User Interfaces for Structural Engineering Relational Databases

by H. Craig Howard and Cynthia Stotts Howard

TECHNICAL REPORT
Number 4

June, 1988

Stanford University

Engineering with Computers 4, 239–249 (1988)

Engineering with Computers

© Springer-Verlag New York Inc. 1988

User Interfaces for Structural Engineering Relational Data Base

H. Craig Howard and Cynthia Stotts Howard
Department of Civil Engineering, Stanford University, Stanford, California

Abstract: The needs of engineers in their interaction with engineering data bases are very different from those of their counterparts in the business world. Business data base management system interfaces typically provide only a single mode of textual communication, usually a structured query language. However, in an ideal *engineering* data base interface, an engineer would be able to define constraints, give examples, point at parts of pictures, and (sometimes) use several modes of communication simultaneously. The paper presents an example from an engineering design application to show how a traditional query language can be enhanced to accommodate the engineering needs. The paper further describes a conceptual approach for multimodal engineering data base interface combining multipurpose graphics, an engineering query language, and other interface methodologies in an engineering workstation environment.

1 Introduction

Data management is important in engineering computer applications where the volume of data is very large and where the same data must be shared by several users and programs. Engineering applications are making increasing use of data base management systems (DBMSs) to provide these capabilities. However, the needs of engineering systems are somewhat different from those of the business systems that stimulated the initial development of the DBMS technology [1,2]. The needs of engineering data bases with respect to data representation, consistency, integrity, and so on have been explored in a number of research efforts (e.g., see Refs. [3,4]). One result of this research has been the widespread adoption of relational data base[1] technology for management of engineering data.

[1] The relational data base model represents objects with a single level of *relations*. A relation defines the collection of attributes that describe a set of objects. An individual object is represented within the relation as a *tuple* (a set of attribute values describing the object). A relation may be pictured as a table in which the columns are the attributes and the rows are the tuples.

Reprint requests: H. Craig Howard, Department of Civil Engineering, Stanford University, Stanford, CA 94305–4020, USA.

Similarly, the interaction between engineers and engineering DBMSs can be very different from their business counterparts. In business data bases, there tends to be a small number of different types of requests that are well known in advance and are repeated over and over again (e.g., the query "get account balance for *x*" in a banking data base). However, in engineering data bases, the requests tend to be much more varied because engineers (especially structural engineers) frequently deal with *one-of-a-kind* objects (e.g., a design for a particular building). The data base requests involved in the design process for those unique elements are necessarily different. Therefore, an engineering data base interface must possess a maximum flexibility for the composition of many types of requests.

One possible approach to increasing flexibility in DBMSs is found in current research into natural languages interfaces. However, this research is applicable to engineering data bases only to the extent that engineers use natural language. Engineers also like to define constraints, give examples, point at parts of pictures, and (sometimes) use several modes of query specification simultaneously. In an ideal data base interface, an engineer would be able to formulate requests using the most natural means of expression, be it text, formulas, graphics, or combinations thereof.

Most existing DBMSs front-ends are unimodal; typically, the user interface is driven by a structured query language. Some DBMSs have multiple, independent user interfaces; that is, the same physical data base can be accessed by different query language interfaces. However, to achieve the level of flexibility described earlier, we need a DBMS interface with multiple, *interdependent* interface modes. In such a system a single request could be formed by pointing at an object on a graphics display, entering a set of applicable design constraints, and selecting required attributes from a graphical display of the standard object labeled with its attributes. Thus the engineer can formulate a request using the mode or combination of modes that best suits that request.

Table 1. Wide-flange-shape relation.

Designation	Nominal depth	Weight	Depth	Section modulus
W4 × 13	4	13	4.16	5.46
W14 × 233	14	233	16.04	375
W30 × 116	30	116	30.01	329

Our research in this area has already led to a first version of an engineering data base query language for relational data bases. Section 2 describes that language and its rationale along with a brief overview of relational data base query languages. Section 3 provides some background on alternative query methodologies drawn from data base research and practice. Finally, Section 4 presents a conceptual overview of a multimodal engineering data base interface to be implemented in an engineering workstation environment.

2 An Engineering Query Language

Our interest in the area of engineering data base interfaces arose from previous research on interfaces between expert systems and DBMSs. From that work it became apparent that traditional DBMS query languages can be awkward to use in the specification of complex engineering design queries. The first part of this section briefly discusses the characteristics of those traditional structured query language interfaces for relational data bases. The second part describes how a traditional query language was extended to facilitate a class of engineering queries.

The sample queries in this section address the "wide-flange-shape" relation shown in Table 1. The sample relation is an abbreviated version of the table of wide-flange structural steel shapes drawn from the *AISC Manual of Steel Construction* [5].

2.1 Structured Query Languages

With a structured query language,[2] the user formulates queries using special commands according to a rigidly defined syntax. The user is responsible for expressing requests within the limitations of the query language. Structured query languages come in many different forms. Two of the best known are the SQL (originally SEQUEL) language [6] and the QUEL language of INGRES [7]. Full definitions of these languages are beyond the scope of this section. The typical use of these query languages is illustrated in the following examples involving queries for values from the sample relation of wide-flange shapes (Fig. 1).

