CONSTRAINT ENFORCEMENT IN A STRUCTURAL DESIGN DATABASE

INTRODUCTION

RELATIONAL DATA MODEL

- Model Description - what it is
- Model Applicability - it applies to engineering & has advantages

DATABASE INTEGRITY - db's perform integrity checks using constraints
define integrity, example

DESIGN CONSTRAINTS

- Constraint Representation - information net, sample spec
  (specifications)
- Constraint Types - code, physical, user-defined
  (equations)
- Constraint Usage - checking & assignment defined & exemplified

RELATIONAL MODELING OF CHECKING CONSTRAINTS

- Relational Representation - of constraints in relations
  (relations)
- Implementation - of constraints in RIM
  (computed attributes)
- Advantages - of this model

SUMMARY

ACKNOWLEDGEMENT

REFERENCES

* The items in parentheses follow constraints through four changing representations.
CONSTRRAINT ENFORCEMENT IN A
STRUCTURAL DESIGN DATABASE

INTRODUCTION

RELATIONAL DATA MODEL
   Model Description
   Model Applicability

DATABASE INTEGRITY

DESIGN CONSTRAINTS
   Constraint Representation (specifications)
   Constraint Types (equations)
   Constraint Usage

RELATIONAL MODELING OF CHECKING CONSTRAINTS
   Relational Representation (relations)
   Implementation (computed attributes)
   Advantages

SUMMARY

ACKNOWLEDGEMENT

REFERENCES

* The items in parentheses follow constraints through four changing representations.
CONSTRAINT ENFORCEMENT IN A STRUCTURAL DESIGN DATABASE

By William J. Rasdorf, P.E.\textsuperscript{a} and
Steven J. Fenves, M. ASCE\textsuperscript{b}

ABSTRACT

During the design of a commercial structure large amounts of information pertaining to all aspects of the project 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 access to it, provides one way to generate, represent, manage, and use this information.

This paper uses the relational database model to represent structural design data and constraints. Data integrity is defined and its enforcement through the use of engineering design constraints is described. A model that enables the engineer to incorporate design constraints into a relational database, is presented. The use of constraints in both passive checking and active design modes is explored. An example is presented that demonstrates the concepts discussed using a commercially available DBMS designed for use by the engineering community.

\textsuperscript{a}Assistant Professor of Civil Engineering and Computer Science, North Carolina State University, Box 7908, Raleigh, NC 27695.

\textsuperscript{b}University Professor of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA 15213.
RELATIONAL DATA MODEL

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 called tuples and its columns are called attributes. The domain of an attribute is the set of allowable values the attribute may possess. The attribute values in Figure 1 are drawn from the domain consisting of the set of positive real 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 domains have no order; they may be arbitrarily interchanged without changing the data content and the meaning of the relation. 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 (ATTRIBUTE1name, ATTRIBUTE2name, . . .)
```

with the BEAMS relation of Figure 1 being represented as:

