1989

A SQL front-end semantic data model

Marc Richard Lodico

ROCHESTER INSTITUTE OF TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY

SQLSDM

"A SQL *Front-End* Semantic Data Model"

by

Marc Richard Lodico

A thesis, submitted to

The Faculty of the School of Computer Science and Technology

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science.

Approved by:

11/9/89

Dr. William Stratton

11/8/89

Dr. Andrew Kitchen

11/9/89

November 9, 1989

1. Title of thesis SQLSDM

"A SQL "FRONT-END" SEMANTIC DATA MODEL" I

I hereby **grant permission** to the Wallace Memorial Library of RIT to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date 11/16/89

SQLSDM

"A SQL FRONT END SEMANTIC DATA MODEL"

A MASTER'S THESIS

BY

MARC RICHARD LODICO

ABSTRACT

SQLSDM is a "front end" semantic data model to a SQL relational database management system (RDBMS). SQLSDM provides a more semantically complete RDBMS through the implementation of a Domain and Relational Integrity scheme. SQLSDM provides integrity definition functions and a sub-system to interpret SQL commands. Integrity system tables are created through the use of SQLSDM's domain definition command and SQL's "CREATE TABLE" command. As SQL database update commands are interpreted, SQLSDM uses these integrity tables to enforce domain and referential integrity. SQLSDM operates virtually transparent to the user and provides for greater database consistency and semantic control. Furthermore, SQLSDM is designed and engineered to be a portable "front-end" that may be implemented on any SQL relational database management system.

ACKNOWLEDGEMENT

Many thanks to Dr. William Stratton whose foresight and understanding of the subject matter were invaluable to the success of this thesis. His patience and kindness is very much appreciated.

Thanks also go out to Dr. Andrew Kitchen who served as a faithful committee member and to Professor Henry Etlinger who sat in at the defense.

Special thanks to my wife Bernadette whose continual support helped me through all the difficult times.

Table of Contents

1.0. INTRODUCTION AND BACKGROUND

1.1. PROBLEM STATEMENT

The objective of this thesis is to further the development of a more semantically complete relational database management system (DBMS) by implementing a "semantic data model" that supports two semantic integrity rules proposed by Date [DATE2]:
(1) Domain Integrity and (2) Relational Integrity. Furthermore, this will be implemented as a portable "front end" to a target relational DBMS, operating independently of that DBMS.

Specifically, the Domain Integrity rules (constraints stating that the values of a specific attribute are required to belong to the set of values constituting the underlying domain) to be implemented are of the form:

```
 domain-definition ::= DOMAIN domain-name [PRIMARY]
                              constraint
                              terminator
 domain-name       ::= substr(domain-name,1,1) = alpha
                       and substr(domain-name,2,x) =
                           alpha | '_' | '-' | '#'
                       and  2 >= x <= 8


 constraint        ::= data-type[(data-spec)][predicate
```

```
data-type          ::= /* any std. SQL data type [DATE4] */

data-spec          ::= /* integer x, where 1 <= x <= 40  */

predicate          ::=   condition [logop condition]

condition          ::=   value1 relop value2

relop              ::= <  |  >  |  !  |  <=  |  >=  |  =

logop              ::= AND | OR

terminator         ::= ;

value              ::=  /* any alpha numeric characters */
```

Relational Integrity (constraints that govern the admissability of a given tuple as a candidate for insertion into a given relation, or the relationship between tuples of one relation and those of another) embodies the two rules of Entity Integrity and Referential Integrity. Entity Integrity states that no attribute participating in the Primary Key (PK) of a base relation is allowed to accept null values. Referential Integrity says that if a base relation R2 includes a FOREIGN KEY (FK) matching the PK of some base relation R1, then every value of the FK in R2 must either be

(a) equal to the value of PK in some tuple R1 or (b) be wholly null
(i.e., each attribute value participating in that FK value must be null).

Specifically, Entity Integrity is implemented by not allowing NULL values to be entered into the proposed system as a PK or FK value. Referential Integrity is implemented through the use of system tables. These tables record the relations in which Referential Integrity is to be enforced as specified by the user or database administrator.

In the present study, these integrity rules will be implemented as a "front-end" to a predefined relational database management system. This "semantic data model" enhancement is attractive from a number of practical standpoints. First, it preserves all of the advantages and technology reflected in relational features of the target database package. Secondly, it can be applied to any DBMS that supports the pre-defined features. Third, it is in line with the current approach of careful packaging and tooling of software (e.g., [SCHUL]).

1.2. PREVIOUS WORK

1.2.1. RELATIONAL DATABASE AND SEMANTICS

Since the introduction of the Relational Model by Codd in 1968 [CODD1], relational DBMSs have become increasingly more popular, both academically and commercially. Many relational DBMSs now available (e.g., INGRES [STONE1], and DB2 [DATE1]) are considered by some to be "state-of-the-art" [STONE1].

However, significant research in the area has shown that these systems suffer from a lack of sufficient semantic control [TSUR, STONE1, WILSON, DATE3]. Indeed, at least one author asserts that the lack of sufficient semantic control prevents relational schemas from completely and expressively modeling the natural relationships and mutual constraints among entities [STONE1]. This lack of semantic control of the relational model has been the subject of much controversy (e.g., 1974 ACM SIGFIDET "The Great Debate") [RUSTIN] and has stimulated significant research activity (e.g., CHEN, SandS).

1.2.2. EVOLUTION OF THE RELATIONAL MODEL

The popularity of the Relational Model [CODD1] over other proposed models (e.g., the Hierarchical model, the DBTG network model) lies in its theoretical elegance and simplicity. Data is described by tables; there are no additional structures superimposed for the purpose of machine representation. The network model [DATE1], represents relationships and entities via nodes and links. However, in the Relational Model, there is no explicit representation of relationships; relationships among entities are created through the application of relational operators. Data independence (defined as the immunity of applications to change in storage structure and access strategy) is a major objective of database management systems [DATE1]. The simple table structure of the Relational Model allows for greater data independence between programs and machine representation and the organization of data than found in the network and hierarchical models [CODD1].

The Relational Model is based on the concrete and well tested principles of Set Theory. These principles are applied in the Relational Algebra of the model, and in Normalization theory.

Relational Algebra, the data manipulation language of the Relational Model, has operators such as JOIN, SELECT, and PROJECT which are derived from the Set operators (e.g. UNION, INTERSECT).

The second application of Set Theory, normalization, is the process of systematically eliminating such anomalies as data redundancy and inconsistency [DATE1], by reducing a relation into equivalent separate relations resulting in a more desirable form. Three levels of normalization were defined in the original Relational Model [CODD1] : First Normal Form (1NF), Second Normal Form (2NF) and Third Normal Form (3NF), where each successive level of normalization includes all the properties of the preceding normal forms. A relation is in 3NF if it has all its non-key attributes mutually independent of one another, and fully dependent on the relation's candidate keys. A non-key attribute does not participate in any candidate key of the relation. Two or more attributes are mutually independent if none of the attributes concerned are functionally dependent on any of the others. Functional dependence refers to an X and Y value pair in a relation, where one specific value of Y is always determined by a specific value of X and no other Y value may be associated with that X value [DATE1]. If a relation is in 3NF, redundancy (multiple occurrences of the same attribute/value pairs replicated in more than one tuple in a relation), a property not recognized in Set Theory, is eliminated. As shown by Date [DATE1] and others (e.g., Codd [CODD1]) redundancy can lead to inconsistencies within a database for various reasons. For example, if a relation is in 2NF (all underlying domains contain atomic values only and every non-key attribute is fully dependent on the primary key), then that relation may contain tuples where a non-key attribute is transitively dependent (its value is determined by another non-key attribute) on some other non-key attribute. Thus, if there is redundancy between multiple tuples, one of these non-key attributes could be updated in one tuple but not in the others, in turn creating an inconsistent state in the database [DATE1]. However, through normalization to 3NF, the transitive dependencies of this relation are eliminated, thereby preventing occurrences of this type of anomaly.

The Relational Model is recognized as simple and theoretically sound, but a number of scholars assert that the relational model suffers from semantic inadequacies [DATE1, DATE2, TSUR, CHEN, SandS]. One of the first papers addressing this concern was presented at the 1975 ACM SIGMOD International Conference of Data by Schmid and Swenson [SandS]. These authors state that questions of a semantic nature can't be answered by the highly mathematical relational model. Specifically, the Relational Model gives no indication of the way in which real world data is to be represented by a collection of relations. They also demonstrated that functional dependencies are inadequate for expressing certain kinds of knowledge. For example, if R.b -> R.c (where '->' is taken to mean that attribute c is functionally dependent on b in relation R), and R.a !->

R.b, (b is not functionally dependent on a), and there is a "real world" requirement that a is related to c, then this requirement can't be expressed as a functional dependency. Schmid and Swenson note also that if there is an inter-relational transitivity of functional dependencies, then another semantic shortcoming occurs. For example, if a JOIN on MGR# was performed between relation EMP_MGR, where EMP_MGR.emp# -> EMP_MGR.mgr#, and relation MGR_SALARY, where MGR_SALARY.mgr# -> MGR_SALARY.salary the resulting relation would contain the attributes emp#, mgr#, and salary. Without proper user knowledge, the salary of a tuple could assumed to be the employee's salary, although it would be erroneous. Thus, due to the lack of semantic control, the theoretically valid application of a JOIN command resulted in a meaningless "real world" relation.

Chen also criticized the semantic inadequacies of the Relational Model ("The Entity-Relationship Model--Toward a Unified View of Data" presented by Chen [CHEN] at the Association for Computing Machinery in 1975). Chen describes the Entity-Relationship Model (E/R) that in his view incorporates all the positive features of the Network Model, the Relational Model and the Entity Set Model [CHEN], while eliminating the shortcomings of each. The E/R Model separates the information about entities from the information about relations by organizing data based on an Entity-Relationship diagram. This diagram is used to develop a semantically rich data description by separating entities from relationships and distinguishing between 1:n, m:n, and 1:1 mappings. Chen asserts that the Relational Model does not include this type of information and therefore is semantically inadequate [CHEN].

1.2.3. SEMANTIC DATA MODELS

Scholars in the field of Database management realize both the advantages of the Relational Model (e.g., data independence, the power of relational algebra, and its "user friendly" interface) and the need to incorporate greater semantic control into the model. This need has sparked investigations [e.g., DATE1, TSUR, STONE1, WILSON, DATE3] into "semantic data modeling" (the task of capturing more of the meaning of data [DATE4]). As Codd points out, "semantic data modeling" is important because it can bring greater understanding and order into the field of database design. In addition, a meaning-oriented data model stored in a computer should enable it to respond to queries and other transactions in a more intelligent manner [DATE4].

One such "semantic data model", Cooperative Overt Passive Error-Detection (COPE), was designed as a system to represent and apply semantic integrity knowledge to detect semantic integrity errors (a state of the database which causes one or more assertions about the database to be false) [WILSON]. The COPE architecture follows that suggested by the ANSI/SPARC [CODASY] guidelines, where three different views or schemas of the database are established. They are (1) the internal schema (how data is physically stored), (2) the external schema (the logical view the user has of the database), and (3) the conceptual schema (enforces consistency and provides a mapping between the internal and external schemas). While other DBMSs embody some form of the internal and external views, most of these support only a minimal conceptual view, and usually not separate from the other two views. Therefore, in these DBMSs it is extremely difficult for the semantic integrity of a database to be checked and maintained by the DBMS [WILSON]. However, COPE's conceptual model component embodies the ANSI/SPARC conceptual model of the DBMS, in that it contains the "real world knowledge" about the database coupled with the ability to monitor the integrity of the existing database state or transitions between states [WILSON].

COPE's conceptual model consists of four major components:

1. Database Structure Description - provides a mapping between COPE's view of the database and the structures of the underlying database.

2. Relational Templates - a relational representation of the underlying database used for checking integrity constraints.

3. Semantic Network - which maps the relational templates to the appropriate portions of the database maintained by the DBMS, along with specifying some integrity constraints.

4. Constraint Rules - rules representing the major specifications of the conditions for semantic integrity to be imposed on the database.

Each of the components are described by a specialized language called the _Internal Conceptual Model Description Language_ (ICMDL). The ICMDL is a declarative language based on a combination of first order predicate logic, set theory, and semantic networks [WILSON].

There are essentially four steps to the integrity checking process of COPE. First, the data or update to be checked is translated into COPE's own internal form. The corresponding constraint rules to be checked are then selected. Each rule is then deciphered to obtain the specific checks required for the data contained in the database and the data of the proposed update. Finally, COPE performs the checks and takes the appropriate action specified in the constraint rules.

The COPE system, due to its conceptual model component, provides many important features that include greater DBMS _independence_ (COPE does not depend upon special capabilities of the DBMS), _specialization_ (it is functionally independent of the DBMS , allowing COPE to be tailored to the job of semantic integrity checking rather than performing DBMS tasks) and _versatility_ (COPE acts as a independent "front-end" filter, it can be used with any target DBMS) [WILSON].

RM\T is another "semantic data model" [CODD2]. Codd defined RM\T in an effort to incorporate more semantic control into the basic Relational Model. In this model, two types of relations are defined to represent all entities in a database: *E-relations* and *P-relations*. E-relations record the existence of the all the entities within the database. P-relations record the properties of those entities. A formal catalog structure which defines all the various relationships that exist among the different entities is also included. The system uses this catalog of relationships to enforce the various integrity constraints implied by the existence of such relationships [DATE1]. High level operators provide for the manipulation of these RM/T objects (E-relations, P-relations and the catalog structure). Three types of entities exist in this model: *kernal entities*, *associative entities*, and *characteristic entities*. A *kernal entity* is one that can exist independently of any other entity (e.g., suppliers, parts or employees). *Associative entities* are functions that represent many-to-many (or many-to-many-to-many, etc.) relationships among two or more otherwise independent entities (Independent entity meaning that none of the entities concerned is existence-dependent on any of the others. A *characteristic entity* describes other entities, either kernal, other characteristic or associative entities. In a given application, any of these three entity types can be specified as a designative entity or an entity in a many-to-one relationship with at least two otherwise independent entities [DATE2]. Entities are also categorized as subtypes, supertypes or both; the "type hierarchy" that results for this categorization is used to classify entities by property categories. For example, if entity E is a subtype of entity F, then F is a supertype of E and E inherits all the properties of entity F. This entire entity classification scheme is used largely to impose some structure on what would otherwise be an unstructured collection of information, thereby introducing some discipline into the integrity enforcement scheme [DATE1].

RM/T also introduces six new integrity rules in addition to the (1) Entity integrity and (2) Referential integrity rules of the basic relational model.
These rules are [DATE2]:

3. Entity Integrity in RM/T: E-relations accept insertions and deletions but not updates.

4. Property Integrity: If a tuple appears in a P-relation P, then the surrogate key (defined shortly) or primary key value of t must appear in the E-relation corresponding to P.

