AN APPROACH TO IMPLEMENTING A GEO-DATA SYSTEM

by

A. Go. M. Stonebraker and C. Williams

Memorandum No. ERL-M529

27 June 1975

AN APPROACH TO IMPLEMENTING A GEO-DATA SYSTEM

by

Angela Go, Michael Stonebraker and Carol Williams

Memorandum No. ERL-M529

27 June 1975

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

AN APPROACH TO IMPLEMENTING A GEO-DATA SYSTEM

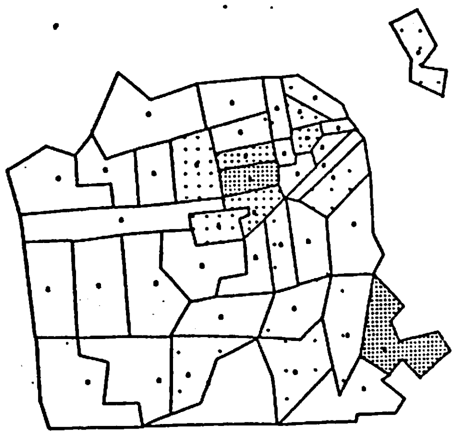
ANGELA GO, MICHAEL STONEBRAKER and CAROL WILLIAMS
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES
AND THE ELECTRONICS RESEARCH LABORATORY
UNIVERSITY OF CALIFORNIA, BERKELEY, CALIFORNIA 94720

ABSTRACT

This paper sketches the implementation approach being taken in building a special purpose geographic information retrieval and display system. The basic mechanism is to construct a special purpose "front end" to the general purpose relational data base management system, INGRES, now operational at Berkeley. It is indicated that this extra software is not difficult to construct and examples of the utility of this approach are given.

I INTRODUCTION

The capability for graphical display of data (especially data that has a spatial character) has been recognized as a valuable component of information retrieval systems. Clearly, Figure 1 which displays the distribution of non-white families earning less than $2,900 for San Francisco County, California by zones defined by the Metropolitan Transit Commission (MTC zones) is more meaningful than a table of numbers to an urban analyst.



Distribution of poor non-white families

Figure 1

Applications which could utilize a graphical facility include those interrogating spatial demand data (e.g. requests for police services, fire department services, etc.), network or grid data (e.g. automobile flows on streets or highways, water and solid waste distribution) and data involving partitions of a geographic area (e.g. census tracts, school districts, election districts etc.).

Several such geodata systems currently exist including BUDS [1], GADS [2,3], NISP [4] and FIRS [5]. All are special purpose information retrieval and display facilities requiring their own query language and internal data structures for data.

In this paper we sketch the implementation of GEO-QUEL which offers comparable facilities to the above systems. GEO-QUEL differs dramatically in its implementation approach as it is constructed on top of a powerful relational data base management system. This approach has the following features:

Research sponsored by the National Science Foundation Grant DCR75-03839 and the U.S. Army Research Office Durham Grant DAHC04-74-G0087.

1) All geographic data (including maps) are treated as relations. Hence, the full power of the relational data sublanguage is available to manipulate geographic data.

2) Because of 1), most features needed in a geo-data system can be implemented easily. Very little special code is required as most tasks can be turned into interactions in the data sublanguage.

The suggestion of using relations for graphics data structures appears in [6]. Here, we indicate the utility of a general purpose relational data base management system for graphics applications.

In order to indicate our implementation, we first discuss the relational data base system INGRES and its data sublanguage QUEL on top of which GEO-QUEL is being implemented.

II. QUEL

QUEL (QUEry Language) has points in common with Data Language/ALPHA [7], SQUARE [8] and SEQUEL [9] in that it is a complete [10] query language which frees the programmer from concern for how data structures are implemented and what algorithms are operating on stored data. As such it facilitates a considerable degree of data independence [11]. Since basic entities in QUEL are relations, we define them and indicate the sample relation which is used in the examples in this section.

Given sets D1,...,DN (not necessarily distinct) a RELATION R(D1,...,DN) is a subset of the cartesian product D1x...xDN. In other words, R is a collection of N-tuples X = (X1,...,XN) where Xi is an element of Di for i in (1,...,N). The sets D1,...,DN are called DOMAINS of R and R has degree N. The only restriction put on relations in QUEL is that they be normalized [12]. Hence, every domain must be SIMPLE, i.e. it cannot have members which are themselves relations.