```
BEAMS (Designation, A, d, b_f, t_f, t_w).
```

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

<table>
<thead>
<tr>
<th>Designation</th>
<th>A</th>
<th>d</th>
<th>b_f</th>
<th>t_f</th>
<th>t_w</th>
</tr>
</thead>
<tbody>
<tr>
<td>W12x50</td>
<td>14.70</td>
<td>12.19</td>
<td>8.077</td>
<td>0.641</td>
<td>0.571</td>
</tr>
<tr>
<td>W12x58</td>
<td>17.10</td>
<td>12.19</td>
<td>10.014</td>
<td>0.641</td>
<td>0.359</td>
</tr>
<tr>
<td>W14x26</td>
<td>7.67</td>
<td>13.89</td>
<td>5.025</td>
<td>0.418</td>
<td>0.255</td>
</tr>
<tr>
<td>W14x38</td>
<td>11.20</td>
<td>14.12</td>
<td>6.776</td>
<td>0.519</td>
<td>0.313</td>
</tr>
</tbody>
</table>

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 properties 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. An introduction to relational databases using structural engineering examples was presented in reference [Schaefer82b].
Model Applicability

One of the advantages of the relational database model is its flexibility. It allows data and relationships to be defined in an ad-hoc fashion; new relations can be defined and incorporated into the database as needs dictate. Another advantage of the relational database model is its simplicity. Its use requires knowledge of only one data construct (the relation) and its underlying access mechanisms are hidden from the user. The user needs to be concerned only with the content of individual relations.

Additionally, a relational database provides well-structured centralized storage and control of data, allowing common data access to all users. Multiple data files are integrated into a single database, thus eliminating data redundancy and insuring data consistency. When data values are changed, all users have immediate access to the most recent value. Flexibility, then, is another advantage [Fenves83].

In view of the conceptual advantages of the relational model and the vast amount of research and development for making it general and efficient, it is likely that in the future it will become the predominant data model. RIN, [Erickson81], BCS83] in particular, will see significant use because it combines the versatility and flexibility of the relational model with engineering oriented extensions, which make it ideal for managing structural engineering data.

DATABASE INTEGRITY

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, DBMS systems have been incorporating into their structure specifications for a set of integrity constraints and mechanisms for performing this task automatically [Scheeer82a].

Integrity of a database refers to the maintenance of functional relationships between data items [Date81]. Integrity deals with dependencies among data items, where dependent values are derived on the basis of the constraints among 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 on its value.

Constraints are relationships among data item values. Constraints are used to guarantee the integrity of the database when operations are performed on it. Integrity constraints are therefore concerned with the meaning of data. They represent relationships between the data that are not directly embodied in the database [Fenves82].
The constraint
\[ S = 2 \times I / d, \]
introduced in the next section, can be used to illustrate the significance of integrity. If the values of \( S \), \( I \), and \( d \) are all stored, their integrity must be maintained with regard to the constraint; i.e., if the value of any one of these data items is changed, the value of one of the others must also be changed. Otherwise, the integrity of the relationship and the validity of the database will be violated. The constraint is thus used in a passive checking mode. Alternatively, it is possible to store any two of the three data item values, calculating the third as needed. In this manner the constraint is used in an active assignment mode and integrity is automatically maintained. A method is presented below whereby integrity constraints are defined, entered into a database, and properly invoked.

DESIGN CONSTRAINTS

The process of structural design can be viewed as:

- the definition of the topology and geometry of a structure and the selection of its structural system type; and
- the assignment of values to data items specifying the physical and geometrical makeup of elements.

Definition and assignment are not arbitrary operations; both are governed by applicable constraints. Integrity in a structural design database is enforced by constraints.

Constraint Representation

