The authors proposed a research program to develop a generalized database manager to support diverse kinds of data including text, icons, forms, maps and other spatial data. The proposed research also included investigating support for integrated data browsers to allow end-users to query, step through, and update diverse data. Specific topics to be investigated included query language facilities to support text and geometric data, user-defined abstract data types in a DBMS, an ordered relation access method for the text and other ordered data, extended secondary indexes, main memory databases, concurrency control (cont'd on back).
for data browsers, and an application program interface based on windows.

This paper reports on our progress during the first year of this program. The major advances have been made in the areas of abstract data types, main memory data bases and extended secondary indexes.
REQUEST FOR INSTRUMENTATION

M. Stonebraker
Principle Investigator

September 1, 1983 - August 31, 1984

ELECTRONICS RESEARCH LABORATORY
College of Engineering
University of California, Berkeley
94720
The goals of this project are an adjunct to those of AFOSR-83-0254. Moreover, the proposal was an attachment to the aforementioned grant. Rather than cosmetically alter that report, it has been included verbatim.

ABSTRACT

The authors proposed a research program to develop a generalized database manager to support diverse kinds of data including text, icons, forms, maps and other spatial data. The proposed research also included investigating support for integrated data browsers to allow end-users to query, step through, and update diverse data. Specific topics to be investigated included query language facilities to support text and geometric data, user-defined abstract data types in a DBMS, an ordered relation access method for text and other ordered data, extended secondary indexes, main memory databases, concurrency control for data browsers, and an application program interface based on windows.

RESEARCH RESULTS DURING THE 1983-1984 YEAR

We have made significant progress in our exploration of abstract data types, main memory data bases and extended secondary indexes during the last year. We comment briefly on significant accomplishments in each of these areas in the following subsections.

Abstract Data Types

One of our goals was to investigate ways of representing more complex objects such as geometric objects, text, maps, etc. in a relational data base system. Our major research result involves the possibility of using a query language to represent data types. This research is explored in detail in [STON84], which is included as an appendix to this document. Here we briefly review two previous proposals, then indicate the significance of our current contribution.
Our previous approach to supporting complex objects in a data base system was through abstract data types. We suggested allowing new types of columns to be added to a data base system along with new operators on these columns [STONE83]. For example, a skilled programmer could define polygons as a new data type along with a collection of operators on data of type polygon including an intersection operator (!!). Then, other users could use polygons in the same way they use the built-in types of a data manager (floating point numbers, integers, and character strings). For example, a user could define the following POLYGON relation

create POLYGON (pid = i4, p-desc = polygon)

and then find the polygons overlapping the unit square as follows:

retrieve (POLYGON.all) where POLYGON.p-desc != "0,0,1,1"

Support for user defined types and new operators has been constructed in about 2500 lines of code for the INGRES relational data base system. Implementation details are addressed in [FOGG82, ONG82], and ADTs run with a modest performance degradation [FOGG82]. Initial suggestions concerning how to integrate new operators into query processing heuristics and access methods are contained in [STON83, ONG84].

The abstract data type approach to complex objects is conceptually clean because no facilities peculiar to a specific kind of data are required. However, it has the disadvantage that one cannot easily "open up" an object and examine its component sub-objects. For example, suppose an airplane wing is defined as an abstract data type composed of a collection of components (e.g. cowlings, engines, etc.). In turn, each component could be composed of sub-components. Then, suppose a user wished to isolate a turbine blade inside a specific engine on a wing. To perform this task he would need two operators, one to "open up" the wing and identify a specific engine and a second to "open up" the engine and...
identify the desired turbine blade. Two cascaded operators are an awkward way to search in a complex object for a specific sub-object.

A second approach is to extend a relational data base system with specific facilities for particular complex objects. This is the approach taken in [LOR183] for objects appropriate for CAD applications. It has the advantage that component objects can be addressed but requires special-purpose services from a DBMS.