Clearly, R can be thought of as a table with elements of R appearing as rows and with columns labeled by domain names as illustrated by the following example.

	NAME	DEPT	SALARY	MANAGER	AGE
	Smith	toy	10000	Jones	25
EMPLOYEE	Jones	toy	10000	Johnson	32
	Adams	candy	12000	Baker	36
	Johnson	toy	14000	Harding	29
	Baker	admin	20000	Harding	47
	Harding	admin	40000	none	58

The above indicates an EMPLOYEE relation with domains NAME, DEPT, SALARY, MANAGER and AGE. Each employee has a manager (except for Harding who is presumably the company president), a salary, an age, and is in a department.

Each column in a tabular representation for R can be thought of as a function mapping R into Di. These functions will be called ATTRIBUTES. An attribute will not be separately designated but will be identified by the domain defining it.

A QUEL interaction includes at least one RANGE statement of the form:

```
RANGE of variable-list IS relation-name
```

The symbols declared in the range statement are variables which will be used as arguments for tuples. These are called TUPLE VARIABLES. The purpose of this statement is to specify the relation over which each variable ranges.

Moreover, an interaction includes one or more statements of the form:

```
Command Result-name (Target-list)
        WHERE Qualification
```

Here, Command is either RETRIEVE, APPEND, REPLACE, or DELETE. For RETRIEVE and APPEND, Result-name is the name of the relation which qualifying tuples will be retrieved into or appended to. For REPLACE and DELETE, Result-name is the name of a tuple variable which, through the qualification, identifies tuples to be modified or deleted. The Target-list is a list of the form:

```
Result-domain = Function, ...
```

Here, the Result-domains are domain names in the result relation which are to be assigned the value of the corresponding function. As a shorthand, the 'Result-domain =' may be omitted if the function is a simple attribute (i.e. NAME = E.NAME may be written as E.NAME - see example 2.6).

The following suggest valid QUEL interactions. A complete description of the language is presented in [13,14].

Example 2.1. Find the birth date of employee Jones

```
RANGE OF E IS EMPLOYEE
RETRIEVE INTO W(BDATE = 1975 - E.AGE)
WHERE E.NAME = "Jones"
```

Here, E is a tuple variable which ranges over the EMPLOYEE relation and all tuples in that relation are found which satisfy the qualification E.NAME = 'Jones'. The result of the query is a new relation, W, which has a single attribute, BDATE, that has been calculated for each qualifying tuple. If the result relation is omitted, qualifying tuples are printed on the user's terminal or returned to the user's program.

Example 2.2. Insert the tuple (Jackson,candy,13000,Baker,30) into EMPLOYEE.

```
APPEND TO EMPLOYEE(NAME = "Jackson", DEPT = "candy",
                   SALARY = 13000, MGR = "Baker", AGE = 30)
```

Here, the result relation EMPLOYEE is formed by adding the indicated tuple to the EMPLOYEE relation.

Example 2.3. Delete the information about employee Jackson.

```
RANGE OF E IS EMPLOYEE
DELETE E WHERE E.NAME = "Jackson"
```

Here, the tuples corresponding to all employees named Jackson are deleted from EMPLOYEE.

Example 2.4. Give a 10 percent raise to Jones

```
RANGE OF E IS EMPLOYEE
REPLACE E(SALARY BY 1.1 * E.SALARY)
where E.NAME = "Jones"
```

Here, E.SALARY is to be replaced by 1.1*E.SALARY for those tuples in EMPLOYEE where E.NAME = "Jones". (Note that the keywords IS and BY may be used interchangeably with '=' in any QUEL statement.)

Also, QUEL contains aggregation operators including COUNT, COUNT', SUM, SUM', AVG, AVG', MAX, MIN, and the set operators, SET and SET'. Two examples of the use of aggregation follow.

