CDIS: AN ENGINEERING CONSTRAINT DEFINITION AND INTEGRITY ENFORCEMENT SYSTEM FOR RELATIONAL DATABASES

By William J. Rasdorf, P.E. and
TsoJen E. Wang

CDIS: AN ENGINEERING CONSTRAINT DEFINITION AND INTEGRITY ENFORCEMENT SYSTEM FOR RELATIONAL DATABASES

By William J. Rasdorf, P.E. * and
TsoJen E. Wang **

ABSTRACT

During the design of a manufactured component large amounts of information pertaining to all aspects of the design must be stored, accessed, and operated upon. A database management system (DBMS), composed of a central repository of data and the associated software for controlling accesses to it and operations on it, provides one way to generate, represent, manage, and use this information.

This paper uses the relational database model to represent both design data and design constraints. Data integrity is defined and its enforcement through the use of engineering design constraints is described. Existing methods for handling constraints are discussed. A model that enables the engineer to associate design constraints with a relational database is presented and an example is given that demonstrates the concepts discussed. Extensions to a commercially available DBMS (designed for use by the engineering community) to implement the concepts presented is described.

THE RELATIONAL DATA MODEL FOR ENGINEERING DATABASES

Model Description

A data model defines the overall logical structure of a database [Ullman80]. It provides a structural framework into which the data is placed. A relational data model is a single level model consisting of a collection of interrelated relations represented in two-dimensional tabular form as shown in Figure 1 [Sandberg81]. Associated with the relations is a set of operators that perform the insertion, deletion, modification, and retrieval of data.

Figure 1 illustrates the structure of a relation. The rows of a relation are called tuples and its columns are called attributes. The domain of an attributes is the set of allowable values the attribute may possess. The values in all of the attributes of Figure 1 are drawn from either the domain of characters, the domain of positive real numbers, or the domain of positive integer numbers. Each tuple represents an individual component and contains a value for each attribute. All tuples are distinct; duplicates are not permitted [Rasdorf82]. Tuples and attributes have no order; they may be arbitrarily interchanged without changing the data content and the meaning of the relation.

* Assistant Professor of Civil Engineering and Computer Science, North Carolina State University, Box 7908, Raleigh, North Carolina, 27695.

** Graduate Research Assistant, Computer Studies Program, North Carolina State University, Box 8207, Raleigh, North Carolina, 27695.

Tuples are accessed by means of a key, a single attribute or a combination of attributes that uniquely identifies a tuple.

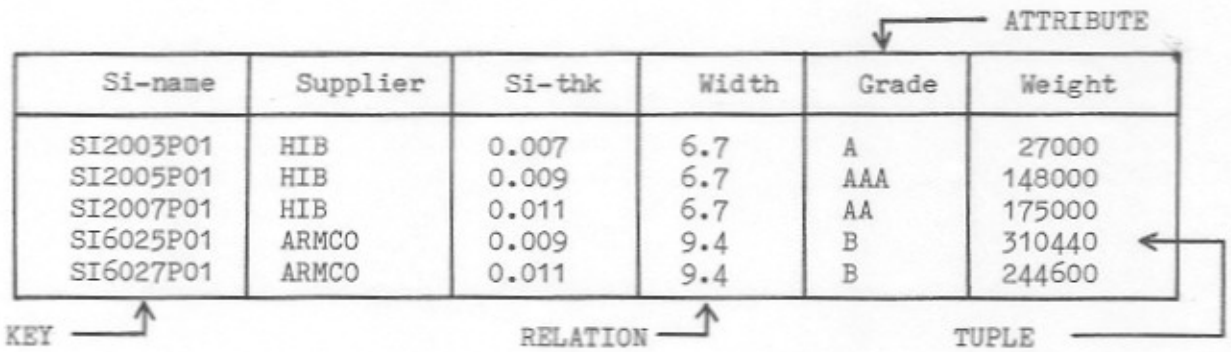
A standard shorthand notation to represent relations is as follows:

RELATIONname (KEYATTRIBUTEname, ATTRIBUTE2name, ...)

with the SI-IRON relation of Figure 1 being represented as:

SI-IRON(Si-name, Supplier, Si-thk, Width, Grade, Weight)

The name of the relation is listed first, followed in parentheses by the names of all of its attributes. The underlined attribute of a relation is the key.



Si-name	Supplier	Si-thk	Width	Grade	Weight
SI2003P01	HIB	0.007	6.7	A	27000
SI2005P01	HIB	0.009	6.7	AAA	148000
SI2007P01	HIB	0.011	6.7	AA	175000
SI6025P01	ARMCO	0.009	9.4	B	310440
SI6027P01	ARMCO	0.011	9.4	B	244600

Figure 1. The Structure of Relations

The overall logical structural of a database is called its schema. When all relations have been defined, and their attributes and associated domains specified, the schema is defined. A subset of the total schema is referred to as a view or subschema. The subschema allows the user to access a limited amount of data enabling him to have a unique restricted perspective of the database.