ENGLISH

Get the designation of the wide-flange shape having a depth greater than 12 inches.

SQL

```
select designation
from  wide-flange-shape
where depth > 12
```

QUEL

```
range of wf is wide-flange-shape
retrieve (wf.designation)
      where wf.depth > 12
```

Fig. 1. Structured query language example queries

The two sample queries are fundamentally mathematical expressions, building on the formal definition of relational data bases. In those terms, the SQL and QUEL languages are based on *tuple-oriented relational calculus.* A few languages are based on *relational algebra,* but they tend to be less widely used because relational algebra expressions specify *how* the data is to be retrieved, whereas relational calculus expressions describe *what* is to be retrieved without specifying how.

2.2 Engineering Query Language Example

The example discussed in this section arose during research into interfacing engineering expert systems with DBMS [8]. That research yielded some basic observations about the characteristics of engineering queries. In particular, for a single-object query formulated during the design process, it is very useful to be able to separate the elements of the query that address constraints on the object being designed from the elements of the query that specify the method for selecting the optimum object. This section presents some examples to illustrate the complications of representing such queries in conventional structured query languages and the ease of representation in the engineering query language.

Queries to a data base for information about a single object take the general form

Find ⟨*attributes of object*⟩ where ⟨*set of qualifiers*⟩ is satisfied.

[2] In formal terms, the query language is called a *data manipulation language* (DML), as distinguished from a *data definition language* (DDL), which is used to define the logical and physical data structures in the data base.

ENGLISH

Get the designation of the lightest wide-flange shape
having a section-modulus greater than 300 cubic inches.

QUEL

```
range of wf is wide-flange-shape
retrieve (wf.designation)
         where wf.section-modulus > 300
         and wf.weight = MIN(wf.weight
                             where wf.section-modulus > 300)
```

SQL

```
SELECT wf.designation
FROM   wide-flange-shape
WHERE  wf.section-modulus > 300
AND    wf.weight =
       (SELECT MIN(wf.weight)
        FROM   wide-flange-shape
        WHERE  wf.section-modulus > 300)
```

Fig. 2. Sample engineering queries

Frequently, that set of qualifiers can be partitioned into two subsets that reveal more about the substance of the query. This partition can be illustrated with a simple structural engineering example. In relational query languages like QUEL and SQL, a query to select a suitable cross section from a data base of standard steel wide-flange shapes would take on the forms shown in Fig. 2: These queries contain two qualifiers: (1) the section modulus must be greater than the required section modulus and (2) the weight must be the minimum weight for the set of sections that satisfy the first qualifier. The two qualifiers are excellent paradigms of two basic qualifier types:

- *Constraints*—those qualifiers that test the entity against an absolute standard; for example, section-modulus is greater than 300. Constraints determine a *satisficing* set of entities, that is, those entities that satisfy the constraints without necessarily being optimum solutions.
- *Optimizing criteria*—those qualifiers that test the entities against a relative standard by comparing the entity to other entities of its class; for example, weight = min(weight). In relational data bases, *min* and *max* are called aggregate operators because they return a single aggregate value based on attribute values for a set of objects. Optimizing criteria are normally used to select a single object from the satisficing set determined by the constraints.

This division of qualifiers is not represented clearly in either of the earlier sample queries because both formulations require that the constraint be repeated in the aggregate operation, since the scoping of the variables in the aggregate expression is independent of the variables in the rest of the query. When an aggregate expression is encountered, the query processor usually suspends its translation of the query to evaluate the aggregate. The value returned by the aggregate is substituted for the aggregate function expression, and the query processor continues with the remainder of the query.

```
retrieve (designation)
from   wide-flange-shape
       where section-modulus >= 300
       subject to (min(weight))
```

Fig. 3. Engineering query language example query

The queries can become very complicated as the optimizing criteria are applied sequentially; that is, the satisficing set is pared by the application of a first criterion, and the resulting set is subjected to a second criterion.

Therefore, our engineering query language extends the basic query form to be

Find ⟨*attributes of object*⟩ where ⟨*set of constraints*⟩ is satisfied, subject to ⟨*ordered set of optimizing criteria*⟩

The constraints are applied first to determine the satisficing set, and the optimizing criteria are applied in sequential order to narrow the set to the desired entity. Adopting the QUEL syntax shown previously, the result for the first example would look something like the query shown in Fig. 3.