Example 2.5. Replace the salary of all toy department employees by the average toy department salary.

```
RANGE OF E IS EMPLOYEE
REPLACE E(SALARY BY AVG'(E.SALARY WHERE E.DEPT = "toy"))
        WHERE E.DEPT = "toy"
```

Here, AVG' is to be taken of the salary domain for those tuples satisfying the qualification E.DEPT = "toy". Note that AVG'(E.SALARY WHERE E.DEPT= "toy") is scalar valued and consequently will be called an AGGREGATE. For the example chosen this aggregate has the value (1/3)*(10000+10000+14000) which equals 11,333. It is sometimes useful to allow aggregates to be taken in such a way that duplicate tuples are not included. Non primed aggregates (SET, AVG, COUNT, and SUM) perform this function. For example, AVG(E.SALARY WHERE E.DEPT = "toy") has a value 12,000.

More general aggregations are possible as suggested by the following example.

Example 2.6. Find those departments whose average salary exceeds the company wide average salary, both averages to be taken only for those employees whose salary exceeds $10000.

```
RANGE OF E IS EMPLOYEE
RETRIEVE INTO HIGHPAY(E.DEPT)
WHERE AVG'(E.SALARY BY E.DEPT WHERE E.SALARY > 10000)

                     >
      AVG'(E.SALARY WHERE E.SALARY > 10000)
```

Here, AVG'(E.SALARY BY E.DEPT WHERE E.SALARY>10000) is an AGGREGATE FUNCTION and takes a value for each value of E.DEPT. This value is the aggregate AVG'(E.SALARY WHERE E.SALARY 10000 AND E.DEPT = value) as indicated below.

E.DEPT	AVG'(E.SALARY BY E.DEPT WHERE E.SALARY>10000)
toy	14,000
candy	12,000
admin	30,000

The qualification expression for the statement is then true for departments for which this aggregate function exceeds the aggregate AVG'(E.SALARY WHERE E.SALARY>1000). The latter is simply the scalar 21,500. Hence, admin is the only qualifying department.

As with aggregates, aggregate functions can have duplicates deleted with an unprimed operator.

The implementation of relations and the mechanism of decomposing interactions in QUEL are discussed in [13,15] and are beyond the scope of this paper.

III MAPS

A basic entity in GEO-QUEL is a map. Intuitively, a map is a collection of:

points	{(x,y)}
lines	{(x1,y1,x2,y2)}
line groups	
zones	

Points and lines are the obvious spatial construct. A line group is simply a collection of lines. A special case of a line group that will often be useful is a polygon. A POLYGON is a line group (of say n lines) such that:

a) each of the 2n points of the polygon belongs to exactly 2 lines

b) no two lines cross

Lastly, a zone is a collection of one or more polygons.

Line groups are required so that structures such as transportation networks can be represented. Polygons are required for partitions of a geographic area and zones are required so that disjoint collections of polygons can be considered as one entity. (Such an entity might be New York City which is composed of several polygons.)

As mentioned in a previous section a map is a relation with a given relation name and containing the following domains: X1, Y1, X2, Y2, PLZTYPE, GRAPHCHAR, INTENSITY, GROUPID, ZONEID, AOD. AOD is a shorthand for any other domains that may be present, so a map is simply a relation of the form

```
RELNAME(X1, Y1, X2, Y2, PLZTYPE, GRAPHCHAR, INTENSITY,
        GROUPID, ZONEID, ... other domains ...)
```

Here:

X1,Y1,X2,Y2 - are coordinates of a point or line.
X1 = X2, Y1 = Y2 for a point.

PLZTYPE - indicates whether the tuple corresponds to a point, a single line or a line segment in a line group or zone.

GRAPHCHAR - applies to tuples of type point only. If not null the point will be represented by the character in GRAPHCHAR.

INTENSITY - indicates the display screen intensity of the tuple.

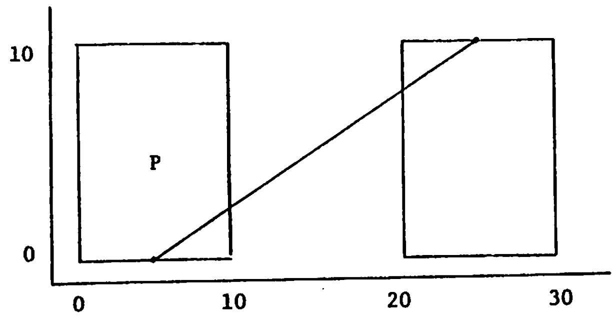
GROUPID - indicates what line group this tuple belongs to.
Points and lines are degenerate line groups.

