal data base system interface. In addition, the domain definition is again checked after the external procedure has terminated.

6. The problem of implementing the domain definition scheme and evaluating its effectiveness and efficiency has yet to be fully addressed.

7. It may be useful to consider the automatic generation of domain definitions by attempting to generalize upon a few examples of domain values which are given by a user. This is, of course, a part of the general problem of the detailed specification of the user interface which supports the construction of domain definitions.

4. RELATION STRUCTURE

Relation structure specification is the description of the fundamental structure of the (base) relations of a data base. When a relation is created, at least the following must be done:

1. The relation must be given a name, which is unique with respect to all names of relations in the data base.
2. The number of columns in the relation must be specified.
3. Each column of the relation must be assigned a unique name (unique with respect to the names of the columns of the relation).
4. The name of the underlying domain of each column must be specified. A definition for each domain thus referenced must exist at the time the relation is created.

It is possible to include other types of information as a part of the fundamental structure of a relation. For example, the primary key [Codd 1970] of the relation may be identified. However, at the level of abstraction at which our discussion of semantic integrity is focused, the identification of the primary key may be viewed as a type of relation constraint (and expressed as such). Furthermore, there is no compelling reason for distinguishing the primary key from other candidate keys [Codd 1970]. It is most logical for a primary key specification to be viewed as a relation constraint, as is the case for other types of functional dependencies.

Many higher level semantic models for data base design and abstraction (data definition), e.g., [Smith 1976], consider certain types of relation constraints (such as functional dependencies) to be special. Functional dependencies are one important type of constraint, but there are other types which may be equally important (in some application

environment). We believe that it is essential to provide for a broad spectrum of relation constraint types, and to integrate the formulation of these constraints with the process of data base design and abstraction. In chapter 6, our approach to relation constraints is further discussed.

4.1. Additional Column Information

In addition to the column name and the name of its underlying domain, it is useful in practice to allow two additional attributes to be associated with each column:

1. a narrative description of the column, for documentation purposes,
2. an indicator specifying whether "null" (undefined) values may be present in the column (thus allowing "null" values to be selectively prohibited from columns).

4.2. Comparability

The kinds of comparisons and manipulations of column entries that are allowed relates to the semantic integrity requirements of a data base. The term comparability is used herein to refer to the general problem of determining when and how two or more column entries may be compared or otherwise manipulated by structured operations. There are two basic types of comparisons: intradomain comparisons and interdomain comparisons.

Intradomain comparisons are those in which two values from the same domain are compared. In this case, the information in the ordering clause of the domain definition is sufficient to determine how the comparison is to be made.

Interdomain comparisons are those in which two values from different domains are compared. In this case, values are compared as atomic strings or numbers using a domain conversion, as defined below.

4.2.1. Domain Conversions

A data base has associated with it a set of domain conversions. Each domain conversion is specified by means of a domain conversion module. Each such conversion is a specification of how values in a given domain are converted into "equivalent" values in another domain, and vice versa. Explicit specification of domain conversions is necessary because values in different domains belong to different abstract sets, and converting a value in one domain into an "equivalent" value in another requires knowledge of the precise nature of the abstract sets corresponding to the two domains involved. For example, both FEET and INCHES are numbers, but they cannot be meaningfully added without the use of an appropriate conversion.

Domain conversions are defined independent of the domains (and relations) of a data base, in the sense that domain conversion modules have no access to the internal details of a domain definition; domain conversions thus map atomic values in one domain into atomic values in another. Domain conversion modules can be dynamically created, deleted, and modified, with the restrictions that:

1. both domains referenced in a domain conversion module must exist at the time the conversion is created,

2. if either of the domains referenced in the domain conversion is deleted, the domain conversion is deleted.

For the purposes of this thesis, it is assumed that domain conversion modules are written in some high level programming language. This language may be a specialized one, similar to the domain definition language. For generality, it is permissible to allow this language to invoke external procedures written in a high level general purpose programming language.

For example, a conversion for domain DOLLARS and

THOUSANDS_OF_DOLLARS can be defined as:

```
domain conversion DOLLARS, THOUSANDS_OF_DOLLARS
        DOLLARS = THOUSANDS_OF_DOLLARS * 1000
        THOUSANDS_OF_DOLLARS = DOLLARS / 1000
```

Conversions may be unidirectional as well as bidirectional, and this is the reason for the seemingly redundant specification in the above example. For more complex types of conversions, external procedures may be used; for example, we may have:

```
domain conversion DATE, JULIAN_DATE
        DATE = p1(JULIAN_DATE)
        JULIAN_DATE = p2(DATE)
```

where p1 and p2 are external procedures.

Structured operations may perform various types of domain comparability operations on entries in a data base. The standardized set of such domain operations includes "=", "~=", ">", ">=", "<", "<=", "+", "-", "*", "/", "**", and string and user-defined operations. For example, some structured operation may check to see if, for some tuple in relation R, the entry in column A is larger than then entry in column B. (It is assumed tha both columns A and B contain numbers.)

Whether or not values from different domains may be utilized together (compared or otherwise manipulated) depends upon the nature of the domains and the particular type of operation that is to be performed on the values in those domains. In order to establish a first approximation to a set of comparability rules (for the standardized set of domain operations), three types of comparability are distinguished:

1. equality-type, which is invoked when one of the following types of manipulations occurs:

    a. values are compared for equality ("=") or inequality ("~="),

    b. numbers are added ("+") or subtracted ("-"),

    c. sets of numbers are manipulated via set operations, such as "maximum" and

"minimum",

d. sets of values are manipulated by "union", "intersection", or "difference",

2. ordering-type, which is invoked when values are compared via "<", "<=", ">", or ">=",

3. mixed-type, which is invoked when values are manipulated via multiplication ("*"), division ("/"), exponentiation ("**"), or any string operation or user-defined operation.

Equality-type comparisons are always allowed if the two values being compared (or manipulated) are from the same domain, i.e., if the values are from the same column or from columns with the same underlying domain. If the values are not from the same domain, i.e., they are from distinct columns with different underlying domains, then they may be compared if and only if a domain conversion exists between those domains. (All domain conversions must be explicitly defined.) The domain conversion is used to convert the value in one of the domains into an "equivalent" value in the other domain, and the resulting values are then compared. (Another type of conversion could be supported, by assigning units to each column, and defining units conversions [McLeod 1976b].)

Ordering-type comparisons are allowed if two values are from the same underlying domain and the ordering of that domain is not "none". The ordering information in the domain definition is used to determine how the values are to be compared. Ordering-type comparisons are also allowed if the two values are from different columns, these columns have different underlying domains, and a domain conversion exists between those two underlying domains. In this case, the values are compared by using the domain conversion, as for equality-type comparisons. In any other case, ordering-type comparisons are not allowed.

Mixed-type comparisons are always allowed. Values can always be manipulated by a mixed-type operation (with no restrictions). Values that are numbers may be multiplied,

divided, and exponentiated with no limitations, except of course for the requirement that the values be numbers. Although numbers may be added and subtracted only when they have the same "units", multiplication, division, and exponentiation can be performed without any such restriction. It presumably makes sense to divide a value in domain FEET by a value in domain POUNDS, but it is (normally) not sensible to add these two values. For mixed-type comparisons, values being manipulated are treated as atomic and domain conversions are not used. Note that if user-defined domain operations are allowed, they may be placed in this category by default. More generally, it may be best to allow the user to specify the comparability type (equality, ordering, or mixed) of each user-defined domain operation.

If the user wishes to state an unusual type of query, such as asking for all employees whose name is the same as the name of their department, the user may be allowed to "force" the comparison, by explicitly overriding the restrictions. Entries in the two columns are then compared using the default numeric or lexicographic ordering, treating the values as atomic numbers or strings, respectively. The idea is to permit the system to be flexible and not to allow comparability rules to get in the way when they should not. The best approach may be to warn the user that an operation may be meaningless, but allow it to proceed if he demands it. (The semantic integrity of the data base is not really in danger anyway).

Domain conversions are also useful when a structured operation retrieves an entry from some column of a tuple in a relation and assigns it to be the new value of some other entry (in a different column of some tuple in a relation). For example, suppose that the date an item was shipped by some company (the entry in column Date of relation ORDERS in the example data base of figure 1-2) is to be copied into the Date column of another relation, say BIG_ORDERS. (BIG_ORDERS records all orders which request over $1000 of merchandise.) The Date column in BIG_ORDERS has underlying domain

JULIAN_DATE (i.e., dates of the form "76.194"), while the Date column in ORDERS has underlying domain DATE (i.e., dates of the form "1/20/1976"). Thus the domain conversion from DATE to JULIAN_DATE can be used to effect the desired assignment.

The general rule for an assignment which takes the entry in a column (A) and assigns it as the new value of an entry in another column (B) is as follows:

1. If A and B have the same underlying domain, the assignment is performed with no conversion.

2. If A and B have different underlying domains, then:

   a. if a domain conversion exists from A to B, the conversion is used to affect the assignment,

   b. if no such conversion exists, the assignment is not allowed.

5. STRUCTURED OPERATIONS

A very important aspect of data base semantic integrity is the set of operations a user may employ to examine and manipulate the data base. It is possible to describe a user's view of a data base as consisting of data structures plus operations. Alternatively, one may conceptually characterize the user's abstract view completely by a set of abstract operations, as is done in abstract data types [Liskov 1974]. These operations provide a behavioral specification of the semantics of the data base.

For these reasons, the concept of a structured operation is included in our approach to semantic integrity. The principal purpose of a structured operation is to embody a conceptual data base transaction: an action which is meaningful and permissible in the context of the application environment. For the example data base of figure 1-2, structured operations may include: hire_employee, fire_employee, raise_salary, place_order, create_new_department, etc.

5.1. Semantic Integrity Information in Structured Operations

One approach to preserving the semantic integrity of a data base is impose the restriction that the operations that may be performed on a data base are only those in some given set. This set of operations should be defined so that it contains only meaningful actions. However, the approach of allowing only semantically meaningful operations has several problems:

1. Operations which are not semantically meaningful in the context of the application environment must be allowed, e.g., to permit errors to be corrected.

2. The set of operations that are to be allowed may depend upon some characteristics of the data base state. For example, the set of operations O1 may be legal if the data

base is in state S1, but if the data base is in state S2, the set of legal operations may be O2.

3. The uses of a data base are not fixed, but rather evolve with time. Operations change and new operations need to be created. If the semantic integrity information is embedded in these operations, a scan of all data base operations may be necessary to make such modifications.

4. Often data is maintained in a data base before uses for it are discovered. Thus it is difficult to characterize the data via a behavioral semantics approach; in some sense the semantics of the data is known, but the exact nature of the set of operations on that data is not.

5.2. The Definition of Structured Operations

Despite the problems mentioned above, it is important to be able to define a set of abstract operations on a data base. To this end, we allow structured operations to be defined. Structured operations are constructed using:

1. the primitive data base operations (e.g., see figure 1-3),

2. statements in a very high level data selection (query) and data modification language, such as SEQUEL (or QUEL or Query by Example).

Structured operations are ordered lists of: primitive operations, statements in a data selection and modification language, and previously defined structured operations. Allowing previously defined structured operations within new operations enables a hierarchic organization.

For the example data base of figure 1-2, a structured operation to raise an employee's salary could be defined:

```
operation raise_salary (employee_name, new_salary)
        update EMP
                where Name = employee_name
                Salary = new_salary
```

This structured operation consists of a single SEQUEL-like statement, which updates the Salary column of the tuple in EMP with a value in the Name column equal to the first parameter of the operation (presumably there is one such tuple). The new Salary value is specified as the second parameter.

Consider an operation to place an order (again in the context of the example data base of figure 1-2):

```
operation place_order (customer_id, item_id)
        insert_tuple (ORDERS)
        Item = item_id
        Customer = customer_id
        Date_shipped = date()
        Order_number = generate_order_number()
```

In this example operation, a tuple consisting of all null values is first created, and then its columns are given values. Note that two external procedures are called, one to return the current date and the other to generate a unique order number. The types of names (identifiers) used in the definition of the operation include those of parameters, a relation, columns, and external procedures.