5. Characteristic Integrity: A characteristic entity cannot exist in the database unless the entity it most immediately describes is also in the database.

6. Association Integrity: Let A be an associative entity type, and let E be the set of E-attributes (attributes defined on the E-domain, which is the

domain of all possible surrogate values)
whose function is to identify the participants in A. Then a given instance of A can exist in the database only if, for that instance, each E-attribute in E either (a) has the value E-null, or (b) identifies an existing entity of the appropriate type.

7. Designative Integrity: Let D be a designative entity type, and let E be the set of E-attributes representing designations by D. Then a given instance of D can exist in the database only if, for that instance, each E-attribute in E either (a) has the value E-null, or (b) identifies an existing entity of the appropriate type.

8. Subtype Integrity: Whenever a surrogate key (say e) belongs to the E-relation for an entity of type E, e must also belong to the E-relation for each entity type for which E is a subtype.

Surrogate keys, an important aspect of the RM/T model, are system generated primary keys used and known internally by the system and not by the user. They eliminate potential problems of user-generated primary keys. For example, if the value of primary key needs to be changed, it involves considerable effort to implement that change, whereas with surrogate keys no change to the primary key or foreign keys is necessary. Another potential problem of user-defined keys is their inability to record the possible "real world" existence of an entity that doesn't have a primary key. For instance, consider an employee who left a company but is entitled to certain benefits; he no longer has an employee number, but still must be kept on the records [DATE]. Surrogate keys provide a mechanism for this type of situation.

The RM\T relational model hasn't been implemented [DATE2], but it theoretically incorporates many of the previously mentioned features that are designed to address many of the criticisms aimed at the original relational model. (For a more detailed explanation of RM/T model see [CODD2] and [DATE2]).

1.2.4. IMPROVEMENTS ON THE RELATIONAL MODEL

A number of noted scholars (e.g., Date [DATE2]; McLeod [MCLEOD]; Fagin [FAGIN]) and others have expanded on existing concepts (e.g., higher forms of normalization and Relational Integrity) and introduced new ones (e.g., Domain Integrity [DATE2]), to incorporate greater semantic control into Codd's original Relational Model.

Two researchers have attempted to replace the definition of 3NF with a stronger definition [DATE2]. This definition known as the Boyce/Codd Normal Form (BCNF) is conceptually simpler than 3NF, in that it makes no explicit reference to first and second normal forms, nor the concepts of full and transitive functional dependence [DATE1]. It simply states that a relation R is in BCNF, if and only if every determinant is a candidate key. A determinant is any attribute on which some other attribute is fully functionally dependent. BCNF was introduced primarily because redundancies and update anomalies occur when a relation in 3NF possesses two or more composite and overlapping candidate keys (see reference [DATE1] for detailed examples). A composite key is a candidate key that consists of more than one attribute. Overlapping candidate keys are composite candidate keys that have a common attribute among them.

Fagin defined a 4NF relation in which a set of projections (vertical subsets of a given relation obtained by selecting specified attributes in a specified left-to-right order, and eliminating duplicate tuples within the attributes selected) are derived from a relation in BCNF. This in turn eliminates any multivalued dependencies that are not also functional dependencies. (A multivalued dependence in relation R with attributes A, B, C, holds in R if and only if the set of B-values matching a given A-value/C-value pair in R depends only on the A-value and is independent of the C-value [DATE2]). This also eliminates redundancies such as those identified by Schmid & Swenson [SandS], that lead to update problems.

An even higher normal form, 5NF, has been identified [DATE2]. However, Date notes that normalization to this level should be treated as a guideline only, and that normalizing to an extreme is sometimes to be avoided. He asserts that it should be regarded primarily as a discipline by which the database designer can capture some of the semantics of the "real-world" enterprise that the database represents [DATE2].

Another approach to further the development of a semantically complete DBMS was introduced by McLeod [DATE2]. He notes that violations of domain integrity rules occur sufficiently often to justify a special facility to handle them. (However, Date warns that few existing systems provide much in the way of this support.) The facility proposed by Mcleod [DATE2], requires a mechanism to define, store and process domain integrity rules. The basic strategy involves declaring a domain-definition which is comprised of a domain-name, a constraint, and a terminator. A domain-name is a label used to identify a domain that exists in a database; where certain domains are used as a pool of values for the primary keys of the database and must be declared as such. A constraint is the set of rules or a condition that the domain values must adhere to, if they are to be considered to belong to that domain. This condition must at least specify a data type and optionally a predicate condition. A terminator is a specification describing the action the system will take if a constraint is violated. Such a violation is triggered by an update or insert of data to the database. The action so taken may include simple rejection (ideally, along with a message to the user), correcting the invalid value and then allowing the update, or failing the update and forcing a rollback (returning the database to its last known consistent state) [DATE2]. This definition constitutes a simple but important integrity rule that should be applied when any update or insert is made to the database [DATE2], thus realizing an important semantic control for the database concerned.

Date has also expanded upon the concept of Relational Integrity (the generic term for Entity Integrity and Referential Integrity) to incorporate greater semantic control into the Relational Model [DATE2]. Both the Entity Integrity rule and the Referential Integrity rule apply specifically to base tables. A base table is a relation

created via a **CREATE** (or similar) command and is physically stored in a database.
The Entity integrity rule, introduced by Codd [CODD1], in his original model was not spelled out explicitly. The rule states that no attribute participating in a primary key is allowed to accept null values. The justification for explicitly defining this rule is intuitively obvious. Base relations correspond to "real world" entities. Entity occurrences, by definition, are distinguishable by a unique indentifier. In the relational model unique entity occurrences are identified by primary keys. If a primary key value is wholly null then that entity occurrence has no identity. This is a contradiction in terms [DATE1]. Therefore, the rule of Entity integrity is a necessary one. This rule also applies to partially null primary keys [DATE1].

Referential Integrity, introduced by Codd [CODD1] as a semantic improvement to his Relational Model, was originally considered an "abstract principle" with several shortcomings [STONE2]. However, Date proposed solutions to these shortcomings. He also proposed a language for general integrity constraints that may be used to express referential constraints in a concrete manner.

Codd's original definition of referential integrity is as follows:

> Let relation R1 have a multi-attribute primary key. If attribute A of that multi-attribute

primary key is defined on primary domain D (a primary domain is any domain on which some single-attribute primary key is defined), then, at all times, for each value of k of A in R1 there must exist a relation R2 in the database with primary key k defined on D, such that k occurs as a value of k in R2 [STONE2].

A relation such as R1 in this rule is a referencing relation, a relation such as R2 is a referenced relation. Attribute r1.A is a referential attribute [STONE2].

The three shortcomings of this definition, identified by Date, become apparent by looking at a few examples.

First, consider a relation EMPLOYEE (EMP#,DEPT#) that includes references to some DEPARTMENT relation (DEPT#, MGR#, etc.) via its DEPT# attribute. Typically every value of the DEPT# attribute in EMPLOYEE is required to appear as the value of the DEPT# attribute in some tuple of the DEPARTMENT relation . However, attribute EMPLOYEE.DEPT# is not a component of the primary key of EMPLOYEE; this leaves no means to enforce referential integrity when the referential attribute is not part of the primary key. Therefore, Date modified the original Referential Integrity Rule to allow the referential attribute to be any attribute that is defined on the primary domain, not just one that is a component of a multi-attribute primary key [STONE2].

A second problem involves the definition of a primary domain. For example, if a relation CITYTAX (CITY,TAX) where CITY is the primary key, then, by definition of primary domain, the domain LOCATION upon which CITY is defined, is primary. This definition disallows a situation in which, say, there is a legitimate reason that any tuple with a TAX value of zero not be included in the relation CITYTAX. To deal with this anomaly, Date modified the definition of primary domain. This new definition states that a given domain may optionally be designated as primary if and only if there exists a relation in the database with a single-attribute, primary key defined on that domain. Under this definition, in the example above, domain LOCATION would not be designated as a primary domain. This would allow, for example, some relation SUPPLIER (SUPP#, SNAME, CITY) to have values of SUPPLIER.CITY that doesn't appear in referenced relation CITYTAX [STONE2].

The third shortcoming is related to entity integrity. Under the original definition, null values in the referential attribute were disallowed because they could not appear in the primary key of the referenced relation and therefore violate the rule of entity integrity. However, there are "real world" situations that require null values to occur in a referential attribute. Consider the relations EMPLOYEE (EMP#, DEPT#) where EMP# is a primary key and DEPT# is a referential attribute, and DEPARTMENT (DEPT#, MGR_NAME) where DEPT# is a primary key and the referenced attribute.

Here the requirement to keep all EMP# in the relation EMPLOYEE even if that employee is not assigned to a DEPT# (e.g., a tuple exists in relation EMPLOYEE with a null DEPT#) is disallowed under the original definition. Date accommodates this situation by allowing the referential attribute DEPT# to take on null values (in the absence of any explicit constraint to the contrary and if the referential attribute is not a component of the primary key) without the need to have corresponding null value in the referenced relation DEPARTMENT [STONE2].

The three modifications give rise to Date's current definition of Referential Integrity:

> If a base relation R2 includes a foreign key(FK) matching the PK of some base relation R1, then every value of the FK in R2 must either be
>
> (a) equal to the value of a PK in some tuple R1
>
> or
>
> (b) be wholly null(i.e., each attribute value participating in that FK value must be null) [DATE2].

Date asserts that this current definition is too theoretical and, while it is a necessary component of the relational database model, it cannot in its theoretical form address "real world" constraints [STONE2]. He states that while the general rule insists that each referenced relation have a single-attribute primary key (i.e., "references" can be made only to relations having such a primary key), today many applications cannot realistically adhere to such a constraint [STONE2]. The rule also relies on the notion of a primary domain, a notion not supported in most current implementations. The need for more additional rules applicable only to specific situations is also necessary if these, and other "real world" constraints, are to be accommodated and enforced.

Because of these concerns Date defines a language for expressing and implementing integrity constraints, and proposes a more specific "integrity" subsystem to monitor transactions (e.g., update operations), to detect violations of integrity rules, and, in the event of a violation, take appropriate action. The system proposed in the current paper attempts to implement this "integrity" subsystem.

2.0. PROJECT DESCRIPTION

2.1. FUNCTIONAL SPECIFICATION

2.1.1. FUNCTIONS PERFORMED

The proposed system SQLSDM (structured query language "semantic data model") supports Domain Integrity and Relational Integrity rules proposed by Date [DATE2]. SQLSDM is to be implemented as a portable "front end" to any target relational database that supports the relational database language SQL. SQL is a good choice for the target relational database language of this system. SQL is the official standard of the American National Standards Institute (ANSI). In addition, many SQL based products are already available in the marketplace [DATE4].

SQLSDM supports an interactive user interface where the user may define Domain Integrity rules and perform standard interactive SQL tasks using standard SQL commands and syntax in order to build and manipulate a relational database that supports both Domain Integrity Rules and Relational Integrity Rules. A relevant set of domain integrity rules is declared when construction of a relational application on the underlying SQL DBMS is undertaken. SQLSDM interprets these rules and stores them in a system file called the Domain Integrity Table (DIT). Secondly, the user defines all the required table intensions needed to implement the intended application's database. SQLSDM stores the relevant information for each table in another system file called the Relation Attribute Table

(RAT). Referential integrity rules are derived by SQLSDM by analyzing the DIT for a primary domain specification as each relation is defined. These rules are stored in the system file called the Referential Integrity Table (RIT).

SQLSDM support for domain integrity and relational integrity consists of two main functions:

I. Integrity Rules Definition

II. Integrity Enforcement

Domain Integrity

I. Integrity Rules Definition

The DIT is used to enforce Domain integrity. The DIT is a table where all the domains in the underlying database are defined. In order to create the DIT, the user must declare a domain name, a data type , a data specification, an optional PRIMARY specification, and/or an optional integrity rule. For example:

```
DOMAIN S# PRIMARY NUMBER  > 0 AND < 999;
```

is interpreted by SQLSDM and stored as a tuple within the DIT. In this way, SQLSDM records that there exists a domain called S# whose data type is a number. Optionally, when the word PRIMARY follows the domain name it signals SQLSDM to enforce referential integrity for any attributes in the underlying database that draw their values from the domain S#. Additionally, an optional domain integrity predicate may follow and is recorded in the DIT to be enforced by SQLSDM.

II. Integrity **Enforcement**

When the necessary domains and relevant domain integrity rules have been declared, SQLSDM can enforce Domain Integrity. Any SQL **INSERT** or **UPDATE** command "triggers" SQLSDM to compare each submitted attribute-value with its corresponding domain definition in the DIT. If a violation is detected (i.e., the value of the attribute is not consistent with the domain integrity rule for the domain it is drawn from) SQLSDM rejects the proposed **INSERT** or UPDATE command with an appropriate error message.

SQLSDM also supports inter-domain comparisons (e.g., the JOIN command, similar to a join via the SQL SELECT command). For instance, if two values, d and e, drawn from two distinct domains, D and E, are compared, SQLSDM will consider this a violation of domain integrity and an appropriate error message will be displayed.

The ability to enforce domain integrity necessitates the two SQLSDM deviation from standard SQL syntax: first, the CREATE TABLE command will include the Domain Name of the attribute defined rather than simply the standard SQL data type and specification (e.g., **CHARACTER**(3) or **NUMBER**); secondly, a JOIN command is used instead of the SQL SELECT command for the join operation.

Relational Integrity

Relational Integrity embodies the two rules of Entity Integrity and Referential Integrity. The rule of Entity Integrity states that no attribute participating in the primary key(PK) of a base relation is allowed to accept null values. The rule of Referential Integrity states that if a base relation R2 includes a foreign key(FK) matching the PK of some base relation R1, then every value of the FK in R2 must either be (a) equal to the value of PK in some tuple R1 or (b) be wholly null.

Entity Integrity is enforced throughout the SQLSDM system (i.e., any attribute-value defined on a domain that was declared **PRIMARY** may not accept NULL as a value). Enforcement of this rule occurs when a tuple is presented to SQLSDM to be inserted or updated. Therefore, any null value submitted as a PK or FK value is invalid and is rejected upon submission to the system. (A further scheme for handling nulls is complex and beyond the scope of this system [DATE2]).