ZONEID - indicates what zone the current tuple belongs to.

As far as QUEL is concerned, a map relation is indistinguishable from any other kind of relation. Moreover, there can be an arbitrary number of such maps.

It is expected that the data base administrator will "set up" in the proper format all maps which are available. Users can then form other maps from the base set by using the query language.

An example of a simple map now follows:



X1	Y1	X2	Y2	PLZTYPE	GRAPHCHAR	INTENSITY	GROUPID	ZONEID
0	0	0	10	zone		7	2	1
0	0	10	0	zone		7	2	1
0	10	10	10	zone		7	2	1
10	0	10	10	zone		7	2	1
20	0	30	0	zone		5	1	1
20	0	20	10	zone		5	1	1
30	0	30	10	zone		5	1	1
20	10	30	10	zone		5	1	1
5	0	25	10	line		3	1	
5	5	5	5	point	P	1	1	

It might be noted that a certain amount of duplication is present in the above relation. This is motivated by the desire to have map relations that are normalized (i.e. whose domains are not themselves relations) and by a desire to store a map as a single relation so that it can be easily manipulated in ways to be presently discussed.

Clearly note, however, that our data base system, INGRES, can compress relations if desired to achieve economy of storage [15]. Maps may be likely candidates for storage by the access method that accomplishes this.

Besides an arbitrary number of map relations there are three special relations which graphics software manipulates. These are now discussed.

There exists a relation MAPRELATION which contains one tuple for each map in the data base. This relation contains the following domains:

```
MAPRELATION (MRELNAME , MRELOWNER , XCENTER , YCENTER ,
             XMAG , YMAG , SHADEK )
```

Here:

MRELNAME - map relation name

MRELOWNER - owner of this map (different users may have different maps with the same name).

XCENTER, YCENTER - x, y coordinates of the map which corresponded to the center of the screen on the graphics terminal the last time the map was displayed.

XMAG, YMAG - the x and y limits that were used to scale the map the last time it was displayed.

SHADEK - the default distance k between shading characters

The third, fourth, fifth and sixth domains contain the center and magnitude of the map the last time it was displayed on the screen. The final domain contains information only relevant to shaded maps and its use will be presently explained.

The relation DSPTEMP is the second special relation that is now discussed.

This relation corresponds to the map that is presently on the screen of the graphics terminal (in our case a Digital Equipment Corporation GT/42). Its owner is the device on which it is displayed and its form is:

```
DSPTEMP ( DSPX1 , DSPY1 , DSPX2 , DSPY2 , DSPPLZTYPE , DSPINTEN , MUID)
```

All domains are similar to those described for RELNAME except MUID. MUID can either be translated directly to a graph character for 0 < = MUID < = 255, or else it is the ID of a message string that can be found in the relation MENU.

Relation MENU is of the form:

MENU (MUID , MURELNAME , MUOWNER , MUSTRING)

MURELNAME is the relation name of the relation that 'owns' this message string.
MUOWNER is the owner of relation MURELNAME.

Basically MENU stores strings, MUSTRING, that are appended to most displayed maps (for example, the coordinates of the center of the map and the current x and y magnitudes). Text strings of more than one character in length are segregated in a separate relation so they can be properly displayed.

IV GRAPHICS COMMANDS

The set of graphics commands being implemented is the following.

DISPLAY

The command DISPLAY processes DSPTEMP to a form suitable for low level display software which puts it on the screen. This routine contains the only interface to device dependent software. Most other commands alter DSPTEMP then call DISPLAY.

OUTLINE relname

OUTLINE attempts to find and display the physical outline of a map regardless of the values in XCENTER, YCENTER, XMAG and YMAG. This is accomplished in two steps. First, OUTLINE builds the desired figure in DSPTEMP then it calls DISPLAY to put the result on the screen. This command is useful in providing appropriate visual information to center the map as described below.

