Role of database management systems in structural engineering

Autor(en): **Fenves, Steven J. / Rasdorf, William J.**

Objekttyp: **Article**

Zeitschrift: **IABSE reports = Rapports AIPC = IVBH Berichte**

Band (Jahr): **40 (1982)**

PDF erstellt am: **14.08.2024**

Persistenter Link: https://doi.org/10.5169/seals-30900

Nutzungsbedingungen
Die ETH-Bibliothek ist Anbieterin der digitalisierten Zeitschriften. Sie besitzt keine Urheberrechte an den Inhalten der Zeitschriften. Die Rechte liegen in der Regel bei den Herausgebern.
Die auf der Plattform e-periodica veröffentlichten Dokumente stehen für nicht-kommerzielle Zwecke in Lehre und Forschung sowie für die private Nutzung frei zur Verfügung. Einzelne Dateien oder Ausdrucke aus diesem Angebot können zusammen mit diesen Nutzungsbedingungen und den korrekten Herkunftsbezeichnungen weitergegeben werden.
Das Veröffentlichen von Bildern in Print- und Online-Publikationen ist nur mit vorheriger Genehmigung der Rechteinhaber erlaubt. Die systematische Speicherung von Teilen des elektronischen Angebots auf anderen Servern bedarf ebenfalls des schriftlichen Einverständnisses der Rechteinhaber.

Haftungsausschluss
Alle Angaben erfolgen ohne Gewähr für Vollständigkeit oder Richtigkeit. Es wird keine Haftung übernommen für Schäden durch die Verwendung von Informationen aus diesem Online-Angebot oder durch das Fehlen von Informationen. Dies gilt auch für Inhalte Dritter, die über dieses Angebot zugänglich sind.

Ein Dienst der *ETH-Bibliothek*
ETH Zürich, Rämistrasse 101, 8092 Zürich, Schweiz, www.library.ethz.ch

http://www.e-periodica.ch

Role of Database Management Systems in Structural Engineering

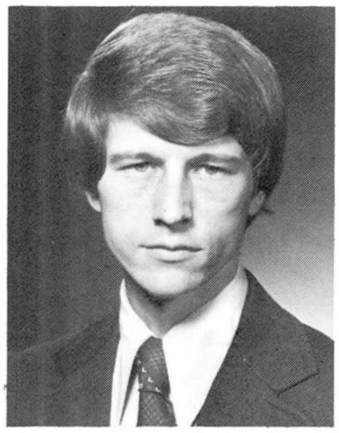
Systèmes de gestion de base de données dans le domaine des structures

Datenbank-Management-Systeme im Bauingenieurwesen

Steven J. FENVES
Professor
Carnegie-Mellon University
Pittsburgh, PA, USA

Steven J. Fenves, born in 1931, received his degrees in civil engineering from the University of Illinois, where he taught until 1972. His teaching and research activities deal with computer-aided engineering, with emphasis on representation of standards, databases and expert systems.

William J. RASDORF
Assistant Professor
North Carolina State Univ.
Raleigh, NC, USA



William J. Rasdorf, born in 1951, received Bachelor's and Master's degrees in Architectural Engineering from the Pennsylvania State University. He worked as an Architectural Engineer for 3 years. He received his Ph.D. from Carnegie-Mellon University where he became interested in computer representation of engineering design data.

SUMMARY
The future integration of structural engineering application programs will depend critically on integrated databases which provide access to information in essentially arbitrary sequences, and which automatically perform a large portion of integrity checking on the data. One source of such design databases are the database management systems (DBMS) evolving from management applications. The paper surveys such systems and presents some extensions needed.

RESUME
L'intégration future des programmes d'application dans le domaine des structures dépend de façon critique de la mise en place de bases de données intégrées. Celles-ci doivent permettre l'accès aux informations selon des séquences essentiellement arbitraires et assurer automatiquement une large part des contrôles d'intégrité des données. Les systèmes de gestion de bases de données (SGBD) issus des applications de gestion constituent un point de référence. L'article examine ces systèmes et propose certaines extensions nécessaires à leurs applications dans le domaine technique.