The operation check_credit_and_order could be defined as:

```
operation check_credit_and_order (customer_id, item_id)
        if check_credit (customer_id)
                then place_order (customer, item)
                else error
```

The operations check_credit and place_order used in this definition are assumed to have been previously defined. Note that this operation contains a conditional expression: a useful construct we may include in the structured operation language. This of course motivates the need for other types of constructs, e.g., for iteration. We may for instance want to have an operation that takes an arbitrary number of items as parameters and

places an order for each.

Thus, in general, it might be desirable to have a structured operation language which has many of the capabilities of a general purpose programming language. We could consequently allow structured operations to be written in some high level general purpose programming language. The details of this are not persued here.

One important point to note in passing, is that structured operations are important with regard to the specification of when relation constraint assertions are to hold (be checked). This is further discussed in chapter 6.

6. RELATION CONSTRAINTS

The fourth aspect of semantic integrity in a relational data base system concerns relation constraints. In this chapter, the requirements for relation constraints are detailed, and an approach to their specification is presented.

Codd [Codd 1971b, Codd 1971c] has identified the "third normal form" of relations [Codd 1974a]: "A relation R is in third normal form if it is in first normal form and, for every attribute collection C of R, if any attribute not in C is functionally dependent on C, then all attributes in R are functionally dependent on C." Third normal form facilitates the straightforward expression of some types of relation constraints, namely functional dependencies. But the class of data properties describable via functional dependencies is limited.

Boyce and Chamberlin [Boyce 1973a] observed that a high level language, such as SEQUEL [Chamberlin 1974b, Chamberlin 1975], may be used as a vehicle for the expression of data properties other than functional dependencies. SEQUEL expressions were shown to be useful in expressing such types of properties as "uniqueness of key", "functional dependency", "validity check", and "inter-relational constraints".

The integrity assertions of SEQUEL [Boyce 1973a, Eswaran 1975], INGRES [Stonebraker 1974c], and Query by Example [Zloof 1975b] are used to express varied types of data properties. However, these facilities basically provide for the unstructured specification of arbitrary predicates. Although the assertion expression capabilities of SEQUEL and INGRES are "complete", they do not allow for the analysis of the types of possible assertions.

Furthermore, the assertions of SEQUEL and INGRES are rather inflexible with regard to when they are to hold, and what action is to occur if they do not. In SEQUEL

and INGRES, if a data base change is specified which would cause some assertion to be violated, the data base change is immediately rejected and an error signaled [Eswaran 1975], or the data base change is modified such that the assertion will be satisfied [Stonebraker 1975c].

In response to this latter objection, a relation constraint is herein defined as an abstract statement, having three components:

1. the assertion (a property), which is a predicate on the state of the data base or on transitions between data base states,
2. the validity requirement, which specifies the occasion(s) at which the assertion is to hold,
3. the violation-action, which is the action that is to occur if the assertion is not satisfied at a time when it should be.

In response to the former objection, a detailed classification of relation constraints is presented below. The emphasis is placed on providing a structured framework, which may be used to construct a high level, abstraction-based, well-directed, and disciplined relation constraint specification methodology. In so doing, a principal goal is to impose some structure on the problem of semantic errors in data bases. In this approach, it is important to keep "an eye toward implementation", although no specific implementation considerations are included in this thesis.

6.1. Whither Assertion Structure?

We subscribe to the view that the assertion component of a data base relation constraint should not be viewed as an arbitrary predicate of the first-order predicate calculus, ranging over tuples of the relations of a data base. Rather, every assertion should have a well-defined, uniform structure. There are several advantages to taking a

disciplined approach to assertion expression:

1. It provides the data base administrator (or other authority responsible for expressing the constraints) with a conceptual framework in terms of which to organize his thinking and structure the formulation of assertion specifications. Reducing abstract, problem-oriented limitations on configurations of the application environment to concrete restrictions on values in the data base is essentially a programming problem. By providing the "programmer" with a theoretical and general framework for his problem, it is possible to significantly ease his task.

2. The issues of constraint specification which are ancillary to assertion expression, namely the validity requirement and violation-action, cannot be satisfactorily addressed in the absence of the kind of structure proposed herein. The degree to which a semantic integrity subsystem can respond "intelligently" to a constraint violation depends upon how well the formulation of the constraint captures the intent of its expressor.

3. A useful conceptual framework for assertions will provide some measure of the complexity of individual assertions, providing their expressor with a guide to the cost of their implementation. Indeed, the structure of an assertion can be used by an implementation facility as a guide to the strategy for the implementation of its checking.

It is important to note that insuring that there is a single, unique specification of a given conceptual constraint is not a major objective here. Rather, the emphasis is placed on encouraging a "reasonable" formulation, one which accurately models the application environment abstraction and which is useable by an implementation facility.

6.2. Relation Constraint Assertions

The assertion component of a relation constraint is a logical predicate on the state of the data base or transitions between data base states. It expresses some semantic property of the data base.

Each assertion is either a simple assertion or a combination of simple assertions (a derived assertion). Simple assertions may be combined using boolean operators and other connectors (such as "if then else"). The remainder of this section deals with simple assertions; the generalization to derived assertions is more-or-less straightforward. When no ambiguity is possible, "assertion" will be used in place of "simple assertion".

6.2.1. Simple Assertions

Every (simple) assertion may be viewed as delimiting certain values of the data base in terms of certain others. That is, an assertion does not merely express some relationship among different values in the data base. Rather, it singles out certain values, and identifies them as being the constrained data of the predicate. The predicate delimits the legal values of the constrained data in terms of the constraining data. Thus, every assertion constrains some data with respect to some other; the two are not being bilaterally restricted.

As a consequence, there are two distinct steps in the process of stating an assertion:

1. The data that is being constrained is described. This description is accomplished in two sequential substeps, in which the following are identified:

    a. the set of all data objects in the data base that are being restricted (the constrained collection),

    b. the precise aspect of each of these data objects that is being delimited (the restricted expression).

Part a of step 1 utilizes data selection predicates. The predicate expression

capabilities of any data selection or query language may be adapted to accomplish this task [Chamberlin 1974b, Chamberlin 1975, Codd 1971a, Codd 1971d, Hall 1975, McLeod 1976c, Held 1975b, Zloof 1974, Zloof 1975a]. For example, consider the assertion that the salary of each employee in the sales department is less than the salary of his manager. Here, the constrained collection consists of those tuples in relation EMP which have "sales" in the Department column. The restricted expression is the Salary entry of each such tuple. The necessity of first identifying the constrained collection and then the restricted expression is occasioned by more rich and complex assertions, as discussed below.

2. The actual predicate of the assertion is stated, which asserts a restriction on the value of the restricted expression for each member of the constrained collection. The predicates used therein are called assertion predicates. In general, this restriction depends on other data in the data base. The other data which participates in the assertion is called the constraining data, and the expression which computes the precise delimiting value is called the restricting expression. For example, for the assertion above, the constraining data (for each tuple) is the tuple in relation EMP whose Name entry equals the Manager entry of the constrained tuple; the restricting expression is the Salary entry of the constraining tuple.

Figure 6-1 contains some examples of simple assertions. For each assertion, the constrained collection and assertion predicate are identified. Note that the "language" used to specify the assertion predicates is intended only to be illustrative, but is more-or-less consistent with the "level" of (and directly translatable into) relational data selection languages such as SEQUEL, QUEL, and Query by Example.

6.2.2. Identification of the Constrained Collection

As introduced above, the first step in the specification of an assertion is the identification of the constrained collection: that which is conceptually being delimited by the assertion. In general, the constrained collection is a collection of data objects, and the assertion applies to each of them. In this sense, every assertion is in effect an assertion schema, which is instantiated for each element of the constrained collection.

An assertion may either express a property of an individual tuple (a tuple assertion), or a property of a set of tuples considered as a whole (a set assertion). In figure 6-1, examples 1-4 are tuple assertions, while examples 5-8 are set assertions.

The constrained collection for a tuple assertion is a collection of tuples, to each of which the assertion applies. The constrained collection for a set assertion, similarly, is a collection of sets of tuples. The set assertion applies to each tuple set in the constrained collection. An important (and frequent) special case of a set assertion is that in which the constrained collection consists of a single set. Note the difference between this special case and a tuple assertion: in the former, the assertion applies to the tuple set as a whole, while in the latter it applies to each individual member of it. Thus, in example 1, the constrained collection has many elements, each of which is a tuple of the EMP relation; in example 5, the constrained collection consists of a single element, which is the entire EMP relation; in example 6, the constrained collection has several elements, each of which is a subset of the EMP relation.

Both for tuple and set assertions, defining the constrained collection begins with identifying some set of tuples (called the underlying relation of the assertion). This tuple set can then be manipulated by means of data selection predicates, to ultimately define the constrained collection.

The underlying relation of an assertion need not be a relation defined as part of the

data base. In general, it may be any of the following:

1. a base relation (a relation explicitly present in the set of data base relations),
2. the cross product of two or more base relations,
3. the union of two or more base relations,
4. the cross product of two or more relations of types 1 and 3, at least one of which is not a base relation,
5. any relation which can be defined in terms of base relations, not included in the above (these relations may be constructed using the various selection criteria and retrieval operators of a data selection language).

For example, EMP is a relation of type 1, EMP cross BUDGET is of type 2. An example of a relation of type 3 would be the union of relations CURRENT_EMP and OLD_EMP (where both have the same structure as EMP). An example of a relation of type 5 is SAL_TOTAL (Department, Sum_salaries), where Sum_salaries is the sum of the salaries of employees working for the associated department.

The foregoing classification of underlying relations is in order of increasing complexity, and exhibits the different kinds of relations to which assertions may apply. It is important to observe that an assertion need not apply to a relation explicitly present in the data base, but may hold for a derived relation.

Once the underlying relation is defined, the precise specification of the constrained collection can be accomplished. In the case of tuple assertions, the constrained collection is obtained from the underlying relation by means of data selection predicates. The complexity of the selection process can be described in terms of the operators of the data selection language. Selection of the constrained collection is a problem in the specification of a relation.

However, in the case of set assertions, there is a need to specify a collection of tuple

sets; each such set is a member of the constrained collection. For illustration, consider the following tentative taxonomy of the first stage of the specification process for a constrained collection which consists of tuple sets:

1. The constrained collection may contain a single set of tuples, selected from the underlying relation. (simple set)

2. A set of tuples may be selected from the underlying relation, and then divided into groups, e.g., by common value in one or more columns or by intervals of column values (such as 21 < Age < 30, 31 < Age < 40, etc.). Certain of these groups may then be chosen based on properties they possess. The constrained collection is thus a collection of tuple sets, namely the groups that were so chosen. The assertion then applies to each tuple set in the constrained collection. (grouped set)

3. A set of tuples may be selected from the underlying relation, and those subsets of it which satisfy a specified property are chosen. An example of such a property might be that the number of tuples in the subset equals three. These chosen subsets comprise the constrained collection, and the assertion is applied to each of them. (property-defined set)

There is a noticeable degree of flexibility in the foregoing framework for identifying the constrained collection, in that it does not impose a rigid specification methodology on the expressor of assertions. The criterion of completeness would not demand all the options for the underlying relation allowed above; it is clear that any assertion can be satisfactorily specified by letting the underlying relation be the cross product of all the base relations and performing various operations thereon to compute the constrained collection. However, in many instances such an "all-at-once" approach would be cumbersome and unnatural. It might be more convenient to follow a "top-down", step-by-step approach and define a sequence of derived relations, the last of which is the underlying relation. This can

facilitate the straightforward expression of the assertion.

Consider the following assertion: the sum of salaries of employees of each department is less than the budget of that department. An all-at-once approach to expressing this assertion would proceed to identify the constrained collection as the set of tuples in EMP, grouped by common Department (grouped set). The restricted expression would be the sum of the Salaries (for each group). The assertion predicate is then "sum(Salary) < BUDGET.Salary_budget where BUDGET.Department = common_value_of(Department) (in the constrained tuple set)". Thus the constraining data is the tuple in BUDGET having the Department column entry equal to the common value of the entries in the Department column for the constrained tuple set, and the restricting expression is the Salary_budget column entry of the constraining tuple.

