TIMBER: A SOPHISTICATED RELATION BROWSER

Michael Stonebraker and Joseph Kalash

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
UNIVERSITY OF CALIFORNIA
BERKELEY, CA.

ABSTRACT
This paper discusses the functions present in a sophisticated relation browser with support for icons, maps, text and normal fixed format relations. A discussion of some of the required data base extensions to support such a browser is also presented.

I INTRODUCTION

This paper suggests the design of TIMBER (Text, Icon and Map Browser for Extended Relations), a user friendly, graphics-oriented browser for a relational data base. There are four motivations for TIMBER which will be discussed in turn.

1) Relation Editor

There is a need for a software facility which would allow a user to "browse" through a relation. Commands such as "next tuple" and "search for the first tuple" satisfying a given constraint are required. A proposal along these lines is presented in [CATE80] for a CODASYL data base.

The various proposals for interaction with a data base through forms, e.g. [ROWE82, DEJO80], can be used to implement a relation editor. One would define a default form which can be used with any relation. Tuples from any given relation can then be displayed through the form. However, most forms proposals are limited to a single form on the screen at one time and to a single data base tuple in any form. What we propose is a more sophisticated facility.

2) ICON Display

Both Query By Example [ZLOO75, ZLOO77] and CUPID [MCDO75] were early examples of the application of graphics to naive user interfaces. More recently, SDMS [HERO80] presents a more sophisticated approach. All three systems have the common feature that the representation of either the query or the output data is done through the use of "icons", which are graphical tokens representing data base objects.

Query By Example (QBE) and CUPID allow a user to specify a relational query by using icons. In QBE the icon is a skeleton of a table while in CUPID it is a collection of rectangles, hexagons and circles. To specify a query in QBE one inserts values in these skeletons; in CUPID he connects the rectangles, hexagons and circles together. Both systems then "answer" the user command by displaying a tabular representation of desired data on the screen.

On the other hand, SDMS allows a user a graphical query language who commands are essentially:

move cursor
zoom
unzoom

A user is shown the entire data base represented as icons on multiple graphics terminals. In a reported example [HERO80], one portion of the data base concerned Navy ships and the initial display was an icon for each ship in the data base. A user could then move his cursor to a ship of interest and zoom to find more detail.

The problem with this approach is that only a very weak query language can be provided. For example, suppose a user wants to know whether the Enterprise has more seamen than the Chicago. He must first move to the Enterprise, zoom and then write down the number of seamen from the display. Then he zooms on the Chicago and does the comparison. It is impossible to get detailed information for both ships on the screen at once unless they happen to be next to each other in the initial display.

SDMS is a sophisticated data browser but lacks a sophisticated means to specify non-trivial data base interactions. On the other hand, QBE and CUPID lack a mechanism to browse the result of queries. A basic

motivation of TIMBER is to support a more comprehensive data manipulation facility for icon oriented data bases.

3) Text Editors

Modern visual oriented editors, e.g. VI [JOY 79] or EMACS, support sophisticated visual interaction with the data in a text file. For example, a user is allowed to browse and update his text with comprehensive cursor commands.

It appears to be beneficial to have normal text stored in a data base system. For example, suppose one has a mailing list stored in a relational data manager and has an application which requires sending the same letter to everybody on the mailing list. Basically, one would like to "join" the letter to the mailing list. If the letter is stored in the data base, such a join is possible.

A basic goal of TIMBER is the support of visual oriented editing of text stored as data base objects.

4) Geographic Data

In urban planning applications there is a need for display of geographic data. Moreover, several geo-data systems have been implemented, some on top of relational data base managers, e.g. [GO75, MANT73]. More recently, computer aided design (CAD) and VLSI design systems have been suggested as application areas for data base management. Again, one requires the storage and manipulation of spatial information. Our last goal for TIMBER is to provide support for spatial objects.

In summary, we are proposing a user interface to a relational data base system that can be:

a) a relation browser for fixed format relations
b) a sophisticated browser for relations with icons
c) an editor for text data stored in relations
d) a map browser for geographic data

It is the thesis of this proposal that these four application areas require essentially the same functions and can be served by common software.