ZUSAMMENFASSUNG
Die zukünftige Vereinheitlichung von strukturellen Applikationsprogrammen wird kritisch von integrierten Datenbanken abhängen, die Zugang zu Information in wichtigen, willkürlichen Abläufen verschafft und die automatisch eine breite Integritätsüberprüfung ausführen. Eine Quelle solcher Entwurfsdatenbanken sind die Datenbank-Management-Systeme, die von Management-Anwendungen stammen. Dieser Bericht betrachtet solche Systeme und behandelt einige erforderliche Zusätze.

1. INTRODUCTION

The decade of the 80's will see a major trend towards the integration of stand-alone structural engineering application programs into comprehensive design systems. The bunifying element of such systems will be a central data base, containing up-to-date information about the evolving design. The organization and operation of such a database has to be drastically different from the file or secondary storage management systems now in common use. Integrated databases must allow users and application programs to access information in essentially arbitrary sequences without regard to the internal organization of data. Furthermore, such databases must automatically perform a large portion of the consistency and integrity checking on the data, both within the structural design process itself and in integrating structural data with other disciplines participating in the design.

Database management systems (DBMS's) incorporating physical and logical data independence and consistency enforcement have evolved for management applications, and many DBMS's are commercially available. A major challenge is to evaluate such systems for their applicability to structural design and to implement necessary additions or extensions.

The paper is organized into three parts. First, the the present use of data are critically evaluated. Second, the relevant concepts of DBMS's and the major existing database models are briefly introduced, with primary emphasis on the relational model. Third, a class of necessary extensions to the relational model is presented, dealing with the representation and processing of constraints that arise in structural design. Illustrations are drawn from a prototype structural design database system recently developed.

2. DATABASE IMPLICATIONS OF STRUCTURAL ENGINEERING COMPUTER USE

2.1 Trends in Computer Use

Computer usage in structural engineering design has had an explosive growth in the past 15 years. Around 1970, the only common structural engineering applications were analysis (usually restricted to linear elastic models), and detailing of certain repetitive structural components. Except for some simple file structures for storing input data and results or for providing restart capabilities on long runs, essentially all data had a lifespan restricted to the duration of a single computer run.

Since the early 1970's, computer usage has grown in four distinct directions, all of which have a major impact on the lifespan of the data and the manner in which they are used:

- Increase in depth. Computer programs are now commonly used at all stages of design of a structure (conceptual, preliminary and detailed); and for fabrication and construction scheduling and control. It has therefore become important to "capture data at the source," so that the output of one stage can serve directly as input to the next one.

- Increase in breadth. The range of programs used at any one stage has grown similarly. For example, it is now common practice to perform a linear, static analysis for one set of loads, and a dynamic and/or nonlinear analysis for another set of environmental conditions. While some large analysis packages support all these analysis options, it is not uncommon that different programs must be used for the different analyses. It is therefore necessary to have the common input data available in a *program-independent* fashion.

- Increase in integration. As the other design disciplines (e.g., architecture, mechanical and electrical engineering) have increased their computer usage, there has been an increasing need for two-way interchange of information between the structural engineer and the other disciplines; it becomes less and less desirable to re-enter the shared data manually or even to re-format them with ad-hoc programs.

- Increased need for flexibility. The combination of the three factors described produces a fourth one, namely, the need to accommodate much more flexible design sequences than was the practice in the past. Thus, programs must accommodate a much larger range of "inputs," in terms of the decisions previously made.

2.2 Present Usage of Data

The present methods of data storage and access in structural engineering fall short of the needs. These methods generally belong to one of the following three categories:

- Temporary files. Most large structural analysis programs store and access information in temporary files (usually unformatted or binary) for segmentation purposes, backup/restart, or postprocessing).

- Explicit interface programs. When the output of one program serves as input to another, interface programs are written for performing the necessary conversions and reformatting.

- Text files. Many organizations save both input and output data in alphanumeric formatted text files. Input files can be edited for changes and resubmitted for reanalysis, and output files can be scanned for relevant information retrieved.

The first two methods exhibit strong physical and logical data dependence, i.e., the content and organization of the data is totally dependent on the needs of the programs using them. The last method, by contrast, exhibits *data ignorance*; the file management system only knows the name of the file, and knows nothing about its contents.

These are a number of notable exceptions to the limited range data storage and access methods discussed above. Only a few of these can be reviewed here.