A top-down, step-by-step approach to the expression of the above assertion may proceed by noting that the assertion could be expressed as a tuple assertion, if there existed a relation of the form DEPARTMENTS (Department, Sum_of_emp_salaries, Salary_budget). If such a relation existed, the constrained collection would be each tuple in relation DEPARTMENTS. The restricted expression would be the column entry Sum_of_emp_salaries. The assertion predicate would be "Sum_of_emp_salaries < Salary_budget". Here the restricting expression is the column entry Salary_budget in the constrained tuple, and the constraining data is the constrained tuple itself.

However, the relation DEPARTMENTS does not exist. Consequently, it is necessary to specify how it is to be derived from existing base relations. The underlying relation of the constrained collection is thus a derived relation, i.e., the relation DEPARTMENTS. A data selection language would be used to construct this derived relation; for example, the specification could be in a SEQUEL-like language

```
DEPARTMENTS (Department, Sum_of_emp_salaries, Salary_budget) =
    select EMP.Department, sum(EMP.Salary), BUDGET.Salary_budget
    from EMP, BUDGET
    where EMP.Department = BUDGET.Department
    group by EMP.Department
```

6.2.3. Tuple Assertions

It is now appropriate to examine more closely the structure of tuple assertions. In this case, the constrained collection is a collection of tuples, obtained from the underlying relation by the application of data selection predicates. The assertion predicate then applies to each individual tuple in the constrained collection. Tuple predicates are used to specify tuple assertions. The restricted expression defines that aspect of each constrained tuple that is being delimited. In the simplest case, the restricted expression is some column name of the underlying relation. More generally, it may be an expression: an appropriate combination of column names, system-provided operators, and user-defined operators.

It may be possible to formulate a given conceptual assertion in different ways, with different restricted expressions. For example, though the tuple assertions "Credit_line - Debt < 50000" and "Credit_line < Debt + 50000" are logically equivalent, in the former case the restricted expression is "Credit_line - Debt", while in the latter case it is just "Credit_line". This flexibility enables the assertion expressor to precisely identify which data values are to be regarded as dominant, and which as subordinate. In the first case, it is a combination of the entries Credit_line and Debt that is being delimited, while in the latter case it is simply the Credit_line entry. This distinction contributes to the abstraction power of assertion expression, and has implications for the implementation of constraints and for the actions that are to be taken upon the detection of an assertion violation.

The value which delimits the restricted expression is the restricting expression, which is computed from some data values which may reside anywhere in the data base. In particular, these data values (the constraining data) may be outside the constrained tuple.

Tuple predicates may be classified on the basis of the relationship between the constrained collection and the constraining data:

1. A tuple predicate is local (L) if the constraining data is present in the constrained tuple. That is, for a local tuple predicate, all data referenced in the predicate is within the constrained tuple itself.

2. A tuple predicate is nonlocal independent (NI) if the constraining data is data selected from elsewhere in the data base, but whose selection does not depend on any data in the constrained tuple.

3. A tuple predicate is nonlocal dependent (ND) if the selection of the constraining data does depend on data in the constrained tuple.

In figure 6-1, examples 1 and 4 involve L-type tuple predicates, example 2 is an NI-type tuple predicate, and example 3 is an ND-type tuple predicate.

This classification is in order of increasing complexity. For L-type tuple predicates, one has only to look at the constrained tuple to determine the restricting expression; the constraining data is present in the constrained tuple itself. For type-NI tuple predicates, this is no longer the case. The restricting expression is now computed from data arbitrarily located in the data base, not confined to the constrained tuple. However, the data from which the restricting expression is computed is the same for each tuple in the constrained collection. Thus the restricting expression admits of a one-time computation, with the result being used for each constrained tuple. For type-ND tuple predicates, the computation of the restricting expression depends on data in the constrained tuple. It is therefore necessary to recompute the restricting expression for each individual constrained tuple.

There are two dimensions by which we classify local tuple predicates. The first dimension measures the complexity of the restricting expression, and has three levels:

1. The restricted expression is compared via a scalar comparator to a constant, a single

column entry from the constrained tuple, or an expression involving several column entries from the constrained tuple. (types 1-3)

2. The restricted expression is compared via a set comparator to a set of constants, a set of column entries from the constrained tuple, a set of single-valued expressions computed from entries from the constrained tuple, or some expression which yields a set of values and depends on entries in the constrained tuple. (types 4-7)

3. The restricted expression is compared via a set comparator to a set of constant tuples, a set of tuples involving entries from the constrained tuple, a set of tuples composed of single-valued expressions computed from entries from the constrained tuple, or some expression which yields a set of tuples and depends on entries in the constrained tuple. (types 8-11)

The second dimension reflects the complexity of the restricted expression, and also has three levels:

a. For types 1-7, the restricted expression is a column entry in the constrained tuple. For types 8-11, it is a subtuple of the constrained tuple.

b. The restricted expression is a single-valued expression. For types 1-7, the restricted expression is computed from column entries in the constrained tuple, and yields a scalar value. For types 8-11, it yields a tuple composed of such column entry expressions.

c. The restricted expression is a set-valued expression. For types 4-7, it yields a set of scalars. For types 8-11, it yields a set of tuples. (This level does not apply to types 1-3.)

Figure 6-2 illustrates this classification for local tuple predicates of types 1a-11a. Consider the relation R (A, B, C, D, E, F) (where columns A, B, and C have underlying domain real number and columns D, E, and F have underlying domain character string). Some examples of local tuple predicates may be classified, as follows:

1. A < 15 (1a),

2. A < B (2a),

3. A < B/C (3a),

4. A is in {"x", "y", "z"} (4a),

5. A is in {"x", E, F} (6a)

(This means that, for each constrained tuple, the entry in column A is in the set containing the constant "x" and the entries in columns E and F.),

6. (D, E) is in {("x", "y"), ("z", F)} (10a)

(This means that, for each constrained tuple, the subtuple consisting of the entries from columns D and E equals either the tuple ("x","y"), or a tuple whose first component is "z" and whose second component is the F entry of the constrained tuple.),

7. A + B < C (2b),

8. A + B is in {C + 1, C + 2, C + 3} (6b),

9. {D, E} intersect {"w", "x"} contains {"y", "z"} (4c)

(This means that the intersection of the sets consisting of the entries in columns D and E and the constants "w" and "x", is a superset of the set containing the constants "y" and "z".).

As for local tuple predicates, nonlocal tuple predicates may be classified on two dimensions. The first dimension again consists of three levels:

1. The restricted expression is compared via a scalar comparator to a a single-valued expression, which yields a scalar value (and which is computed from data elsewhere in the data base). (type 1)

2. The restricted expression is compared via a set comparator to a set-valued expression, which yields a set of scalars. (type 2)

3. The restricted expression is compared via a set comparator to a set-valued expression, which yields a set of tuples. (type 3)

Again, the second dimension consists of three levels:

a. For types 1-2, the restricted expression is a column entry. For type 3, it is a tuple of entries which constitutes a subtuple of the constrained tuple.

b. The restricted expression is a single-valued expression. For types 1-2, this expression is computed from entries in the constrained tuple, and yields a scalar. For type 3, it yields a tuple composed of such column entry expressions.

c. The restricted expression is a set-valued expression. For type 2, it yields a set of scalars. For type 3, it yields a set of tuples. (This level does not apply to type 1.)

Figure 6-3 illustrates this classification for nonlocal tuple predicates of types 1a-3a. Note that the computation of the restricting expression (scalarval or setval) is independent of the constrained tuple for NI-type tuple predicates, but dependent for ND-type predicates. The data selection language must now serve the added role of identifying the constraining data. For this reason, the classification is coarser for nonlocal tuple predicates than for local tuple predicates.

6.2.4. Set Assertions

For set assertions, the constrained collection is a collection of tuple sets, obtained from the underlying relation, as discussed in section 6.2.2. The assertion predicate then applies to each tuple set in the constrained collection. Set predicates are used to specify set assertions. The restricted expression is that aspect of each constrained tuple set that is being delimited. In the simplest case, the restricted expression is the set of entries in some column of the underlying relation (e.g., the set of Salary entries in EMP). More generally, it may be an expression: an appropriate combination of column names, system-provided operators, and

user-defined operators. These operators include aggregate arithmetic operators which are applied to sets of values.

As for tuple assertions, the restricting expression is the value that delimits the restricted expression. The constraining data may be, in general, data anywhere in the data base. Again, as for tuple assertions, it may be possible to express a given conceptual set assertion in several ways.

Set predicates may be classified on the basis of the relationship between the constrained collection and the constraining data:

1. A set predicate is local (L) if the constraining data is present in the constrained tuple set. That is, the restricting expression may be computed solely from the constrained tuple set.

2. A set predicate is nonlocal independent (NI) if the constraining data is data selected from elsewhere in the data base, but where this selection does not depend upon the constrained tuple set.

3. A set predicate is nonlocal dependent (ND) if the selection of the constraining data does depend upon the constrained tuple set.

In figure 6-1, examples 6 and 8 are L-type set predicates, and examples 5 and 7 are NI-type set predicates.

As for tuple predicates, there are two dimensions on which local set predicates may be classified. One dimension reflects the complexity of the restricting expression, and the other reflects the complexity of the restricted expression. The first dimension has four levels:

1. The restricted expression is compared via a scalar comparator to a constant, an aggregate function of the entries in some column of the constrained tuple set, or an expression involving several such aggregates. (types 1-3)

2. As in 1, except that the aggregate functions in the constraining expression are not computed for a set of scalars, but for a set of tuples; namely, the collection of subtuples obtained by projecting the constrained tuple set onto two or more columns. (types 4-6)

3. The restricted expression is compared via a set comparator to a set of constants, the set of entries in some column of the constrained tuple set, or an expression involving several such sets. (types 7-9)

4. This is analogous to 3 in the same way that 2 is analogous to 1. That is, the restricting expression does not deal with scalars, but with sets of subtuples of the constrained tuple set. (types 10-12)

The second dimension consists of two levels:

a. For types 1-6, the restricted expression is an aggregate function. For types 7-12, it is an instantiation of the function "set", which generates the set of values in some column or the set of subtuples for some group of columns, taken over the constrained tuple set.

b. For types 1-6, the restricted expression is a single-valued expression computed from two or more of the aggregate functions described above. For types 7-12, it is a set-valued expression, computed from two or more instantiations of "set", as described above.

A special type of local set predicates, the column relationship predicates, are not included in the above scheme. Column relationship predicates are used to express properties such as one-to-one correspondences and functional dependencies. To state a column relationship predicate, two groups of column names from the constrained tuple set are specified. The relationship between these two groups of columns is then stated. For example, one may state that for the relation R (A, B, C, D, E, F), there is a one-to-one

correspondence between the column A and the column group (B, C). This means that there is a one-to-one relationship between the entry in column A and the subtuple formed from the entries in columns B and C. Note that column relationship predicates are always local.

Figure 6-4 illustrates this classification for local set predicates, types 1a-16a. For example, for the relation R (A, B, C, D, E, F) (where columns A, B, and C have underlying domain real number and columns D, E, and F have underlying domain character string), various local set predicates may be classified, as follows:

1. avg(A) < 15 (1a),

2. avg(A) < sum(B) (2a),

3. count(D, E) < 50 (4a)

(This means that the number of tuples in the relation formed by projecting the constrained tuple set on columns D and E is less than 50.),

4. set(D) contains {"x", "y", "z"} (7a),

5. set(D) properly contains set(E) union {"y", "z"} (9a),

6. set(D, E) is in {("w", "x"), ("y", "z")} (10a)

(This means that the set of tuples obtained by projecting on columns D and E is a subset of the set of constant tuples containing ("w", "x") and ("y", "z"),

7. D one-to-one (E, F) (14a),

8. set (D) union set (E) is in set (F) (8b).

Nonlocal set predicates may be similarly classified. The first dimension has three levels:

1. The restricted expression is compared via a scalar comparator to a single-valued expression, which yields a scalar value (and which is computed from some data in the data base) (types 1-2).

2. The restricted expression is compared via a set comparator to a set-valued

expression, which yields a set of scalars. (type 3)

3. The restricted expression is compared via a set comparator to a set-valued expression, which yields a set of tuples. (type 4)

The second dimension consists of two levels:

a. For types 1-2, the restricted expression is an aggregate function. For types 3-4, it is an instantiation of the function "set", which generates the set of values in some column or the set of subtuples for some group of columns, taken over the constrained tuple set.

b. For types 1-2, the restricted expression is a single-valued expression computed from two or more of the aggregate functions described above. For types 3-4, it is a set-valued expression, computed from two or more instantiations of "set", as described above.