To illustrate further the power of this method of expression for engineering design queries, we present an example taken from the HI-RISE expert system for the preliminary design of high-rise buildings [9]. The example concerns the selection of a steel wide-flange section for a floor beam in a high-rise structure. The constraint is the same as in the previous example—the section modulus must be greater than the required section modulus (300), but the criteria are different. Since ceiling-to-floor depth is critical in high-rise construction, the primary optimizing criterion is to find the minimum nominal depth. The criterion for minimum weight remains, but it is secondary. The query formulations in QUEL, SQL, and the engineering query language are shown in Fig. 4. The intent of the query is much clearer in the engineering formulation than either the QUEL or SQL forms. The engineering form is also more amenable to query optimization because the structure of the query is not procedurally restrictive. When confronted by this kind of query, SQL and QUEL users may opt to use three separate queries with temporary relations to cache intermediate results, but once again the basic intent of query is lost. Similarly, languages based on

QUEL

```
range of wf is wide-flange-shape
retrieve (wf.designation)
        where wf.section-modulus > 300
        and wf.nominal-depth
            = MIN(wf.nominal-depth
                  where wf.section-modulus > 300)
        and wf.weight
            = MIN(wf.weight
                  where wf.nominal-depth = MIN(wf.nominal-depth
                                               where wf.section-modulus > 300))
```

SQL

```
SELECT designation
FROM   wide-flange-shape
WHERE  section-modulus > 300
AND    nominal-depth =
       (SELECT MIN(nominal-depth)
        FROM   wide-flange-shape
        WHERE  section-modulus > 300)
AND    weight =
       (SELECT MIN(weight)
        FROM   wide-flange-shape
        WHERE  section-modulus > 300
        AND    nominal-depth =
               (SELECT MIN(nominal-depth)
                FROM   wide-flange-shape
                WHERE  section-modulus > 300))
```

Engineering Query Language

```
retrieve (designation)
from     wide-flange-shape
         where section-modulus > 300
         subject to (min(nominal-depth),min(weight))
```

Fig. 4. Complex engineering queries

relational algebra can represent this query as a set of nested "select" commands. However, as noted previously, the relational algebra formulations require the user to describe a query in terms of *how* it is to be processed instead of *what* is to be found.

The optimization criteria feature has been implemented in the KADBASE query language (KQL) [10] and used successfully in data base interfaces for the HI-RISE expert system [9] and the SPEX structural component design expert system [11]. In the implementation, KADBASE translates the KQL constraints and criteria into a sequence of QUEL queries to an INGRES data base using temporary relations. KADBASE takes care of the administrative details required to chain the queries together through the correct temporary relations and to delete the temporary relations when the query processing has been completed. The translation process and attendant administrative details are completely hidden from the querying component.

3 Interface Methodologies

In addition to the structured query languages described earlier, other paradigms for data base access have been proposed. In the following section we discuss various interface methodologies and their uses in accessing engineering data bases. While these methodologies vary in ease of use and, in some cases, incorporate graphics, no single paradigm is adequate for an engineering data base interface.

wide-flange-shape	designation	nominal-depth	weight	depth	section-modulus
	P.			> 12	

Fig. 5. Query-by-example sample query

3.1 *Formal Structured Query Interfaces*

The query-by-example (QBE) [12] language is a different approach to a structured query language. Queries in QBE are represented as entries in tables. QBE utilizes the tabular nature of relational data bases and the natural desire of human users to demonstrate what they want rather than express a query according to some arbitrary syntax. In QBE, the sample query used in the structured query language section would be represented as shown in Fig. 5. The table represents the sample relation wide-flange shape and its attributes as described in the last section. The notation "> 12" in the depth column represents the qualifier "depth > 12," and the notation "P." in the designation column indicates that the values for the corresponding designations are to be printed.

Like SQL and QUEL, QBE has a mathematical basis in relational data base theory. Formally speaking, QBE is a *domain-oriented relational calculus* language.

3.2 *Natural Language Data Base Interfaces*

In a natural language (NL) data base interface, the user expresses the request in the same terms that would be used in communicating with a human data base manager. The user does not need to learn any special language syntax or command structure. Natural language data base interfaces are useful when requests are easily expressed in English, but engineers tend to talk about data in mathematical terms, so a natural language interface is not a complete answer.

A variety of systems have been implemented to provide restricted natural language (NL) access to software systems including data bases. Among these are:

- *CO-OP* [13]—a portable NL interface for data base systems.
- *XCALIBUR* [14]—a domain-independent NL interface for an expert system.
- *MDX* [15]—a medical expert system with a NL

front-end that has also been implemented for another expert system and a relational data base.
- *RESADA* [16]—a deductive NL data base system for querying about biographical data.

None of these systems is capable of a full range of discourse, even within the domain of the application or data base. However, each has the power to parse simple queries expressed in conversational English into data base access requests and to ask for clarifications where ambiguities exist. The CO-OP system provides several good examples of the types of queries that can be processed using the current technology. For example (taken from Ref. [13])

- "Who advises projects in area 36?"
- "Which programmers from the ASD group are in superdivision 5000?"
- "Which users work on every project in area 55?"

3.3 Graphical Extensions to Formal Structured Query Interfaces

Several graphical query languages have been developed by extending structured query languages. For example, the pictorial structured query language (PSQL) [17] is based on SQL, with the extension of two extra clauses: "on ⟨picture-list⟩" and "at ⟨area-specification⟩." The area specification can include spatial operators such as "covering," is-covered-by," and so on. PSQL accommodates direct (or explicitly specified) spatial search, and indirect spatial search, in which the location is determined by the constraints listed in the "where" clause. Query by pictorial example (QPE) [18] provides similar capabilities and also allows users to enter areas or shapes to be matched using graphical input devices.