Referential Integrity

I. Integrity Rules Definition

It is up to the database designer to choose the relations in which referential integrity is enforced. This is accomplished through the SQL **CREATE TABLE** command and the use of the optional **PRIMARY** constant in the domain declaration phase of the application's database. When SQLSDM encounters the SQL **CREATE TABLE** command, it looks at each domain from which each attribute is defined, and searches the DIT table for a match. If a match is found, on any attribute other than the first (assumed to be a PK), and its domain is designated as **PRIMARY**, then referential integrity is applicable to that attribute in the newly created table. To record this, SQLSDM inserts a tuple comprised of the relation name (designated the referencing relation), the referential attribute (acts as a FK), and the domain name (designated as PRIMARY) into the RIT. This logic is applied to the subsequent create of all tables. In addition, each time a new table is created, SQLSDM uses the domain of the first specified attribute specified (assumed to be the primary key of that relation) to search the RIT for a match on the domain. If a match is found, then that table is designated as the referenced relation and the matching RIT tuple is updated with the referenced relation name. After the **CREATE TABLE** phase is complete for the entire database, the SQLSDM **RITEND** command is entered which triggers SQLSDM to read through the RIT, flag any tuple that does not have a value in its referenced relation attribute, and generate an error message to the user. The user is required to reconcile all such discrepancies before any SQL data manipulation commands are processed. This is enforced by requiring the user to create a new table to be used as the referenced relation, or to delete the referencing relation (therefore removing the corresponding RIT tuple).

Additionally, the **CREATE TABLE** command triggers SQLSDM to record the intensional data of a newly created relation. This data is stored in a system file called the Relation Attribute Table (RAT), one of tables used to enforce Referential Integrity. It records the relation name, the relation's attribute names, the domain names on which those attributes are defined, and an indicator designating the Primary Key attribute.

This SQLSDM implementation suggests that to implement an integrity scheme within the current standard SQL framework would neccessitate a change to the current SQL standard. Currently, there appears to be no way to define domains or to specify the relations and attributes to enforce referential integrity constraints upon. There is the need to have a command to define domains, and the SQL **CREATE TABLE** command must be modified to accept domain names instead of the data type and length specification required under the SQL standard.

II. Integrity Rules Enforcement

SQLSDM supports Referential Integrity for the non-cursor SQL data manipulation commands: **DELETE**, **UPDATE**, and **INSERT** [DATE4].

DELETE

When a **DELETE** command is encountered, SQLSDM enforces Referential Integrity by not allowing any set of one or more tuples to be deleted when one or more foreign keys exist in some other referencing relation that matched the set of primary keys of those tuples targeted for the delete.

Specifically, when a **DELETE** command is entered, SQLSDM reads the RIT is read to obtain all relations and corresponding referential attributes, that reference the relation to delete. For instance, consider relations, R1 and R2, their corresponding attributes A1 and A2, that reference relation, RD, targeted for deletion. SQLSDM searches R1 and R2 (by formulation of the appropriate standard SQL DBMS calls) for any foreign key values equal to the primary key of the tuples to delete from RD. SQLSDM accomplishes this by reading the RD tuples targeted for delete, in the application database, and by reading R1 and R2 for any foreign key values that match the primary key values of the RD tuples targeted for deletion. If any foreign key values match the primary key values of the tuples targeted for deletion, then SQLSDM rejects the **DELETE** command and issues an appropriate error message.

UPDATE

SQLSDM enforces Referential Integrity for the SQL **UPDATE** command as well. SQLSDM accomplishes this by ensuring that if the new value to be updated is a foreign key in some referencing relation(s), then there exists a matching primary key value in its corresponding referenced relation.

Specifically, SQLSDM reads RIT to determine if the relation to update is a referencing relation. The referential attribute for this relation is compared to the attribute that is targeted for the update. If they match, it is a signal to SQLSDM to enforce referential integrity this update operation; if they do not, then the command is passed on to SQL DBMS for normal processing. If referential integrity is to be enforced, SQLSDM reads the RAT using the referenced relation name, and the corresponding primary domain name, to obtain the attribute name that serves as the primary key of the referenced relation. That referenced relation is now searched, and SQLSDM formulates the appropriate standard SQL DBMS calls to verify the existence of a primary key value equal to the value of the attribute to update in the referencing relation. If no primary key value is found, SQLSDM rejects the update command with an appropriate error message. If there is a matching primary key value then SQLSDM passes the **UPDATE** command to the SQL DBMS and normal processing continues.

INSERT

If the user attempts an SQL INSERT command on a relation that appears as a referencing relation in the RIT, SQLSDM enforces Referential Integrity.

The RIT is first read by SQLSDM to determine the primary domain names of the referential attributes, and their corresponding referenced relation names. If RIT entries are found, SQLSDM uses the domain names of the referential attributes to read the RAT to obtain all referenced attribute names defined in their corresponding referenced relations. SQLSDM then reads each referenced relation by formulating SQL calls to the SQL DBMS. If the values to insert exist as primary keys in their corresponding referenced relation, then SQLSDM permits the INSERT command and passes it on to the SQL DBMS. However, if any value to insert has no corresponding primary key value in its referenced relation, SQLSDM rejects the INSERT command an issues an appropriate error message.

2.1.2. LIMITATIONS AND RESTRICTIONS

The initial goal of SQLSDM was to operate as a portable "front end" to any relational database system that supports standard ANSI SQL [DATE4]. However, this intial goal has not been proven at this time. SQLSDM was written using ANSI defined "C' to allow for portability, but the SQL call interface is ORACLE specific. Although the ORACLE SQL call interface appears to be generic it would most likely require modifications if it were imported to another system.

Support for NULL values are non-existent for primary and foreign keys only, due to the complexity of referential integrity enforcement needed to support such a feature. Therefore, the **NOT NULL** option on the SQL CREATE **TABLE** command is always be assumed.

SQLSDM is designed for interactive SQL only. However, programs that enforce Relational Integrity and Domain Integrity may be used, with minor interface changes, in an application program designed for batch type processing.

To support Domain Integrity, SQLSDM requires a modified syntax for the SQL **CREATE TABLE** command. A domain name must be input instead of the standard data type specification that normally accompanies this command. In addition, SQLSDM will always assume that the first attribute named in the **CREATE TABLE** command is a primary key. Another deviation is that the SQL join operation, implemented by using the

SELECT command, may by implemented using the SQLSDM **JOIN** command if inter-domain consistency is required.

Multiple attribute primary keys are not supported. To add this support requires a method to differentiate between an attribute that is to act as a primary key versus an attribute intended to be a foreign key. Under the current implementation, the first attribute specified in a **CREATE TABLE** command must be from a primary domain and assumed to be a primary key.

Finally, SQLSDM does not support Relational Integrity or Domain Integrity for VIEWS.

2.1.3. USER INPUTS

SQLSDM accepts all user input through an on-line source. Any valid ANSI standard SQL command will be accepted, along with any valid SQLSDM data definition command.

SQLSDM identifies each command input and compares the command to an internal table of commands. If the command is identified as an SQLSDM data definition command or a SQL data manipulation command, then SQLSDM processes the command accordingly. If the command is identified as a non-SQLSDM command, then that command is passed directly to the SQL DBMS.

<u>SQLSDM DATA DEFINITION</u> **COMMANDS:**

```
CREATE TABLE  relation-name attribute-name-1  domain-name-1,
                            attribute-name-2  domain-name-2,
                            ...,
                            attribute-name-n  domain-name-n;
```

- This modified SQL command triggers SQLSDM to record the relation being created in the RAT system file. SQLSDM modifies the command by replacing each domain-name with its corresponding data type and specification (recorded in the DIT system file). SQLSDM then passes the modified command to the SQL DBMS for normal processing.

DOMAIN domain-name [PRIMARY] constraint;

- This SQLSDM command is used to define a domain. SQLSDM records the domain information in the DIT system file. It is used to enforce Domain Integrity, and as a means to identify attributes on which Referential Integrity will be enforced (see 5.1.1. DOMAIN INTEGRITY RULES DEFINITION).

DOMDEL domain-name;

- This SQLSDM command triggers SQLSDM to delete the domain-name from the DIT system file. SQLSDM reads the RAT to check for any relation that still uses that domain, if none exists, SQLSDM will then delete the domain from the DIT.

RITDEL relation-name;

- This SQLSDM command triggers SQLSDM to delete any record that exists in the RIT system file for the relation-name entered. This eliminates Referential Integrity support for that particular relation.

RITEND;

- This SQLSDM is used when the user has **CREATEd** all the the tables needed in the particular application. The command triggers SQLSDM to "reconcile" the RIT system file (i.e., all "referencing" relations in the RIT must have a corresponding "referenced" relation).

SQL DML COMMANDS [DATE4] THAT INTERACT WITH SQLSDM:

SELECT

- This command is used by SQL for "read only" access to the database.

JOIN

- The JOIN command triggers SQLSDM to read the DIT system file and verify that the attributes to join on are drawn from the same domain (see 5.1.1. Domain Integrity Enforcement). If both attributes are from the same domain, then SQLSDM replaces the 'JOIN' literal with 'SELECT' literal and passes the command to the SQL DBMS for processing.

INSERT

- This SQL command is used to add a new tuple to the specified relation or table. It triggers SQLSDM to enforce Domain Integrity and Referential Integrity. However, Referential Integrity is only enforced if the relation targeted for INSERT is recorded in the RIT as a "referencing" relation. If all the values to INSERT are valid (see 5.1. Domain Integrity Enforcement) and Referential Integrity is maintained (see 5.1. Referential Integrity Enforcement), then SQLSDM passes the command to the SQL DBMS for actual execution.

UPDATE

- This SQL command changes values in the relation specified. SQLSDM enforces Domain Integrity and Referential Integrity for this command in the same manner as the SQL **INSERT** command, except that only values to be updated are verified (see 5.1.1. Domain Integrity Enforcement and Referential Integrity Enforcement).

DELETE

- The **DELETE** command is used to remove one or more tuples from the specified relation. This command triggers SQLSDM to enforce Referential Integrity. SQLSDM allows for the removal of a tuple if and only if its primary key value has no corresponding foreign key value in a "referencing" relation (see 5.1.1. Referential Integrity Enforcement).

2.1.4. USER OUTPUTS

The two categories of SQLSDM output include (1) **SQL COMMANDS** and (2) **SQLSDM ERROR MESSAGES**.

SQL COMMANDS

- SQLSDM examines each command input into the system. If the command needs to be processed by SQLSDM (i.e., Domain Integrity or Referential Integrity Enforcement), then SQLSDM will formulate standard SQL SELECT commands (to be passed to SQL) that are used to retrieve records to determine if any integrity constraints will be violated upon execution of that input command by the SQL DBMS. SQLSDM, after integrity has been verified or after determining that the command input does not need any SQLSDM processing, outputs that input command to the SQL DBMS for processing.

SQLSDM ERROR MESSAGES

An **SQLSDM ERROR MESSAGE** is produced whenever SQLSDM determines that execution of the entered SQL input command, by the SQL DBMS, will lead to a database state that violates the Domain and Relational Integrity laws for that underlying database.
There are other instances when an **ERROR MESSAGE** is output to the user. They occur when SQLSDM encounters an erroneous or invalid input command. The error are as follows:

EM_1_1	INVALID COMMAND SYNTAX
EM_1_2	INCOMPLETE COMMAND
EM_2_1	DOMAIN INTEGRITY VIOLATION - "JOIN" COMMAND USES ATTRIBUTES FROM DIFFERENT DOMAINS
EM_2_2	ENTITY INTEGRITY VIOLATION - NULLS NOT ALLOWED FOR INSERT OR UPDATE
EM_2_3	PRIMARY KEY NOT UNIQUE - INSERT REJECTED
EM_2_4	PRIMARY KEY VALUE DOES NOT EXIST IN REFERENCED RELATION
EM_2_5	INVALID UPDATE ATTEMPTED ON A PRIMARY KEY
EM_2_6	INVALID UPDATE ATTEMPTED - VALUE TO UPDATE IS A FK AND IS NOT A PK IN A REFERENCED RELATION
EM_2_7	INVALID DELETE ATTEMPTED - PK VALUE EXISTS AS A FK
EM_2_8	DOMAIN INTEGRITY VIOLATION - ILLEGAL VALUE SUBMITTED
EM_3_1	DOMAIN ALREADY EXISTS
EM_3_2	DOMAIN IS STILL UTILIZED IN DB - CANNOT DELETE IT
EM_3_3	TABLE NAME ALREADY EXISTS - CANNOT CREATE

EM_3_4	RIT RECONCILE ERROR - NO REFERENCED RELATION FOUND IN RAT FOR RELATION:
EM_3_5	DOMAIN DOES NOT EXIST - CANNOT CREATE TABLE
EM_3_6	DOMAIN DOES NOT EXIST - NOTHING TO DELETE
EM_3_7	RELATION NAME NOT FOUND IN RIT - NOTHING TO DELETE
EM_3_8	INVALID DOMAIN NAME
EM_3_9	INVALID SQL DATA TYPE
EM_3_10	INVALID SQL DATA SPECIFICATION
EM_3_11	SQLSDM DATABASE ERROR ... CONTACT DBA
EM_3_12	INVALID SQL RELATION NAME
EM_3_13	INVALID ATTRIBUTE NAME
EM_3_14	DUPLICATE ATTRIBUTE NAMES

2.1.5. SYSTEM FILES

SQLSDM utilizes three system files: the RIT, the RAT, and the DIT.

RIT

The Relational Integrity Table is used to record all the relationships in which Referential Integrity is to be enforced. The RIT is a relation with the following attributes:

REL_NAME	:	The referencing relation name.
REF_ATTR	:	The attribute that acts as a foreign key
DOM_NAME	:	The domain name on which the REF_ATTR is drawn from. Always a PRIMARY domain.
REFD_REL	:	The referenced relation name.

NOTE - all relation names and attribute names adhere to the ANSI SQL standards [DATE4]. The SQLSDM DOM_NAME is the same data type as a SQL attribute.

RAT

The Relational Attribute Table records the existence of all the relations and their attributes of the underlying application database. The RAT consists of:

RELATION	:	The relation name.
ATTR_NAME	:	The attribute name.
DOM_NAME	:	The domain from which the attribute is drawn.
PRIMARY	:	Indicates 'Y' for a primary domain.

Blank otherwise. Character (1).

DIT

The Domain Integrity Table. Records the domain information of all the domains in the underlying application's database. The DIT consists of:

DOM_NAME	:	The domain name. Where name is substr(domain-name,1,1) = alpha and substr(domain-name,2,x) = alpha or or '_' or '-' or '#' and 4 >= x <= 8
DATA_TYPE	:	The ANSI SQL standard data types. Character (8).
DATA_SPEC	:	An Integer length if required, otherwise 0.
PRIMARY	:	Indicates a primary domain. 'y' or blank
DI_RULE	:	A 30 character descriptor of the domain Integrity rule to be enforced.

2.2. SYSTEMS SPECIFICATION

2.2.1. SYSTEMS ORGANIZATIONAL CHART

- (see APPENDIX A1)

2.2.2. SYSTEMS DATA FLOW CHART

- (see APPENDIX A2)

2.2.3. EQUIPMENT CONFIGURATION

- ANY IBM PC OR COMPATABLE WITH A MONITOR.
- AT LEAST 1.64 MEGABYTES RAM (required for ORACLE), MINIMUM 10 MEGABYTE HARD DISK (required for ORACLE).
- PRINTER (NOT REQUIRED)

2.2.4. IMPLEMENTATION TOOLS

SQLSDM is implemented using the C programming language and ORACLE SQL DBMS. ORACLE is a good choice as the relational database management systems because it is one of the few available relational DBMS that supports the ANSI SQL standard. ORACLE also provides a programmable call interface between the C and the SQL DBMS.