Constraint relationships between data items comprise a dependency network, a directed graph obtained by assigning a node to each datum and assigning a directed branch from each node to its dependents. The dependents of a node are those data items that are a function of the node and its ingredients are those data items that are needed to evaluate it [Fenves77]. Each node in the network represents a single datum, which may have a quantitative, qualitative, or boolean value. The lowest level nodes represent input data and the higher level nodes represent derived data. Any dependency network can be conceptually broken into multiple subnetworks, each representing a lower level constraint.

The following fictional simplified specification for the design of steel beams will be used as an example:
"Each beam of the structure shall be proportioned so that its flexural, material, and clearance requirements are satisfied (beamOK).

The **flexural requirement** (flexOK) is satisfied if

\[ M_u \leq M_c \]

where \( M_u \) = applied moment (required moment), and \( M_c \) = moment capacity (design moment).

The applied moment, \( M_u \), is computed by analysis as a function of the applied loads.

The moment capacity, \( M_c \), is given as

\[ M_c = \Phi_i \times M_n \]

where \( \Phi_i = 0.66 \), and \( M_n \) = nominal moment capacity.

The nominal moment, \( M_n \), is given as

\[ M_n = S \times F_y \]

where \( F_y \) = specified yield strength, and \( S \) = elastic section modulus.

The **material requirement** (materialOK) is satisfied if

\[ F_y \leq 50 \text{ ksi} \]

where \( F_y \) = yield strength.

The **section requirement** (sectionOK) is satisfied if

\[ S = \frac{2 \times I}{d} \]

where \( I \) = beam moment of inertia, and \( d \) = depth of a beam.

The **clearance requirement** (clearanceOK) is satisfied if

\[ d \leq d_c \]

where \( d \) = depth of beam, and \( d_c \) = available clearance."

Figure 2 (a) shows the constraints arising from the sample specification and Figure 2 (b) shows the dependency network representing them. The network can be divided into multiple subnetworks, each of which represents one of the constraints. The portion of the dependency network including \( S, I, \) and \( d \), for example, constitutes a subnetwork representing constraint (4). The top four nodes comprise a subnetwork representing constraint (1).

**Constraint Types**

Generalizing from the above example, constraints may be subdivided into the following different categories: code, standard, and specification constraints; physical constraints; and user-defined constraints.

Design codes, standards, and specifications provide one class of constraints on data item values; they are constraints intended to assure the functionality and usability of a building. They represent a set of practices and requirements that govern the design and behavior of the components of a building [Fenves77]. Item (2) in Figure 2 (a) is such a constraint.
(1) BeamOK := (FlexOK AND MaterialOK AND SectionOK AND ClearanceOK)

(2) FlexOK := (M_u <= Phi * M_n)

(3) M_n := S * F_y

(4) S := 2 * I / d

(5) ClearanceOK := d <= d_c

(6) SectionOK := (S = 2 * I / d)

(7) MaterialOK := F_y <= 50

(a) Constraints

(b) Dependency Network

Figure 2: Dependency Network Representation of Constraints

Physical constraints define permissible physical situations within a structure. Item (5) in Figure 2 (a) illustrates a constraint of this type. Similarly, property constraints define permissible physical properties of structures components. Items (4) and (6) illustrate constraints of this type.
A fourth class of design constraints, referred to as user-defined constraints, are constraints which arise from each user's personal design style. They are not predefined and may be issued at any point during the design process. User-defined constraints are incorporated into the database at the time of their definition. The sample specification includes an example of a user-defined constraint (item (7) above) on the yield stress of components limiting it to a maximum value of 50 ksi.

Other constraints are also considered in design although they may not always be explicitly stated. Examples of these are equilibrium and compatibility. It is the function of application programs, rather than the database to enforce functional integrity with respect to such constraints. For example, a structural analysis program outputs a set of member forces and joint displacements consistent with input topology, geometry, loads, and member properties in accordance with structural laws of equilibrium and compatibility. Related to these constraints are analysis constraints such as item (3) in Figure 2(a).

**Constraint Use**

Constraints restrict the domain of a data item, i.e., they restrict its allowable values. As mentioned above, constraints can be used for two distinct purposes: checking and assignment [Fenves82, 83].