- Both GENESYS [9] and ICES [18] contain a common system for data storage and communication among subsystems. ICES contains a subsystem, TABLE, specifically for storage and access of data. POLO [10] goes a step further and contains a database management system, FILES [11].

- A number of application-independent centralized databases have been developed. In systems such as IPAD [3, 15] or COMRADE [1, 17], all data are stored in a single common pool and are accessible to all users.

- General finite element pre- and post-processors such as UNISTRUC [22] and FASTDRAW [13, 14] represent the model in a "neutral file"; when model generation and manipulation is completed, a "source file" for a specified analysis program is generated. Similarly, results from any of the supported analysis program can be reformatted into a "neutral file" and post-processed.

- GLIDE (Graphical Language for Interactive Design in Engineering) [5, 6] is a prototype system using a flexible database supporting high level data abstractions and geometric modelling capabilities.

A recent survey of database applications in engineering practice [2] shows that 53 out of 153 firms surveyed use some form of database applications, but that the majority of the use is in accounting, personnel records and project management. Only 23 firms use databases for analysis or design.

2.3 Shortcomings of Present Usage

The present methods of data storage and access have three major shortcomings. First, these methods are not suited for supporting flexible design sequences, where the programs may interact in a variety of sequences, where multiple iterations have to be performed, and where multiple alternative designs may have to be generated and results compared. Such flexibility cannot be provided if the internal representation (content and organization) of the data is integrally dependent on the specific program(s) using the data.

Second, these methods provide no general mechanisms for either querying or updating the database. All updates or queries must be specifically programmed, using the specific internal representation of the data.

Third, and possibly most importantly, there are no general ways of insuring or monitoring the integrity and consistency of the stored data. One cannot impose constraints such as "beam depth $\leq$ clearance allowed" unless one knows exactly where and in what format "beam depth" and "clearance allowed" have been stored.

3. DATABASE MANAGEMENT SYSTEMS

3.1 The System

In administrative data processing applications there has been a major trend toward the use of DBMS systems. DBMS's are designed to store data in a manner independent from programs, to allow programs to access and retrieve data, and to provide a means of inserting, deleting, and modifying data. A DBMS can be represented as a series of layers:

- Core. The actual data as it is stored on a physical device.
- Interface. The layer immediately surrounding the central data core is a collection of software that enables the core to be used. It is the communications link between the stored data and users; it acts as a transfer mechanism that converts the "raw" data from its stored physical form to a logically structured format.
- Users and applications. The outermost layer consists of database users and application programs accessing and performing operations on the data.

Each level of the DBMS contains a different *view* or description of the data. A view of the core shows the physical layout of the data as it resides on a storage device. This is the view of systems designers and programmers concerned with system performance, data indexing, and data location [4]. The organization of the data at this level is referred to as the physical structure of the data.

Software associated with the second layer of the DBMS converts the physical structure of the core data to a logical structure or *schema*. The schema is an overall *representation* of the data describing it in logical rather than physical terms. The overall schema is further converted by the software associated with the outermost layer into smaller views referred to as *subschemas*, portions of the overall logical data representation used by application programs. The schema provides an overall view of the logical representation of all of the data, while a subschema provides a limited view structured for a particular application.

One of the key objectives of a DBMS is to allow different applications to use the same data in a program-independent fashion. To do so requires a data structure where all data is pooled together at the physical level and selected views of the data are appropriately available.

The DBMS must also provide the capability for the database to grow and change as needs dictate. Over the life of the database it must be possible to dynamically change its logical structure as new types of data are added and new programs developed.

Flexibility in a database is possible only if *physical* and *logical data independence* is maintained between the layers of the DBMS. The physical data structures of the core must be entirely separate from the data structures of the logical outer layers perceived by users and programs. The connection between the layers is maintained by the DBMS software. Changes to either the data or the programs can be made without requiring changes to the other. Only the layer of DBMS software that interfaces with the change needs to be modified. In this manner the database need not be restructured when new programs are written and existing programs need not be rewritten when changes are made to the data structure.

There are three major capabilities provided by a DBMS:

- Data definition. Data definition defines the schema and builds the *framework* into which attribute values are placed. Data definition is performed by the database administrator using a data definition language (DDL).