While these languages do combine pictorial and textual data, they still require that the user apply a formal (and sometimes awkward) structure for specifying queries. Furthermore, these languages focus on the characteristics of two-dimensional images rather than the spatial relationships inherent to structural engineering data.

3.4 Graphical Data Base Interfaces

In a graphical query interface, the user expresses requests by pointing at objects on a graphics display in conjunction with commands entered via the keyboard or selected from a displayed menu. A graphical interface permits a user to express the concept of "*that beam*" (by pointing at it) without having to know some unique beam identifier (e.g., beam number "123"). However, if the user wants to ask some complex, multifaceted question about that beam, then the capabilities of a query language (structured or natural) are required.

Most existing computer-aided design and drafting (CADD) systems support their own ad hoc data bases of graphical information. The CADD commands for displaying, drawings, deleting, and so on act as a specialized data base language; for example, the "display" and "draw" commands roughly correspond to the data base commands "retrieve" and "insert." This level of data base interaction is sufficient when the engineer is only interested in graphical objects, but it is inadequate when the engineer wants to address nongraphical information related to the graphical objects. Complicating the problem, the graphical data base for the CADD system is frequently kept separate from the data base of nongraphical attributes used by the rest of the engineering software.

3.5 Navigational Data Base Interfaces

Structured query languages require that the user know the exact structure of the data base being queried [19]. Navigational data base interfaces relieve the user of this burden by providing a "map" of the database. The user formulates a query by indicating the area of interest on the map.

An example of a navigational data base interface is the spatial data management system (SDMS) [20,21]. In the SDMS, users are presented with a "world view" of the data base. Using a joy stick, the user selects an area of the world view and "zooms in" for more detail. By repeating these steps, the user can obtain data on specific items in the data base. The data base administrator determines how the data base will be protrayed; the portrayal of the data does not necessarily reflect the underlying structure of the data base.

Other navigational data base interfaces are based more closely on the structure of the data base. The data base interface aid described by Burgess [19] provides the user with a tree diagram of the data base. Only the current and next lowest levels of the data base are portrayed. The user navigates through the data base by specifying which leaf to visit next.

Navigational data base interfaces provide the user with a clear picture (literally) of the structure and contents of the data base. They are very easy to learn and encourage browsing. However, they do not lend themselves to the complex types of queries possible in engineering tasks.

3.6 Nonsimultaneous Multimode Data Base Interfaces

Many current DBMSs provide multiple, *separate* user interfaces. Frequently, the separate interfaces consist of a formal query language interface on the one hand and a specialized report writer on the other. IBM's Query Management Facility (QMF) [22] provides two formal query language interfaces: SQL and QBE. Relational Technologies has developed a version of INGRES that accepts requests in either QUEL or SQL. The principal drawback of these systems is that the two modes cannot be used simultaneously in the formulation of a single request.

3.7 Conclusions on Existing Methodologies

Each of the query representation techniques discussed earlier has its own advantages and disadvantages. No one form is ideally suited to all query formulation requirements. In every language there are queries that are awkward or impossible to express. The basic question is, *How can we combine these query languages in such a way as to accumulate their advantages while omitting the necessity of dealing with their disadvantages?* The following section draws upon the various query representation paradigms presented here to paint a broad picture of a multimodal query interface for engineering data base.

4 Conceptual Overview of a Multimodal Engineering Data Base Interface

The three basic characteristics of our data base interface model are:

- *The engineering user should have multiple modes of query formulation simultaneously available;* for example, an engineering query language, a QBE template, a graphical display, and so on.
- *The various query formulation modes will interact by contributing to the formulation of a single query;* for example, selecting several components in a graphical display and requesting attribute values in a QBE template will result in a single query for the values of the specified attributes for the specified components.
- *Each mode will update its query formulation information to correspond to query components entered in through other modes;* for example, if the engineer has selected the component type beam in one mode, all other modes can specialize their responses based on the definition of the object beam and its attributes.

On an engineering workstation each mode is implemented in an individual window on the screen display. The query formulation windows are linked through a central query processor that communicates with a DBMS. Figure 6 shows the conceptual architecture of such a system, illustrating the following query formulation modes:

- The *data base map* shows the organization of the data in a graphical format, allowing the engineer to navigate through the data base by following the natural interrelationships of the data base objects. This interface would be similar to the world view of the SDMS.
- The *engineering query language* (EQL) provides a powerful, structured, relationally complete query formulation capability. The language includes the extensions for optimizing criteria described previously. Additional engineering-oriented language features may be desirable, especially in the area of spatially oriented query operators (as in PSQL).
- The *query-by-example* (QBE) interface provides the engineer with a tabular paradigm for query formulation.
- The *graphical attribute selection* adds a pictorial element to the basic idea of QBE. Engineers indicate the attributes to be returned by selecting them via a pointer on a graphical representation of the component. This interface mode is particularly desirable in complex, dimensioned objects.
- The *graphical object selection* provides graphical, CADD interface to the data. Through this interface mode the engineer can specify an individual component, several components, or entire systems of components by using a graphical cursor to select components and systems on a screen display. The level of selection can be controlled by other interface components; for example, if the engineer has selected an object type through another mode, that object type governs the response in the graphical object selection.
- The *natural language* (NL) interface is included in this model to emphasize the flexibility envisioned for the multimodal system. While we noted earlier that engineers need much more than natural (textual) langauge to express engineering data requests, an unstructured NL subsystem can be a useful complement to the more mathematical and graphical query formulation modes, which can be used to establish a context for the NL discourse.