Our major contribution under the current grant is a third approach which may offer the good features of each of the above proposals. It involves supporting commands in the query language as a data type in a DBMS. In our environment this means that a column of a relation can have values which are one (or more) commands in the data manipulation language QUEL. We explain our proposal using the following example. Suppose a complex object is composed of lines, text and polygons. Each component is described in a separate relation as follows:

\[
\begin{align*}
\text{LINE} & \quad (\text{Lid}, \text{l-desc}) \\
\text{TEXT} & \quad (\text{Tid}, \text{t-desc}) \\
\text{POLYGON} & \quad (\text{Pid}, \text{p-desc})
\end{align*}
\]

Example inserts into the LINE and POLYGON relation are:

\[
\begin{align*}
\text{append to LINE} & \quad (\text{Lid} = 22, \\
& \quad \text{description} = "(0,0) (14.25)"
\end{align*}
\]

\[
\begin{align*}
\text{append to POLYGON} & \quad (\text{Pid} = 44, \\
& \quad \text{description} = "(1,10) (14,22) (6,19) (12,22)"
\end{align*}
\]

Then, the object as a whole would be stored in the OBJECT relation:

\[
\begin{align*}
\text{OBJECT} & \quad (\text{oid}, \text{o-desc})
\end{align*}
\]

The description field in OBJECT would be of type QUEL and contain queries to assemble the pieces of any given object from the other relations. For example, the following insert would make object 6 composed of line 22 and polygon 44.

\[
\begin{align*}
\text{append to OBJECT} & \quad (\text{oid} = 6, \\
& \quad \text{description} = \text{assemble line 22 and polygon 44})
\end{align*}
\]
oid = 6,
  o-desc = "retrieve (LINE.all) where LINE.id = 22
           retrieve (POLYGON.all) where pid = 44")

We have proposed extensions to QUEL which allow the components of an object to be addressed. For example, one could retrieve all the line descriptions making up object 6 which were of length greater than 10 as follows:

range of 0 is OBJECT
retreive of (0.o-desc.1-desc) where
   length (0.o-desc.1-desc)>10

This notation has many points in common with the data manipulation language GEM [ZAN83], and allows one to conveniently discuss subsets of components of complex objects. In addition we can support clean sharing of lines, text and polygons among multiple composite objects by having the same query in the description field of more than one object, a feature lacking in the proposal of [LORI83].

Main Memory Data Bases

We have spent considerable time exploring the use of large amounts of main memory to speed data base processing. In [DEWI84] we have presented our results. The main contributions are:

1) an investigation of the viability of AVL trees in an environment where most of a relation may be present in main memory

2) an investigation of new algorithms for performing relational joins which can effectively utilize large quantities of main memory

3) an investigation of efficient means of obtaining crash recovery for a data base mostly resident in main memory

Our conclusions are somewhat counterintuitive. For example, it is found that merge-sort [SELI79] is rarely effective as a join tactic in an environment
with much main memory because it is outperformed by several variations of hashing joins. A full discussion of this point and others appears in [DEW184], which is also included in the appendix.

Extended Secondary Indices

During the past year we have concentrated on designing a secondary indexing structure that would be appropriate for solid geometric objects such as the polygons and rectangles which appear in CAD applications. Conventional spatial indexing schemes (e.g. KDB trees [ROB181] are only appropriate for point data. Our scheme, R-trees, allows efficient access to solid spatial objects according to their location [GUT84]. Leaf nodes in an R-tree contain index entries, each consisting of a pointer to a spatial object and a rectangle that bounds the object. Higher nodes contain similar entries, with pointers to lower nodes and rectangles bounding the objects in the lower nodes. This hierarchy of covering rectangles is built and maintained dynamically in a manner similar to a B+tree.

To search for all data overlapping a given rectangle, we examine the root node to find which entries have rectangles overlapping the search area. The corresponding subtrees can have data in the search area, and therefore we apply the search algorithm recursively to each one. In this way we find all qualifying data, but avoid searching parts of the tree corresponding to objects that are far from the search area.

R-trees can be built for any number of dimensions. In addition they are useful for overlapping objects of non-zero size, a characteristic not shared by most multi-dimensional indexing schemes, for example quad trees [FINK74], k-d trees [BENT75], and K-D-B trees [ROB181].

We have implemented R-trees, and in spatial search tests using VLSI data, only about 150 usec of CPU time was required to find each qualifying item. This indicates that the structure effectively restricts processing to qualifying or
near-qualifying data. Our paper on R-trees [GUTT84] is included in the appendix to this proposal.

REFERENCES


PROFESSIONAL PERSONNEL

Michael Stonebraker, PI


Antonin Gutman: PhD August 1984, "Application of Relational Data Base Systems to CAD Data"

PUBLICATION CITATIONS


INTERACTIONS

The first three papers in the Publication Citations section were presented at ACM SIGMOD annual conference.

A presentation on "QUEL as a Data Type" was given at Wang Institute and IBM Federal Systems Division.