- Data modification. Data modification includes insertion, modification, and deletion of data values and is performed by the database user using a data manipulation language (DML).

- Data retrieval. Data retrieval consists of obtaining desired information from the database and includes the ability to search, manipulate, and query without the necessity of writing application programs. Data retrieval is performed by the user using a *query language*.

The DBMS communicates with application programs written in a standard programming language through a *host language interface*. The interface consists of a set of call statements to DML procedures that initiate the desired operations on the database.

3.2 Common Data Models

A *data model* defines the overall logical structure of a database [12]. It provides the structural framework into which the data are placed. Three database models have come into common usage: the hierarchical, network, and relational models.

Hierarchical Data Model. The hierarchical model is *tree structured*. It is composed of *nodes* connected together by *links* as shown in Figure 1. The nodes may be grouped into horizontal layers called *levels*. As Figure 1 shows, a hierarchy is a *multilevel* data model. The tree structure of the hierarchical model implies that each node may be linked to more than one node below but to only one node above itself [12].

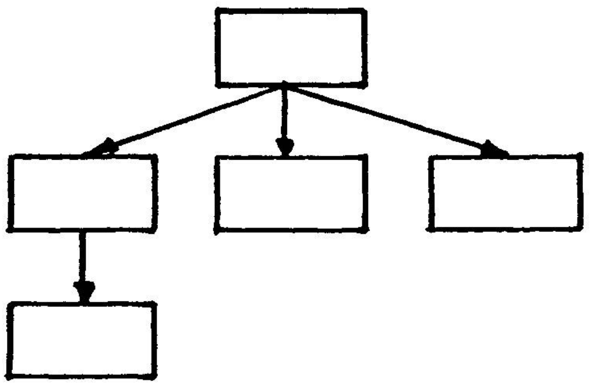


Figure 1: Hierarchical Data Structure

A node represents a *type* of entity about which information is stored. An entity may be an object such as a bolt, a column, or an entire frame. Each entity has certain descriptive information associated with it. This information determines the entity *type* and is referred to as the *attributes* of the entity. Links represent relationships between the entity types. The direction of the link indicates a relationship of one to many from the tail of the arrow to its head.

Network Data Model. A *network* is a directed graph. The network model is a multilevel data model in which each node may be linked to more than one other node in *both* upward and downward directions [4]. This is the distinguishing difference between the hierarchical and network models; it allows relationships to be established horizontally within levels between different entity types as well as vertically between levels.

Figure 2 illustrates an example of a network data structure. It has basically the same structure as the hierarchy of Figure 1 with the addition of multiple network links. It can readily be seen that the hierarchical model is a special case of the network model.

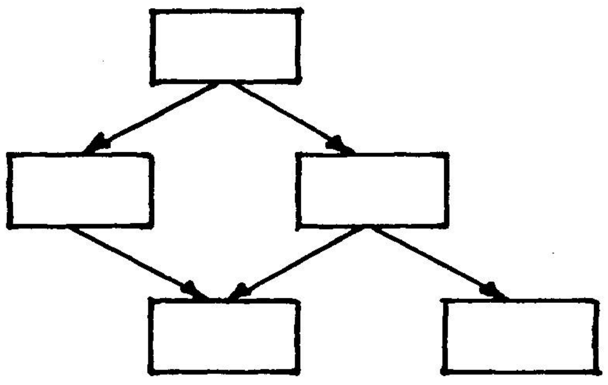


Figure 2: Network Data Structure

Relational Data Model. A *relational model* is a *single level* model consisting of a collection of *relations* represented in two-dimensional tabular form [19]. Associated with the relations is a set of operators that allow for the insertion, deletion, modification, and retrieval of data. A figure for the relational model similar to Figures 1 and 2 would simply contain a collection of nodes without any links between them. There are no predefined hierarchies or networks in the relational model. Links needed between nodes are automatically created by the relational DBMS upon demand and an access path is established to any node.

Figure 3 illustrates the structure of a relation. The rows of a relation called *tuples* and its columns are called *attributes*. All attribute values are drawn from the same *domain* i.e., they are of the same data type. Each tuple represents an entity and contains a value for each attribute. All tuples are distinct; duplicates are not permitted [16]. Tuples and domains have no order; they may be arbitrarily interchanged without changing the data content and 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 3 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.