Hence, in the next section we turn to the TIMBER user interface and then in Section III to the architecture of this system. Section IV contains some comments on extending a relational data manager such as INGRES [STON76, STON80] to effectively support icon, map and text oriented data. Lastly, Section V contains some conclusions.

II TIMBER

TIMBER has two basic concepts: a window on a sophisticated graphics terminal and a relation in a data base. The screen of the terminal can be split into a collection of rectangular windows and relations "bound" to such windows. In this manner tuples from the relation appear in the window and can be manipulated by TIMBER commands. These commands are presented in an English textual syntax which is solely for ease of reading this presentation. As will be noted in Section III a small application program will control the syntax seen by an end user. Presumably, this will involve function keys, the cursor and menus.

2.1 Windows

Windows can be created, destroyed and expanded as follows:

CREATE WINDOW (X1, Y1, X2, Y2)

This command creates a window which is rectangular shaped with lower left hand corner at (X1, Y1) and upper right corner at (X2, Y2). To achieve some terminal independence, 0 <= X1, Y1, X2, Y2 <= 1. Hence,

CREATE WINDOW (0,0,1.,1.)

would be a window incorporating the whole screen.

DESTROY WINDOW

This command destroys the window in which the cursor is currently positioned.

EXPAND WINDOW (X1, Y1, X2, Y2)

This command changes the window size of the window in which the cursor is currently positioned to the coordinates indicated. There is no restriction that windows cannot overlap; however, TIMBER will not do visually pleasant things for such occurences.

2.2 The Cursor

Another of TIMBER's fundamental concepts is a cursor. It is positioned at a specific location on the screen, and is controlled in our environment by a bit pad and mouse. Many TIMBER commands affect the window in which the cursor is currently positioned.

The windows on the screen are arbitrarily ordered, and the following commands move the cursor to a different window:

NEXT WINDOW
PREVIOUS WINDOW

2.3 Relations

Another fundamental concept in TIMBER is a relation. A relation can be "bound" to a window by the following TIMBER command:

BROWSE REL-NAME

REL-NAME is assumed to be the name of a relation which may be specified in one of three ways:

1) by specifying a QUEL RETRIEVE command whose output is a relation

2) by specifying a relation in a data base

3) by specifying a view on a relation or relations in a data base

This relation is "bound" to the window in which the cursor is currently positioned. Binding entails three operations:

a) the screen format for a tuple must be ascertained. This will determine the visual representation of data on the terminal.

b) a two dimensional geometry must be associated with a relation. This will give any relation an (X,Y) coordinate system and a location to any tuple in the relation. This coordinate system will be called "the relation coordinate system" or more briefly "relation space".

c) the window must be positioned in relation space and its size determined. This will determine which tuples are initially visible on the screen and where they are positioned.

To accomplish a) and b), each relation has a default coordinate system and screen format. TIMBER recognizes four kinds of coordinate systems:

1) normal fixed format relations

Such relations are exactly like existing INGRES relations. Display of such relations on a graphics terminal conforms to the current INGRES display of relations. For example, suppose

EMP(name = c12, salary = i2)

is a relation, then its screen image would be:

```
name        salary
Kalash      10000
Stonebraker 20000
```

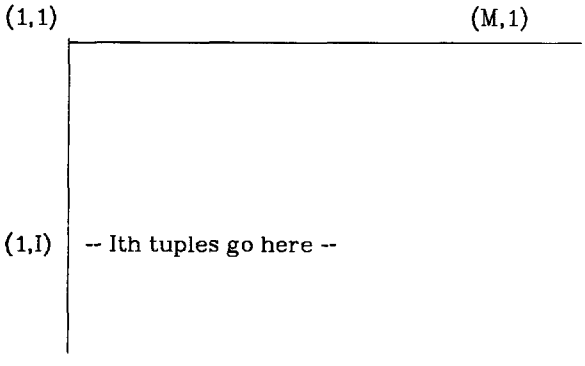
More precisely, the coordinate system for a relation with N tuples, each M bytes wide is a rectangle with corners (1,1), (1,N), (M,1) and (M,N). The Ith tuple occupies locations (1,I) to (M,I) as noted in Figure 1. The relation header giving the names of the columns appears in the 0th row of relation space.