A Manufacturing Database Schema

All of the illustrations in this paper are based on a manufacturing database which includes the following, SI-IRON, CA-STEEL, AL-ROD, RND-WIRE, and RCT-WIRE relations:

Relations:
SI-IRON(Si-name, Supplier, Si-thk, Width, Grade, Weight)
CA-STEEL(Ca-name, Supplier, Ca-thk, Width, Grade, Weight)
AL-ROD(Dia, Grade, Supplier, Weight)
RND-WIRE(Fw-name, Supplier, Dia, Matl, Insl-thk, Weight)
RCT-WIRE(Fw-name, Supplier, Width, Height, Matl, Insl-thk, Weight)

These relations have been extracted from a transformer manufacturing database schema. The attributes are defined below:

Attributes:
Si-name = Identification name of silicon iron sheets
Supplier = Identification name of material supplier

Si-thk	=	Thickness of silicon iron sheets (inches)
Ca-thk	=	Thickness of carbon steel sheets (inches)
Width	=	Width of silicon iron or carbon steel sheets (inches), or Horizontal dimension of rectangular finished wire (inches)
Grade	=	Quality level of silicon iron or carbon steel
Weight	=	Weight of the rolls of silicon iron, carbon steel, round wire, or rectangular wire currently in stock (pounds)
Ca-name	=	Identification name of carbon steel sheets
Dia	=	Diameter of aluminum rod or round finished wire (inches)
Fw-name	=	Identification name of finished wire
Matl	=	Identification name of basic material (Copper or Aluminum)
Insl-thk	=	Thickness of the insulation on finished wire (inches)
Height	=	Vertical dimension of rectangular finished wire (inches)

The relations listed above describe the physical characteristics and materials that are used in manufacturing a transformer. During the transformer manufacturing process silicon iron sheets are used to make the core of the transformer and carbon steel sheets are used to make the outer covering of the tank that protects the core. Both the silicon iron and carbon steel are purchased in units of rolls. Thus, for each sheet type, information such as the supplier name, the dimensions of the cross-section, the quality level, and the total weight currently in stock must be stored in the database. SI-IRON and CA-STEEL are the relations that store this information about silicon iron and carbon steel, respectively.

AL-ROD contains information about the aluminum rods that are used when the core is made. Diameter and Grade form the composite key that uniquely distinguishs each tuple. A composite key is needed because no particular name is given to aluminum rods. As in the previous relations, the name of the rod supplier is given as well as the total weight of rod currently in stock.

The core of a transformer is surrounded by either round or rectangular wires. RND-WIRE contains an identification name used to distinguish between round wire types, the supplier name, the diameter of the wire, the material that the wire is made of, the thickness of the insulation surrounding the wire, and the total weight of such wire currently in stock. RCT-WIRE contains a similar set of attributes except that diameter is replaced by the dimensions of a rectangular section.

DATABASE INTEGRITY

Integrity of a database refers to the maintenance of functional relationships between data items [Date83, 86]. Dependent data item values are derived on the basis of the constraints among a collection of ingredient data items. The value of the dependent data may either be computed when it is needed, or it may be redundantly stored and its consistency maintained through constraints.

Constraints are relationships among data item values that are directly embedded in the database [Fenves82]. Constraints are used to guarantee the integrity of the database when operations are performed on it. Constraints are therefore concerned with the correctness of data.

Integrity plays an important and essential role in databases. In the past, much of the integrity checking in both business and engineering databases

was performed by application programs. More recently, DBMSs have been incorporating into their structure specifications for a limited set of integrity constraints and the mechanisms for invoking them automatically. If an engineering design DBMS lacks a constraint processing mechanism, there can be no guarantee that the design data will be valid. To ensure the correctness of design data, an effective constraint management capability must be incorporated into any proposed engineering design DBMS. This paper discusses how this can be achieved and proposes a systematic way to classify constraints so that integrity can be maintained efficiently.

Rasdorf [Rasdorf82] first utilized functions and procedures to monitor constraints in a structural design database for buildings. Fenves and Rasdorf [Fenves85, Rasdorf86a] then presented a mechanism for representing and processing single-relation, single-tuple constraints in a relational engineering design database. Schaefer [Schaefer82] extended constraint processing considerations to include data stored in more than one relation. Rasdorf and Chung [Rasdorf84] proposed further extensions and developed a formalism for classifying all possible types of constraints that can be imposed on relations. Rasdorf and Ulberg [Rasdorf86b] then examined and asserted the applicability of the constraint classification scheme. This paper illustrates the implementation of a portion of the constraint processing mechanism using an existing DBMS.

ENGINEERING CONSTRAINTS

Constraints restrict the domain of a data item, i.e., they restrict its allowable values. Constraints can be used for two distinct purposes: checking and assignment [Fenves82, 85].