Checking constraints are passive constraints that use existing data item values to determine whether a constraint is satisfied i.e., whether the prescribed functional relationship between the constrained data items exists. A checking constraint results in the assignment of a boolean value (true meaning satisfied or false meaning violated) representing the value of the constraint. Item (2) of Figure 2 (a) is an example of a checking constraint for satisfying flexural strength requirements of a beam. Its corresponding subnetwork consists of the data items FlexOK, M_u, Phi, and M_n.

Assignment constraints 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 is automatically satisfied. Items (3) and (4) of Figure 1 (a) illustrate assignment constraints: each assigns a numerical value to a data item, M_b and S, respectively. Holtz [Holtz82] has investigated the conversion of passive constraints to active assignment procedures that give bounds on the values of unknown data items as a function of their ingredients.
To more clearly exemplify the characteristics of checking and assignment constraints consider again item (4) in Figure 2 (a), \( S := 2 \times I / d \). If the values of all three of \( S \), \( I \), and \( d \) are stored, then their integrity must be maintained in accordance with item (6), SectionOK := \( S = 2 \times I / d \). If the value of any one of these data items is changed, SectionOK evaluates to false (violated). In a case such as this, the constraint is used in a passive checking mode. On the other hand, it is possible to recalculate the third data item any time the other two change such that the constraint is satisfied. In this case, the constraint is used in an active assignment mode and integrity is automatically maintained.

RELATIONAL MODELING OF CHECKING CONSTRAINTS

Because an engineering constraint network is generally large and complex, an entire network cannot yet be represented in a database. Currently only small portions of the network have been represented in databases; namely its subnetworks.

Reference [Fenves83, Rasdorf82] describes a relational data structuring scheme for representing design constraints that eliminates problems of redundancy, update anomalies and other irregularities caused by dependencies among subnetwork data items. The new model does so by introducing into relations new attributes that record the status of all constraints defined on the relation. The integrity of data item dependencies is thus always monitored. At the same time, the model retains immediate local access to all of a relation's attribute values.

As an example consider the relation

\[
\text{PROPORTIONS (ID, S, I, d)}
\]

containing the data items of item (4) of Figure 2. The introduction into the relation of the new attribute SectionOK

\[
\text{PROPORTIONS2(ID, S, I, d, SectionOK)}
\]

allows one to monitor whether or not the constraint is satisfied. SectionOK is a boolean attribute whose value is determined by the constraint. The constraint itself is stored in the database as a computed attribute. Whenever one of \( S \), \( I \), or \( d \) is changed, or when a new tuple is added to the relation, the constraint is automatically invoked and its value is computed and stored in the tuple. Such a model provides a way to directly associate engineering constraint checking with data in a relational database. The model can also be generalized to handle multiple constraints as shown below. In addition, it can handle constraints among a multiple number of relations as long as there is a corresponding distinct tuple in each relation. The model does not currently handle hierarchical and inheritance relationships among relations.
Relational Representation

In the relational model a means of constraining data item values stored in single attributes already exists [RasdorfF83]. Requiring the yield stress, $F_y$, to be less than 50 ksi is an example of such a single attribute constraint. For constraints of a broader nature, i.e., for subnetworks, integrity can only be enforced by removing constrained data items from large relations and separately maintaining them in protected relations that can be modified only by authorized database users. As an example consider the following candidate relations for representing the information network of Figure 2 (b):

1. BEAMS (\textit{beamID}, designation, BeamOK);
2. STRENGTH (\textit{beamID}, $M_y$, $M_n$, Phi, FlexOK);
3. PROPERTIES (\textit{beamID}, $F_y$, ..., MaterialOK);
4. PROPORTIONS (\textit{beamID}, $S$, $I$, $d$, SectionOK); and
5. PHYSICAL (\textit{beamID}, $d_o$, ClearanceOK).

These relations group the information normally used in analysis and design in a "natural" way. The BEAMS relations contains the status indicator of item (1) of Figure 2 (a) and the beam designation. The STRENGTH relation contains the values of the ingredient data items of item (2) of Figure 2 (a) as well as the status indicator of that constraint. The PROPERTIES relation contains values for additional properties that might be associated with the beam including $F_y$, an ingredient datum of items (3) and (7) of Figure 2 (a). The PROPORTIONS and PHYSICAL relations contain the values of the ingredient data items and the boolean status indicators necessary for evaluating items (5) and (6), respectively, of Figure 2 (a). These five relations, along with the indicated constraints, provide one of the possible data structures capable of representing the information network of Figure 2 (b).