2) A text relation

If a user creates a document, it will typically have the format:

CREATE DOCUMENT (line-id = i4, text = c132)

In Section 4.2 we propose the notion of an ordered relation to support documents. INGRES will automatically



Relation Space for
Fixed Format Relations

Figure 1

maintain line numbers for ordered relations. Hence, TIMBER will assume that any text relation is an ordered relation. It will then suppress line numbers as distracting and display the remainder of the relation as a fixed format object. Hence, the coordinate system for an ordered relation is identical to that of a fixed format relation, except for the suppression of line numbers.

3) relations containing an ICON field

Any relation can have a field whose data type is "icon" In this field a graphical token is stored which is used to assist in the visual representation of the row. Creation of icons for data base objects is discussed in Section 4.1.

For example, suppose an EMP relation is declared as follows:

CREATE EMP (name = c10, salary = i2, visual = icon)

Suppose the graphical token for an employee has been specified as a rectangle. It might be represented as some coding for the following information:

lines:

(0,0) -> (24,0)
(24,0) -> (24,4)
(24,4) -> (0,4)
(0,4) -> (0,0)

text:

"name =" at (1,3)
"salary =" at (2,3)

fields:

name at (1,10)
salary at (2,10)

scale-x = 1/24
scale-y = 1/24

and have a screen image of:

name = Kalash
salary = 10000

All entries above are self explanatory except for the scaling factors. These are used to alter the relative size of the icon when it is placed on the screen.

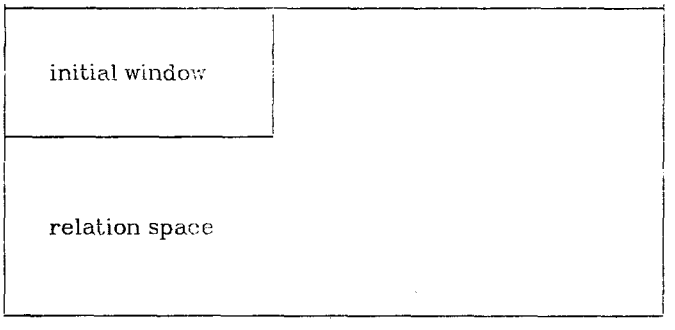
Icon relations are assumed to be stored as two dimensional ordered relations, a concept to be discussed in Section 4.2. Hence, every tuple in this kind of relation has an X-line-id and a Y-line-id. Both of these ids are non negative integers, and the location of any tuple in relation space is this X-Y pair. The icon for each tuple is scaled by the scaling factors noted in the example above and all coordinates in the icon definition are translated by the X-Y location of the tuple. This scaling and translation produces a representation for a tuple which is intended to fit in the square bounded by (X,Y), (X+1,Y), (X,Y+1) and (X+1,Y+1). As long as the scaling factors are chosen wisely, the icon for a given tuple will not overlap the icons for any of its neighbors.

4) Map Relations

Any relation can have a field who data type is "map". In this field is stored a coding of a geographic object. A "map" data type differs from an "icon" data type in only one way; icons are translated to the coordinates of a tuple whereas all map components are relative to (0,0) and no translation is performed.

Once the coordinate system for a relation has been ascertained, task c) above can be accomplished by placing the window in the upper left hand corner of relation space as indicated in Figure 2. For fixed format and text relations, the initial zoom is set so as to put the first 80 columns of each of the first 24 tuples on the initial display. For icon and map relations, the initial window will contain all objects within an arbitrarily set square with boundaries of (0,0), (10,0), (0,10), and (10,10).