BEAMS

```
 ------------------------------------------------------------
 | Designation |    A    |   d    |   bf    |   tf    |   tw    |<---ATTRIBUTES
 |=============|=========|========|=========|=========|=========|
 |    W12x50   |  14.70  | 12.19  |  8.077  |  0.641  |  0.371  |
 |-------------|---------|--------|---------|---------|---------|
 |    W12x58   |  17.10  | 12.19  | 10.014  |  0.641  |  0.359  |<---TUPLE
 |-------------|---------|--------|---------|---------|---------|
 |    W14x26   |   7.67  | 13.89  |  5.025  |  0.418  |  0.255  |
 |-------------|---------|--------|---------|---------|---------|<-----
 |    W14x38   |  11.20  | 14.12  |  6.776  |  0.513  |  0.313  |       |
 ------------------------------------------------------------          |
                                   ^                                   |
                                   |                                   |
                                   --DOMAIN          RELATION----
```

Figure 3: The Structure of Relations

3.3 Model Comparison

The similarities between the multilevel hierarchical and network models are evident. The network model is more flexible, allowing non-hierarchical or, multi-hierarchical relations to be defined. This added flexibility results in greater representational power, although it still does not afford the representational capabilities of the relational model.

Relationships among entities and constraints among attributes impose critical requirements on a database. All user queries and updates generally cannot be anticipated prior to the establishment of the structure of the database. To satisfy diverse access needs, links may be required between any of the components of the database. Both the hierarchical and network models are composed of precisely defined links between the nodes. As a result, the structure of the database is fixed and cannot easily be changed.

Both the hierarchical and network models present difficulties in representing many to many relationships. An additional disadvantage is that *loops* are not permitted, i.e., relationships cannot be established between a record type and itself. This is particularly disadvantageous in a structural engineering database where such relationships are common: beams are connected to adjacent beams; columns to adjacent columns; etc..

Neither of these disadvantages occurs with the relational model. Its use requires knowledge of only one data construct and its underlying access mechanisms are hidden from the user. The user needs to be concerned only with the content of individual relations. The hierarchical and network models do, however, allow for efficient implementations. Because hierarchy and network links are implemented as pointers, node traversal is direct and fast. These efficiencies contrast with the relational model whose primary disadvantage at the present is its lower efficiency of accessing.

An additional advantage of the relational model is its ability to avoid common anomalies through *normalization*, the process of removing dependencies from among the attributes of relations. The concept of normalized relations is an integral part of the relational model and it promotes the achievement of well structured data while providing a degree of automatic integrity and consistency checking.

Most existing commercial database systems are based on the hierarchical and network models. The relational database model is the newest model and only recently have commercial relational database systems such as RIMS [7] and SQL [20, 21] become available. However, as efficiency problems are reduced, relational implementations will see far greater use. The versatility and flexibility of the relational model make it ideal for managing structural engineering data.

4. THE FUNCTION OF DATABASE MANAGEMENT SYSTEMS IN STRUCTURAL ENGINEERING

4.1 Potential Role

DBMS systems offer many advantages for structural engineering use. Program-data independence is one of these. A DBMS stores data in such a way that many different applications, both existing and planned, can use it. The data is structured for efficient use but it is not tailored to a particular application program.

DBMS systems provide a structure for storing a wide variety of data. A collection of data far more global than that needed by a single application program or a single design discipline can be achieved, linking together many different applications around commonly shared data. This integration offers promise of significant improvement in overall information storage and handling.

DBMS systems provide the conceptual framework for organizing design data so that all database needs can be efficiently and uniformly supported. Two aspects of DBMS's are of particular significance. One is the intellectual discipline required to formulate the overall framework or schema of the entire database, an absolutely essential step but one likely to be ignored when a system of programs is allowed to grow in an uncoördinated fashion. It is worth emphasizing that the schema can be extended and the database reorganized as needed without affecting existing applications. The second significant aspect is the availability of query languages, so that many accesses to the database can be made directly, without the need to first develop an application program.

It is to be expected that structural engineering design systems will be developed based on both the network and relational data models (hierarchical models have already been essentially supplanted by network models). In view of the conceptual advantages of the relational model and the vast amount of research and development for making it more general and efficient, it is likely that eventually it will become predominant. The comments that follow are based on this development.