Figure 6-5 illustrates this two dimensional classification for types 1a-4a. Note that the computation of the restricting expression (scalarval or setval) is independent of the constrained tuple set for NI-type set predicates, but dependent for ND-type predicates.

6.2.5. Scope of Assertions

It was stated in section 6.2.2 that each assertion is actually an assertion schema: an assertion is instantiated for and applies to each element of the constrained collection. But there is another sense in which an assertion may be viewed as a schema. This is by allowing described rather than explicit references to relation and column names within an assertion.

It may be desirable to state a "second order" assertion, e.g., each column in some relation of the data base which has underlying domain NAME must be a subset of the Name column in relation EMP. This may be handled by allowing column names (and

relation names) to be variables which range over the set of all columns or relations in the data base (or some specified subset thereof). This is basically a universal quantification of second order.

Without proposing a specific detailed solution to this problem of explicit scope vs. described scope, we may observe that such a solution must facilitate a second order quantification, on a level above the constrained collection. Consider the assertion that, for each column in the data base named Ci, every pair of entries in this column sums to less than 100. Here the constrained collection is a set of pairs of tuples. The property must hold for each element of the constrained collection. Furthermore, the assertion actually applies to each element in a set of constrained collections, viz., one such constrained collection for each column (in the data base) which is named Ci.

It has been stated that the scope of a relation constraint assertion can either be explicit (apply to relations and columns which are constants) or described (apply to relations and columns which are variables whose ranges are described). It is certainly valid to question the desirability and practicality of assertions with described scope, and we shall not take a position on this matter here. Rather, for the purposes of the remainder of this thesis, it is sufficient to assume that we are dealing with assertions having explicit scope, although we believe that the extension to assertions having described scope is straightforward.

6.3. Relation Constraint Validity Requirement

Another component of a relation constraint is the validity requirement(s): the occasion(s) at which the assetion component of the constraint must hold.

One possibility is that an assertion must hold at all times, and consequently must be checked after any data base change that may cause its violation. Such assertions must

theoretically be checked (verified) after every primitive data base change (such as update, insert, or delete tuple). Assertions actually need to be checked only if some value(s) are changed which may cause the assertion to be violated. Some success has been achieved in automatically determining when an assertion actually needs verification [Eswaran 1975, Stonebraker 1975c].

In some cases, it is necessary to specify than an assertion need not hold during some complex data base transaction(s), because it may not be meaningful to verify the assertion until after the transaction(s) are completed. Such assertions are checked only at the end of these transactions.

Suppose, for example, that there is an assertion for the example data base of figure 1 which states that exactly two employees in the sales department have a salary of more than $15,000. Assume that at some time the assertion holds, as employees "Smith" and "Jones" both have salary $20,000 and work in the sales department. It is now desired to transfer employee "Smith" out of the sales department, replacing him with employee "Davis" (with salary $30,000). If the primitive operations update row, insert row, and delete row are the only operations available and the assertion is checked after each primitive operation, the desired change cannot be legally accomplished. Thus the verification of this assertion must be deferred until the entire transaction (which consists of two primitive operations) is completed.

Consequently, it can be semantically necessary and/or desirable for the constraint expressor to specify precisely when an assertion is to be checked. For reasons of efficiency, it is also important to have the ability to specify that an assertion need only be checked at certain limited times, because verifying it after every data base change that could cause its violation might be catastrophically expensive.

Accordingly, the validity requirement of a relation constraint should be expressed in

terms of structured operations. For example, the validity requirement of some assertions might be that the assertion is to be checked after operation raise-salary. Each relation constraint validity requirement should consist of a list of structured operations after which the assertion component is to be checked. The special validity requirement "always" has the function of assuring that the assertion will be checked after any data base change that may cause its violation.

It may be necessary to check one or more relation constraint assertions after each data base change is attempted (by a structured operation). The simplest type of data base change is a primitive update, insert, or delete tuple operation. Slightly more complex is the set-oriented tuple update, insert, or delete which may be expressed in the high level nonprocedural data selection and modification language (e.g., SEQUEL). Since structured operations are hierarchically organized, it may be necessary to check some assertions after each hierarchic structured operation. Consider, for example, the structured operation A, which is defined to have the effect of executing a delete tuple operation, followed by the execution of operation B. Operation B consists of a single update tuple operation. It may then be necessary to check some assertions after the delete tuple operation, after operation B, after the update tuple operation (in B), and after operation A.

A special treatment of "null" (undefined) values as column entries is required. As noted by Eswaran and Chamberlin [Eswaran 1975], the checking of a relation constraint assertion should be such that the presence of "null" values should never cause the assertion to succeed if it would otherwise fail (be violated), and should never cause it to fail if it would otherwise succeed. An exception to this rule is made for assertions which explicitly reference "null" values (e.g., "Sex = null").

6.4. Relation Constraint Violation-Action

Associated with every occasion at which an assertion is to be checked, is a violation-action to be taken if the assertion is not satisfied upon attempted verification. Several types of violation-action can be specified:

1. An error can be signalled, and the requested data base change rejected. A message is issued informing the user of the problem; the nature of this message may be explicitly specified as a part of the violation-action, or it may be chosen by the system.

2. A warning can be issued, but the illegal data base change allowed. The user may be warned with a system-generated message, or a message specified as part of the violation-action. The warning may be persistent, in which case it appears whenever the potentially bad data is referenced.

3. A corrective action can be specified, which attempts to repair the error; the assertion is then rechecked. This approach may be dangerous, but is appropriate in some cases. There are several types of corrective action:

    a. a substitute value may be specified to replace the offending data,

    b. a structured operation may be performed,

    c. an external procedure may be called.

If a corrective violation-action is attempted, the relation constraint assertion which caused its invocation is rechecked after the corrective action is performed. It is intended that corrected value and structured operation corrective actions handle the bulk of the corrective violation-action needs. However, it is possible to call an external procedure (which is written in some high level general purpose programming language) as a corrective action. This external procedure receives no special privliges with regard to data base interaction. There are of course other

problems which result from permitting such external procedures to be used, which are similar to those discussed in the context of domain definition violation-action (see section 3.4). (A more far-reaching set of problems of this type is discussed by Minsky [Minsky 1976].)

The actual interface which reports relation constraint violations to the user should actually allow this user to control the violation-action. The user should be consulted, if appropriate. For instance, assume that the user wishes to perform an operation which gives employee "Jones" a 10% raise in salary. Assume also that there is a relation constraint assertion which states that the sum of salaries of all the employees in each department of the company must be less than the budget of that department. Suppose also that this assertion would be violated if the salary of "Jones" is increased by 10%. A reasonable violation-action might be to raise the salary of "Jones" to its maximum permissible value, while reporting this to the user and asking for approval before actually performing the action.

In this scheme, the violation-actions are associated with the assertion; they are part of the relation constaint. This means that violation-action information is not a part of the specification of the structured operations. All information regarding the checking of an assertion is localized in the relation constraint. This has the desirable effect of eliminating the arbitrary procedural embedding of violation-action information.

6.5. Implementation Considerations

A relation constraint language processor may be used to "compile" relation constraints into an internal form. Relation constraints may be added to and deleted from a data base. (A constraint may be changed by deleting it and adding a revised version.) Adding a relation constraint consists of its compilation and initial checking. Normally, the constraint

must be satisfied when it is added to the data base.

The internal form into which a relation constraint is compiled is used by the semantic integrity subsystem to check the integrity of the data base, and to take appropriate action which violations are detected. Moreover, the integrity subsystem manages all four aspects of semantic integrity, as discussed above and in chapter 7.

6.6. Remarks

The principal purpose of this chapter has been to impose some structure on the problem of relation constraint specification in the context of the semantic integrity of a relational data base. Important issues to be considered in future work include:

1. a detailed analysis of the applicability of specific high level, nonprocedural data selection languages to assertion specification (e.g., SEQUEL, QUEL, or Query by Example),
2. a complete description of a disciplined specification methodology for relation constraints (including detailed example(s) of relation constraint specification),
3. specifications of the user interface of the semantic integrity subsystem, vis-a-vis relation constraints,
4. an analysis of the impact of the semantic integrity subsystem on other aspects of the data base system (e.g., data security),
5. an assessment of the ramifications of various problems concerning relation constraints, including:
    a. redundancies,
    b. contradictions,
    c. circularities (because of corrective action side effects),
6. a study of implementation techniques for relation constraint checking.

7. ON THE DESIGN OF A SEMANTIC INTEGRITY SUBSYSTEM

The purpose of this chapter is to present some brief comments on several important aspects of the design of a semantic integrity subsystem. The purpose of such a subsystem is to manage the semantic integrity of a data base, as indicated by the semantic integrity specifications for that data base.

7.1. Components of a Semantic Integrity Subsystem

We propose that a semantic integrity subsystem possess four principal components:

1. The semantic integrity language processors translate the specifications in the high level semantic integrity languages into internal forms useful to the semantic integrity subsystem. As discussed in this thesis, there are four semantic integrity languages, for domain definition, relation structure, structured operations, and relation constraints. (Actually, these four langauges may be viewed as sublanguages of a single semantic integrity language.)

2. The semantic integrity checker determines which domain definitions and relation constraints need to be checked after a given data base change is performed, and performs that checking.

3. The semantic integrity violation-action processor takes appropriate action when a domain definition or relation constraint is violated.

4. The relation constraint compatibility checker is responsible for insuring that the set of relation constraints currently extant for a data base is free from contradictions and other undesirable properties. The compatability checker may be called by the relation constraint language processor when adding a new relation constraint, to make sure that it is acceptable to add it. The problem of designing and implementing a

compatability checker involves general techniques of deductive inference, automated theorem provers, etc. Only a very limited compatability checker could be practical at the present time.

7.2. The User's View of the Integrity Mechanism

It is extremely important to provide an effective user - data base system interface, especially with regard to the creation, maintenance, and reporting of semantic integrity information. There are actually three major types of users with which one needs to be concerned:

1. the data base administrator (DBA), which may in fact be a single person or many persons, whose job is to create and maintain the semantic integrity specifications,
2. the nonprogramming user, who deals with the data base by means of generalized data selection and modification languges (e.g. SEQUEL, QUEL, or Query by Example),
3. the applications program, which calls upon data base system facilities.

Of course, a single person may serve both as a DBA and a (nonprogramming) user. The distinction between nonprogramming users and applications programs is made in order to distinguish the types of communication with the semantic integrity subsystem which are necessary.

The DBA should be provided facilities which allow the following types of actions:

1. add relation,
2. delete relation,
3. add domain,
4. delete domain,
5. add structured operation,

6. delete structured operation,

7. add relation constraint,

8. delete relation constraint.

It should also be possible for a DBA to change the structure of relations, and modify the definition of domains, structured operations, and relation constraints. It is furthermore desirable to allow the DBA to ask questions about the semantic integrity specifications, especially the relation constraints. For example, it should be possible to ask which constraints may possibly be violated if an entry in a given column is changed, or which constraints have a given column entry as constrained data.

The nonprogramming user must be provided with high level reporting of semantic integrity violations and violation-actions. In general, a (nonprogramming) user sees a set of data structures (domains and relations), a set of structured operations, and a set of relation constraints. When a domain definition or relation constraint is found to be violated, the user is either informed of this fact or an automatic corrective action is attempted. In any case, it must be possible to provide the user with a high level "error message". The semantic integrity subsystem must not be completely silent (e.g., see [Stonebraker 1974d, Stonebraker 1975c]). It must also be possible for the user to interact with the semantic integrity subsystem to attempt to repair an error, should that be appropriate.

The applications program must be provided with capabilities similar to those for nonprogramming users, but all communication must be accomplished via procedure call and return, and message passing protocols.

7.3. Some Thoughts on Integrity Subsystem Implementation

Although a detailed investigation of implementation techniques for semantic integrity subsystems is an important research topic, little has been done on it to date. Stonebraker

and Wong [Stonebraker 1974d, Stonebraker 1975c] have proposed a very clean "query modification" approach to integrity checking, but this scheme has some limitations (e.g., some useful types of techniques for the optimization of integrity checking are not handled). Sarin [Sarin 1976] is currently investigating this topic in some detail. In this thesis, we are not principally concerned with the specifics of implementation techniques. However, we shall discuss a few important aspects of semantic integrity subsystem implementation.