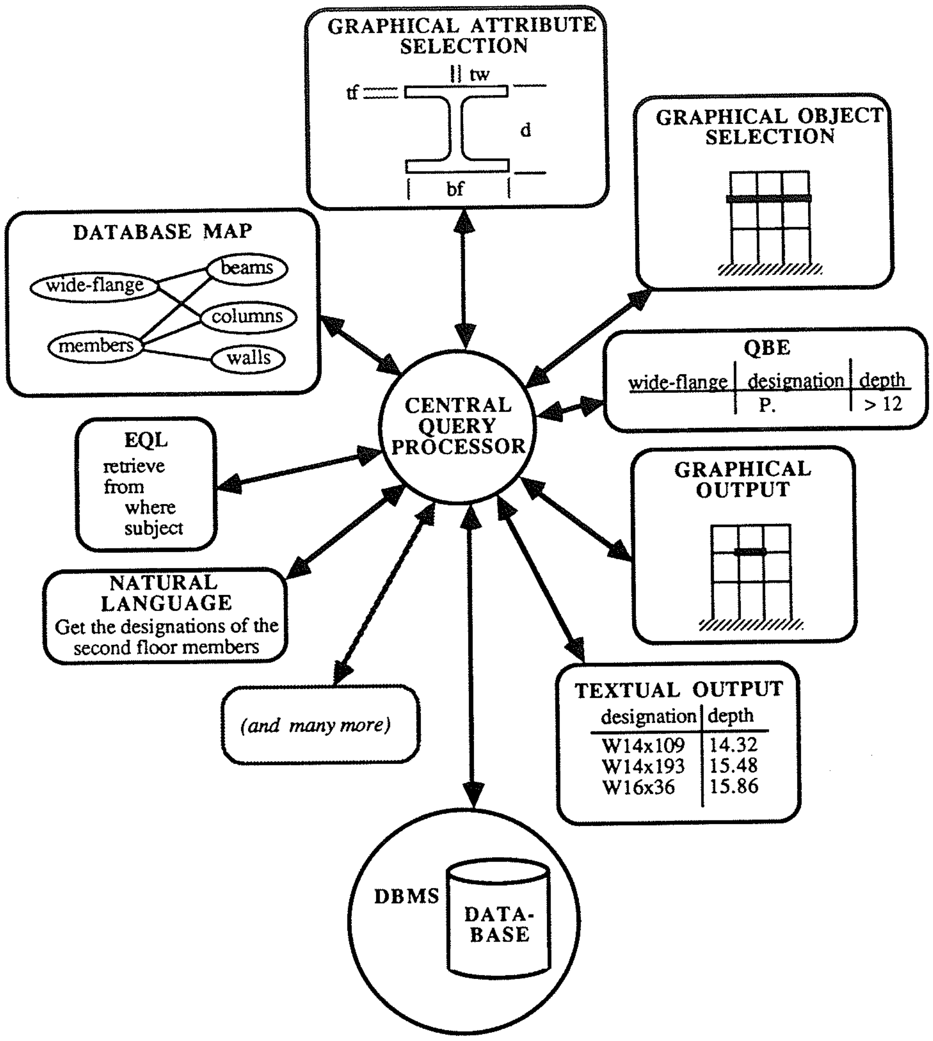


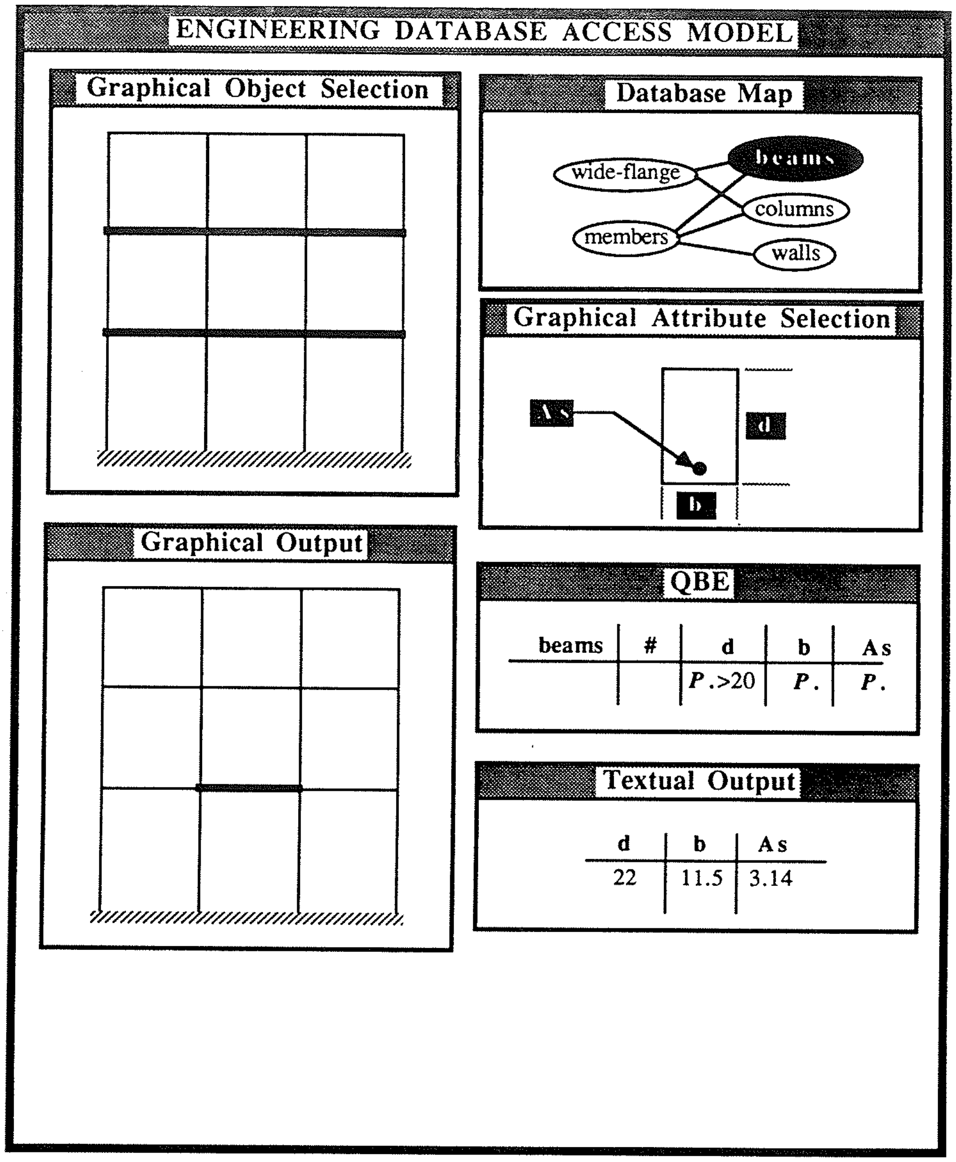
Fig. 6. Integration of multiple query modes

These query input windows are complemented by output windows displaying textual and graphical results.

The two keys to the integration of the multiple modes into a single coherent system are the central query processor and the standardized language for communicating about partially formed queries. The central query processor is responsible for combining the partial queries into a single query suitable for the associated DBMS. It also communicates each step of the query formulation process to each interface mode; therefore, in the subsequent steps all modes know about object types, attributes, and constraints already specified and can adjust their displays and responses accordingly. The two-way communication of partial queries requires an internal language that includes a full complement of query representation features.

Figures 7 and 8 demonstrate sample formulations of engineering queries in a multimodal environment. The first query (Fig. 7) asks which beams on the second and third floors of the concrete frame have depths greater than 20 inches. It is intended to illustrate the basic ideas of combining query formulation modes to naturally express the elements of the request. The engineer would formulate this query using the following steps:

1. In the *data base map* window, the engineer selects the object *beams*. In response to the engineer's selection, the oval representing the *beams* object is highlighted, and the *graphical attribute selection* window displays a cross-sectional view of a concrete beam, labeled with the attributes that describe concrete beams.
2. In the *graphical attribute selection* window, the


ENGINEERING DATABASE ACCESS MODEL
Graphical Object Selection
Database Map
wide-flange
beams
columns
members
walls
Graphical Attribute Selection
As
d
b
Graphical Output
QBE
Textual Output

beams	#	d	b	As
		P.>20	*P*.	*P*.