2.3. VERIFICATION AND VALIDATION

2.3 1. DELIVERABLE ITEMS

- completed thesis proposal
- C program listings
- hard copies of system files before and after the Phase I test suite. The Phase I test suite includes a series of SQLSDM Data Definition Commands designed to completely test the execution of such commands on the DIT, RAT and RIT.
- hard copies of application files before and after the Phase II test suite. The Phase II test suite includes a series of SQL Data Manipulation commands designed to completely test the Domain Integrity and Relational Integrity aspects of SQLSDM.
- "portability" test. This includes hard copies of both the SQLSDM system files and the SQL application database before and after Phase I and Phase II test suites using another DBMS that supports ANSI SQL. Specifically, the Relational DBMS RBASE-V which also supports SQL will be used to test "portability".

NOTE - This last deliverable is dependent upon the availability, and licensing and copyright agreements of RBASE-V.

2.3.2. MILESTONE IDENTIFICATION AND SCHEDULE

10/30/88	- acceptance of Masters Thesis Proposal.
1/24/89	- command parsing routines.
	- routines to execute SQLSDM Data Definition commands.
4/25/88	- Phase I test suite.
	- routines to formulate SQL inquiry commands.
	- Domain and Relational Integrity interfaces.
	- Phase II test suite.
07/10/89	- complete all testing and system integration
	- "portability" test
10/30/89	- Masters Thesis Defense

3.0 CONCLUSIONS

SQLSDM functions as it was originally intended, with one exception: portability. The portability aspect of the system was not proven. Although the C code was written using standard ANSI C to allow for portability, the ORACLE SQL call interface has some ORACLE DBMS SQL specific commands which may not be compatible with the DBMS SQL call interface of other relational database management systems.

The Domain Integrity scheme functions as was originally intended, but a further scheme is required in order to be more functional. There is the need to be able to specify mathmetical constraints upon domains. Constraints that specify the validity of multiplying one domain by another to get a valid value in a third domain is one example. There must also be a way to enforce these constraints once they are defined. Other domain integrity improvements include: addtional data types, ordering, and composite domains [DATE2].

Perhaps the most signifcant finding is the apparent inadequacy of the ANSI SQL standard to incorporate any commands to implement a domain or referential integrity scheme. This inadequacy became apparent during the system design phase of SQLSDM. Therefore, it was necessary to include a command to define domains and to modify the SQL CREATE TABLE command.

4.0 QUALIFICATIONS

4.1. PERSONAL BACKGROUND

While working for Bausch & Lomb in Rochester, N.Y. during the years of 1982 through 1984, I was extensively involved in database

design and implementation on micro-computers.

For the past four years I have been working as a Consultant in a Technical Analyst position at EASTMAN KODAK I have been extensively involved in design and implementation of IMS Database application systems.

4.2. COURSES TAKEN

PROGRAMMING LANGUAGES AND DESIGN

ICSP 305	**ASSEMBLY LANGUAGE**
ICSS 320	**DATA STRUCTURE AND ANALYSIS**
ICSS 709	**PROGRAMMING LANGUAGE THEORY - C** programming was undertaken in this course.
ICSS 708	**COMPUTER ORGANIZATION AND PROGRAMMING** - structure charts, data flow diagrams and top-down design were investigated.
ICSS 730	**MODELING AND SIMULATION I**

DATABASES

ICSS 735	**ON-LINE INFORMATION SYSTEMS**
ICSS 738	**DATA BASE CONCEPTS** - studied the relational database model and designed and implemented an application database.
ICSS 739	**DATA BASE IMPLEMENTATION** - extensive relational database investigation. Including design and implementation, using **UNIX** and C, of a Relational DBMS.
ICSS 846	**INFORMATION STORAGE AND RETRIEVAL**

COMPUTING THEORY

CTDS 230	DISCRETE STRUCTURE
ICSS 706	FOUNDATIONS OF COMPUTING THEORY

MISCELLANEOUS

ICSS 720	COMPUTER ARCHITECTURE
ICSS 740	COMPUTER COMMUNICATION NETWORKS
ICSS 770	FUNDAMENTALS OF COMPUTER GRAPHICS

4.4. PROGRAMS WRITTEN

I have written an SQL-like relational DBMS for ICSS-739 "DATABASE IMPLEMENTATION". This Relational DBMS was an on-line system that involved interpretation and execution of SQL-like DDL and DML commands. Including the ability to define and manipulate indices for the application tables.

5.0. GLOSSARY

Associative Entities:

RM/T functions that represent many-to-many (or many-to-many-to-many, etc.) relationships among two or more otherwise independent entities.

Base Tables:

a relation or table created via a **CREATE** (or similar) command and is physically stored in a database.

BCNF:

a relation is in Boyce/Codd Normal Form if and only if every determinant is a candidate key.

Candidate key:

the set of attributes in a relation that satisfies the two time-independent properties of Uniqueness and Minimality.

Characteristic Entity:

an RM/T entity that describes other entities, either kernal, other characteristic or associative entities.

Composite Key:

a candidate key that consists of more than one attribute.

Conceptual Schema:

enforces consistency and provides a mapping between the internal and external schemas.

Constraint:

a the set of rules or a condition that the domain values must adhere to, if they are to be considered to belong to that domain. This condition must at least specify a data type and optionally a predicate condition.

Constraint Rules:

COPE's rules representing the major specifications of the conditions for semantic integrity to be imposed on the database.

COPE:

Cooperative Overt Passive Error-Detection, a system designed to represent and apply semantic integrity knowledge to detect semantic integrity errors in relational database management systems.

Data Independence:

the immunity of applications to change in storage structure and access strategy.

Database Structure Description:

provides a mapping between COPE's view of the database and the structures of the underlying database.

Designative Entity:

an RM/T entity in a many-to-one relationship with at least two otherwise independent entities.

Determinant:

any attribute on which some other attribute is fully functionally dependent.

Domain definition:

a syntax to declare a domain to be used in a relational database.

Domain Integrity Rule:

a constraint stating that the values of a specific attribute are required to belong to the set of values constituting the underlying domain.

Domain-name:

a label used to identify a domain that exists in a database; where certain domains are used as a pool of values for the primary keys of the database and must be declared as such.

Entity Integrity:

a rule of the Relational Model that states that no attribute participating in a primary key of a base relation is allowed to accept null values.

E-relations:

in RM/T used to record the existence of the all the entities within the database.

External Schema:

the logical view the user has of the database.

Foreign key:

an attribute (or attribute combination) in one relation R2 whose values are required to match those of the primary key of some relation R1.

Functional Dependence:

refers to an X and Y value pair in a relation, where one specific value of Y is always determined by a specific value of X and no other Y value may be associated with that X value.

Hierarchical Model:

a database model that consists of an ordered set of multiple occurrences of a single type of tree. A tree type consists of a single "root" record type, together with an ordered set of zero or more dependent (lower level) subtree types. A subtree type also consists of a single record type - the root of the subtree type - together with an ordered set of zero or more lower-level dependent subtree types [DATE1].

ICMDL:

Internal Conceptual Model Description Language is COPE's declarative language based on a combination of first order predicate logic, set theory, and semantic networks.

Independence:

a COPE feature that refers to the ability that COPE's functionality does not depend upon special capabilities of the DBMS.

Independent Entity:

an RM/t entity where none of the entities concerned is existence-dependent on any of the others.

Internal Schema:

how data is physically stored.

Kernal Entities:

an RM/T entity that can exist independently of any other entity.

Multivalued dependency:

a multivalued dependence in relation R with attributes A, B, C, holds in R if and only if the set of B-values matching a given A-value/C-value pair in R depends only on the A-value and is independent of the C-value.

Mutually Independent:

two or more attributes are mutually independent if none of the attributes concerned are functionally dependent on any of the other attributes.

Network Model:

a database model that consists of two sets, a set a set of records and a set of links. A set of multiple occurrences of each of several types of records, together with a set of multiple occurrences of each of several types of link. Each link type involves two record types, a parent record type and a child record type. Each occurrence of a given link type consists of a single occurrence of the parent record type, together with an ordered set of multiple occurrences of the child record type [DATE1].

Non-key attribute:

an attribute that doesn't participate in any candidate key of the relation.

Normalization:

the process of systematically eliminating such anomalies as data redundancy and inconsistency , by reducing a relation into equivalent separate relations resulting in a more desirable form.

Overlapping candidate keys:

composite candidate keys that have a common attribute.

P-relations:

in RM/T used record the properties of entities.

Primary Domain:

a primary domain is any domain on which some single-attribute primary key is defined.

Primary Key:

a designated candidate key.

Projections:

vertical subsets of a given relation obtained by selecting specified attributes in a specified left-to-right order, and eliminating duplicate tuples within the attributes selected.

Redundancy:

multiple occurrences of the same attribute/value pairs replicated in more than one tuple in a relation.

Referenced Relation:

a relation in which the primary key occurs as a foreign key in some other relation (known as the referencing relation).

Referencing Relation:

a relation in which a foreign key is found.

Referential Attribute:

an attribute occurring in a referencing relation whose value occurs in some referenced relation.

Referential Integrity:

states that if a base relation **R2** includes a foreign key(FK) matching the PK of some base relation R1, then every value of the FK in R2 must either be (a) equal to the value of PK in some tuple R1 or (b) be wholly null(i.e., each attribute value participating in that FK value must be null).

Relational **Algebra**:

the data manipulation language of the Relational Model.

Relational Integrity:

the ability of a database system to adhere to constraints that govern the admissability of a given tuple as a candidate for insertion into a given relation, or the relationship between tuples of one relation and those of another) embodies the two rules of Entity Integrity and Referential Integrity.

Relational Model:

originally defined by Codd [CODD1]. The model consists of three major parts: a structural part, an integrity part, and a manipulative part. The structural part consists essentially of any n-ary relations(together with their underlying domains). The integrity part consists of two general integrity rules , namely "entity integrity" and "referential integrity". Finally, the manipulative part provides a set of algebraic operators for data manipulation

Relational Templates:

COPE's relational representation of the underlying database used for checking integrity constraints.

RM/T:

a semantic data model Codd defined in an effort to incorporate more semantic control into the basic Relational Model.

Rollback:

the process which returns a database to its last known consistent state.

Semantic **control**:

the ability for relational schemas to completely and expressively model the natural relationships and mutual constraints among entities.

Semantic Data Modeling:

the task of capturing more of the meaning of data.

Semantic Network:used by COPE to map the relational templates to the appropriate portions of the database maintained by the DBMS, along with specifying some integrity constraints.

Specialization:

a feature of COPE which allows COPE to be functionally independent of the DBMS.

Subtypes, Supertypes:

in RM/T they are used to classify entities by property category. if entity E is a subtype of entity F, then F is a supertype of E and E inherits all the properties of entity F.

Surrogate Key:

system generated primary keys used and known internally by the system and not by the user.

Terminator:

a specification describing the action the system will take if a constraint is violated.

Transitively Dependent:

a attributes value determined by another non-key attribute.

Versatility:

a feature that refers to COPE's ability to act as an independent "front-end" filter to be used with any target DBMS.

Views:

a "virtual" table. A table that does not exist in physical storage, but looks to the user as if it did.

3NF:

a relation is in third normal form if and only if the non-key attributes of that relation is "mutually independent"
and "fully dependent" on the primary key of that relation.

4NF:

a relation R is in fourth normal form if and only if, whenever there exists a "multivalued dependency" in **R**, say **A** ->-> B, then all attributes of R are also "functionally dependent" on A.

6. BIBLIOGRAPHY

CHEN P.P. Chen, "The Entity-Relationship Model - Toward an Unified View of Data", ACM Trans. Database Systems, Vol.1, No.1, 1976.

CODASY CODASYL Systems Committee, "Introduction to 'Feature Analysis of Generalized Data Base Management Systems'", CACM, Vol.14, No.5, May 1971.

CODD1 E.F. Codd, "A Relational Model of Data for Large Shared Data Banks", CACM, Vol.13, No.6, June 1970.

CODD2 E.J. Codd, "Extending The Database Relational Model to Capture More Meaning", ACM TODS, Vol.4, No.4, Dec 1979.

DATE1 C.J. DATE, An Introduction to Database Systems, Vol.I, 4th Edition, Addison-Wesley, Reading, Mass., 1986.

DATE2 C.J. DATE, An Introduction to Database Systems, Vol.II, Addison-Wesley, Reading, Mass., 1985.

DATE3 C.J. Date, "Referential Integrity", Proc. 7th International Conf. on Very Large Data Bases, Cannes, Sept 1983.

DATE4 C.J. Date, A Guide to The SQL Standard, Addison-Wesley, Reading, Mass., 1987.

FAGIN R. Fagin, "Normal Forms and Relational Database Operators", Proc.1979 ACM SIGMOD International Conference on Management of Data.