4.2 Extensions Needed

To assume a major role in structural engineering, extensions to the present relational DBMS capabilities are needed. These are particularly necessary in the areas of run-time storage management, extension of data types, and integrity management.

Run-Time Storage Management. The efficient operation of data dependent systems and the integrating capabilities of data independent systems are both desirable characteristics. The logical extension to existing DBMS systems to achieve both simultaneously is a facility for run time storage management. Such a facility would temporarily restructure those portions of the database needed by an application program during its operation. Application programs interfacing using a DBMS with a run time storage management system would be treated as modules that utilize the DBMS only for input and output. For program input the DBMS would *export* the data to the storage management system. Output data from the application program could be processed directly using DBMS facilities.

Extended Data Types. From a structural engineering perspective one of the significant drawbacks of existing relational DBMS's is their limited number of available data types: integers, reals, and character strings. These types are sufficient for business applications but engineering applications require the use of a wider range of data types including vectors, arrays, matrices, etc..

Integrity Management Enhancements. Extensions are also needed in the area of integrity management. The built-in integrity controls of present relational DBMS systems are extremely limited. In the majority of cases only constraints on single attributes of single tuples can be enforced. These are not sufficient to satisfy engineering design integrity management needs. Functional dependencies between multiple attributes of tuples must be established. Where functional dependencies result in defined constraints, those constraints must be automatically enforced. Mechanisms to achieve this integrity capability are introduced below.

5. INTEGRITY AND CONSISTENCY ISSUES

5.1 Functional Integrity

Functional integrity and consistency with respect to governing laws must be enforced by application programs. These constraints cannot be built into the database because they must invoke high-level complex sequences of computations.

For example, a structural analysis application program must output a set of member forces and member properties in accordance with structural laws of compatibility and equilibrium. As a result the interface between the database and the program is very limited. The interaction between them is simply an input/output transformation. It is taken for granted by the database that the output it receives from the application program is consistent with the constraints built into the program.

5.2 Constraint Needs

Although the global constraints mentioned above cannot be built into the database, there are many constraints and design functions that can be. Design criteria, interaction constraints, consistency constraints on redundant data, and iteration control are examples of constraints that must be enforced in a structural engineering database.

Design Criteria. Design criteria specify limitations on attribute values and may be defined by codes, standards, and specifications or by a database user or designer. Codes, standards, and specifications comprise a class of constraints designed to assure the functionality and usability of the entity which they govern. They consist of a set of requirements that govern the design and behavior of the components of the entity [8]. Examples of such design criteria are:

$\mathbf{M_{provided} \geq M_{required}}$ for a member, and

$\mathbf{b \; / \; t \leq K * \sqrt{F_y}}$ for a plate element of a steel member.

User-defined constraints arise from each user's personal design style. They may be issued at any point during the design process and incorporated into the database. An example of a user-defined design criterion would be:

$$\mathbf{F_y \leq 50ksi}$$

limiting yield stress to a maximum value of 50 ksi.

Interaction. Each design discipline deals with a set of attributes commonly used by designers of the discipline. Often individual attributes of one discipline are related to attributes of other disciplines. Such attributes are referred to as boundary attributes. Interaction constraints are those which define relationships between boundary attributes. One example of an interaction constraint is:

$$\mathbf{d \leq clearance.}$$

This constraint relates the beam depth, **d**, under control of the structural designer, to the available **clearance**, dealt with by the architectural subsystem.

Consistency. Consistency is a special case of integrity and deals either with an attribute redundantly stored in more than one database location or with dependencies among attributes where dependent values are calculated on the basis of independent data and constraints among the attributes. In the first case each occurrence of the attribute is equivalent and must have the same value. In the second case the value of the dependent data may be computed when it is needed.

The constraint among beam section properties:

$$S = 2 * I / d$$

can be used to illustrate redundancy and consistency. 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 attributes is changed, the value of one of the others must also be changed; otherwise the consistency of the relationship and the integrity of the database are violated. The constraint is thus used in a passive *checking* mode. Alternatively, it is possible to store any two of the three attribute values, calculating the third as needed. In this manner the constraint is used in an active *assignment* mode and integrity is automatically maintained.