d	b	As
22	11.5	3.14

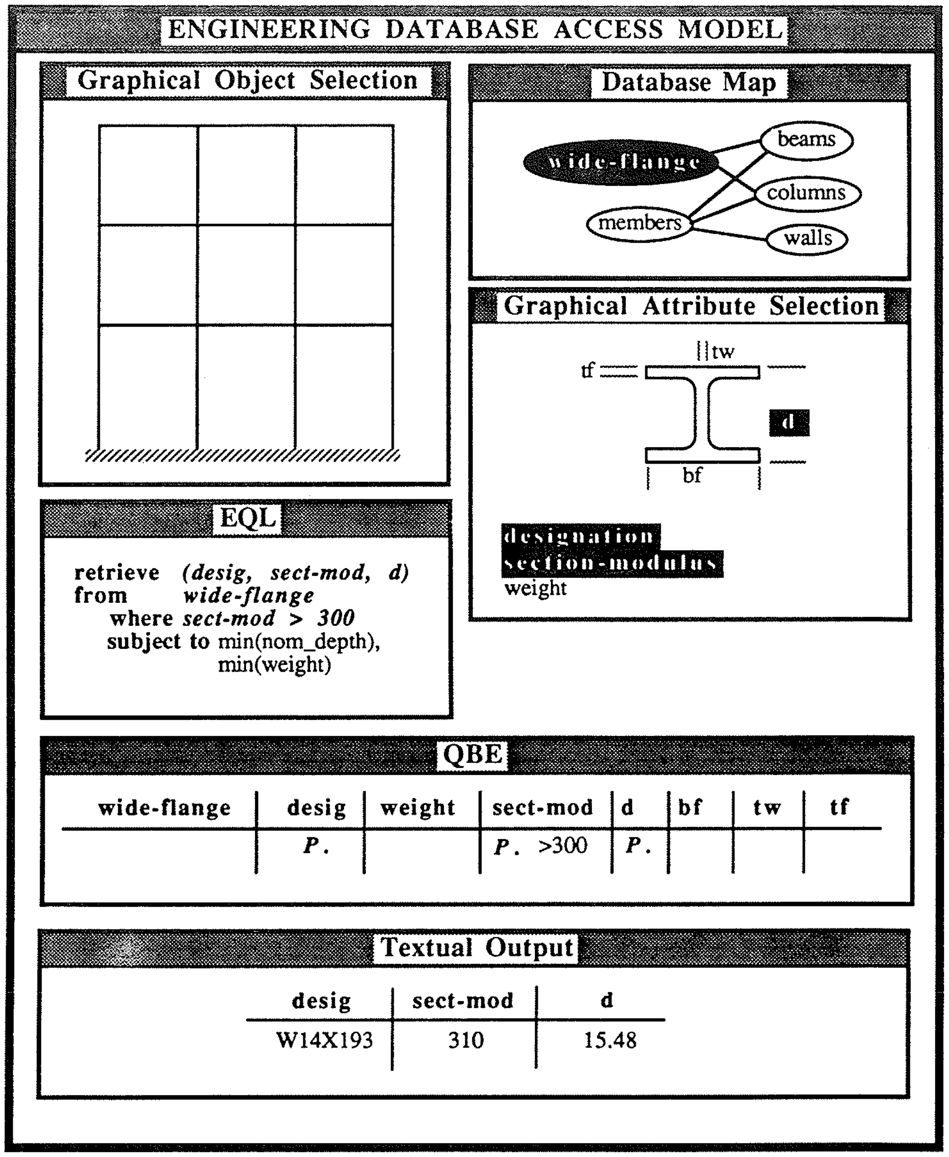
Fig. 7. Multimodal formulation of simple query

engineer selects the attributes *b, d, As* on the concrete beam diagram. As the engineer points to each attribute, it is highlighted on the display. In response to the selections in this window, the *query-by-example* (QBE) window is updated to show which attributes have been selected for output (represented by the "P." in the applicable columns).
3. In the *QBE* window, the engineer specifies the constraint "depth > 20" by typing "> 20" in the depth column.
4. In the *graphical object selection* window, the engineer selects an area containing the second and third floors. The window knows that the engineer has already selected *beams* as the object of interest, so for the chosen area only the beams are highlighted in the display.

The beams satisfying the constraint are highlighted in the *graphical output* window, and the associated attribute values are shown in the *textual output* window. Now the engineer is free to compose another query or modify the current one.

Figure 8 represents a multimodal formulation of the complex engineering design query described earlier in the paper. The query is composed as follows:

1. In the *data base map* window, the engineer selects the object wide-flange. The *graphical at-*



```
retrieve  (desig, sect-mod, d)
from      wide-flange
   where sect-mod > 300
   subject to min(nom_depth),
              min(weight)
```

wide-flange	desig	weight	sect-mod	d	bf	tw	tf
	P.		*P.* >300	*P.*			

desig	sect-mod	d
W14X193	310	15.48

Fig. 8. Multimodal formulation of engineering design query

tribute selection and *query-by-example* (QBE) windows immediately display the templates associated with wide-flange sections.

2. In the *graphical attribute selection* window, the engineer indicates the attribute values to be returned, in this case designation, depth, and section modulus. The *QBE* and *engineering query language* (EQL) windows are also updated to show which attributes have been selected for output.
3. In the *QBE* window, the engineer specifies the constraint on the section modulus by typing "> 300" in the section-modulus column. The EQL window also updates its display to include the constraint.
4. In the *EQL* window, the engineer describes the two optimizing criteria: min(nom_depth), min(weight).

The *textual output* window shows that the desired wide-flange shape is a W14 × 193.

Our intent in these examples has been to demonstrate the basic functionality of a multimodal engineering data base interface, to show how multiple query formulation options can be combined to simplify and clarify the engineer's data access needs. The engineering data base interface can be easily expanded to incorporate new query formulation modes and new data bases (even multiple data base systems).