Checking constraints are passive constraints that use existing data item values to determine whether a relationship is satisfied, i.e., whether the prescribed functional relationship between the constrained data items exists. A checking constraint results in the determination of a boolean value (true meaning satisfied or false meaning violated) representing the value of the constraint. Examples of checking constraints for satisfying diameter restrictions on round wire in the RND-WIRE relation are:

```
Dia10K := Dia >= 0.0308  (for aluminum wire)
Dia20K := Dia <= 0.1290  (for aluminum wire)

Dia30K := Dia >= 0.0210  (for copper wire)
Dia40K := Dia <= 0.0987  (for copper wire)
```

In order for the transformer core to be successfully wrapped, the above constraints state that the diameter of the round wire must be within the specified range. Since constraints are directly associated with relations, their values are determined on a tuple by tuple basis when data values are entered into the database. If the incoming data are within the range, then DiaOK is set to TRUE for that tuple, otherwise it is set to FALSE. The addition of this constraint to the DBMS facilitates the design checking process since a FALSE value in immediately indicates that the range requirement has not been fulfilled and the corresponding data must not be used.

Checking constraints can also involve ingredient data items from more than one tuple. An example of such a checking constraint on the SI-IRON relation is:

WeightOK := SUM(Weight WHERE Grade = A) >= 300000

So that scheduled productivity can be maintained, this constraint states that the total weight of grade A silicon iron currently in stock must be greater than the specified value. To enforce the constraint requires a calculation of the sum of all of the weights, each of which must be drawn from a qualifying tuple.

Assignment constraints, on the other hand, are active constraints that assign a value to an unknown, or designable, data item as a function of given values of its ingredient data items consistent with an applicable constraint [Rasdorf84]. Since the assignment is made through the use of a constraint, the constraint itself is automatically satisfied. An example of an assignment constraint that assigns a numerical value to the aspect ratio of a rectangular wire follows:

Aspect-ratio := Width / Height

As values for "Width" and "Height" are entered into the database, the value of the corresponding "Aspect-ratio" is calculated and immediately stored. During the lifespan of the database, the value of either "Width" or "Height" can be changed and a new value for "Aspect-ratio" will be generated. Thus, not only can the engineer be freed from tedious calculation but he can have confidence that data item values are determined according to a precisely predefined relationship.

CONSTRAINT ENFORCEMENT IN THE RELATIONAL DATABASE MODEL

The last section introduced and described constraints. This section indicates methods by which constraints can be associated with, and enforced with respect to, relational databases. Two approaches to constraint handling are described. The first shows how an existing DBMS currently supports constraint handling; the second shows extensions to enable the DBMS to more broadly encompass the wide range of engineering constraints encountered in practice.

To implement the concepts presented herein a public domain version of a commercially available database management system, Boeing Computer Service's Relational Information Manager (BCS RIM), was used [Erickson81, BCS83].

Computed Attributes

In the RIM DBMS, assignment constraints are referred to as computed attributes. These are specified in the familiar form:

attname = "expression"

where attname is the name of the computed attribute and expression is a mathematical combination of defined attributes, constants, and/or operators [BCS83]. The value of a computed attribute need not be explicitly loaded into the database, it is automatically computed according to the associated expression.

Computed attributes can be used for checking constraints as well as assignment constraints. Suppose that the attribute "Aspect-ratio" is included

in the RCT-WIRE relation. Using a computed attribute to determine a value for "Aspect-ratio" results in the assignment seen earlier:

Aspect-ratio = "Width / Height"

A different approach, however, is taken to enforce the checking constraint "Aspect-ratioOK" on relation RCT-WIRE. A computed attribute "Differ" is first defined to compute the difference between "Aspect-ratio" and "Width" divided by "Height":

Differ = "ABS(Aspect-ratio - Width / Height)"

"Aspect-ratioOK" can then be expressed as:

Aspect-ratioOK = "IFLT(Differ,0.1,"TRUE","FALSE")"

IFLT(A1,A2,A3,A4) is a multi-operand function which returns A3 if A1 is less than A2; otherwise it returns A4. Because of potential errors due to rounding of values "Aspect-ratioOK" is determined to be TRUE if the value of "Differ" is within the specified tolerance (0.1).

Rules

In the RIM DBMS, checking constraints which do not return a value are referred to as rules. Rules are specified in the form:

```
RULES
attname [IN relname] {EQS} value [{AND} attname ...]
                     NES         OR
                      .
                      .

attname1 IN relname {EQA} attname2 IN relname [{AND} ...]
                    NEA                        OR
                      .
                      .
```

where attname is the name of the attribute involved in a rule and relname is the relation in which the attribute resides. Through the use of rules an attribute value can be related to a constant, a range of values, or another attribute (in the same relation or in a different relation) [BCS83, Stonebraker83]. The checking constraint "DiaOK" on relation RND-WIRE can be enforced using the following rules:

Dia IN RND-WIRE GE 0.0308 AND Dia IN RND-WIRE LE 0.1290
(for aluminum wire)

Dia IN RND-WIRE GE 0.0210 AND Dia IN RND-WIRE LE 0.0987
(for copper wire)

In this example each rule relates the value of the attribute "Dia" to a constant.