2.4 Screen Manipulation



The Initial Window for
Brouwed Relations

Figure 2

There are three ways to alter the contents of the screen.

1) by moving the cursor

Cursor motion is supported in a window. If the cursor moves off the edge of a window, the window is automatically repositioned in relation space. Intelligent buffering, discussed in Section III, attempts to make this movement possible with good response time.

2) by zooming

The user can alter the size of the display by a zoom as follows:

ZOOM (X-factor, Y-factor)

This command zooms the current window by the X and Y factors indicated. If these factors are less than one, the display is "shrunk" and if more than one it is "blown up".

3) by relational commands affecting the window

Initially TIMBER has six relational commands as follows:

a) NARROW domain-list

This command performs a projection on the relation in the current window preserving only the named fields in the domain list.

b) RESTRICT expression

This command removes the icons or the rows which do not satisfy the expression. This is a relational restriction operation as in [CODD72]. We expect to allow the following possible expressions:

i) string-constant

A row would be removed unless it had some field which contained the indicated string. For example,

RESTRICT se

would remove all rows which failed to contain the string "se".

ii) domain-name rel-op value

A row would be removed unless the specified domain name had an appropriate value. Rel-op is one of {<, <=, =, !=, >=, >}. For example

RESTRICT salary > 20000

would remove all low paid employees. Of course, we would allow any boolean combination of such clauses.

c) SEARCH DIRECTION expression

This command would search in the given direction for the first icon or row which satisfied the expression. It would then position the window over the desired object. Direction can be U (up), D (down), R (right) or L (left).

It is possible to pick up some of the data from one window and move them to another window. This is accomplished as follows.

d) PICK expression

This command will cause all tuples in the current window which satisfy the indicated expression to be "picked". These tuples are now associated with the cursor and are no longer in the window. If no expression is indicated, the icon or tuple at the current cursor position is picked.

The user can now move the cursor to another spot on the screen in the same or a different window and then drop anything picked up as follows.

e) DROP

This command will drop all picked objects at the location on the screen where the cursor is now positioned.

A user is expected to create "scratch" windows. Then he can pick various objects of interest and move them to a scratch window for detailed study. However a user may sometimes wish to make a copy of data in his scratch window. If a copy of objects is desired the user can do so as follows:

f) COPY expression

COPY is a PICK without deleting the picked objects from the window.

2.5 Data Update

When the cursor is positioned in a field of a relation, the user can modify the value of this field using the following commands. The scope of the command is either the whole field for numeric data or the character string between the first blank or punctuation mark in both directions from the cursor for ASCII fields.

R string

This will replace a numeric field or portion of a character field with the string indicated.

D

This will delete the string at the current cursor position

DD

This will delete the whole tuple at the current cursor position

I string

This will insert the indicated string into the field at the current cursor position

A tuple

This command will insert a new tuple into the browsed relation. It should be noted that the new tuple may not be placed in relation space at the current position of the cursor. For example, if the EMP relation is stored indexed sequential on employee name and a new

employee "aardvark" is inserted, he will be placed at the top of relation space and not at the position of the cursor.

X

This will delete the specific character that the cursor is currently pointing to

C character

This will change the character at the current cursor position to the indicated one.

Using these commands a user can browse through the relation associated with a window and make random updates, inserts and deletes. A commit mode is required to make these changes appear in the data base:

COMMIT commit-mode

When a collection of changes are made to the data in a window, a decision must be made concerning when to update the actual data base. Most text editors require specific user commands to write the contents of the editor workspace back to the file in question. TIMBER requires the same sort of advice, as follows:

commit-mode = now

Every time a user changes a data item, the change is immediately spooled to a data base update process and the change is accomplished. No further screen manipulation is possible until the update is installed in the data base. In order to allow a user to continue browsing while the data base update is occuring, the second commit mode is:

commit-mode = at-your-convenience

TIMBER spools all changes to a data base update process. This process may lag the user by an indeterminate amount.

commit-mode = end-of-session

TIMBER saves all changes, and at the end of a session it commits all updates. This is similar to the mechanism employed by many text editors.