First of all, it is important that a data base logging and backup facility exist. This is crucial in allowing the actions of a structured operation (transaction) to be "backed out" and "undone", if occasioned by the violation of a domain definition or relation constraint.

It is sometimes the case that a data base change will cause several domain definitions and relation constraints to be checked. (A data base change is accomplished by the invocation of a primitive or structured operation.) A scheme must be developed for determining in what order these are to be checked. One way to handle this is to assign priorities to domain definitions and relation constraints; this may be done by the DBA or automatically by the semantic integrity subsystem. Domain definitions should receive priority over relation constraints (since they are always checked after primitive operations), and the various types of relation constraints can be ordered by their complexity, importance, or some other metric.

Since relation constraint checking is potentially a costly undertaking, it is crucial that efficient checking techniques be developed. Much of the work on optimizing data selection and modifiction languages is relevant here. Heuristics may be developed for determining, on the basis of the patterns of data base interaction, which access paths and aids to maintain [Hammer 1976b]. One type of useful heuristic involves the maintenance of aggregate values. For example, if there is a relation constraint assertion which states that the sum of employee salaries is less than $100,000, it may be helpful to maintain the sum

and update it as necessary, rather than constantly recalculating it when the assertion is checked. Other types of heuristics may also prove useful, e.g., dealing with characteristics of individual types of physical storage devices (such as data clustering and page arrangement), or dealing with the maintenance and use of inversions (indices).

7.3.1. The Use of Inversions in Relation Constraint Checking (An Example)

As an example illustrative of the usefulness of inversions in relation constraint checking, consider an example assertion. Suppose that the assertion (for the example data base of figure 1-2) states that for each tuple B in relation BUDGET, the entry in the Salary_budget column (B.Salary_budget) is greater than or equal to the sum of the entries in the Salary column of the tuples in EMP (E1, ..., En) which have Department = B.Department. Several primitive operations which may require this assertion to be checked are listed below, along with the method by which the necessary checking may be accomplished and an indication of which inversions would be helpful in such checking:

1. for some tuple B in BUDGET, Salary_budget is changed:
    a. find all tuples in EMP (E1, ..., En) which have Department = B.Department,
    b. calculate S = E1.Salary + ... + En.Salary,
    c. check that S <= B.Salary_budget,

    useful inversions: Department in EMP (for step a),

2. for some tuple E in EMP, Salary is changed:
    a. find all tuples in EMP (E1, ..., En) which have Department = E.Department,
    b. calculate S = E1.Salary + ... + En.Salary,
    c. find the tuple in BUDGET (B) which has Department = E.Department,
    d. check that S <= B.Salary_budget,

    useful inversions: Department in EMP (for step a), Department in BUDGET (for

step c),

3. for some tuple in BUDGET (B), Department is changed:

(same as 1),

4. for some tuple in EMP (E), Department is changed,

(same as 2),

5. a new tuple is inserted into BUDGET (B),

(same as 1),

6. a new tuple is inserted into EMP (E),

(same as 2).

In this particular example, no checking needs to be done when tuples are deleted from EMP, since that can only cause the sum (S) to decrease. Of course, this is not true for all assertions involving sums of this type.

8. REMARKS AND DIRECTIONS

The major purpose of this thesis has been to provide a comprehensive, detailed analysis of the issues and problems associated with maintaining semantic integrity in a generalized (relational) data base system. The principal emphasis has been on the high level expression of semantic integrity specifications. The major portion of the work described herein has been concerned with providing a framework for semantic integrity specifications. Both the functional requirements for a solution to the semantic integrity problem and a specific approach to providing such a solution have been emphasized. An attempt has been made to indicate important directions for further work on semantic integrity.

By way of conclusion, there are several important general directions for the extension of the work described in this thesis. The following are most significant:

1. an analysis of important integrity specification language design issues (e.g., the usefulness of constructs in languages like SEQUEL, QUEL, and Query by Example, the adequacy of nonprocedural specification methodologies, the importance of iteration and recursion, etc.),

2. the complete design of a language for semantic integrity specification, including sublanguages for each of the four aspects of semantic integrity (in the relational data model),

3. the development of a well-directed, structured, disciplined approach to data base design (based on the semantic integrity framework),

4. a comprehensive example of the application of the semantic integrity specification methodology described herein to a "real" application domain,

5. the implementation of the semantic integrity subsystem outlined in this thesis,

6. an analysis of the cost of building, maintaining, and enforcing semantic integrity rules,

7. a study of the relationship of semantic integrity issues with those of security, concurrent consistency, and query processing (including the use of deductive techniques),

8. an evaluation of the ramifications of separating the four aspects of integrity to the extent described above (e.g., an analysis of whether it is necessary to allow the information within a domain definition to be referenced in relation constraint assertions), and a study of the appropriateness of this approach,

9. an evaluation of the applicability of a behavioral approach to the description of data semantics in an integrated data base environment,

10. the extension of the semantic integrity scheme to allow multiple "views" of a data base,

11. an evaluation of possible extensions to permit a nonabsolutist approach to integrity (involving the notions of quantized truth and confidence measures [Zadeh 1975]),

12. a study of the ability of the approach to the semantic integrity problem described in this thesis to improve the overall effectiveness of a data base system.

REFERENCES AND BIBLIOGRAPHY

[Abrial 1974]

Abrial, J. R., "Data Semantics", Data Base Management, North Holland, 1974.

[Allman 1975]

Allman, E., M. Stonebraker, and G. Held, Embedding a Relational Data Sublanguage in a General Purpose Programming Language, Electronics Research Laboratory Report ERL-M564, University of California; Berkeley CA, 10 October 1975.

[Allman 1976]

Allman, E., M. Stonebraker, and G. Held, "Embedding a Relational Data Sublanguage in a General Purpose Programming Language", Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure, Salt Lake City UT, 22-24 March 1976.

[Armstrong 1974]

Armstrong, W. W, "Dependency Structures of Data Base Relationships", Information Processing 74, North Holland, 1974.

[Astrahan 1975]

Astrahan, M. M. and D. D. Chamberlin, "Implementation of a Structured English Query Language", Proceedings of ACM SIGMOD International Conference on the Management of Data, San Jose CA, 14-16 May 1975.

[Bachman 1973]

Bachman, C. W., "The Programmer as Navigator", Communications of the ACM, Volume 16, Number 11, November 1973.

[Bernstein 1975]

Bernstein, P. A., J. R. Swenson, and D. C. Tsichritzis, "A Unified Approach to Functional Dependencies and Relations", Proceedings of ACM SIGMOD International Conference on the Management of Data, San Jose CA, 14-16 May 1975.

[Bjorner 1973]

Bjorner, D., E. F. Codd, K. L. Deckert, and I. L. Traiger, The Gamma-0 N-ary Relational Data Base Interface Specifications of Objects and Operations, IBM Research Report RJ1200, San Jose CA, 11 April 1973.

[Borgida 1975]

Borgida, A. T., Topics in the Understanding of English Sentences by Computer, Technical Report 78, Department of Computer Science, University of Toronto, Toronto, Canada, February 1975.

[Boyce 1973a]

Boyce, R. F. and D. D. Chamberlin, Using a Structured English Query Language as a Data Defnition Facility, IBM Research Report RJ1318, San Jose CA, 10 December 1973.

[Boyce 1973b]

Boyce, R. F., D. D. Chamberlin, W. F. King III, and M. M. Hammer, "Specifying Queries as

Relational Expressions: SQUARE", Proceedings of ACM SIGPLAN-SIGIR Interface Meeting, Gaithersburg MD, 4-6 November 1973.

[Boyce 1975]

Boyce, R. F., D. D. Chamberlin, W. F. King III, and M. M. Hammer, "Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage", Communications of the ACM, Volume 18, Number 11, November 1975.

[Bracchi 1972]

Bracchi, G. A., A. Fedeli, and P. Paolini, "A Language for a Relational Data Base Management System", Sixth Annual Princeton Conference on Information Sciences and Systems, Princeton NJ, 23-24 March 1972.

[Bracchi 1974]

Bracchi, G., A. Fedeli, and P. Paolini, "A Multi-Level Relational Model for Data Base Management Systems", Data Base Management, North Holland, 1974.

[Cardenas 1975]

Cardenas, A. F., "Analysis and Performance of Inverted Data Base Structures", Communications of the ACM, Volume 18, Number 5, May 1975.

[Chamberlin 1974a]

Chamberlin, D. D., R. F. Boyce, and I. L. Traiger, "A Deadlock-Free Scheme for Resource Locking in a Data Base Environment", Information Processing '74, North-Holland, 1974.

[Chamberlin 1974b]

Chamberlin, D. D. and R. F. Boyce, "SEQUEL: A Structured English Query Language", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, Ann Arbor MI, 1-3 May 1974.

[Chamberlin 1975]

Chamberlin, D. D., J. N. Gray, and I. L. Traiger, "Views, Authorization, and Locking in a Relational Data Base System", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

[Chan 1974]

Chan, A. Y., Automatic Selection of Inversions in an Integrated Data Base Environment, S. M. thesis proposal, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge MA, 18 December 1974.

[Chen 1975]

Chen, P. P. S., "The Entity-Relationship Model: Toward a Unified View of Data", ACM Transactions on Data Base Systems, Volume 1, Number 1, March 1976 (to appear).

[Codasyl 1971a]

Codasyl Committee on Data System Languages, Codasyl Data Base Task Group Report, ACM, New York NY, 1971.

[Codd 1970]

Codd, E. F., "A Relational Model for Large Shared Data Banks", Communications of the

ACM, Volume 13, Number 6, June 1970.

[Codd 1971a]

Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego CA, 1971.

[Codd 1971b]

Codd, E. F., "Further Normalization of the Data Base Relational Model", Courant Computer Science Symposia 6, New York NY, 24-25 May 1971, in Data Base Systems, Prentice Hall, 1971.

[Codd 1971c]

Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego CA, 1971.

[Codd 1971d]

Codd, E. F., "Relational Completeness of Data Base Sublanguages", Courant Computer Science Symposia 6, New York NY, 24-25 May 1971, in Data Base Systems, Prentice Hall, 1971.

[Codd 1974a]

Codd, E. F., "Recent Investigations in Relational Data Base Systems", Information Processing '74, North Holland, 1974.

[Codd 1974b]

Codd, E. F., "Seven Steps to Rendezvous with the Casual User", Proceedings of IFIP TC-2 Working Conference on Data Base Management Systems, Cargese, Corsica, 1-5 April 1974, North Holland, 1974.

[Codd 1974c]

Codd, E. F. and C. J. Date, "Interactive Support for Non-Programmers: The Relational and Network Approaches", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, Ann Arbor MI, 1-3 May 1974.

[Codd 1975a]

Codd, E. F., A List of References Pertaining to Relational Data Base Management, IBM Research Laboratory, San Jose CA, 1975.

[Codd 1975b]

Codd, E. F. (editor), "Implementation of Relational Data Base Management Systems", (Transcription of 1975 National Computer Conference Panel Discussion on Relational Data Base Management), FDT - Quarterly Bulletin of ACM SIGMOD, Volume 7, Nmuber 2, September 1975.

[Conway 1974]

Conway, R. W., W. L. Maxwell, and H. L. Morgan, "A Technique for File Surveillance", Information Processing '74, North Holland, 1974.

[Date 1971a]

Date, C. J. and P. Hopewell, "File Definition and Logical Data Independence", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego CA, 1971.

[Date 1971b]

Date, C. J. and P. Hopewell, "Storage Structure and Physical Data Independence", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego CA, 1971.

[Date 1972]

Date, C. J., "Relational Data Base Systems: A Tutorial", Proceedings of Fourth Annual Symposium on Computers and Information Science, Miami Beach FL, 14-16 December 1972, Plenum Press, 1972.

[Date 1974]

Date, C. J. and E. F. Codd, "The Relational and Network Approaches: Comparison of the Application Programming Interfaces", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, Ann Arbor MI, 13 May 1974.

[Date 1975]

Date, C. J., An Introduction to Data Base Systems, Addison-Wesley, 1975.

[Engles 1971]

Engles, R. W., "An Analysis of the April 1971 DBTG Report", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego CA, 1971.