One disadvantage of the rule facility is that there is no way to specify a subset of tuples to which a constraint is to be applied. In the example given

above both rules will be applied to every tuple in the RND-WIRE relation regardless of what the associated wire is made of. To guarantee that the rules are applied to the correct tuple, the relation must be broken into two new relations, one containing information for aluminum wire and one for copper. Another disadvantage of the rule facility (and computed attributes as well) is that it cannot be used to enforce a constraint that involves ingredient data items from more than one tuple. Hence the "WeightOK" checking constraint for the SI-IRON relation can only be enforced by coding it in application programs.

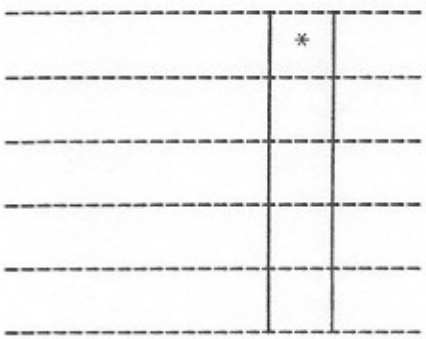
Computed attributes and rules are RIM features that provide a strong underlying level of integrity maintenance. Rules enable the designer to enforce constraint relationships. Computed attributes enable him to assign values to constrained data items in such a way that the constraint is automatically satisfied. Thus, in addition to integrity maintenance, the assignment of attribute values by the DBMS itself is achieved. However, a more versatile and flexible mechanism for handling constraints of both types is needed. In particular, constraints whose ingredient and dependent data items reside in multiple tuples or multiple relations must be handled. Computed attributes can only be used to enforce constraints that involve data items within a single tuple of a single relation. Rules extend beyond the boundaries of a single relation, but the ingredient data items for a rule are still restricted to be drawn from single tuple. It is not possible, therefore, to enforce checking of data with respect to a wide variety of constraints. The extensions described in the following sections, which address the checking aspect of constraint use but do not deal with assignment constraints, overcome these limitations.

Structured Constraints

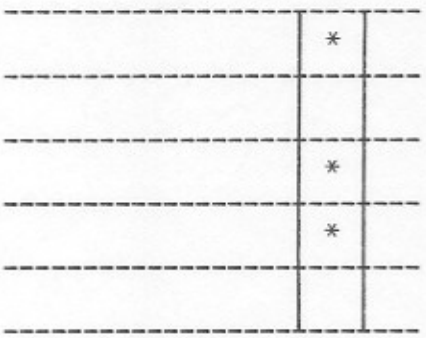
The constraints defined herein are based on the structure of the relations that comprise the relational database model. In fact, they are classified in terms of the location of the ingredient data items (attributes) that are needed to evaluate a constraint when it is invoked [Rasdorf84]. Thus, constraint values may be drawn from one or more relations, rows, columns, or row-column intersections. Hence a constraint can be defined to be any one of the following types:

Single	Relation	- Single	Attribute	- Single	Tuple	<SR-SA-ST>
Single	Relation	- Single	Attribute	- Multiple	Tuples	<SR-SA-MT>
Single	Relation	- Single	Attribute	- All	Tuples	<SR-SA-AT>
Single	Relation	- Multiple	Attributes	- Single	Tuple	<SR-MA-ST>
Single	Relation	- Multiple	Attributes	- Multiple	Tuples	<SR-MA-MT>
Single	Relation	- Multiple	Attributes	- All	Tuples	<SR-MA-AT>
Multiple	Relations	- Single	Attribute	- Single	Tuple	<MR-SA-ST>
Multiple	Relations	- Single	Attribute	- Multiple	Tuples	<MR-SA-MT>
Multiple	Relations	- Single	Attribute	- All	Tuples	<MR-SA-AT>
Multiple	Relations	- Multiple	Attributes	- Single	Tuple	<MR-MA-ST>
Multiple	Relations	- Multiple	Attributes	- Multiple	Tuples	<MR-MA-MT>
Multiple	Relations	- Multiple	Attributes	- All	Tuples	<MR-MA-AT>

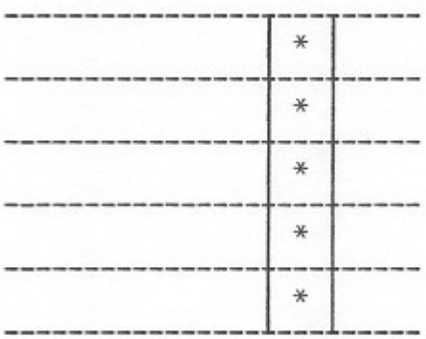
Figure 2 illustrates the physical position of where the ingredient data items reside within the relation for the single relation constraint types. The '*' mark denotes the cell (s) [row(tuple) - column(attribute) intersection] that is involved in the constraint.