In structural engineering databases redundantly stored attribute values and consistency constraints among attributes are necessary. Unfortunately, the only tool available in a relational DBMS to prevent redundancy and consistency problems is normalization. But structural engineering data storage needs cannot be satisfied by normalization because they require that a variety of interrelated data be stored together, as shown above, so that all potentially needed data is immediately available for use.

Iteration. One important capability that must be supported by an engineering database is the design of an entity through multiple iterations and the control of the process of iteration. The database must provide the ability to distinguish between iterations and to orderly assemble subsequent generations of data.

The control of iteration can be achieved using constraints. To determine if the result of a structural analysis-sizing iteration has converged, for example, one must measure the difference between the moment of inertia, I, before and after sizing using the constraint

$$|I_{new} - I_{old}| / I_{new} \leq \text{epsilon.}$$

If the difference is less than the acceptable tolerance, sizing is complete and another design stage may be initiated. Otherwise, additional iterations are necessary.

5.3 A Model

Introduced above were a number of requirements a database must satisfy to be an effective structural engineering tool. Reference [16] describes a new relational data structuring scheme that eliminates problems of redundancy, update anomalies, and other irregularities caused by intrarelation dependencies among attributes. 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 intrarelation 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

MEMBER (ID, S, I, d)

containing the attributes of the section properties constraint introduced above. The alternative to normalizing this relation is to introduce into the relation the new attribute **sectionOK**

MEMBER (<u>ID</u>, S, I, d, sectionOK)

that monitors whether or not the constraint is satisfied. The attribute sectionOK is a boolean domain whose value is determined by the constraint. The constraint itself is stored in the database as a function. Whenever one of S, I, or d is changed, or when a new tuple is added to the relation, the constraint function is invoked and its value (true or false) is computed and stored in the tuple.

This model can be generalized to handle multiple constraints. All of the constraints introduced earlier can be converted to assignment form as follows:

$$\text{strengthOK} := M_{\text{provided}} \geq M_{\text{required}}$$

$$\text{sectionOK} := S = 2 * I / d$$

$$\text{clearanceOK} := d \leq \text{clearance}$$

$$\text{iterationOK} := |I_{\text{new}} - I_{\text{old}}| / I_{\text{new}} \leq \text{epsilon.}$$

These assignment statements would be inserted into functions and stored in the database. The modified relation would then contain the following attributes:

$$\text{MEMBER } (\underline{\text{ID}}, S, I_{\text{new}}, I_{\text{old}}, d, M_{\text{provided}}, M_{\text{required}}, \text{strengthOK},$$
$$\text{sectionOK, clearanceOK, iterationOK).}$$

In this manner the status of multiple constraints among intrarelation attributes can be monitored and recorded.

An additional capability provided by the model is the assignment of attribute values to dependent data items in such a way that the governing constraint(s) is automatically satisfied. To do so requires the constraint checking function to be converted to an assignment procedure in which the left hand side of the assignment statement is the dependent attribute. Converting the sectionOK constraint redundant in this manner results in the new assignment procedure:

```
PROCEDURE  S (I, d : real;  VAR S : real;  VAR sectionOK : boolean);
   BEGIN
       S := 2 * I / d;
       sectionOK := TRUE
   END;
```

The designer thus gains a new tool with which he can assign results known to be consistent with other attribute values in the database.

The primary advantage of the model is that the database user has direct access to the values of all of the attributes in the relation while insuring a measure of intrarelation integrity and consistency. Even though functional interdependencies are retained, the integrity of the tuples can at all times be determined by invoking the stored checking functions and recording the values returned.

6. CONCLUSIONS

The clear trend in structural engineering computer usage is towards higher levels of integration of individual programs, both horizontally (among different structural applications) and vertically (between structural applications and the applications of other participating disciplines, from planning to construction management and facility operations). At the same time, there is a similar trend towards more flexible use of programs in response to the wide range of design and analysis sequences needed in diverse projects. The data exchanged and shared among the applications is the key to integration.

There are three paths that organizations may take in their approach to data usage. One is to continue extending the present mode of high program-data dependence; this approach is highly self-limiting, and there is a clear indication that this mode cannot be fruitfully continued. Second, we can wait for the development of an "ideal" system, incorporating all structural engineering needs. The third approach, advocated in this paper, is to build on the highly developed – and rapidly developing – area of DBMS's, adding those specific extensions necessary for structural design.