[Eswaran 1974]

Eswaran, K. P., J. N. Gray, R. A. Lorie, and I. L. Traiger, The Notions of Consistency and Predicate Locks in a Data Base System, IBM Research Report RJ1487, San Jose CA, 30 December 1974.

[Eswaran 1975]

Eswaran, K. P. and D. D. Chamberlin, "Functional Specifications of a Subsystem for Database Integrity", Proceedings of International Conference on Very Large Data Bases, Framingham MA, 22-24 September 1975.

[Everest 1974a]

Everest, G. C., "Concurrent Update Control and Database Integrity", Proceedings of IFIP TC-2 Working Conference on Data Base Management Systems, Cargese, Corsica, 1-5 April 1974, North Holland, 1974.

[Everest 1974b]

Everest, G. C., "The Futures of Database Management", Proceedings of ACM SIGMOD Conference on Data Description, Access, and Control, Ann Arbor MI, 1-3 May 1974.

[Fadous 1975]

Fadous, R. Y. and J. Forsyth, "Finding Candidate Keys for Relational Data Bases", Proceedings of ACM SIGMOD International Conference on the Management of Data, San Jose CA, 14-16 May 1975.

[Fehder 1974]

Fehder, P., "HQL: A Set-Oriented Transaction Language for Hierarchically Structured Data Bases", Proceedings of ACM National Conference, San Diego CA, November 1974.

[Fernandez 1975]

Fernandez, E. B., R. C. Summers, and T. Lang, "Definition of Access Rules in Data Management Systems", Proceedings of International Conference on Very Large Data Bases, Framingham MA, 22-24 September 1975.

[Florentin 1974]

Florentin, J. J., "Consistency Auditing of Databases", The Computer Journal, Volume 17, Number 1, February 1974.

[Florentin 1976]

Florentin, J. J., "Information Reference Coding", Communications of the ACM, Volume 19, Number 1, January 1976.

[Fossum 1974]

Fossum, B. M., "Data Base Integrity as Provided for by a Particular Data Base Management System", Data Base Management, North Holland, 1974.

[Goldstein 1970]

Goldstein, R. C. and A. L. Strnad, "The MacAims Data Management System", Proceedings of ACM SIGFIDET Workshop on Data Description and Access, November 1970.

[Gosden 1974]

Gosden, J. A., "Large Scale Data Base Systems - Current Deficiencies and User Requirements, Data Base Management Systems, North Holland, 1974.

[Gotlieb 1975]

Gotlieb, L. R., "Computing Joins of Relations", Proceedings of ACM SIGMOD International Conference on the Management of Data, San Jose CA, 14-16 May 1975.

[Graves 1975]

Graves, R. W., "Integrity Control in a Relational Data Description Language", Proceedings of ACM Pacific Conference, San Francisco CA, 17-18 April 1975.

[Gray 1975]

Gray, J. N., R. A. Lorie, and G. R. Putzolu, "Granularity of Locks in a Shared Data Base", Proceedings of International Conference on Very Large Data Bases, Framingham MA, 22-24 September 1975.

[Grossman 1975]

Grossman, R. W., "Representing the Semantics of Natural Language as Constraint Expressions", Working Paper 87, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge MA, January 1975.

[Grossman 1976]

Grossman, R. W., Some Data-base Applications of Constraint Expressions, S. M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of

Technology, Cambridge MA, January 1976.

[Guttag 1976]

Guttag, J., "Abstract Data Types and the Development of Data Structures", Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure, Salt Lake City UT, 22-24 March 1976.

[Hall 1975]

Hall, P. A. V., S. J. P. Todd, and P. Hitchcock, An Algebra of Relations for Machine Computation, IBM Scientific Centre Report UKSC0066, Peterlee, England, January 1975.

[Hammer 1974]

Hammer, M. M., W. G. Howe, and I. Wladawsky, An Interactive Business Definition System, IBM Research Report RC4680, Yorktown Heights NY, 16 January 1974.

[Hammer 1975]

Hammer, M. M. and D. J. McLeod, "Semantic Integrity in a Relational Data Base System", Proceedings of International Conference on Very Large Data Bases, Framingham MA, 22-24 September 1975.

[Hammer 1976a]

Hammer, M. M. and D. J. McLeod, A Framework for Data Base Semantic Integrity Constraints, Very Large Data Bases Group Report, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge MA, January 1976.

[Hammer 1976b]

Hammer, M. M. and A. Y. Chan, "Index Selection in a Self-Adaptive Data Base Management System", Proceedings of ACM SIGMOD International Conference on the Management of Data, Washington D. C., 2-4 June 1976 (to appear).

[Hammer 1976c]

Hammer, M. M., "Error Detection in Data Base Systems", Proceedings of National Computer Conference, New York NY, 7-10 June 1976 (to appear).

[Hawkinson 1975]

Hawkinson, L., "The Representation of Concepts in OWL", Proceedings of Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, 3-8 September 1975.

[Hawley 1975]

Hawley, D. A., J. S. Knowles, and E. E. Tozer, "Database Consistency and the CODASYL DBTG Proposals, The Computer Journal, Volume 16, Number 3, November 1975.

[Hawryskiewycz 1972]

Hawryskiewycz, I. T. and J. B. Dennis, "An Approach to Proving the Correctness of Data Base Operations", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, November 1972.

[Hawryskiewycz 1973]

Hawryskiewycz, I. T., Semantics of Data Base Systems, Massachusetts Institute of

Technology Project MAC Technical Report TR-112, Cambridge MA, December 1973.

[Heath 1971]

Heath, I. J., "Unacceptable File Operations in a Relational Data Base", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego CA, 1971.

[Held 1975a]

Held, G. and M. Stonebraker, Storage Structures and Access Methods in the Relational Data Base Management System INGRES, Electronics Research Laboratory Report ERL-M505, University of California, Berkeley CA, 3 March 1975.

[Held 1975b]

Held, G., M. R. Stonebraker, and E. Wong, "INGRES: A Relational Data Base System", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

[Held 1975c]

Held, G., Storage Structures for Relational Data Base Management Systems, Electronics Research Laboratory Report ERL-M533, University of California, Berkeley CA, 11 August 1975.

[Hewitt 1971]

Hewitt, C. E., Procedural Embedding of Knowledge in PLANNER, Proceedings of International Joint Conference on Artificial Intelligence 2, September 1971.

[Housel 1976]

Housel, B. C. and N. C. Shu, "A High Level Manipulation and Query Language for Hierarchical Data Abstractions", Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure, Salt Lake City UT, 22-24 March 1976.

[IBM]
IBM, IMS/360 Application Description Manual, GH20-0765, White Plains NY.

[Jervis 1974]
Jervis, B. M., Query Languages for Relational Data Base Management Systems, S.M. Thesis, Department of Computer Science, University of British Columbia, Canada, May 1974.

[Joyce 1974]
Joyce, J. D., J. T. Murray, and M. R. Ward, "Data Management System User Requirements", Data Base Management Systems, North Holland, 1974.

[King 1974]
King, W. F. III, On the Selection of Indices for a File, IBM Research Report RJ1341, San Jose CA, January 1974.

[Liskov 1974]
Liskov, B. and S. Zilles, "Programming with Abstract Data Types", Proceedings of a Symposium on Very High Level Languages, Santa Monica CA, March 1974.

[Lorie 1974]

Lorie, R. A., XRM - An Extended (N-ary) Relational Memory, IBM Cambridge Scientific Center Technical Report 320-2096, Cambridge MA, January 1974.

[Machgeels 1976]

Machgeels, C., "A Procedural Language for Expressing Integrity Constraints in the Coexistence Model", Proceedings of IFIP TC-2 Conference on Modelling in Data Base Management Systems, Freudenstadt, W. Germany, 5-9 June 1976 (to appear).

[Marill 1975]

Marill, T. and D. Stern, "The Datacomputer: A Network Utility", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

[Martin 1975]

Martin, J. T., Computer Data-Base Orgranization, Prentice Hall, 1975.

[Maynard 1974]

Maynard, H. S., "User Requirements for Data Base Management Systems (DBMS)", Data Base Management Systems, North Holland, 1974.

[McDonald 1974a]

McDonald, N., M. Stonebraker, and E. Wong, Preliminary Design of INGRES Part I - Query Language, Data Storage and Access, Electronics Research Laboratory Report ERL-M435, University of California, Berkeley CA, 10 April 1974.

[McDonald 1974b]

McDonald, N. M., M. Stonebraker, and E. Wong, Preliminary Design of INGRES Part II - Protection, Concurrency and Graphics, Electronics Research Laboratory Report ERL-M436, University of California, Berkeley CA, 9 May 1974.

[McDonald 1974c]

McDonald, N. and M. Stonebraker, CUPID - The Friedly Query Language, Electronics Research Laboratory Report ERL-M487, University of California, Berkeley CA, 16 October 1974.

[McDonald 1975a]

McDonald, N. and M. Stonebraker, "CUPID: The Friendly Query Language", Proceedings of ACM Pacific Conference, San Francisco CA, 17-18 April 1975.

[McDonald 1975b]

McDonald, N. H., CUPID: A Graphics Oriented Facility for Support of Non-Programmer Interactions with a Data Base, Electronics Research Laboratory Report ERL-M563, University of California, Berkeley CA, 12 November 1975.

[McLeod 1974]

McLeod, D. J., Relational Data Management in Minicomputers, S.B. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge MA, February 1974.

[McLeod 1975]

McLeod, D. J. and M. J. Meldman, "RISS: A Generalized Minicomputer Relational Data

Base Management System", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

[McLeod 1976a]

McLeod, D. J., High Level Domain Definition in a Relational Data Base System, IBM Research Report RJ1718, San Jose CA, 9 February 1976.

[McLeod 1976b]

McLeod, D. J., "High Level Domain Definition in a Relational Data Base System", Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure, Salt Lake City UT, 22-24 March 1976.

[McLeod 1976c]

McLeod, D. J., Query by Example and SEQUEL: Translation and Compatibility, IBM Research Report RJ1730, San Jose CA, 1976.

[Meltzer 1973]

Meltzer, H. S., Current Concepts in Data Base Design, IBM Report to GUIDE 37 Information Systems Division, 2 November 1973.

[Minsky 1974a]

Minsky, N., "On Interaction with Data Bases", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, Ann Arbor MI, 1-3 May 1974.

[Minsky 1974b]

Minsky, N., Protection of Data Bases and the Process of User Data-Base Interaction, Department of Computer Science Technical Report SOSAP-TR-11, Rutgers University, New Brunswick NJ, September 1974.

[Mommens 1975]

Mommens, J. H. and S. E. Smith, "Automatic Generation of Physical Data Base Structures", Proceedings of ACM SIGMOD International Conference on the Management of Data", San Jose CA, 14-16 May 1975.

[Morgan 1970]

Morgan, H. L., "An Interrupt Based Organization for Management Information Systems", Communications of the ACM, Volume 13, Number 12, December 1970.

[MRI 1972]

MRI Systems Corporation, System 2000 General Information Manual, Austin TX, 1972.

[Mylopoulos 1975]

Mylopoulos, J., S. A. Schuster, and D. Tsichritzis, "A Multi-Level Relational System", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

[Nijssen 1974]

Nijssen, G. M., "Data Structuring in the DDL and Relational Data Model", Proceedings of IFIP TC-2 Working Conference on Data Base Management Systems, Cargese, Corsica, 1-5 April 1974, North Holland, 1974.

[Nordstrom 1976]

Nordstrom, B., "An Outline of a Mathematical Model for the Definition and Manipulation of Data", Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure, Salt Lake City UT, 22-24 March 1976.

[Notley 1972]

Notley, M. G., The Peterlee IS/1 System, IBM United Kingdom Scientific Center Report UKSC-0018, England, March 1972.

[Olle 1974]

Olle, T. W., "Current and Future Trends in Data Base Management Systems", Information Processing 74, North Holland, 1974.

[Ozkaran 1974]

Ozkaran, E. A., S. A. Schuster, and K. C. Smith, A Data Base Processor, Technical Report CSRG-43, University of Toronto, Toronto, Canada, November 1974.

[Ozkaran 1975]

Ozkaran, E. A., S. A. Schuster, and K. C. Smith, "RAP: As Associative Processor for Data Base Management", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

[Pfister 1974]

Pfister, G. F., The Computer Control of Changing Pictures, Technical Report TR-135, Project MAC, Massachusetts Institute of Technology, Cambridge MA, September 1974.