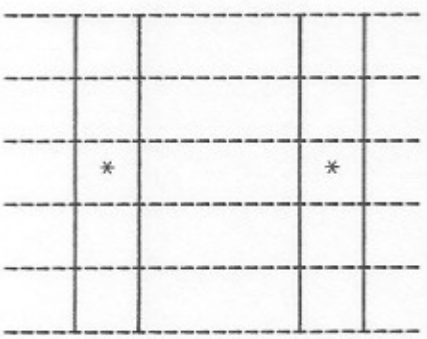
(a) SR-SA-ST



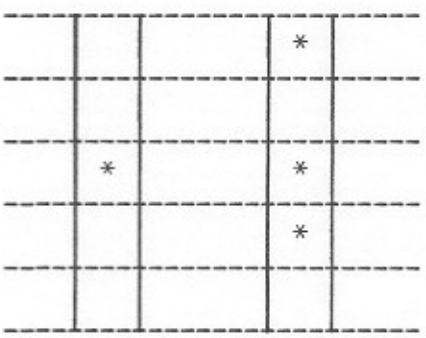
(b) SR-SA-MT



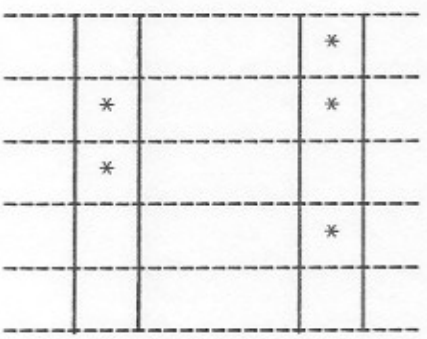
(c) SR-SA-AT



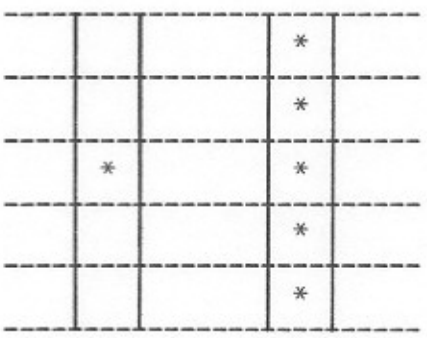
(d) SR-MA-ST



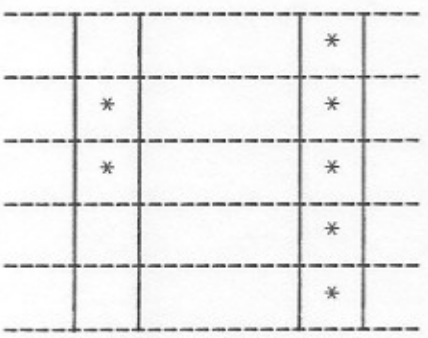
(e) SR-MA-MT



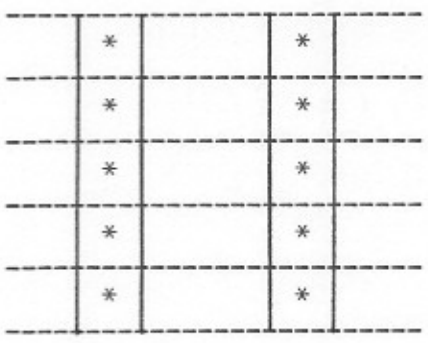
(f) SR-MA-MT



(g) SR-MA-AT



(h) SR-MA-AT



(i) SR-MA-AT

Figure 2. Illustration of single relation Structured Constraints

A SR-SA-ST constraint is illustrated in Figure 2(a) where a single cell provides the data needed to evaluate this type of constraint. The range checking constraint on "Dia" in the RND-WIRE relation is an example of a SR-SA-ST constraint:

DIA IN RND-WIRE <= 0.1290

A SR-SA-MT constraint is illustrated in Figure 2(b) where the data from several cells of a single attribute are needed to evaluate the constraint. The constraint on "WeightOK" in the SI-IRON relation is an example of a SR-SA-MT constraint:

SUM (Weight WHERE Grade = A) >= 300000

Note that the attribute "Grade" is a qualifier; the value of "A" is used only to specify the subset of tuples from which the values of "Weight" are drawn. The actual data needed to evaluate the constraint comes only from the attribute "Weight".

If the evaluation of a constraint requires data from every tuple in a relation, it is a SR-SA-AT constraint as illustrated in Figure 2(c). Such a constraint on the "Weight" attribute of the SI-IRON relation would be:

SUM (Weight) <= 5000000

This constraint indicates that the total weight of all the silicon iron rolls currently in stock must not exceed the specified value (which might represent a warehouse capacity).

In some cases the evaluation of a constraint requires data from more than a single attribute. Such constraints are categorized as SR-MA constraints. Additionally, if the data that is required in the evaluation of a constraint resides in two or more relations the constraint is of the MR type. Examples for these constraint types are not presented here but can be found in references [Rasdorf84, 86b].