MCLEOD D.J. McLeod, M.M. Hammer "Semantic Integrity in a Relational Database System", Proc. 1st Inter. Conf. on Very Large Data Bases. Sept 1975

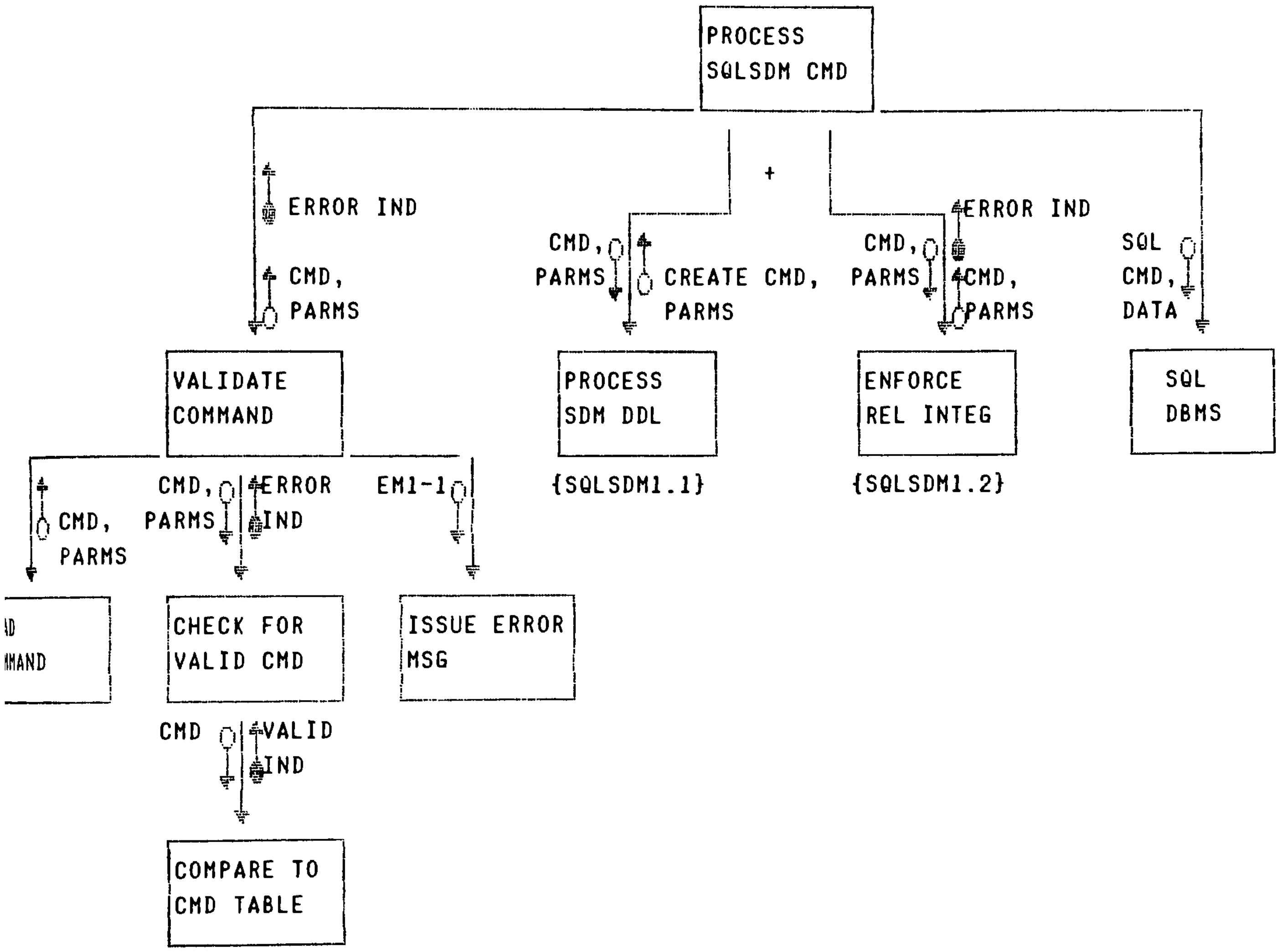
RUSTIN R. Rustin,(Ed.), "Data Models: Data-Structure_Set versus Relational", Workshop on Data Description Access And Control, May 1-3, 1974, ACM SIGMOD proceedings of "the great debate" between Codd and Bachman.

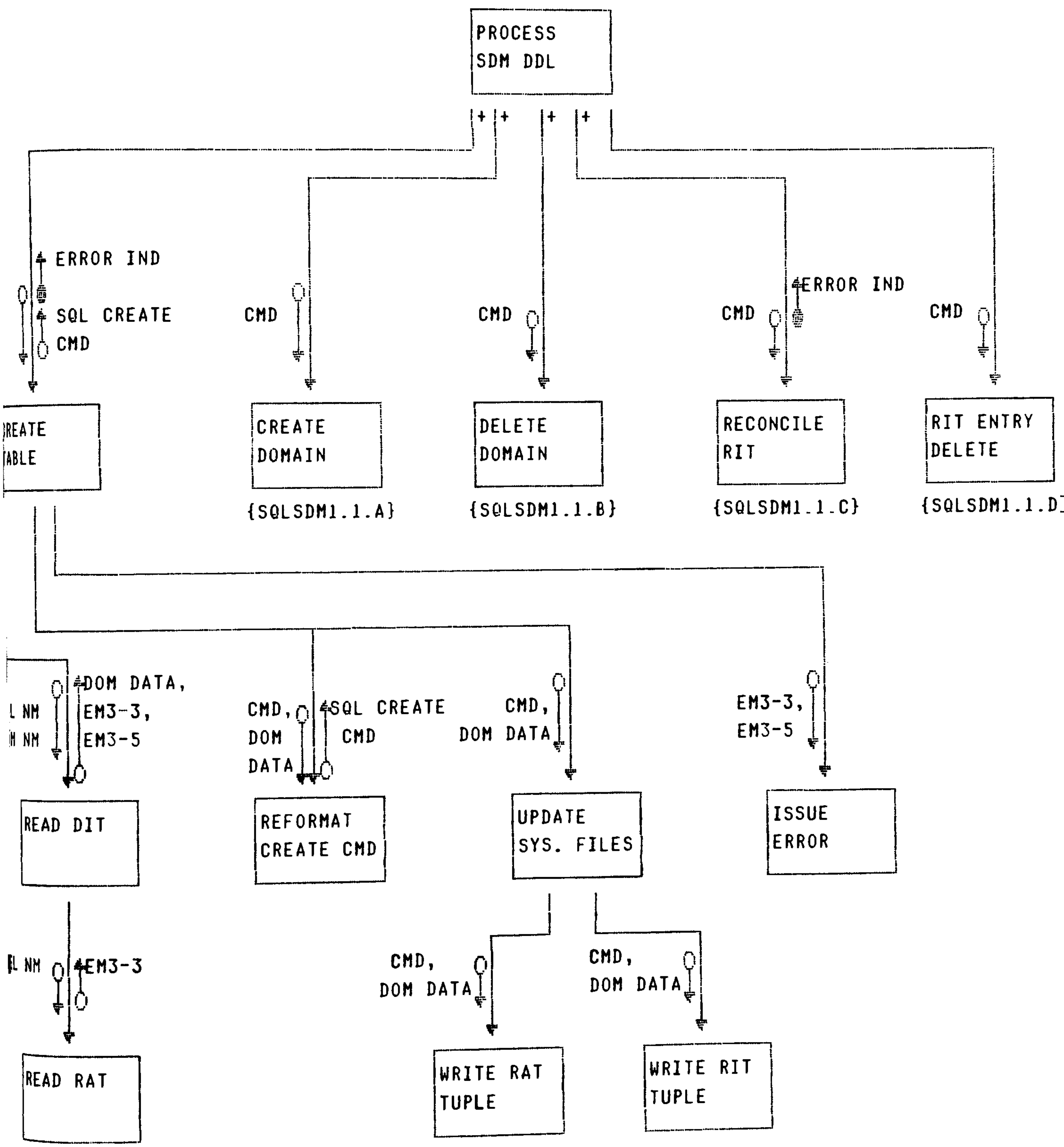
SANDS	H.A. Scmid, J.R. Swenson, "On The Semantics Of The Relational Model", Proc.1975 ACM SIGMOD International Conf. on Mgmt. of Data, May 1975.
SCHUL	G. Schulmeyer, J. McManus, Handbook of Software Quality and Assurance, Van Nostrand Reinhold, N.Y., N.Y., 1987.
STONE1	M. Stonebraker, E. Wong, P. Kreps, G. Held, "The Design and Implementation of INGRES", ACM Trans. on Database Systems, Vol.1, No.3, Sept. 1976.
STONE2	M. Stonebraker, "Implementation of Integrity Constraints and Views by Query Modification", Proc. ACM-SIGMOD Conf., San Jose, Ca., 1975.
TSUR1	S.Tsur, C.Zaniolo, "An Implementation of GEM - supporting a semantic data model on a relational back-end", CACM, Vol.8, No.6, 1984.
WILSON	G.A. WILSON, "A Conceptual Model For Semantic Integrity Checking", Proc. 6th International Conf. on Very Large Data Bases, Oct 1980.