Of course, the commit mode can be changed at any time during a TIMBER session.

2.6 Miscellaneous Commands

Windows can be saved and cleared using the following commands:

SAVE in REL-NAME

This command will save the contents of the current window in a new relation called REL-NAME. An error will result if the user has changed the data in such a way that the window no longer corresponds to a relation. This could happen, for example, if he picked and dropped icons with different data into the same window.

CLEAR

This will cause the current window to be cleared. This is the opposite of the BROWSE command as it disassociates the window with a relation. This is considered the end of the session with respect to commiting updates to that relation. The user can now BROWSE through another relation in the same window.

III ARCHITECTURE OF TIMBER

TIMBER is composed of six major modules as indicated in the diagram below. The INGRES DBMS has been extensively described elsewhere [STON76, STON80]. Hence, in the next four subsections we discuss the other pieces of TIMBER. The CRT in our environment is an AED 512 color graphics terminal with Barco monitor.

3.1 Low Graphics

This module is a standard low level drawing package. It contains routines to draw lines, points, boxes, print text and move the cursor. This level of the system knows only about the display of objects at absolute locations on the screen. All terminal dependence is isolated at this level.

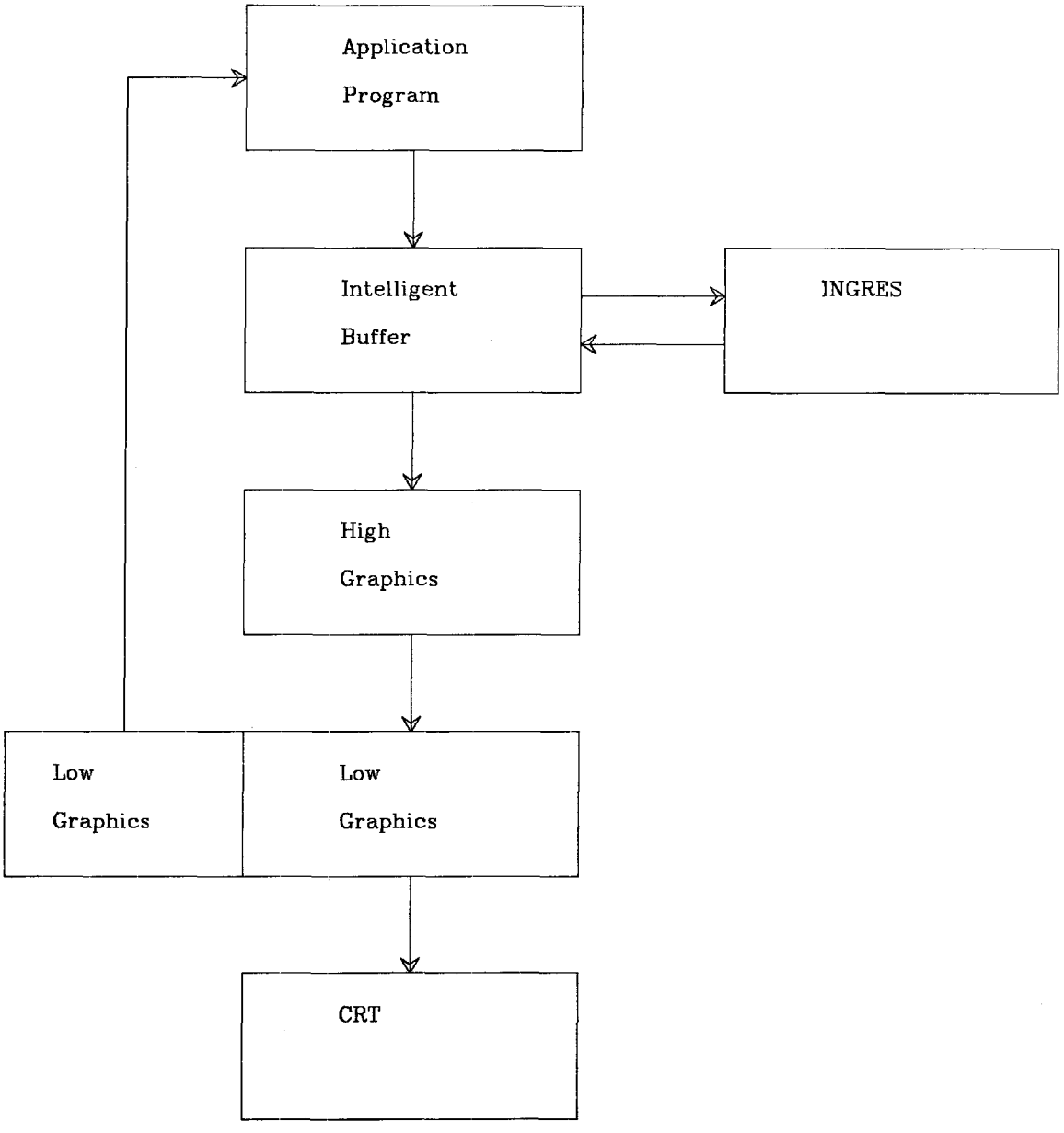
3.2 High Graphics

This module is a higher level graphics package, whose major concepts are windows and objects. An object is a graphical representation for a tuple in a relation. The co-ordinates of each object indicate its position relative to the upper left hand corner of the window. High Graphics contains a collection of routines which:

a) display a list of objects in a window
b) erase an object
c) clear a window
d) move an object
e) update an object

3.3 Intelligent Buffer

This level attempts to provide an intelligent cache for the INGRES relations that are being browsed. On a per window basis, it is responsible for fetching tuples and transforming them to object representation. When a window is moved or zoomed, this module is responsible for finding any new objects which might now fit in the


Application
Program
Intelligent
Buffer
INGRES
High
Graphics