The significant advantage of the Structured Constraint formalism is its generality. Because the formalism is based on the structure of the relational

model, it can represent any definable constraint and is database independent. Since a Structured Constraint can involve attributes that are drawn from several relations, the integrity among relations is maintained without the need for excessive or unnecessary normalization. No other formalism can represent as wide a variety of constraints.

A prototype system called CDIS (Constraint Definition Integrity System) has been successfully built at North Carolina State University to implement a portion of the formalism presented herein. The following section describes the capability of the CDIS and discusses possible extensions to it and improvements in its functionality.

CONSTRAINT DEFINITION INTEGRITY SYSTEM (CDIS)

General View

CDIS is an interactive system developed to implement the integrity system first proposed by Rasdorf and Chung [Rasdorf84]. This implementation was written in FORTRAN-77 and runs on the VMS4.1 operating system of a VAX 11/750 computer. RIM is itself written in FORTRAN-66. CDIS consists of two major components: the first is called the Constraint Definer (CD), and the second is called the Integrity System (IS).

The CD provides a way for the user to define Structured Constraints. The CD currently allows the user to define four types of constraints, namely SR-SA-ST, SR-SA-MT, SR-SA-AT, and SR-MA-ST, on any of the relations of a database. A limit of twenty constraints per relation and 999 constraints per database was chosen for the initial implementation. These limitations are a function of the naming convention of the constraints and can be easily modified as needed.

The IS is the system which actually performs the integrity maintenance during database operation. Its goal is to disallow any operation which would result in invalid data in the database. As long as the IS is active, it will closely monitor the data manipulation process (including data addition and data modification) to ensure that no constraints are violated. If a constraint is violated, the IS will reject the attempted manipulation thereby safeguarding the integrity of the database.

Complicating the implementation of the CDIS is the aspect of time. In the early stages of an engineering design many data item values are as yet unknown. They may be determined at varying points in the design process. Until they are determined null values must be inserted into the database in their place. Although the presence of null data in the database would normally cause the constraint(s) that is (are) associated with the data to be violated, our implementation does tolerate this necessary situation. Whenever a null value is encountered in the integrity checking process, the constraint associated with it is simply not invoked. Later, when the missing data is (are) added to the database, the IS will automatically be activated and the validity of the data will then be checked.

There are two approaches to implementing the formalism introduced herein. The first is to cast all defined constraints as functions, embed them in the DBMS, and record their presence in two constraint relations stored directly in the database. The underlying DBMS routines for handling the relational algebra LOAD and CHANGE commands are then modified so that they check the constraint

relations to determine if constraints exist before data is loaded or changed. The corresponding checking functions are then called as needed and database operations are accepted or rejected accordingly. This is the approach adopted in CDIS.

The other approach to implementing the Structured Constraint formalism is to store the constraints in equation form in the Integrity System. This approach requires symbolic manipulation to achieve the reformulation necessary to solve for arbitrary unknowns, yet the coupling between a symbolic processing language (e.g. LISP, PROLOG, etc.) and a general purpose programming language (e.g. FORTRAN, PASCAL, PL/I, etc.) has not been well established. Even given the presence of a tight language coupling this approach suffers from the fact that each time a constraint is invoked the symbolic manipulation routines have to be called. This process is definitely slower than making a FORTRAN subroutine call.

The Constraint Definer

Prior to defining constraints on a database with the CD, two constraint relations must also be defined on the database to store the information describing the constraints. These two relations are CONATT and CONTBL:

CONATT(Connam, Contyp, Relnam, Contxt)
CONTBL(Attnam,Connam)

where "Connam" is the name of each defined constraint (given by the CD during constraint definition process), "Contyp" is the type of constraint based on the Structured Constraint classification scheme, "Relnam" is the name of the relation whose attributes are being constrained, "Contxt" is the text form of the constraint, and "Attnam" is the name of the attribute being constrained.

There are two reasons for generating these relations. From the perspective of the database user, they provide useful information about the constraints. The user can view these relations with standard DBMS commands, enabling him to determine, for example, how many constraints have been defined, their content, which attributes and which relations are constrained, etc. From the perspective of the Integrity System, these relations provide the information essential for its operation.

To define a constraint using the CD the user simply enters the constraint in equation form according to the syntax given in the Appendix. The CD automatically generates the corresponding FORTRAN checking function and enters information describing the constraint into the two constraint relations. These operations are transparent to the user. Upon the completion of a constraint definition session, the user compiles the FORTRAN functions generated during the session and links them to the IS using standard operating system compile and link commands.

As an example the constraints given earlier for "DiaOK", "WeightOK", "Sum-wtOK", and "Aspect-ratioOK" would be entered by the user in the following form:

RND-WIRE.Dia LE 0.1290

SUM SI-IRON.Weight WHERE Grade EQS A GE 300000.0

SUM SI-IRON.Weight LE 5000000.0

RCT-WIRE.Aspect-ratio EQ RCT-WIRE.Width * RCT-WIRE.Height

Sample on-line sessions illustrating the use of the CD to define a set of Structured Constraints and showing how the IS actually works during the data insertion and manipulation processes can be found in reference [Wang85]. A discussion of efficiency, constraint activation, applicability to other database models, and additional issues will be given in a subsequent paper.