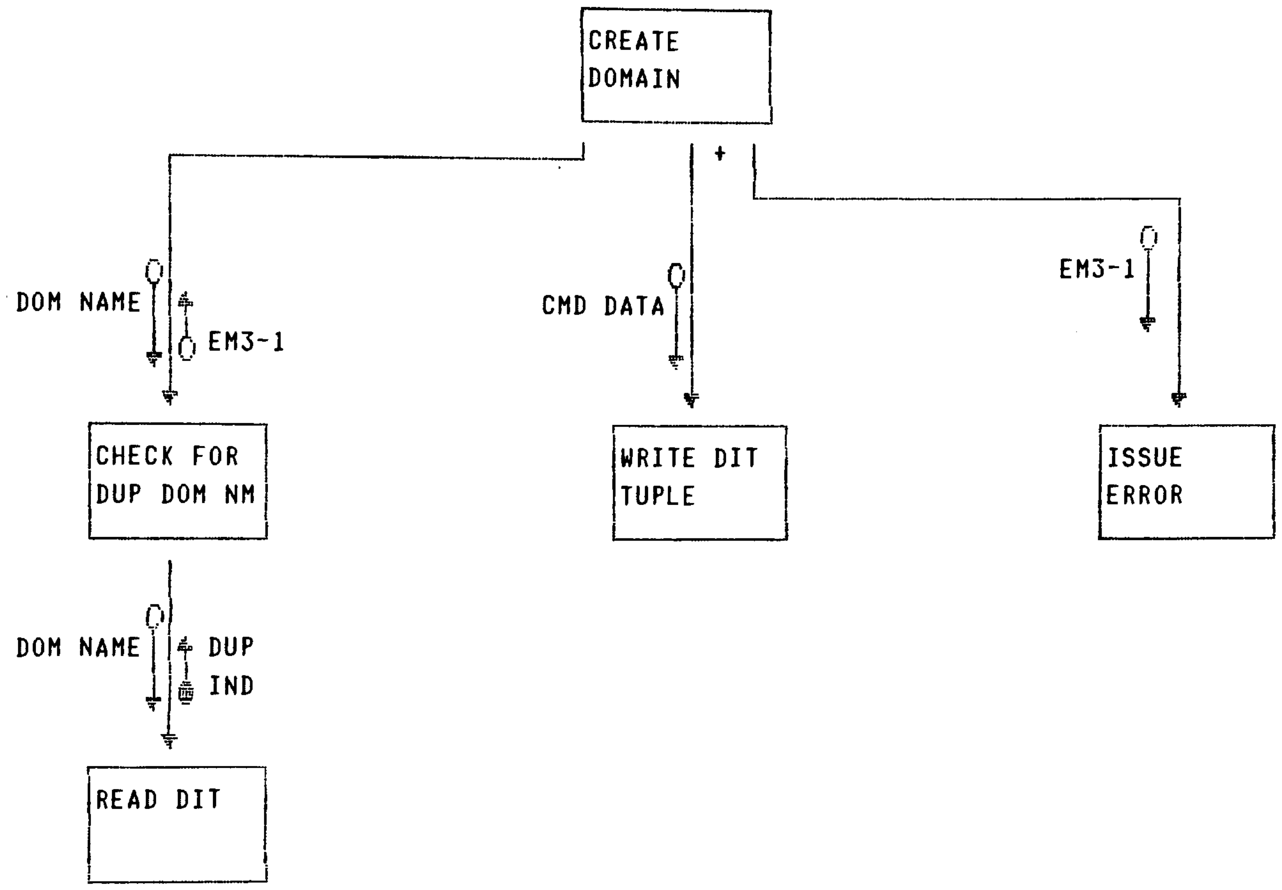
7. APPENDIX

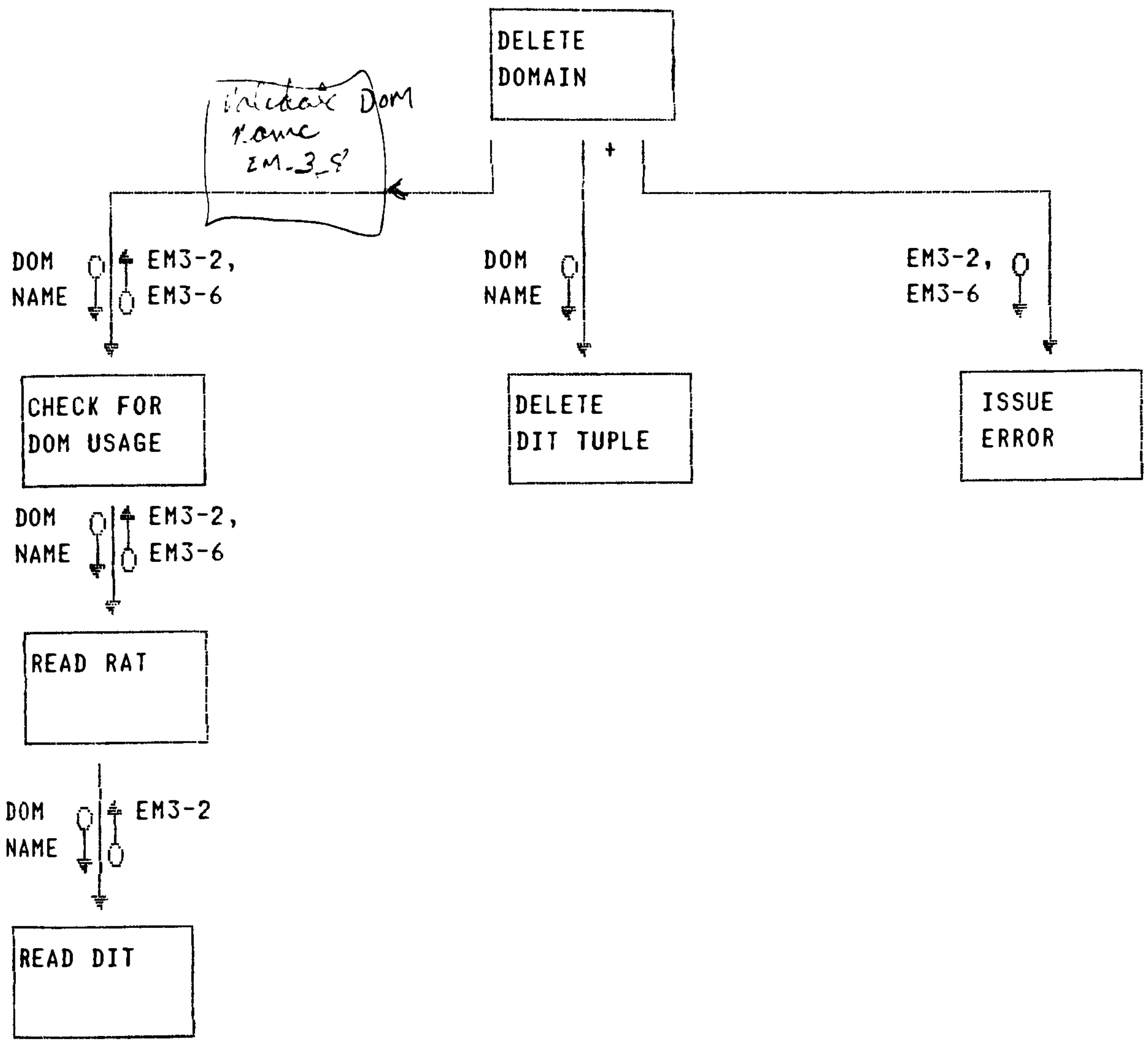
APPENDIX A1

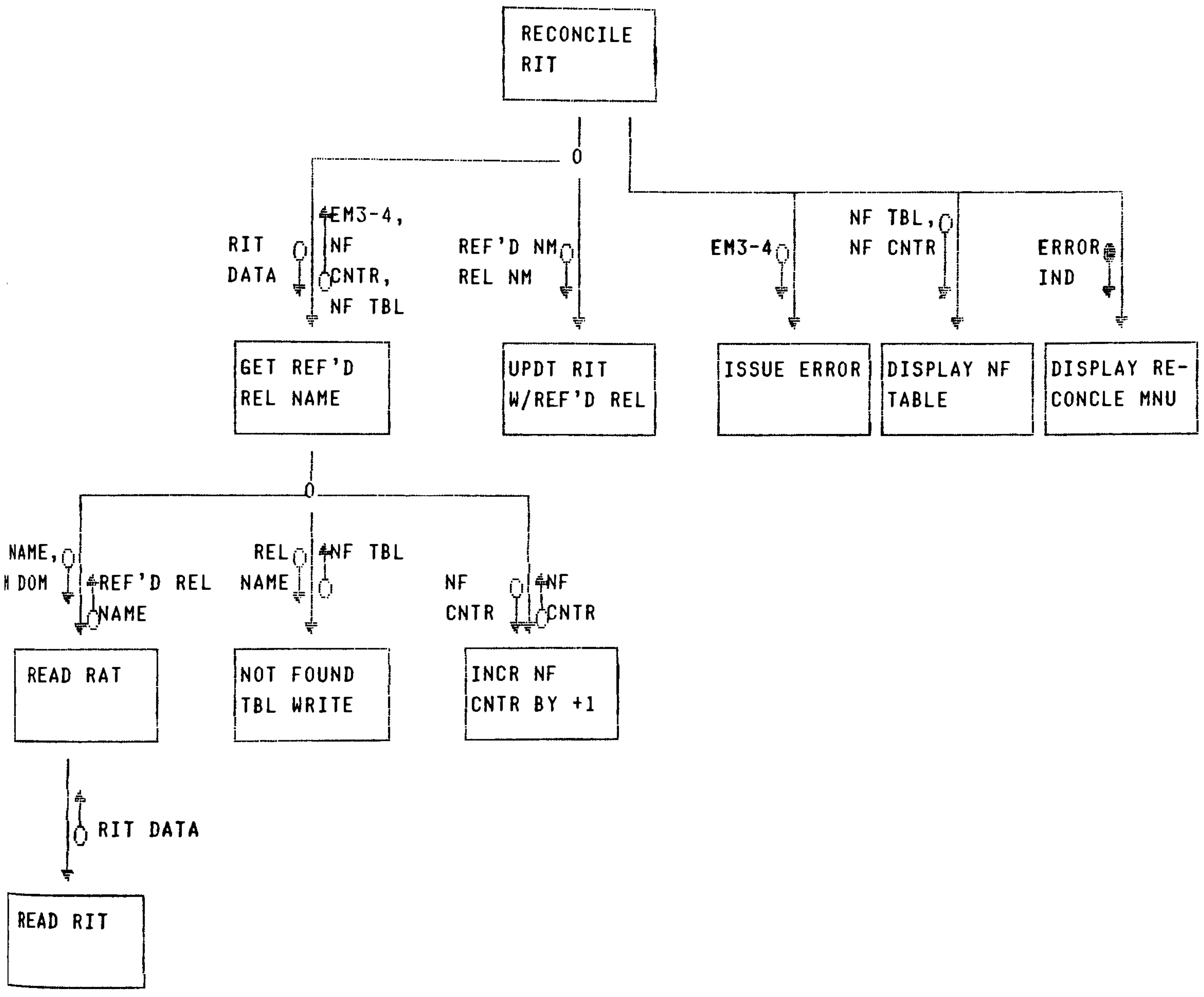
SYSTEM STRUCTURE CHART

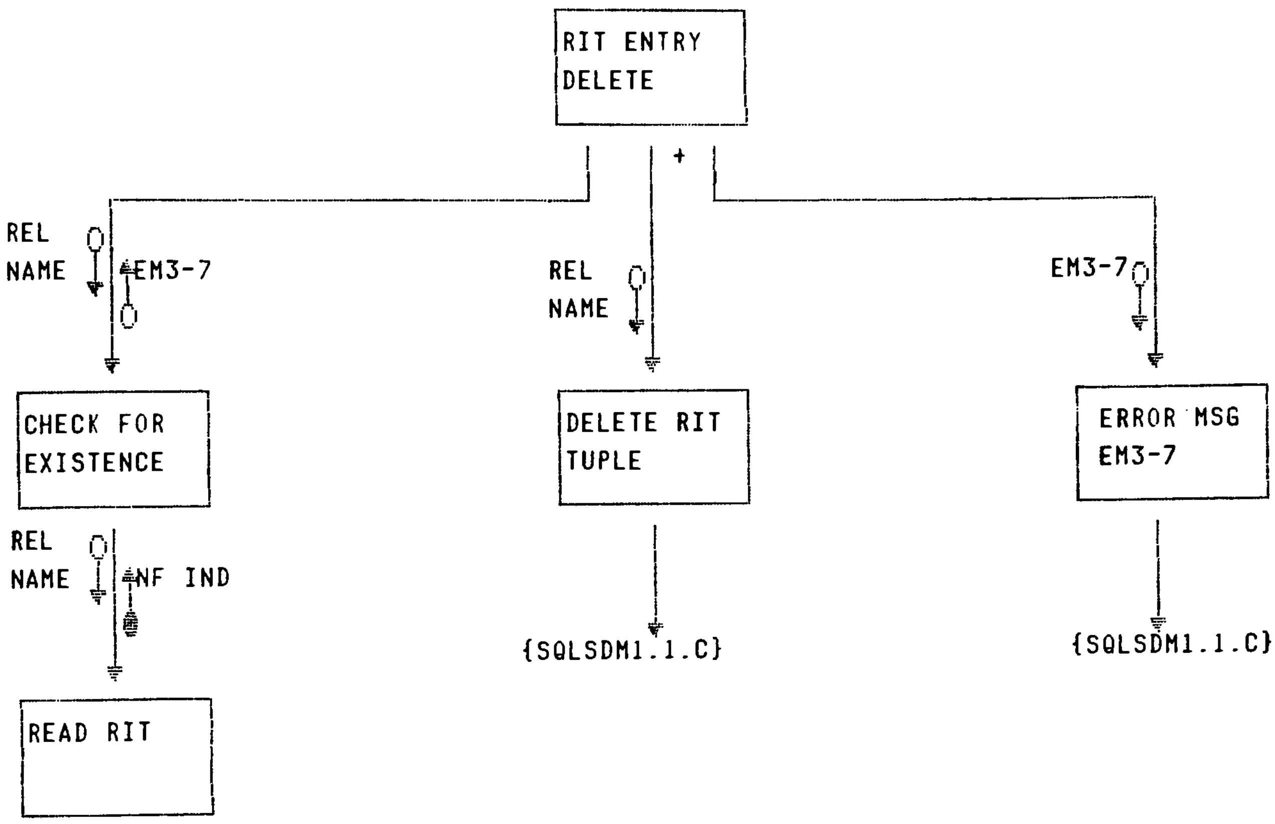

PROCESS SQLSDM CMD
ERROR IND
CMD, PARMS
+
CMD, PARMS
CREATE CMD, PARMS
CMD, PARMS
ERROR IND
CMD, PARMS
SQL CMD, DATA
VALIDATE COMMAND
PROCESS SDM DDL
{SQLSDM1.1}
ENFORCE REL INTEG
{SQLSDM1.2}
SQL DBMS
CMD, PARMS
CMD, PARMS
ERROR IND
EM1-1
AD MMAND
CHECK FOR VALID CMD
ISSUE ERROR MSG
CMD
VALID IND
COMPARE TO CMD TABLE

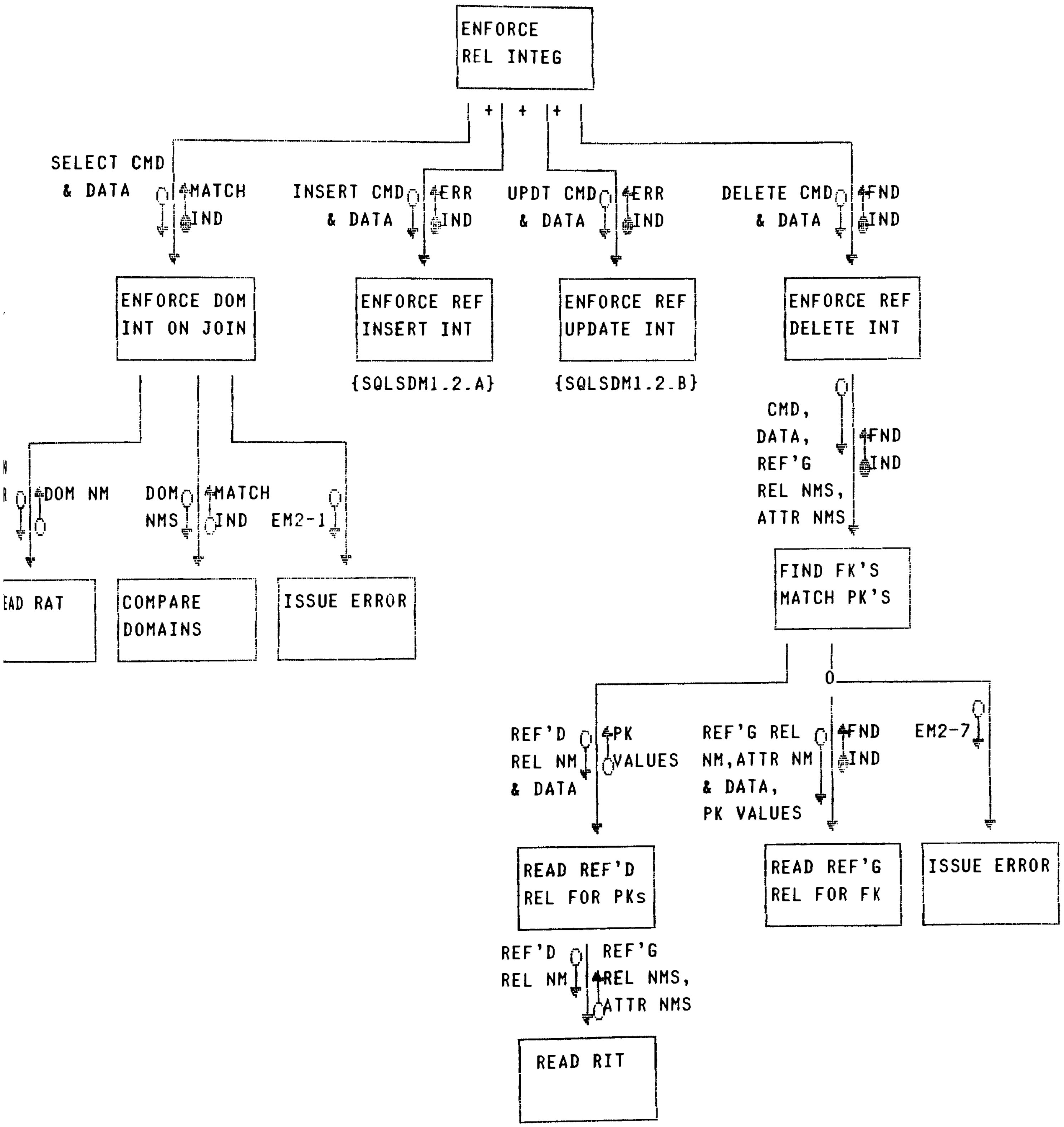

PROCESS SDM DDL
ERROR IND
SQL CREATE CMD
CMD
CMD
CMD
ERROR IND
CMD
REATE ABLE
CREATE DOMAIN
{SQLSDM1.1.A}
DELETE DOMAIN
{SQLSDM1.1.B}
RECONCILE RIT
{SQLSDM1.1.C}
RIT ENTRY DELETE
{SQLSDM1.1.D
L NM
M NM
DOM DATA, EM3-3, EM3-5
CMD, DOM DATA
SQL CREATE CMD
CMD, DOM DATA
EM3-3, EM3-5
READ DIT
REFORMAT CREATE CMD
UPDATE SYS. FILES
ISSUE ERROR
L NM
EM3-3
CMD, DOM DATA
CMD, DOM DATA
READ RAT
WRITE RAT TUPLE
WRITE RIT TUPLE