Low
Graphics
Low
Graphics
CRT

window and passing them to High Graphics. These objects are either in the cache, or the INGRES data base must be accessed to obtain them.

This module is the only place where the relation coordinate system must be understood. The commands from the application program to the intelligent buffer are essentially the commands described in Section II.

3.5 Application Program

This layer supports the actual syntax of user interaction with the screen. It corresponds to the "terminal monitor" of INGRES [STON76] and the "User Friendly Interface" for System R [ASTR76]. It is expected that several terminal monitors might be constructed for TIMBER with various syntactic conventions.

IV INGRES EXTENSIONS

In order to support TIMBER, several extensions to a relational data base manager appear desirable. We discuss three in turn below.

4.1 Icons as a Data Type

It is evident that icons must be added as a new INGRES data type. The sequence of (x,y) pairs which describe the icon must be stored. In addition, there must be a "slot" for each field in the relation which can hold a data value and optionally a label. Although icons are stored initially as character strings, a more elegant solution solution would be desirable.

In [STON82] we suggested a particular use of abstract data types which allow user defined types for the columns of a relation and new data base operations on these types. This proposal is a generalization of the notion of data base experts [STON80a]. Storage of icons appears an ideal use of this facility. Moreover, functions, such as translating or rotating an icon, can be implemented as new QUEL operators.

4.2 Ordered relations

Storage of documents fundamentally requires the notion of an ordered collection of records (i.e. lines of text). Without additional mechanisms a relational data base system which stores unordered tuples has difficulty with documents. We propose to extend INGRES with the notion of an ordered relation. An unordered relation R(A, B, C) can be ordered by a utility as follows:

```
ORDER R
```

An ordered relation contains an extra field called a line identifier (LID), which is a positive integer. Moreover, there exists a sort order of the tuples so that LID's are in ascending sequence. The LID field can be manipulated in QUEL just like an ordinary field with a few changes in semantics.