The Integrity System

Within the IS, there are three commands available to the user to control the invocation of constraints: ACTIVATE, DEACTIVATE, and INVOKE. These commands essentially enable the user to turn the constraints on and off. When the ACTIVATE command is entered, the IS monitors both the addition of data to the database and the modification of data already in the database. Whenever a violation of a constraint is detected, the attempted operation will be rejected. If the user does not want the IS to interfere with the data manipulation processes he can issue the DEACTIVATE command to prevent any checking from occurring.

Normally the IS checks only those transactions carried out after the system has been ACTIVATEd. There are occasions that the user may wish to DEACTIVATE the system for a period of time and reactivate it after some collection of transactions are completed. If the IS is DEACTIVATEd during the data manipulation period, inconsistent or invalid data may be entered into the database. To correct this unacceptable situation and to enforce the integrity maintenance, one can issue the INVOKE command. Upon receiving the command, the IS checks the database for conformance with all defined constraints. When an integrity violation is detected, the IS will report the discovery to the user by showing both the tuple where the bad data resides and the constraint(s) that is (are) violated. The user can then take the appropriate action to correct the violation.

Future Extensions

A number of enhancements for CDIS are planned. One item of priority is to extend the constraint definition capability for multiple-attribute and multiple-relation constraints. Another item of priority is to expand the extent of the checking capability so that the constraints are checked when tuples are DELETEd as well as when they are INSERTed and CHANGEd.

Regarding the constraint control commands, the current commands to ACTIVATE, DEACTIVATE, and INVOKE apply collectively to all constraints that have been defined. CDIS will be modified to allow the user to specify individually the one or more constraints to be ACTIVATEd, DEACTIVATEd, or INVOKEd. An additional command called DISCARD will be provided to allow the user to discard inappropriate, outdated, or unnecessary constraints.

Because engineering databases often contain attributes which have either vector or matrix data types, an additional objective is to revise the current implementation to enable it to handle such extended atomic relational data types. Lastly, a more friendly user interface will be provided to enable the

system to automatically classify as well as process the constraints defined by a user.

CONCLUSIONS

Relational databases possess characteristics that make them desirable for engineering use. In particular they are amenable to flexibly representing complex engineering data relationships, including those defined by codes, standards, and specifications, in the form of constraints which can be used to check and generate data item values. In short, relational databases can be used to directly contribute to the engineering data generation processes.

The formalism presented in this paper is one component of a more general approach (further described in reference [Fenves85]) to handling checking and assignment operations in a relational DBMS environment. The implementation described herein shows how the checking portion of the model could be incorporated into a commercially available DBMS. No currently available DBMSs provides the much needed capabilities proposed here. The contribution of this study, therefore, fulfills a unique and important engineering need.

ACKNOWLEDGEMENTS

The Relational Information Manager (RIM) DBMS was used to implement the database presented herein [Erickson81]. RIM was originally developed by Boeing for NASA and was used for the space shuttle program. A version called BCS RIM is currently available from Boeing Computer Services Co. The BCS RIM DBMS is compatible with the R:BASE Series 5000 microcomputer DBMS of MicroRIM, Inc. [MicroRIM85] allowing data to be transferred between micro and mainframe computers. All RIM DBMSs support an interface to FORTRAN programs, an essential capability for engineering applications.

Portions of this work were sponsored by the National Science Foundation under grant CEE-8319869 entitled "Generative Database Systems for Structural Engineering Design" and by a NSF Presidential Young Investigator Grant MSM-8451465.

REFERENCES

[BCS83] Boeing Computer Services Company, BCS RIM Relational Information Management Systems Version 6.0 Users Guide, Boeing Computer Services Company (BCS), P. O. Box 24346, Seattle, Washington, July, 1983.

[Date83] Date, C. J., An Introduction to Database Systems, Volume 2, First Edition, Addison Wesley, Reading, Massachusetts, 1983.

[Date86] Date, C. J., An Introduction to Database Systems, Volume 1, Fourth Edition, Addison Wesley, Reading, Massachusetts, 1986.

[Erickson81] Erickson, W. J., Gray, F. P. and Limbach, G., Relational Information Management System, Version 5.0 edition, Boeing Commercial Airplane Company, P. O. Box 3707, Seattle, WA, 1981.

[Fenves82] Fenves, S. J. and Rasdorf, W. J., "Role of Database Management Systems in Structural Engineering," Proceedings of the IABSE Colloqium on Informatics in Structural Engineering, International Association of Bridge and Structural Engineers, (IASBE), Bergamo, Italy, Pages 229-242, October, 1982.

[Fenves85] Fenves, S. J. and Rasdorf, W. J., "Treatment of Engineering Design Constraints in a Relational Database," Engineering With Computers, Volume 1, Number 1, Pages 27-37, Spring, 1985.