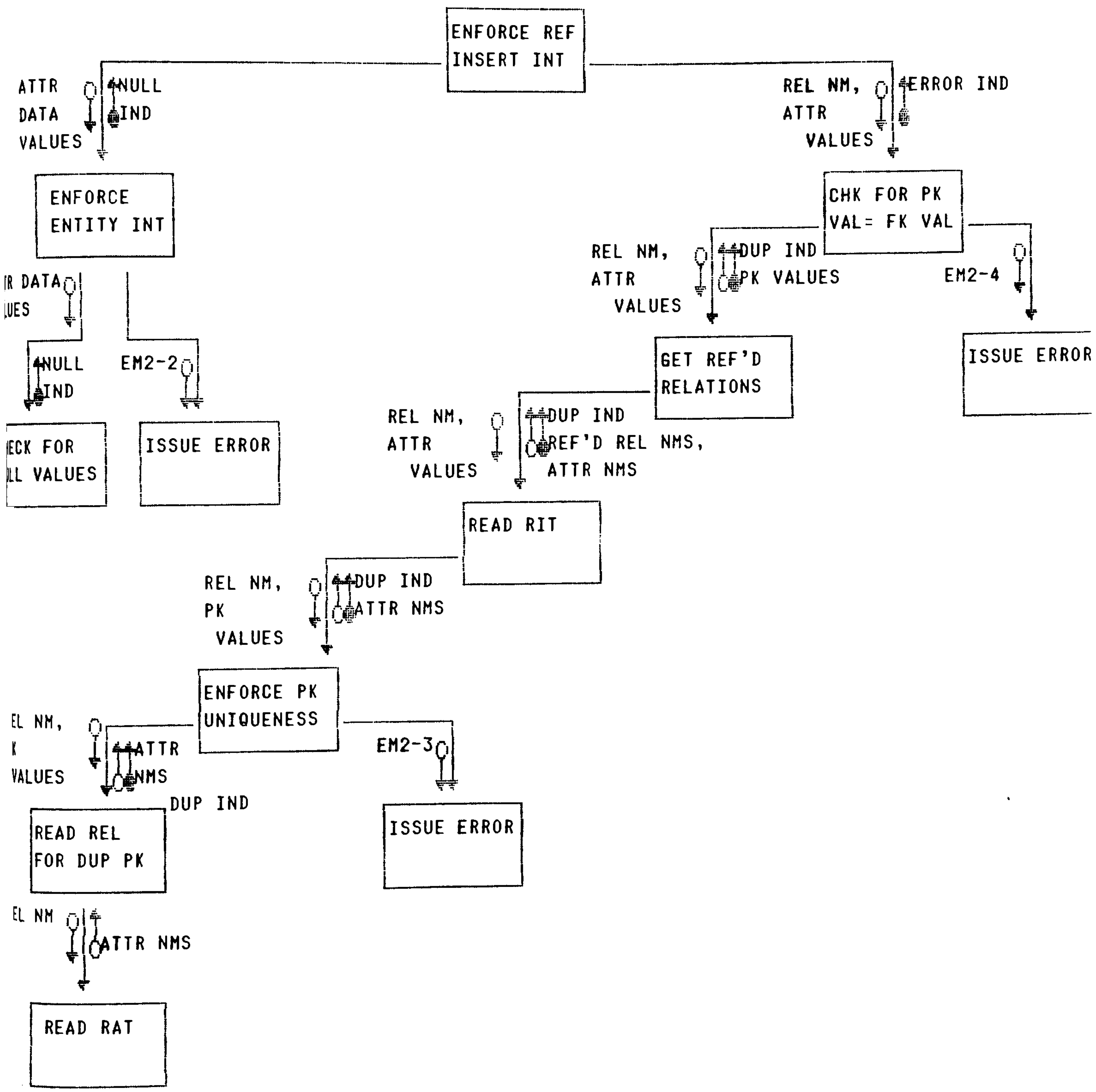

CREATE DOMAIN
+
DOM NAME
EM3-1
CMD DATA
EM3-1
CHECK FOR DUP DOM NM
WRITE DIT TUPLE
ISSUE ERROR
DOM NAME
DUP IND
READ DIT

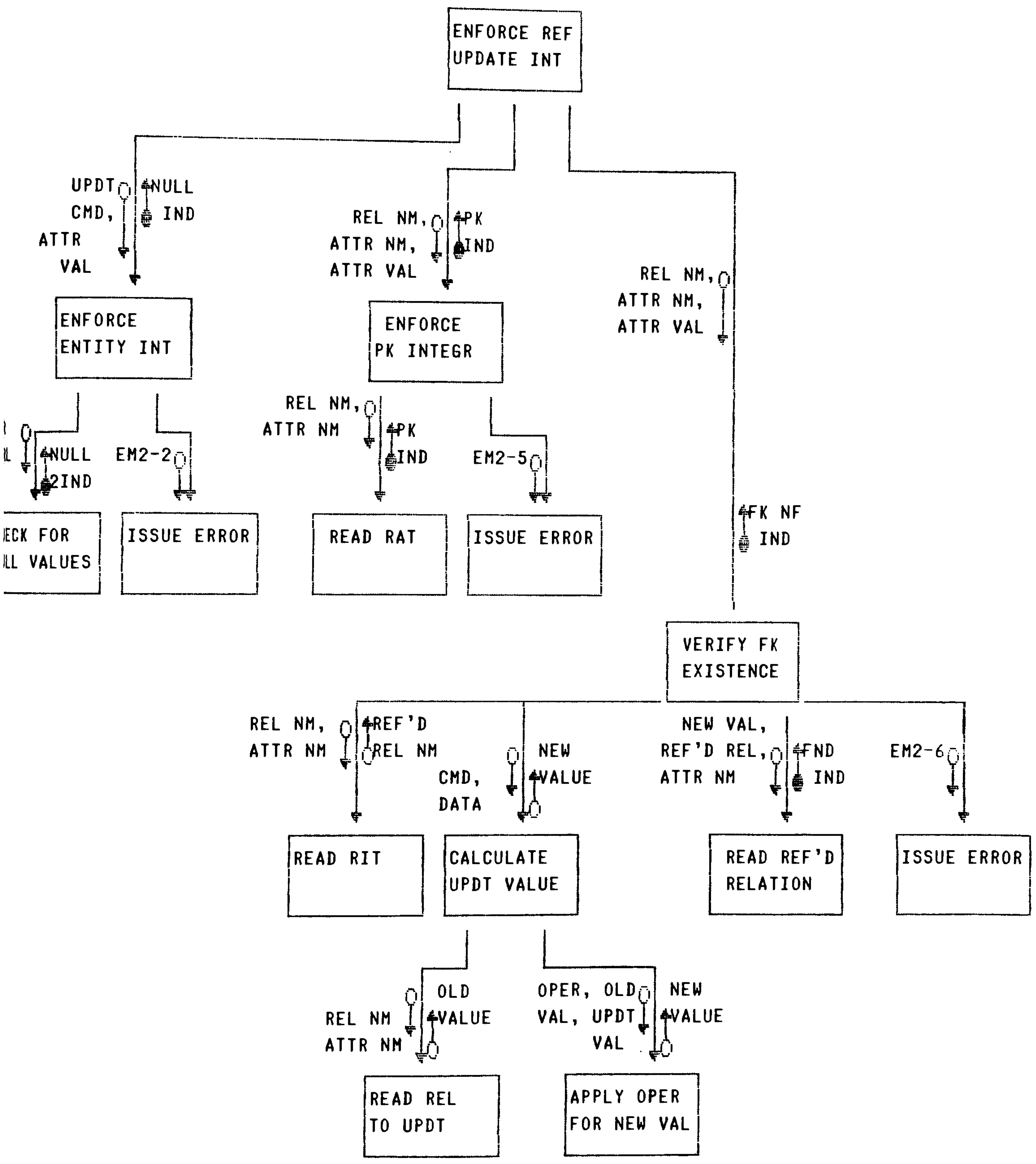

DELETE DOMAIN
Validate Dom name EM-3-5?
+
DOM NAME
EM3-2, EM3-6
CHECK FOR DOM USAGE
DOM NAME
EM3-2, EM3-6
READ RAT
DOM NAME
EM3-2
READ DIT
DOM NAME
DELETE DIT TUPLE
EM3-2, EM3-6
ISSUE ERROR


RECONCILE RIT
RIT DATA
EM3-4, NF CNTR, NF TBL
REF'D NM REL NM
EM3-4
NF TBL, NF CNTR
ERROR IND
GET REF'D REL NAME
UPDT RIT W/REF'D REL
ISSUE ERROR
DISPLAY NF TABLE
DISPLAY RE-CONCLE MNU
NAME, DOM
REF'D REL NAME
REL NAME
NF TBL
NF CNTR
NF CNTR
READ RAT
NOT FOUND TBL WRITE
INCR NF CNTR BY +1
RIT DATA
READ RIT


RIT ENTRY DELETE
+
REL NAME
EM3-7
CHECK FOR EXISTENCE
REL NAME
NF IND
READ RIT
REL NAME
DELETE RIT TUPLE
{SQLSDM1.1.C}
EM3-7
ERROR MSG EM3-7
{SQLSDM1.1.C}

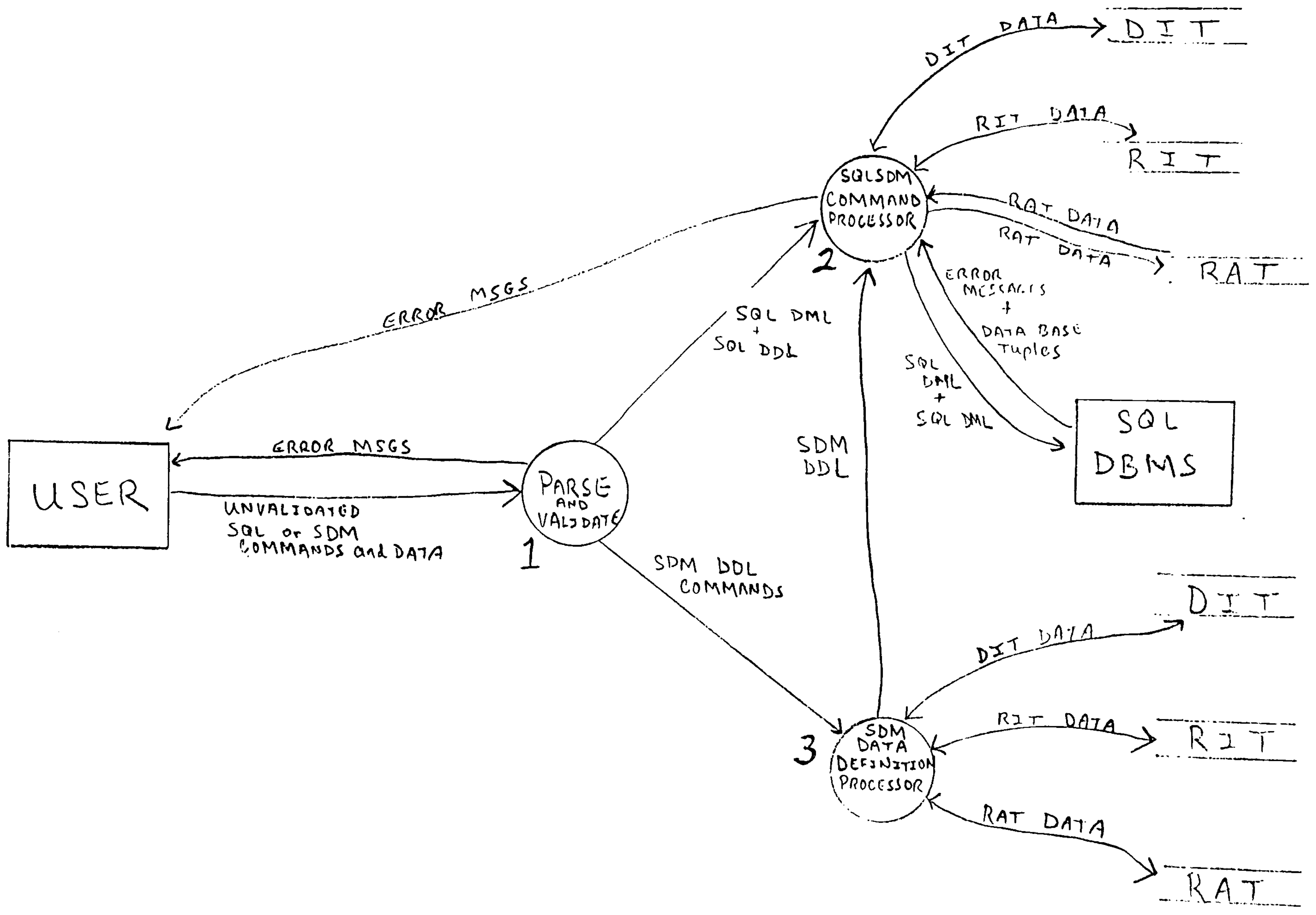

ENFORCE REL INTEG
SELECT CMD & DATA
MATCH IND
INSERT CMD & DATA
ERR IND
UPDT CMD & DATA
ERR IND
DELETE CMD & DATA
FND IND
ENFORCE DOM INT ON JOIN
ENFORCE REF INSERT INT
{SQLSDM1.2.A}
ENFORCE REF UPDATE INT
{SQLSDM1.2.B}
ENFORCE REF DELETE INT
CMD, DATA, REF'G REL NMS, ATTR NMS
FND IND
DOM NM
DOM NMS
MATCH IND
EM2-1
EAD RAT
COMPARE DOMAINS
ISSUE ERROR
FIND FK'S MATCH PK'S
REF'D REL NM & DATA
PK VALUES
REF'G REL NM, ATTR NM & DATA, PK VALUES
FND IND
EM2-7
READ REF'D REL FOR PKs
READ REF'G REL FOR FK
ISSUE ERROR
REF'D REL NM
REF'G REL NMS, ATTR NMS
READ RIT

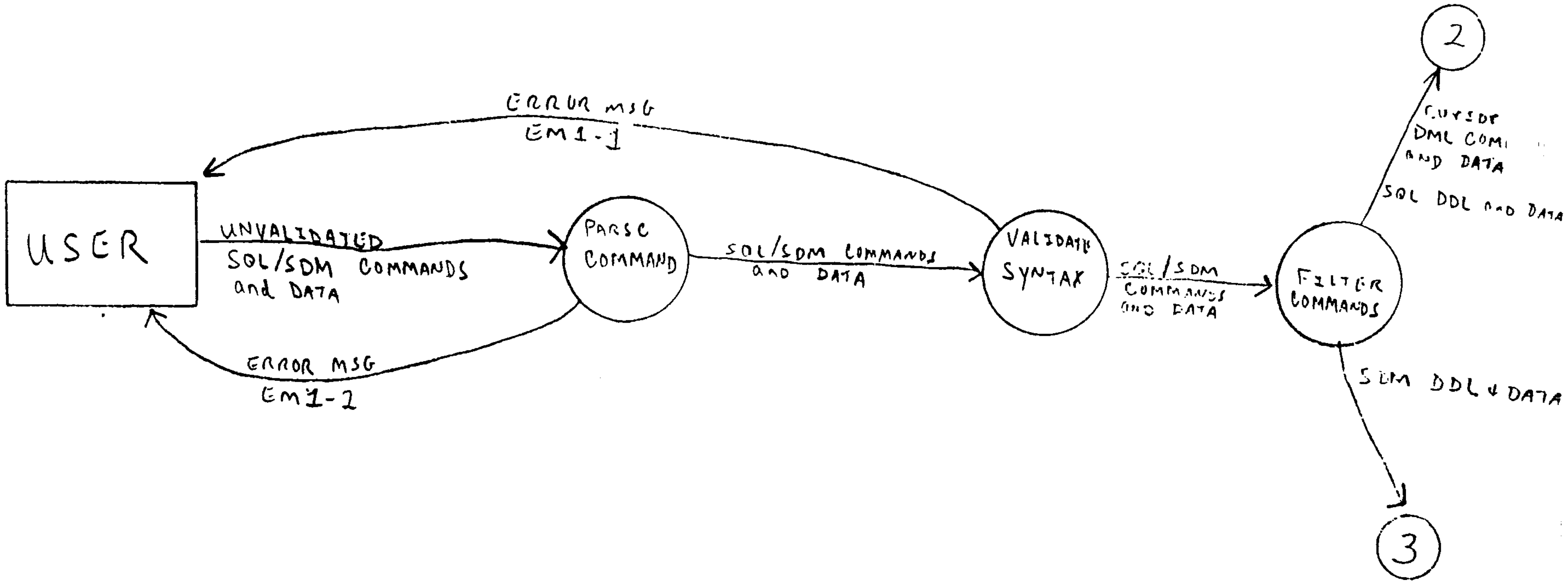

ENFORCE REF INSERT INT
ATTR DATA VALUES
NULL IND
REL NM, ATTR VALUES
ERROR IND
ENFORCE ENTITY INT
CHK FOR PK VAL= FK VAL
IR DATA
LUES
NULL IND
EM2-2
REL NM, ATTR VALUES
DUP IND
PK VALUES
EM2-4
ECK FOR
LL VALUES
ISSUE ERROR
GET REF'D RELATIONS
ISSUE ERROR
REL NM, ATTR VALUES
DUP IND
REF'D REL NMS, ATTR NMS
READ RIT
REL NM, PK VALUES
DUP IND
ATTR NMS
ENFORCE PK UNIQUENESS
EL NM, K VALUES
ATTR NMS
DUP IND
EM2-3
READ REL FOR DUP PK
ISSUE ERROR
EL NM
ATTR NMS
READ RAT