A DBMS-based structural design database will provide much more than a passive repository of data, and will serve as an active agent in the design process. Many of the design control functions will be part of the database management activity. Consistency management with respect to a wide range of design constraints can be made an integral part of the DBMS. Other consistency management functions, such as monitoring spatial conflicts, can also be included in the DBMS. The need for a centralized database incorporating a high level of consistency management will become even more acute as the trend towards more decentralized computing, including personal computing, accelerates.

ACKNOWLEDGEMENTS

This work was sponsored in part by the National Science Foundation under grant MCS7822328 entitled "Data Base Methods for Design."

REFERENCES

[1] Bandurski, A. E., and Wallace, M. A.
COMRADE Data Management System Storage and Retrieval Techniques.
In *Proceedings 1973 National Computer Conference and Exposition*, pages 353-357. AFIPS, Montvale, NJ, 1973.

[2] Bland, R.
Private communication.
1982.

[3] Burner, B., Ives, F., Lixvar, J., and Shovlin, D.
The Design, Evaluation, and Implementation of the IPAD Distributed Computing System.
In *Proceedings of the First Conference on Computing in Civil Engineering*, pages 126-144. American Society of Civil Engineers, New York, NY, June, 1978.

[4] Date, C., J.
An Introduction to Database Systems.
Addison Wesley, Reading, MA, 1977.

[5] Eastman, C., and Thornton, R.
A Report on the GLIDE2 Language Definition.
Technical Report, Computer-Aided Design Group, Institute of Physical Planning, Carnegie-Mellon University, Pittsburgh, PA, March, 1979.

[6] Eastman, C.
An Introduction to GLIDE: Graphical Language for Interactive Design.
Technical Report, Computer-Aided Design Group, Institute of Building Sciences and Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1980.

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

[8] Fenves, S.J. and Wright, R.N.
The Representation and Use of Design Specifications.
In W.J. Hall (editor), *Structural and Geotechnical Mechanics*, pages 277-304. Prentice-Hall, Englewood Cliffs, NJ, 1977.

[9] Genesys Limited.
GENESYS.
Technical Report, Genesys Limited, Loughborough, England, 1976.

[10] Lopez, L. A.
POLO: Problem Oriented Language Organizer.
Journal of Computers and Structures 2(4):555-572, 1972.

[11] Lopez, L. A.
FILES: Automated Engineering Data Management System.
In *Sixth Conference on Electronic Computation*, pages 47-71. American Society of Civil Engineers, New York, NY, 1974.

[12] Martin, James.
Principles of Data-Base Management.
Prentice-Hall, Inc., Englewood Cliffs, NJ, 1976.

[13] *FASTDRAW/3 Reference Manual*
MCAUTO - Graphics Products, McDonnell Douglas Automation Company, St. Louis, Missouri, 1978.

[14] *FASTDRAW/3 Interactive Postprocessing Reference Manual*
MCAUTO - Graphics Products, McDonnell Douglas Automation Company, St. Louis, Missouri, 1980.

[15] Miller, R. E. et al.
Feasibility Study of an Integrated Program for Aerospace Vehicle Design (IPAD).
Technical Report, Boeing Commercial Airplane Company, Seattle, WA, 1973.

[16] 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, PA, April, 1982.

[17] Rhodes, T. H.
The Computer-Aided Design Environment Project (COMRADE).
In *National Computer Conference*, pages 319-324. AFIPS Press, 1973.

[18] Roos, D.
ICES System Design
Second Edition edition, The MIT Press, Cambridge, MA, 1967.

[19] Sandberg, G.
A Primer on Relational Database Concepts.
IBM Systems Journal 20(1):23-40, 1981.

[20] *SQL/ Data System General Information*
International Business Machines (IBM), White Plains, NY, 1981.
Report GH24-5012-0.

[21] *SQL/ Data System Concepts and Facilities*
International Business Machines (IBM), White Plains, NY, 1981.
Report GH24-5013-0.

[22] *UNISTRUC 2 Reference Manual*
Control Data Corporation, Minneapolis, MS, 1979.