[Redell 1974]

Redell, D. D., Naming and Protection in Extendible Operating Systems, Technical Report TR-140, Project MAC, Massachusetts Institute of Technology, Cambridge MA, November 1974.

[Reisner 1975]

Resiner, P., R. F. Boyce, and D. D. Chamberlin, "Human Factors Evaluation of Two Data Base Query Languages: SQUARE and SEQUEL", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

[Robinson 1967]

Robinson, J. A., "A Review of Automatic Theorem Proving", Proceedings of Symposium in Applied Mathematics, American Mathematical Society, Providence RI, Volume 19, 1967.

[Robinson 1975]

Robinson, K. A., "Data Base -- The Ideas Behind the Ideas" The Computer Journal, Volume 18, Number 1, January 1975.

[Rothnie 1972]

Rothnie, J. B., The Design of Generalized Data Management Systems, Ph. D. thesis, Department of Civil Engineering, Massachusetts Institute of Technology, Cambridge MA, September 1972.

[Rothnie 1974]

Rothnie, J. B., "An Approach to Implementing a Relational Data Management System", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, Ann Arbor MI, 1-3 May 1974.

[Rothnie 1975]

Rothnie, J. B., "Evaluating Inter-Entry Retrieval Expressions in a Relational Data Base Management System", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

[Roussopoulos 1975]

Roussopoulos, N. and J. Mylopoulos, "Using Semantic Networks for Data Base Management", Proceedings of International Conference on Very Large Data Bases, Framingham MA, 22-24 September 1975.

[Sarin 1976]

Sarin, S. K., Design of a Semantic Integrity Subsystem for Relational Data Base Systems, S.M. Thesis Proposal, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge MA, 29 January 1976.

[Schlotnick 1975]

Schlotnick, M., "Secondary Index Optimization", Proceedings of ACM SIGMOD International Conference on the Management of Data, San Jose CA, 14-16 May 1975.

[Schmid 1975]

Schmid, H. A. and J. R. Swenson, "On the Semantics of the Relational Data Model",

Proceedings of ACM SIGMOD International Conference on the Management of Data, San Jose CA, 14-16 May 1974.

[Senko 1973]

Senko, M., E. Altman, M. Astrahan, and P. Fehder, "Data Structures and Accessing in Data Base Systems", IBM Systems Journal, Number 1, 1973.

[Senko 1975]

Senko, M. E., "Specifications of Stored Data Structures and Desired Output Results in DIAM II with FORAL", Proceedings of International Conference on Very Large Data Bases, Framingham MA, 22-24 September 1975.

[Sibley 1974]

Sibley, E. H., "Data Management System User Requirements", Data Base Management Systems, North Holland, 1974.

[Smith 1976]

Smith, J. M. and D. C. P. Smith, "A Semantics for Relational Data Bases Founded on Abstraction", Proceedings of ACM SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition, and Structure, Salt Lake City UT, 22-24 March 1976.

[Software AG 1974]

Software AG, ADABAS ADASCRIPT User's Manual, Reston VA, 1974.

[Steuert 1974]

Steuert, J. and J. Goldman, "The Relational Data Management System: A Perspective", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, Ann Arbor MI, 1-3 May 1974.

[Stonebraker 1974a]

Stonebraker, M. R., "The Choice of Partial Inversions and Combined Indices", International Journal of Computer and Information Science, Volume 3, Number 2, June 1974.

[Stonebraker 1974b]

Stonebraker, M. R., "A Functional View of Data Independence", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, Ann Arbor MI, 1-3 May 1974.

[Stonebraker 1974c]

Stonebraker, M. R., High Level Integrity Assurance in Relational Data Base Management Systems, Electronics Research Laboratory Report ERL-M473, University of California, Berkeley CA, 16 August 1974.

[Stonebraker 1974d]

Stonebraker, M. and E. Wong, Access Control in a Relational Data Base Management System by Query Modification, Electronics Research Laboratory Report ERL-M438, University of California, Berkeley CA, 14 May 1974.

[Stonebraker 1975a]

Stonebraker, M. R. and G. Held, Networks, Hierarchies, and Relations in Data Base Management Systems, Electronics Research Laboratory Report ERL-M504, University of California, Berkeley CA, 3 March 1975.

[Stonebraker 1975b]
Stonebraker, M. R., Getting Started in INGRES - A Tutorial, Electronics Research Laboratory Report ERL-M518, University of California, Berkeley CA, 23 April 1975.

[Stonebraker 1975c]
Stonebraker, M. "Implementation of Integrity Constraints and Views by Query Modification", Proceedings of ACM SIGMOD International Conference on the Management of Data, San Jose CA, 14-16 May 1975.

[Summers 1975]
Summers, R. C., C. D. Coleman, and E. B. Fernandez, "A Programming Language Extension for Access to a Shared Data Base", Proceedings of ACM Pacific Conference, San Francisco CA, 17-18 April 1975.

[Taylor 1974]
Taylor, B. J. and S. C. Lloyd, "DUCHESS - A High Level Information System", Proceedings of National Computer Conference, Chicago IL, 6-10 May 1974.

[Thomas 1975]
Thomas, J. C., and J. D. Gould, "A Psychological Study of Query by Example", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

[Tsichritzis 1975]

Tsichritzis, D., Features of a Conceptual Schema, Technical Report CSRG-56, Computer Systems Research Group, University of Toronto, Toronto, Canada, July 1975.

[Valle 1975]

Valle, G., "Interactive Handling of Data Base Relations: Experiments with the Relational Approach", Technical Report, University of Bologna, Bologna, Italy, March 1975.

[Weber 1976]

Weber, H., "A Semantic Model of Integrity Constraints on a Relational Data Base", Proceedings of IFIP TC-2 Conference on Modelling in a Data Base Management Systems, Freudenstadt, W. Germany, January 1976.

[Whitney 1974]

Whitney, K. M., "Relational Data Management Implementation Techniques", Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, Ann Arbor MI, 1-3 May 1974.

[Wilkes 1972]

Wilkes, M. V., "On Preserving the Integrity of Data Bases", The Computer Journal, Volume 15, Number 3, 1972.

[Zadeh 1975]

Zadeh, L. A., Calculus of Fuzzy Restrictions, Electronics Research Laboratory Report ERL-

M502, University of California, Berkeley CA, 19 February 1975.

[Zloof 1974]

Zloof, M. M., Query by Example, IBM Research Report RC4917, Yorktown Heights NY, 2 July 1974.

[Zloof 1975a]

Zloof, M. M., "Query by Example", Proceedings of National Computer Conference, Anaheim CA, 19-22 May 1975.

[Zloof 1975b]

Zloof, M. M., "Query by Example: The Invocation and Definition of Tables and Forms", Proceedings of International Conference on Very Large Data Bases, Framingham MA, 22-24 September 1975.

[Zook 1975]

Zook, W., K. Youssefi, P. Kreps, G. Held, and J. Ford, INGRES - Reference Manual, Electronics Research Laboratory Report ERL-M519, University of California, Berkeley CA, 23 April 1975.

Figure 1-1. Relation EMP

column ->	Name	Sex	Salary	Manager	Department
underlying domain ->	NAME	SEX	MONEY	NAME	DEPT
	Jones, Richard	male	$12,000	Jones, Richard	research
	Phillips, Jeff	male	$10,000	Smith, Kathy	sales
	Smith, Kathy	female	$11,000	Jones, Richard	sales

Figure 1-2. Example Data Base

Domains:

```
NAME        QUAN
SEX         ORDER_NUM
MONEY       CUST
DEPT        DATE
ITEM
```

Relations:

```
EMP (Name, Sex, Salary, Manager, Department)
     NAME  SEX  MONEY    NAME       DEPT

SALES (Item, Department, Quantity_on_hand, Cost)
       ITEM     DEPT          QUAN          MONEY

ORDERS (Order_number, Customer, Item, Date_shipped)
         ORDER_NUM      CUST    ITEM      DATE

BUDGET (Department, Salary_budget)
           DEPT          MONEY
```

Figure 1-3. A Possible Set of Relational Primitive Operations

```
create domain
delete domain          (these operations allow domains and
create relation         relations to be defined and deleted)
delete relation

insert tuple           (these operations allow changes to be
delete tuple            made to data in relations)
update tuple

add column to
  relation
delete column
  from relation        (these operations facilitate relation
copy relation           modification and relational algebraic
intersection            manipulation of a data base)
union
difference
join
```

Figure 3-1. Selected Example Data Base Domain Definitions

```
domain NAME                                        ("Smith, John")
     description
          last: string
          ', '
          first: string
     ordering
          last, first
     violation-action
          error

domain SEX                                         ("female")
     description
          oneof 'female', 'male'
     ordering
          none
     violation-action
          error 'sex must be female or male'

domain MONEY                                       ("$100")
     description
          '$'
          value: number where >=0
          where length(right(*, '.' + 1)) = 2
               or not present *, '.'
     ordering
          value
     violation-action
          substitute null 'value in error, null has been assumed'

domain ITEM                                        ("AB-75-326")
     description
          string where not has numerics, '-'
          i1: '-'
          i2: string where not has alphabetics, '-'
          where repititions i1 through i2 >=1 and <=3
     or
          string where call check_item
     ordering
          call compare_item
     violation-action
          substitute left(*, 5)
```

Figure 3-1. (continued)

```
domain QUAN                                          (17)
     description
          value: number where integer
                    and >=0
     ordering
          atomic
     violation-action
          call fixup_quan

domain DATE                                          ("1/20/1976")
     description
          month: oneof 1, ..., 12
          '/'
          day: number where integer and >=1 and <=31
          '/197'
          year: number where integer and >=5 and <=9
          where (if (month = 4 or =5 or =9 or =11) then day<=30)
               and (if month = 2 then day <= 29)
               and (if (month = 2 and year ~= 6) then day <= 28)
     ordering
          year, month, day
     violation-action
          error
```

Figure 3-2. Syntax of the Domain Definition Language

```
domain-definition ::= DOMAIN domain-name
                          DESCRIPTION
                              description-clause
                          [ORDERING
                              ordering-clause]
                          [VIOLATION-ACTION
                              violation-action-clause]

domain-name ::= string-constant

description-clause ::= description-subclause
                     | description-clause
                       OR
                       description-subclause

description-subclause ::= description
                              [where-restriction]

description ::= [label:] subunit
              | description
                [label:] subunit

label ::= string-constant

subunit ::= STRING [WHERE string-boolean]
          | NUMBER [WHERE number-boolean]
          | ONEOF string-constant-list
          | ONEOF number-constant-list

string-constant-list ::= string-constant-component
                       | string-constant-list, string-constant-component

string-constant-component ::= string-constant
                            | ALPHABETICS
                            | NUMERICS
                            | SPECIALS

number-constant-list ::= number-constant
                       | number-constant-list, number-constant

string-boolean ::= string-boolean-term
                 | string boolean OR string-boolean-term

string-boolean-term ::= string-boolean-factor
                      | string-boolean-term AND string-boolean-factor
```

Figure 3-2. (continued)

```
string-boolean-factor ::= string-boolean-primary
                        | NOT string-boolean-primary

string-boolean-primary ::= string-predicate
                         | (string-boolean)

string-predicate ::= comparator string-constant
                   | IF string-predicate THEN string-predicate
                                          [ELSE string-predicate]
                   | SIZE comparator number-expression
                   | HAS string-constant-list
                   | CALL procedure

comparator ::= = | ~= | >'| >= | < | <=

number-boolean ::= number-boolean-term
                 | number-boolean OR number-boolean-term

number-boolean-term ::= number-boolean-factor
                      | number-boolean-term AND number-boolean-factor

number-boolean-factor ::= number-boolean-primary
                        | NOT number-boolean-primary

number-boolean-primary ::= number-predicate
                         | (number-boolean)

number-predicate ::= comparator number-constant
                   | IF number-predicate THEN number-predicate
                                          [ELSE number-predicate]
                   | INTEGER
                   | EXPONENTIAL
                   | CALL procedure

where-restriction ::= boolean

boolean ::= boolean-term
          | boolean OR boolean-term

boolean-term ::= boolean-factor
               | boolean-term AND boolean-factor

boolean-factor ::= boolean-primary
                 | NOT boolean-primary

boolean-primary ::= predicate
                  | (boolean)
```