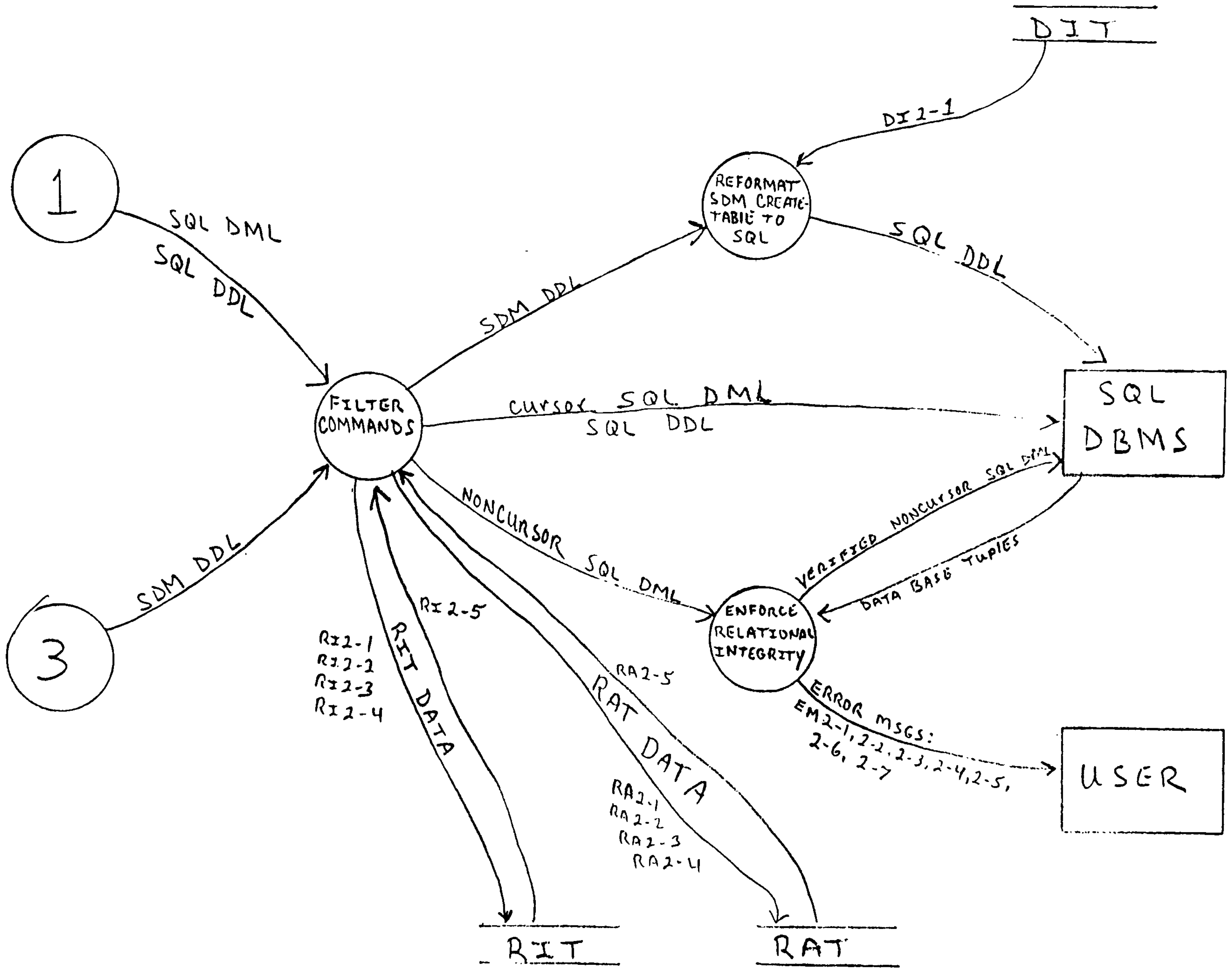

ENFORCE REF UPDATE INT
UPDT CMD, ATTR VAL
NULL IND
ENFORCE ENTITY INT
REL NM, ATTR NM, ATTR VAL
PK IND
ENFORCE PK INTEGR
REL NM, ATTR NM, ATTR VAL
FK NF IND
NULL 2IND
EM2-2
ECK FOR
LL VALUES
ISSUE ERROR
REL NM, ATTR NM
PK IND
READ RAT
EM2-5
ISSUE ERROR
VERIFY FK EXISTENCE
REL NM, ATTR NM
REF'D REL NM
READ RIT
CMD, DATA
NEW VALUE
CALCULATE UPDT VALUE
NEW VAL, REF'D REL, ATTR NM
FND IND
READ REF'D RELATION
EM2-6
ISSUE ERROR
REL NM ATTR NM
OLD VALUE
READ REL TO UPDT
OPER, OLD VAL, UPDT VAL
NEW VALUE
APPLY OPER FOR NEW VAL

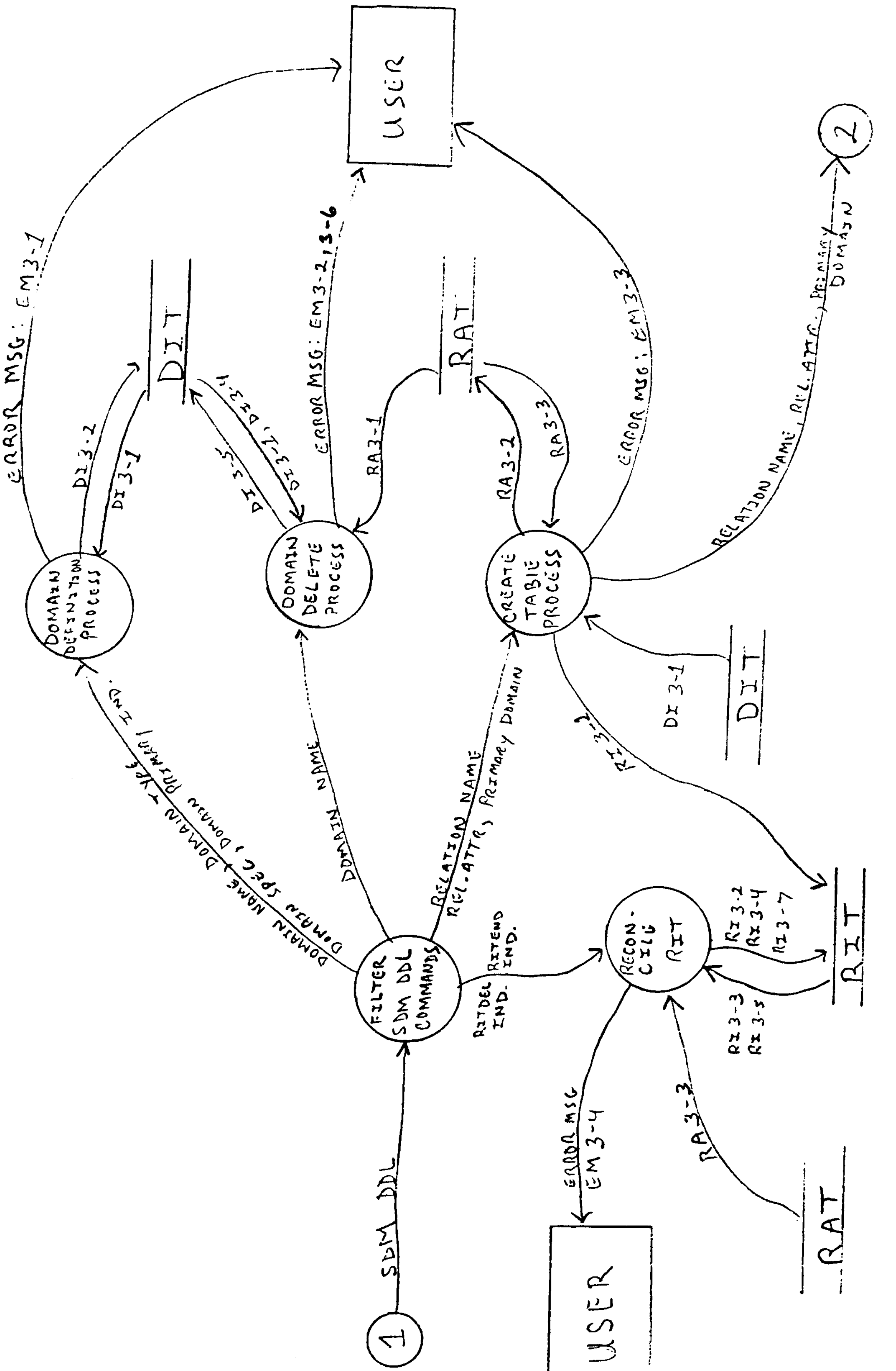
APPENDIX A2

SYSTEM DATA FLOW DIAGRAM


USER
ERROR MSGS
UNVALIDATED SQL or SDM COMMANDS and DATA
PARSE AND VALIDATE
1
ERROR MSGS
SQL DML + SQL DDL
SDM DDL COMMANDS
SQLSDM COMMAND PROCESSOR
2
DIT DATA
DIT
RIT DATA
RIT
RAT DATA
RAT DATA
RAT
ERROR MESSAGES + DATA BASE TUPLES
SQL DML + SQL DML
SQL DBMS
SDM DDL
SDM DATA DEFINITION PROCESSOR
3
DIT DATA
DIT
RIT DATA
RIT
RAT DATA
RAT


USER
UNVALIDATED SOL/SDM COMMANDS and DATA
PARSE COMMAND
ERROR MSG EM1-2
SOL/SDM COMMANDS AND DATA
VALIDATE SYNTAX
ERROR MSG EM1-1
SOL/SDM COMMANDS AND DATA
FILTER COMMANDS
CURSOR DML COM AND DATA
SOL DDL AND DATA
SDM DDL & DATA
2
3


DIT
DI2-1
1
SQL DML
SQL DDL
REFORMAT SDM CREATE-TABLE TO SQL
SQL DDL
SDM DDL
FILTER COMMANDS
CURSOR SQL DML
SQL DDL
SQL DBMS
3
SDM DDL
NONCURSOR SQL DML
VERIFIED NONCURSOR SQL DML
DATA BASE TUPLES
ENFORCE RELATIONAL INTEGRITY
ERROR MSGS: EM2-1, 2-2, 2-3, 2-4, 2-5, 2-6, 2-7
USER
RI2-1
RI2-2
RI2-3
RI2-4
RIT DATA
RI2-5
RA2-5
RAT DATA
RA2-1
RA2-2
RA2-3
RA2-4
RIT
RAT


1
SDM DDL
FILTER SDM DDL COMMANDS
DOMAIN NAME, DOMAIN SPEC, DOMAIN TYPE, DOMAIN PRIMARY IND.
DOMAIN DEFINITION PROCESS
DI 3-2
DI 3-1
DIT
ERROR MSG: EM3-1
USER
DOMAIN NAME
DOMAIN DELETE PROCESS
DI 3-5
DI 3-1, DI 3-4
ERROR MSG: EM3-2, 3-6
RA3-1
RAT
RELATION NAME REL. ATTR, PRIMARY DOMAIN
CREATE TABLE PROCESS
RA3-2
RA3-3
ERROR MSG: EM3-3
RELATION NAME, REL. ATTR., PRIMARY DOMAIN
2
DI 3-1
DIT
RI 3-1
RIT
RITEND IND.
RITDEL IND.
RECONCILE RIT
RI 3-2
RI 3-4
RI 3-7
RI 3-3
RI 3-5
RA3-3
RAT
ERROR MSG EM 3-4
USER

SQLSDM DATA OBJECT LEGEND

DIT DATA

DI 2-1	:=	DOMAIN TYPE AND/OR DATA SPECIFICATIONS
DI 3-1	:=	DOMAIN NAME
DI 3-2	:=	DOMAIN NAME, DOMAIN TYPE/SPEC, PRIMARY IND
DI 3-3	:=	DOMAIN NAME, PRIMARY IND
DI 3-4	:=	ALL DIT TUPLES
DI 3-5	:=	ALL DIT TUPLES EXCLUDING DELETED DIT TUPLES

RAT DATA

RA 2-1	:=	ATTRIBUTE NAMES
RA 2-2	:=	DOMAIN NAME
RA 2-3	:=	PRIMARY KEY INDICATOR
RA 2-4	:=	ALL RAT TUPLES
RA 3-1	:=	DOMAIN NAME
RA 3-2	:=	RELATION NAME, ATTRIBUTE NAME, DOMAIN NAME
RA 3-3	:=	RELATION NAME

RIT DATA

RI 2-1	:=	REFERENCED RELATION NAME
RI 2-2	:=	RELATION NAME
RI 2-3	:=	REFERENCING RELATION ATTRIBUTES (FOREIGN KEYS)
RI 3-1	:=	RELATION NAME, ATTRIBUTE NAME, PRIMARY DOMAIN
RI 3-2	:=	REFERENCED RELATION NAME
RI 3-3	:=	RELATION NAME
RI 3-4	:=	ALL RIT TUPLES
RI 3-5	:=	ALL RIT TUPLES EXCLUDING DELETED TUPLES

EM_3_10	INVALID SQL DATA SPECIFICATION
EM_3_11	SQLSDM DATABASE ERROR ... CONTACT DBA
EM_3_12	INVALID SQL RELATION NAME
EM_3_13	INVALID ATTRIBUTE NAME
EM_3_14	DUPLICATE ATTRIBUTE NAMES

INTER-PROCESS DATA OBJECTS LEGEND

SDM DDL COMMANDS	SQL OR SDM COMMANDS
CREATE TABLE	SDM DDL
CREATE DOMAIN	SQL DDL
DELETE DOMAIN	SQL DML:= CURSOR SQL DML + NON-CURSOR SQL DML
RITDEL	
RITEND	

SQL DDL COMMANDS COMM.	TRANSACTION TERM.
CREATE SCHMA AUTHORIZATION	COMMIT
CREATE TABLE	ROLLBACK
CREATE VIEW	
GRANT	

CURSOR SQL DML

OPEN
FETCH
UPDATE..CURRENT
DELETE..CURRENT
CLOSE

NON-CURSOR SQL DML

SELECT
INSERT
UPDATE
DELETE