For an ordered relation

```
DOCUMENT (LID, TEXT)
```

and a command:

```
APPEND TO DOCUMENT (LID = 100,
        TEXT = "a new line to be inserted")
```

the LID of this tuple becomes 100 and all lines with LID greater than or equal to 100 get incremented by 1. On a delete command,

```
RANGE OF D IS DOCUMENT
DELETE D WHERE D.LID =100
```

then the appropriate tuple is removed and all LID's greater than the LID of the deleted tuple are decremented. On a replace command such as

```
REPLACE D(LID = 50) WHERE D.LID = 100
```

then the semantics are that of a delete followed by an append. On the other hand, if the new LID is greater than the old one, the append must precede the delete.

A command such as

```
RETRIEVE INTO TEMP (D.LID, D.TEXT) WHERE D.TEXT =
"*the*"
```

has the effect of creating a TEMP with all lines containing the string "the". The TEMP formed is an ordinary relation and the LID is a normal data item. As such it does not have the sequencing property of ordered relations. To create an ordered relation, one must:

```
RETRIEVE INTO TEMP (D.TEXT) WHERE D.TEXT = "*the*"
ORDER TEXT
```

Ordered relations can have secondary indexes in the conventional way. That is, every tuple in any relation has a tuple identifier (TID) which is an indirect pointer to its storage address. A secondary index in INGRES for R(A, B,

C) is specified as follows:

INDEX ON R IS INDEX-NAME(B)

This has the effect of creating a relation INDEX-NAME as follows:

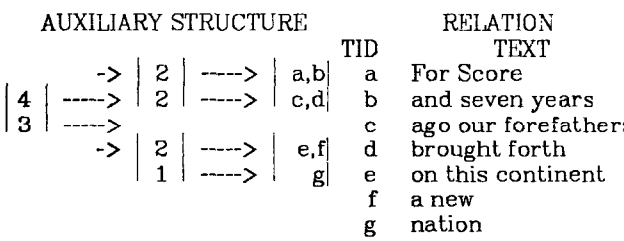
INDEX-NAME (B, TID-of-R)

We now discuss the relationship between TID's and LID's. The basic function of TID's is to facilitate secondary indexes and the deferred update performed to allow crash recovery. As such, it is important that TID's change as infrequently as possible. On the other hand, LID's serve to generate an ordering of a relation and thus change frequently.

It would be possible to use a TID to support ordering. However, it is difficult to to find tuple number 2700 without a sequential scan of the 2699 preceding tuples. Consequently, both LID's and TID's appear necessary.

We propose to support ordered relations as follows. When any relation is ordered, a special access structure is constructed. This structure is similar to a B+ Tree [COME79] whose leaf pages contain TIDs for tuples in the relation being ordered. LIDs are assigned to tuples in ascending sequence as leaf pages are passed left to right. Hence, the leftmost TID on the leftmost leaf page has a LID of 1, its neighbor to the right an LID of 2, etc. Each index block in this access structure stores a collection of pairs <page pointer, N> where N is the number of TIDs in the subtree pointed to by the page pointer, as noted in Figure 3.

To support the correct semantics for LIDs, the insertion of a new tuple into an ordered relation causes an insert of its TID into the new access structure at its correct position in the LID ordering. All values for N on the path to the root must be incremented. A page split via conventional B+ tree techniques is performed as necessary. The delete algorithm is analogous.



The Structure To Support LID's
Figure 3

Ordered relations and the above auxiliary structure provide a one dimensional ordering for a relation. In [STON82a] a generalization of this structure is suggested which supports multiple sequencing fields for a tuple. In this way a tuple can have several LIDs and a resulting multiple dimension ordering.

4.3 Concurrency Control

The intelligent buffer (IB) may want to have a large part of a relation already converted to object representation in virtual memory. In this way cursor motion can be supported with minimal response time. However, potentially any object in IB can be updated; consequently all tuples in IB must be locked. The net effect will be that TIMBER will have potentially large portions of relations locked and may hold such locks for the duration of a browsing session.