Figure 3-2. (continued)

```
predicate ::= expression comparator expression
            | IF predicate THEN predicate
                        [ELSE predicate]
            | PRESENT expression, string-constant-list
            | CALL procedure

expression ::= [addition-operator] unsigned-expression

unsigned-expression ::= arithmetic-term
                      | unsigned-expression addition-operator arithmetic-term

arithmetic-term ::= arithmetic-factor
                  | arithmetic-term multiply-operator arithmetic-factor

arithmetic-factor ::= subexpression
                    | (expression)

subexpression ::= atomic-expression
                | set-function(expression-list)
                | APPEND(expression, expression)
                | SUBSTRING(expression, expression, expression)
                | LEFT(expression, expression)
                | RIGHT(expression, expression)
                | LOCATION(expression, expression)
                | LENGTH(expression)
                | REPITITIONS label THROUGH label

atomic-expression ::= label
                    | string-constant
                    | number-constant
                    | *

expression-list ::= expression
                  | expression-list, expression

set-function ::= MAXIMUM | MAX | MINIMUM | MIN | string-constant

addition-operator ::= + | -

multiply-operator ::= * | / | **

ordering-clause ::= ordering-list
                  | NONE
                  | ATOMIC
                  | CALL procedure
```

Figure 3-2. (continued)

```
ordering-list ::= label
                | ordering-list, label

violation-action-clause ::= violation-action
                          | violation-action-clause
                            violation-action

violation-action ::= ERROR
                   | ERROR message
                   | SUBSTITUTE expression
                   | SUBSTITUTE expression message
                   | CALL procedure
                   | CALL procedure message

message ::= string-constant
          | SYSTEM-GENERATED

procedure ::= string-constant
```

Notes:

The nonterminals string-constant and number-constant are not further defined.

ALPHABETICS refers to the characters "A" through "Z" and "a" through "z", NUMERICS refers to the digits 0 through 9, and SPECIALS refers to all other characters.

SIZE returns the length of a string subunit. HAS s1, ..., sn returns "true" if a subunit has an occurrence of each of the strings s1, ..., sn (otherwise "false"). SIZE and HAS appear only in subunit where restrictions.

SUBSTRING(s,i1,i2) returns the substring of string s starting at character i1 and extending i2 characters. LEFT(s,i) and RIGHT(s,i) return the left and right substring (respectively) of s having length i. SUBSTRING, LEFT, and RIGHT may also be invoked with a second argument which is a string. This means that the substring is to start at the leftmost or rightmost occurrence of the second string argument, e.g., "LEFT(*, '.')" and "LEFT(*, INDEX(*, '.'))" are equivalent. LENGTH(s) returns the length of string s. APPEND(s1,s2) concatenates s1 and s2. LOCATION(s1,s2) returns the index of the first occurrence of s2 in s1 (or 0 if s2 is not a substring of s1). REPETITIONS s1 THROUGH s2 returns the number of repetitions (of the domain value) for subunits labeled s1 through s2.

Figure 6-1. Some Simple Assertions (for data base in figure 1-2)

Note: CC means constrained collection, PR means predicate

1. The salary of every employee is less than $50,000.
   CC: each tuple in EMP
   PR: Salary < 50000

2. The manager of each employee is also an employee.
   CC: each tuple in EMP
   PR: Manager is present in set of all Names from tuples in EMP

3. The salary of each employee in the toy department is less than the salary of his manager.
   CC: each tuple in EMP where Department = 'toy'
   PR: Salary < Salary of the tuple where Name = Manager in constrained tuple

4. The salary of an employee cannot decrease.
   CC: each tuple in EMP
   PR: new Salary >= old Salary

5. The average employee salary is at least equal to the salary of Robert Jones.
   CC: set of tuples in EMP
   PR: average(Salary) >= Salary of tuple where Name = 'Jones, Robert'

6. Each department has at most two employees with a salary of more than $50,000.
   CC: set of tuples in EMP where Salary > 50000, grouped by common Department
   PR: count(Name) <= 2

7. The number of female employees is at least 40% of the total number of employees.
   CC: set of tuples in EMP where Sex = 'female'
   PR: count(Name) >= .4 * count(Name) for tuples in EMP

8. Employee names are unique.
   CC: set of tuples in EMP
   PR: multiset(Name) has no duplicates

Figure 6-2. Local Tuple Predicates

Types of Predicates (a):

```
 1a.  col scalarcomp const
 2a.  col scalarcomp col
 3a.  col scalarcomp colexpr

 4a.  col setcomp {const-1, ..., const-m}
 5a.  col setcomp {col-1, ..., col-m}
 6a.  col setcomp {colexpr-1, ..., colexpr-m}
 7a.  col setcomp setexpr

 8a.  (col-1, ..., col-n) setcomp {(const-11, ..., const-1n), ...,
        (const-m1, ..., const-mn)}
 9a.  (col-1, ..., col-n) setcomp {(col-11, ..., col-1n), ...,
        (col-m1, ..., col-mn)}
10a.  (col-1, ..., col-n) setcomp {(colexpr-11, ..., colexpr-1n), ...,
        (colexpr-m1, ..., colexpr-mn)}
11a.  (col-1, ..., col-n) setcomp setexpr
```

Definitions:

col:	column name with optional "old" or "new" (col-1, col-11, etc., are cols; all cols must reference entries within the constrained tuple)
const:	constant from an appropriate domain
scalarop:	+, -, *, /, **, max, min, etc., or a user-defined scalar operator
setop:	union (also written as {}), intersection, difference, or a user-defined set operator
colexpr:	a legal combination of col, const, op, and setop which yields a single value
setexpr:	same as colexpr except yields a set of values
scalarcomp:	=, ~=, >, >=, <, <=, or a user-defined scalar comparator
setcomp:	is in, contains, properly is in, properly contains, or a user-defined set comparator

Figure 6-3. Nonlocal Tuple Predicates

Types of Predicates (a):

1a. col scalarcomp scalarval

2a. col setcomp setval

3a. (col-1, ..., col-n) setcomp setval

(In type 2a setval is a set of values, and in type 3a setval is a set of tuples.)

Definitions:

Definitions here are the same as figure 6-2, except:

scalarval: a scalar value computed from the data base
setval: a set value computed from the data base

ND predicates are the same as NI predicates, except that the process selecting scalarval and setval may reference the entries in the constrained tuple.

Figure 6-4. Local Set Predicates

Types of Predicates (a):

```
 1a.  aggfn(col) scalarcomp const
 2a.  aggfn(col) scalarcomp aggfn(col)
 3a.  aggfn(col) scalarcomp aggfnexpr

 4a.  aggfn(col-1, ..., col-n) scalarcomp const
 5a.  aggfn(col-1, ..., col-n) scalarcomp aggfn(col-1, ..., col-m)
 6a.  aggfn(col-1, ..., col-n) scalarcomp aggfnexpr

 7a.  set(col) setcomp (const-1, ..., const-n)
 8a.  set(col) setcomp set(col)
 9a.  set(col) setcomp setfnexpr

10a.  set(col-1, ..., col-n) setcomp ((const-11, ..., const-1n), ...,
         (const-m1, ..., const-mn))
11a.  set(col-1, ..., col-n) setcomp ((col-11, ..., col-1n), ...,
         (col-m1, ..., col-mn))
12a.  set(col-1, ..., col-n) setcomp setfnexpr

----
13a.  col crel col
14a.  col crel (col-1, ..., col-m)
15a.  (col-1, ..., col-n) crel col
16a.  (col-1, ..., col-n) crel (col-1, ..., col-m)
```

Definitions:

(col, const, scalarop, setop, colexpr, scalarcomp, setcomp are as in figure 6-2)

aggfn:	set, max, min, avg, sum, count, or a user-defined aggregate function (also all these with "'", e.g., "set'", meaning duplicates are retained)
crel:	one-to-one, functionally-dependent, or a user-defined column relationship comparator
aggfnexpr:	a legal combination of aggfn, col, const, scalarop, setop, and colexpr
setfnexpr:	a legal combination of "set", col, const, scalarop, setop, and colexpr

"Set" returns the set of values in a column (or tuples in a group of columns. It is an aggfn, but is also treated separately since it yields a set value.
(Note that "max(set(Salary))" is equivalent to "max(Salary)".)

Figure 6-5. Nonlocal Set Predicates

Types of Predicates (a):

1a. aggfn(col) scalarcomp scalarval
2a. aggfn(col-1, ..., col-n) scalarcomp scalarval

3a. set(col) setcomp setval

4a. set(col-1, ..., col-n) setcomp setval

(In type 3a, setval is a set of scalars, and in type 4a, setval is a set of tuples.)

Definitions:

Definitions here are the same as figure 6-4, except:

scalarval: a scalar value computed from the data base
setval: a set value computed from the data base

ND predicates are the same as NI predicates, except that the process selecting scalarval and setval may reference the data in the constrained tuple set.

This empty page was substituted for a blank page in the original document.

CS-TR Scanning Project
Document Control Form

Date : 12 / 11 / 95

Report # LCS-TR-165

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 124 (131 - IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:
☒ DOD Form (2) ☐ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages (by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

	Description :	Page Number:
(A)	IMAGE MAP!	(1-124) UNNUMBERED TITLE & BLANK PAGE, 2, UNNUMBERED BLK, 3, UNNUMBERED BLK, 4, UNNUMBERED BLK, 5-119, UNNUMBERED BLK. (125-131) SCANCONTROL, COVER, DOD(2), TARGETS(3)
(B)	1	

Scanning Agent Signoff:
Date Received: 12/11/95 Date Scanned: 1/11/96 Date Returned: 1/18/96

Scanning Agent Signature: Michael N. Cook

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MIT/LCS/TR-165	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: High Level Expression of Semantic Integrity Specifications in a Relational Data Base System	5. TYPE OF REPORT & PERIOD COVERED: S.M. Thesis 1975-1976
	6. PERFORMING ORG. REPORT NUMBER: MIT/LCS/TR-165
7. AUTHOR(s): Dennis J. McLeod	8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0661
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Massachusetts Institute of Technology Laboratory for Computer Science 545 Technology Square; Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency Department of Defense 1400 Wilson Boulevard Arlington, Virginia 22209	12. REPORT DATE: September 1976 13. NUMBER OF PAGES: 121
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Office of Naval Research Department of the Navy Information Systems Program Arlington, Virginia 22217	15. SECURITY CLASS. *(of this report)*: Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Data base management, semantic integrity, error detection and correction, data base design, data definition, data semantics, very high level languages

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The "semantic integrity" of a data base is said to be violated when the data base ceases to represent a legitimate configuration of the application environment it is intended to model. In the context of the relational data model, it is possible to identify multiple levels of semantic integrity information: (1) the description of the domains of the data base as abstract sets of atomic data values (domain definition), (2) the specification of the fundamental structure of the relations of the data base (relation structure

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20.

specification), (3) the definition of the abstract operations which are meaningful in terms of the application environment (structured operations), and (4) the expression of additional semantic information not contained in the structure of the relations nor in the identities of their underlying domains (relation constraints).

A high level, nonprocedural domain definition language facilitates the description of domains. Such a language allows the specification of the properties of the values constituting a domain, and the action that is to occur if an attempt is made to update a column entry such that it does not belong to the underlying domain of that column. The specification of relation structure and structured operations can be accomplished by means of high level integrity (sub)languages.

A relation constraint has three components: (1) the assertion (a predicate on the state of the data base or on transitions between data base states), (2) the validity requirement (the occasion(s) at which the assertion must hold), and (3) the violation-action (the action that is to occur if the assertion does not hold at a time when it should). Relation constraint specification can be related to an expression framework (classification scheme) which is useful for the construction of a relation constraint language and specification methodology. Assertions are more than expressions of some relationships among different values in a data base; an assertion singles out the data that is constrained, and states the properties this data must possess. A classification is provided of the various predicate types used to identify constrained data and to state the properties that they are to possess.

A semantic integrity subsystem (of a generalized relational data base management system) can support the generation and maintenance of integrity specifications, verify that these specifications are met by the data base, and take appropriate action if violations are detected.

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned
Date: 01/12/1996
M.I.T. Libraries
Document Services