The structure of the relations presented above permits an additional distinction to be made among them. Clearly, the constraints whose status is stored in relations 2, 3, and 4 are directly enforceable using the data items available in those relations. These are referred to as single relation constraints. The constraints whose statuses are stored in relations 1 and 5 differ in that some of their ingredient data items are obtained from other relations. These are referred to as multiple relations constraints. This distinction is particularly important because multiple relation constraints are not currently supported in commercially available DBMSs. Hence, to enforce the multiple relation constraints shown here requires that the relations containing the ingredient data items be JOINed together into a single relation.
The structure of the relations introduced above is such that they can automatically satisfy checking constraints thereby enforcing the integrity of the relationships between their attributes. A function is added to the database that checks the constraint for each tuple and returns a value for the status attribute indicating whether or not the defined relationship between the constrained attributes is satisfied. The status is then recorded in the new status attribute added for that purpose.

Implementation

A commercially available database management system, Boeing Computer Service's Relational Information Manager (BCS RIM), was used to implement the concepts presented herein [BCS83]. This section uses BCS RIM to present an example of: defining the attributes and a relation of a schema; loading data; defining rules and computed attributes; and checking the operation of the DBMS. To fully demonstrate all of these operations the PROPORTIONS and PHYSICAL relations introduced above were JOINed and two additional attributes, 'desig' for designation and 'differ' for difference, were introduced. The new relation:

\[
\text{PROPORTIONS(beamID, desig, S, I, d, differ, sectOK, dc, clearOK)}
\]

contains abbreviated names which correspond to those in the original relations.

The text which follows takes the form of an on-line terminal session, interspersed with narration explaining the sequence of operations. The BCS RIM DBMS operates interactively. The BCS RIM commands are given in upper case and are set apart from the text.

**ATTRIBUTES**

- beamID TEXT 8 UNITS="CHARS" "beam identification"
- desig TEXT 8 UNITS="CHARS" "beam designation"
- S REAL 1 UNITS="INCH**3" "elastic section modulus"
- I REAL 1 UNITS="INCH**4" "moment of inertia"
- d REAL 1 UNITS="INCHES" "depth of beam"
- dc REAL 1 UNITS="INCHES" "available clearance"
- differ:"ABS(S-2*I/d)" REAL 1 "computed S error"
- sectOK:"IFGT(differ,0.1,0.1)" INTEGER 1 "section status attribute"
- clearOK:"IFGT(dc,d,0.1)" INTEGER 1 "clearance status attribute"

**RELATIONS**

- PROPORTIONS WITH beamID, desig, S, I, d, differ, sectOK, dc, clearOK

**RULES**

- sectOK IN PROPORTIONS EQ 1
- clearOK IN PROPORTIONS EQ 1

END
The ATTRIBUTES command is used to define all the attributes that are used in the entire database. The definition is composed of six parts: the attribute name, type, length, key, units, and definition. Computed attributes are specified by the form:

\[ \text{attname} = \text{expression} \]

The expression can consist of functions; sectOK, for example, is a computed function which will have an integer value of 0 if 'differ' is greater than 0.1 and an integer value of 1 otherwise. Because of potential errors due to rounding of values sectOK is determined to be true if the value of differ is within the specified tolerance (0.1). Here the 0 should be interpreted as false for a non-complying constraint, and the 1 as true for a complying constraint. Ideally, one would use boolean values for the constraint status attributes, but BCS RIM does not support a boolean data type.

The RELATIONS command is used to specify the attributes in each individual relation. The designer may specify them in any order he chooses. The RULE command is used to specify the constraints imposed on the attributes of a relation. In the example shown the rules mandate that both sectOK and clearOK must be true for values stored in this relation. Any attempt to store data that violates either of these constraints for a beam will be rejected.

Initial data items are manually loaded into the database by the designer using the following BCS RIM commands:

```
NOCHECK
LOAD PROPORTS
B1,W10X112,716,11.36,126,12
B2,W10X100,623,11.10,112,12
B3,W10X88,98.5,934,10.84,10
END
```

The NOCHECK command indicates that the designer has suspended rule checking. This command acts as a switch permitting the designer to turn off checking, i.e., to deactivate the rules thereby allowing any data to be entered. The LOAD command indicates that data for the specified relation will be entered. The data items are then entered in the order designated by the RELATIONS definition, with computed attributes being omitted.

The structure of the new relation and the data that has been entered into it can now be checked. The LISTREL command prints the schema (structure) of any specified relation, and the SELECT ALL (the tuples) command prints the attribute values for each tuple that currently exists in the relation.
LISTREL PROPORTS

RELATION : PROPORTS
LAST MOD : 85/12/30
SCHEMA : BUILD
READ PASSWORD : NONE
MODIFY PASSWORD : NONE

NAME TYPE LENGTH KEY DEFINITION UNITS
----- ----- -------- --- ------------------------ --------
beamID TEXT 8 CHARs YES beam identification CHAR
desig TEXT 8 CHARs    beam type CHAR
S  REAL 1    elastic section modulus INCH**3
I  REAL 1    moment of inertia INCH**4
d  REAL 1    depth of beam INCH
dc REAL 1    available clearance INCH
differ REAL COMPUTED computed S error INCH**3
sectOK REAL COMPUTED section status attribute -0-
clearOK REAL COMPUTED clearance status attribute -0-

CURRENT NUMBER OF ROWS = 3

SELECT ALL FROM PROPORTS
beamID desig S I d dc differ sectOK clearOK
----- ----- --- --- --- --- ------- ------ ------
B1  W10X112 126  716. 11.36 12. .056339  1  1
B2  W10X100 112  623. 11.1 12. .252252  0
B3  W10X88  98.5 534. 10.84 1.00E+1 .023985  1  0

The values of sectOK and clearOK were computed by the computed attributes when the data was loaded. They show that data not currently satisfying the constraints has been entered into the database. Beam B1 satisfies both constraints (both attributes have a value of 1); Beam B2 does not satisfy the section constraint, while beam B3 does not satisfy the clearance constraint. The non-conforming data was accepted by the database because NOCHECK was in effect. If CHECK is now activated, any additional data that is loaded must satisfy the defined constraint rules. Issuing the commands:

CHECK
LOAD PROPORTS
B7,W10X49,54.6,272,9.98,10

causes BCS RIM to respond with:

-ERROR- THE DATA FAILS TO SATISFY THE FOLLOWING RULE
RULE NUMBER 1
sectOK IN PROPORTS EQ 1

An additional use of the computed attribute capability is the assignment of attribute values to unknown data items in such a way that the governing constraint(s) is automatically satisfied. As an example, S could be converted as follows from a data item which must be entered manually to one which is automatically computed:

ATTRIBUTES
S="2*I/d" REAL 1 UNITS="INCH**3" "computed elastic section modulus"
In this case, the LOAD PROPORTS command will require one less data item, S, to be entered for each tuple. The new computed attribute returns a value for S that satisfies the defined relationship and at the same time causes the sectOK rule to be satisfied automatically. In this manner the designer gains a new tool with which he can assign results known to be consistent with other attribute values in the database. To illustrate, the above example is used again.

NOCHECK
LOAD PROPORTS
B1, W10X112, 716, 11.36, 12
B2, W10X100, 623, 11.10, 12
B3, W10X88, 534, 10.84, 10

The result of these operations is that 'differ' will always be computed to be 0.0 and sectOK will always be 1. The constraint will necessarily and automatically be satisfied. This can be seen by again SELECTing the resulting tuples:

```
SELECT ALL FROM PROPORTS
beamID  design S   I    d    dc  differ sectOK  clearOK
------  ----- ---- --- ------- ------- ---- ------- -------
B1      W10X112 126.056 716. 11.36 12.  .0  1       1
B2      W10X100 112.252 623. 11.1 12.  .0  1       1
B3      W10X88  98.524  534. 10.84 1.00E+1 .0  1       0
```

It can be seen that the sectOK constraint has been satisfied for B3 but that the clearOK constraint has been violated. The reason this violation occurred is that NOCHECK was in effect. Had it been in effect, this tuple would not have been permitted to be entered into the database. Used in the way shown here the designer has a mechanism for entering his data even if it is not all correct at the time he enters it. The advantage of the status attribute is that the designer is aware that there is a problem and corrective action can be taken to remedy it. As a result, the designer can proceed to do operations that are for the most part correct and can handle deviations at a later time.

If the designer wishes to locate those tuples that violate a constraint, he may do so by performing a SELECT on the relation where his selection criterion is a value of 0 for that constraint's status. For the example above, to determine which beams do not comply with the clearance constraint, the following command would be issued:

```
SELECT beamID, d, dc, clearOK FROM PROPORTS WHERE clearOK EQ 0
beamID  d    clear  clearOK
------  ---- ------- -------
B3      10.84 10     0       
```

Subsequently this tuple can be modified in any way the designer desires in order to bring it into compliance and to satisfy the constraint.
Limitations of the Implementation

The previous example illustrated a progression whereby constraints were extracted from a design specification and transformed to fit within the framework of a relational DBMS. Two mechanisms for the use of specifications in a DBMS context were described: checking and assignment.

Checking and assignment capabilities are implemented in the BCS RIM DBMS in the form of rules and computed attributes, respectively. These BCS RIM features 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 type is needed. In particular, constraints whose ingredient and dependent data items reside in multiple relations must be handled. Neither computed attributes nor rules extend beyond the boundaries of a single relation. As a result, the database designer has two alternatives, both of which have significant disadvantages. The first is to duplicate attributes in different relations so that there will be at least one relation containing all of the attributes relevant to a particular constraint. The disadvantage of this solution, which was what was done to the PROPORTIONS relation above, is that data is redundantly stored and consistency among that data is difficult to maintain. The net result is that to enforce integrity the mechanism itself introduces potentially more dangerous integrity problems.

The second alternative is to JOIN all relations containing constrained attributes into a single relation and replace the previous relations with the new one. All constraints would then be applied to the new relation. The disadvantage of this approach is that any logical organizational structure of the database disappears and the database becomes a small collection of very large files. Logical groupings of data and logical organizational divisions are thus lost.

The model presented in this paper is a component of a generalized mechanism (further described in reference [Fenves83]) for handling checking and assignment operations in a relational DBMS environment. The example implementation showed how the model could be implemented using a commercially available DBMS. The limitations of the implementation make it clear that additional developments are needed in this area because no currently available DBMSs provides the much needed capabilities presented here.
SUMMARY

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 or determine data item values. In short, relational databases can be used to directly contribute to the design and data representation processes.

The constraint handling system proposed in this paper is based on the concept of object-oriented programming [Johnson81]. The system is user-transparent in that it handles all constraint processing automatically. In a checking mode all constraint statuses are recorded. In an assignment mode constraint statuses are recorded and a new value for the constrained data item is assigned. User intervention is required only when ingredient data item values are missing from a constrained relation and default values have not been defined. What is new in this DBMS model is that a constraint subnetwork is associated directly with the stored data that it constrains. The conceptual result is a new abstraction, consisting of a relation and computations, that has its own characteristics. The database user need only initially define the rules and computed attributes. These are then used by the relation to perform the appropriate checks and assignments. The database user is, to a significant degree, free of constraint checking concerns because the system itself knows what to do.

ACKNOWLEDGEMENTS

The BCS RIM relational database management system was used to implement the database presented herein [BCS83]. The authors express their gratitude to Boeing Computer Services for their contribution to this study. The BCS RIM DBMS is compatible with the R:BASE Series 4000 microcomputer DBMS of MicroRIM, Inc. [MicroRIM84] allowing data to be readily transferred between micro and mainframe computers. Both DBMSs support an interface to FORTRAN application programs.

Portions of this work were sponsored by the National Science Foundation under Grant MCS7622328 entitled "Data Base Methods for Design" and by the National Aeronautics and Space Administration under grant NAG-1-503 entitled "Data Management Technology - Functional Requirements for a Generative Database."
REFERENCES


[Date81] Date, C. J., An Introduction to Database Systems, Addison Wesley, Reading, Massachusetts, 1981.