One solution to a similar situation is proposed in [BROW81] for the Cedar DBMS. One writer and several readers for an object are allowed. When an update to an object occurs, the data manager broadcasts a "dirty data" message to any process that has read a particular object. Each reader must either release the read lock on the object or abort. In a browsing environment, however, it is difficult to tell under what circumstances such a read lock can be released. If an object has appeared on the screen, a browsing user may have noted its value and be using this value in a subsequent action. As a result, an abort of the transaction may be inevitable.

In a separate paper we propose a new kind of user interface to a relational data manager which will allow multiple concurrent browsers which perform updates [STON82b]. This proposal may also allow some of the buffering needed by TIMBER to be done inside the data manager.

V CONCLUSIONS

We have presented the specification and overall architecture for TIMBER in this paper. It is intended as a sophisticated two-dimensional graphical browser for text relations, fixed format relations, and relations containing icons or maps.

At this time (February 1982) substantial portions of Low Graphics and High Graphics are operational. The intelligent buffer has not been addressed, so a "hollow shell" which does no buffering at all is in place. Most of the functions specified for fixed format relations and text relations are operational. We expect to have a com-

plete system for internal experimentation within six months.

REFERENCES

[ASTR76] Astrahan, M. et. al., "System R: A relational Approach to Data," TODS, June 1976.

[BROW81] Brown, M., et. al., "The Cedar DBMS: A Preliminary Report," Proc. 1981 ACM-SIGMOD Conference on Management of Data, Ann Arbor, Mich., May 1981.

[CATE80] Catell, R., "An Entity-based Database User Interface," Proc. 1980 ACM-SIGMOD Conference on management of Data, Santa Monica, Ca., May 1980.

[CODD72] Codd, E., "Relational Completeness of Data Sublanguages," Courant Computer Science Coloquium, New York, 1972.

[COME79] Comer, D., "The Ubiquitous B-Tree," Computing Surveys, June 1979.

[DEJO80] de Jong, P. and Byrd, R., "Intelligent Forms Creation in the System For Business Automation," IBM Research, Yorktown Heights, N.Y., RC 8529, October, 1980.

[GO75] Go, A. et. al., "GEOQUEL: A Relational Geo-data System," Proc. ACM SIGGRAPH-SIGMOD Workshop on Application of Data Management to Graphics, Waterloo Ontario, September 1975.

[HERO80] Herot, C., "Spatial Management of Data," TODS, December 1980.

[JOY79] Joy, W., "The Text Editor, VI," (unpublished working paper)

[MANT73] Mantey, P. et. al., "Information for Problem Solving: The Development of an Interactive Geographic Information System," IEEE Conference on Communications, Seattle, Wash., June 1973.

[MCDO75] McDonald, N. and Stonebraker, M., "CUPID: A User Friendly Graphics Query Language," Proc. 1975 ACM-PACIFIC, San Francisco, Ca., April 1975.

[ROWE82] Rowe, L. and Schoens, K., "A Forms Application Development System," Proc. 1982 ACM-SIGMOD Conference on management of Data, Orlando, Fla., June 1982.

[STON76] Stonebraker, M., "The Design and Implementation of INGRES," TODS, Sept. 1976.

[STON80] Stonebraker, M., "Retrospection on a Data Base System," TODS, June 1980.

[STON80a] Stonebraker, M. and Keller, K., "Embedding Expert Knowledge and Hypothetical Data Bases in a Data Base System," ACM-SIGMOD Conference on Management of Data, Santa Monica, Ca., May 1980.

[STON82] Stonebraker, M., "Application of AI Techniques to Data Base Systems," Proc. NSF Workshop on Data Semantics, Bretton Woods, N.H., June 1982.

[STON82a] Stonebraker, M., et. al., "Document Processing in a Relational Data Base System," University of California, Electronics Research Laboratory, Memo ERL M82/24, April 1982.

[STON82b] Stonebraker, M., et. al., "Data Base Windows: A New Application Program Interface," (in preparation).

[ZLOO75] Zloof, M., "Query By Example," Proc. 1975 National Computer Conference, Anaheim, Ca., June 1975.

[ZLOO77] Zloof, M., "Query By Example: A Data Base Language," IBM Systems Journal, December 1977.