5 Conclusions

We have presented an engineering query language that substantially improves the formulation of a broad class of engineering design queries. The engineering query language has served as a springboard for a broader exploration of the requirements for engineering data base interfaces. Our conclusion is that an engineering data base interface must support multiple, simultaneous modes of query formulation, including both graphical and textual specification of query information. The implementation of a flexible, multimodal interface of the type proposed here has the potential for far-reaching effects on the structural engineering data management. Among these are

- *Greater reliance on formal DBMSs for the storage of engineering data.* Data base management techniques are extremely valuable in maintaining the integrity and consistency of design data, but even the most powerful software systems are of little use if they are not responsive to the needs of their users. The engineering data base interface would encourage the greater use of DBMSs for design data.
- *Integration of graphical and nongraphical engineering data.* One of the stumbling blocks to effective integration of graphical and nongraphical data is the lack of a unified approach to accessing the two types of data. In general, textual data bases cannot draw pictures, and CADD data bases treat nongraphical data as mere attachments to graphical objects. If the digital representations of engineering designs are to be truly cohesive, the designers must have a unified interface to all of the data.
- *Better communication in the design/construction process.* The data *is* the design, and, with the increasing reliance on computer-aided design, more and more of the design data communication is transacted through the computer, specifically via the design data base. The engineering data base interface will give design/construction professionals better access to the data and, therefore, better communication about the design.

References

1. Codd, E.F. (1982) Relational databases: A practical foundation for productivity. Commun. ACM 25 (2), 109–117
2. Fenves, S.J.: Rasdorf, W.J. (1982) Role of Database Management Systems in Structural Engineering. Technical Report DRC-12-13-82, Design Research Center, Carnegie-Mellon University, Pittsburgh, December
3. Fenves, S.J., Rasdorf, W.J. (1982) Treatment of Engineering Design Constraints in a Relational Database. Technical Report DRC-12-14-82, Design Research Center, Carnegie-Mellon University, Pittsburgh, December
4. Rasdorf, W.J., Fenves, S.J. (1983) Organization of a structural design database. In: Proceedings of the Eight Conference on Electronic Computation, (Ed. J.K. Nelson, Jr.). Committee on Electronic Computation of the Structural Division of the American Society of Civil Engineers, pp. 559–571, February
5. Manual of Steel Construction, 8th Ed. (1980) American Institute of Steel Construction, Chicago, IL
6. Chamberlin, D.D., Boyce, R.F. (1974) SEQUEL: A structured English query language. In: Proceedings, 1974 ACM SIGMOD Workshop on Data Description, Access and Control, Ann Arbor, MI
7. Stonebraker, M., Wong, E., Kreps, P. (1976) The design and implementation of INGRES. ACM Trans. Database Syst. (3), 189–222
8. Howard, H.C., Rehak, D.R. (1988) KADBASE: A prototype expert system-database interface for engineering systems. IEEE Expert [in press]
9. Maher, M.L., Fenves, S.J. (1985) HI-RISE: An expert system for the preliminary structural design of high rise buildings. In Knowledge Engineering in Computer-Aided Design, (Ed. J.S. Gero). Amsterdam: North-Holland
10. Howard, H.C., Rehak, D.R. (1986) Interfacing Databases and Knowledge Based Systems for Structural Engineering Applications. Technical Report EDRC-12-06-86, Engineering Design Research Center, Carnegie-Mellon University, Pittsburgh, November
11. Garrett, J.H., Jr., Fenves, S.J. (1987) A knowledge-based standards processor for structural component design. Eng. Comput., 2, 219–238
12. Zloof, M.M. (1974) Query by example. In: Proceedings, National Computer Conference, Vol. 44, Anaheim, CA, pp. 431–437
13. Kaplan, S.J. (1984) Designing a portable natural language database query system. ACM Trans. Database Syst., Assoc. Comput. Mach. 9(1), 1–19
14. Carbonell, J.G., Boggs, W.M., Mauldin, M.L. (1983) The XCALIBUR project: A natural language interface to expert systems. In: Proceedings, International Joint Conference on Artificial Intelligence, Vol. 2, Karlsruhe, West Germany, pp. 653–656, August
15. Obermeier, K.K. (1984) Natural language front-ends for expert systems. In: Proceedings, National Online Meeting, New York, pp. 265–272, April
16. Zarri, C. (1984) Expert systems and information retrieval: An experiment in the domain of biographical data management. Int. J. Man-Mach. Studies 20(1), 87–106
17. Roussopoulos, N., Leifker, D. (1984) An introduction to PSQL: A pictorial structured query language. In: Proceedings, 1984 IEEE Computer Society Workshop on Visual Languages, Hiroshima, Japan, pp. 77–87, December
18. Chang, N.S., Fu, K.S. (1980) A relational database system for images. In: Pictorial Information Systems (Eds: S.K. Chang, K.S. Fu). New York: Springer-Verlag, pp. 288–321
19. Burgess, C.G. (1984) A database interface aid. In: Proceedings, 1984 IEEE Computer Society Workshop on Visual Languages, Hiroshima, Japan, pp. 72–76, December
20. Herot, C.F. (1981) Spatial management of data. In: Data

Base Management in the 1980's (Eds. J.A. Larson, H.A. Freeman). New York: Institute of Electrical and Electronics Engineers, pp. 132–151

21. Herot, C.F. (1982) Graphical user interfaces. In: Human Factors and Interactive Computer Systems. (Ed. Y. Vassiliou). NYU Symposium on User Interfaces, May 26–28, 1982, Ablex Publishing Corporation, Norwood, NJ, pp. 83–103

22. Query management facility general information (1985) IBM Form No. GC26-4071