[MicroRIM85] R:BASE Series 5000 User's Manual and Tutorial, Version 1.15, MicroRIM Inc., Bellevue, WA, October, 1985.

[Rasdorf82] Rasdorf, W. J., "Structure and Integrity of a Structural Engineering Design Database," Technical Report DRC-02-14-82, Design Research Center, Carnegie-Mellon University, Pittsburgh, Pennsylvania, April, 1982.

[Rasdorf84] Rasdorf, W. J. and Chung, T., "Specification and Implementation of Constraints in a Relational Database for Engineering Design," Civil Engineering Report CE-002-84, North Carolina State University, Raleigh, North Carolina, May, 1984.

[Rasdorf86a] Rasdorf, W. J. and Fenves, S. J., "Constraint Enforcement in a Structural Design Database," Journal of the Structural Division, American Society of Civil Engineers, submitted for publication.

[Rasdorf86b] Rasdorf, W. J. and Ulberg, K., "A Model of Semantic Integrity Constraints Based on the Structure of the Relational Database Model," submitted for publication.

[Sandberg81] Sandberg, G., "A Primer on Relational Database Concepts," IBM Systems Journal, Volume 20, Number 1, Pages 23-40, 1981.

[Schaefer82] Schaefer, M. J., "Constraint Processing Alternatives in an Engineering Design Database," Technical Report DRC-01-13-82, Design Research Center, Carnegie-Mellon University, Pittsburgh, PA, October, 1982.

[Stonebraker83] Stonebraker, M., Woodfill, J. and Anderson, E., "Implementation of Rules in Relational Data Base Systems," Database Engineering, Institute of Electrical and Electronic Engineers, Volume 6, Number 4, Pages 65-74, December, 1983.

[Ullman80] Ullman, J. D., Principles of Database Systems, Computer Science Press, Inc., Potomac, MD, 1980.

[Wang85] Wang, T. E. and Rasdorf, W. J., CDIS - Constraint Definition Integrity System Version 1.0 User Guide, Computer Studies Program, Box 8207, North Carolina State University, Raleigh, NC, October, 1985.

APPENDIX

Syntax of Single Relation Structured Constraints

<SR-SA-ST> ::= <attribute> <relational operator> <constant>

<SR-SA-MT> ::= <computational operator> <attribute> <where clause>
<relational operator> <constant>

<SR-SA-AT> ::= <computational operator> <attribute> <relational operator>
<constant>

<SR-MA-ST> ::= <attribute> <relational operator> [<attribute> ¦
<numerical expression>]

<SR-MA-MT> ::= "<computational operator>" <attribute> <where clause>
<relational operator> <computational operator>
<numerical expression> <where clause>

<SR-MA-AT> ::= "<computational operator>" <attribute> "<where clause>"
<relational operator> <computational operator>
<numerical expression>

<arithmetic operator> ::= + ¦ - ¦ * ¦ / ¦ **

<attribute> ::= <relation name> . <attribute name>

<attribute name> ::= (the name of a valid attribute)

<clause 1> ::= <attribute> [EXISTS ¦ FAILS]

<clause 2> ::= <attribute> <relational operator> <constant>

<clause 3> ::= <attribute> EQ [MAX ¦ MIN]

<clause 4> ::= <attribute> EQS <text>

<clause 5> ::= <attribute> <relational operator 2> <attribute>

<clause 6> ::= ROWS <relational operator> <integer>

<clause 7> ::= [<attribute> ¦ ROWS] [EQ ¦ NE] <list 1>

<clause 8> ::= <attribute> EQS <list 2>

<clause 9> ::= LIMIT EQ <integer>

<computational operator> ::= COUNT ¦ AVE ¦ SUM ¦ MAX ¦ MIN

<constant> ::= <integer> ¦ <real>

```
<integer> ::= (an integer value)

<list 1> ::= { <constant> "[ , ¦ <space> ]" }

<list 2> ::= { <text> "[ , ¦ <space> ]" }

<numerical expression> ::= { "<parentheses>" [ <attribute> ¦ <constant> ]
                           "<arithmetic operator>" }

<parentheses> ::= { ( ¦ ) }

<relation name> ::= (the name of a valid relation)

<relational operator> ::= EQ ¦ NE ¦ GT ¦ GE ¦ LT ¦ LE

<relational operator 2> ::= EQA ¦ NEA ¦ GTA ¦ GEA ¦ LTA ¦ LEA

<space> ::= (one or more blank spaces)

<text> ::= (a text string)

<where clause> ::= WHERE { [ <clause 1> ¦ <clause 2> ¦ <clause 3> ¦
                   <clause 4> ¦ <clause 5> ¦ <clause 6> ¦ <clause 7> ¦
                   <clause 8> ¦ <clause 9> ] "[ AND ¦ OR ]" }


NOTATION:

< > : term          ::= : is defined as   ( ) : description   ¦  : or

[ ] : choose one    { } : repetition      " " : optional
```