MAP relname [ON domain]

MAP displays a map with respect to the center and magnitude found in MAPRELATION. If no tuple for this relation exists in MAPRELATION, then the map is scaled to fit entirely on the screen, and an appropriate tuple is inserted in MAPRELATION. If the ON option is used, the value of the domain chosen will be displayed appropriately over the centroid of a zone or on a line group or point. This domain must be in the relation relname. The ON option is useful both to indicate data of a geographic nature and to indicate zone id's when necessary (for example, census tract numbers on a census tract map).

SHADE relname WITH domain = <graphchar or LINE> [SHADE DENSITY=k]

SHADE essentially does a MAP first. Considering only zones completely within the portion of the map which will fit on the screen, characters will be drawn whose density is proportional to the value of the given domain for a polygon. The shade character is specified by graphchar. If the shade density option is omitted, the unit spacing of characters will be adjusted until the maximum number of flicker free characters can be put on the screen. Along with the display appears the value of k used. The user can optionally choose his own value (perhaps so he can compare this map accurately with a previous map). The option LINE uses a similar algorithm to draw diagonal lines through zones whose perpendicular distance is inversely proportional to the value of the domain of interest.

Lines and points are shaded by varying their intensity.

GRAPH relname ON domain WITH {domain = graphchar}
POINTGRAPH relname ON domain WITH {domain = graphchar}

For GRAPH and POINTGRAPH, the first domain specified will serve as the x axis. Each domain thereafter will be a y axis. Each point is first plotted or drawn according to the graph character specified for that domain. The GRAPH command draws line segments connecting points belonging to the same domain.

CENTER relname AT x , y WITH MAGNITUDE xmag, ymag

CENTER changes the XCENTER, YCENTER, XMAG and YMAG of relname in MAPRELATION to the given values.

SAVEMAP relname

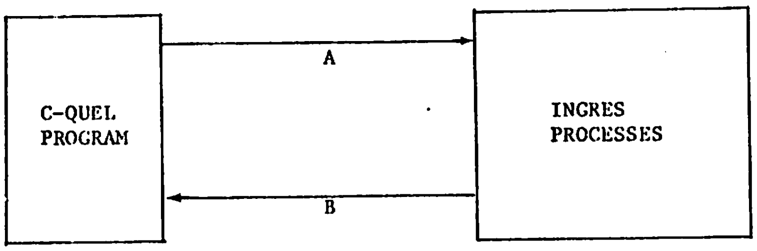
SAVEMAP makes a permanent relation out of the current display relation (DSPTEMP).

V IMPLEMENTATION CONSIDERATIONS

GEO-QUEL consists of QUEL augmented by the commands indicated in the previous section. It is implemented by making use of a macro facility that exists in INGRES and by utilizing a precompiler that embeds QUEL in the general purpose programming language "C" [16]. The macro facility is straightforward and we will indicate its utility presently. First, we show the nature and implementation of the language C-QUEL which is supported by the precompiler.

A C-QUEL program runs as a process on the UNIX operating system [17]. UNIX also supports an interprocess communication facility (pipes) and allows a process to fork subprocesses and execute files in these subprocesses.

The precompiler first inserts the code necessary to fork INGRES (which runs as three UNIX processes and is protected from the C-QUEL program). Figure 2 indicates the process structure that is created.



The Forked Process Structure

Figure 2

Pipe A is the INGRES standard input on which it expects commands; on pipe B error messages and data are returned to the C-QUEL program.

For all INGRES commands that appear in a C-QUEL program (except those RETRIEVE statements in which no result relation is specified), the precompiler simply removes the command from the C-QUEL program and inserts appropriate write commands for pipe A. Then, it inserts a read command for pipe B to obtain completion information. Lastly, if necessary, an error routine is called (either the default system routine or a user supplied one).

For RETRIEVE commands for which no result relation is specified the precompiler must do additional work, namely, it must return tuples to the C-QUEL program one by one through pipe B.

More specifically, a target list for such a RETRIEVE command must have components of the form:

C-variable = QUEL function

Moreover, each such C-variable must have been previously declared. In this case after the operations described in the previous paragraph, the precompiler inserts a read of pipe B to obtain format information for the tuples which follow. It then inserts the necessary control statements to read a single tuple, convert any domains to the types of the declared C-variables and assign appropriate values to these C-variables. Control is then passed to the next block of the C-QUEL program which may operate on these C-variables (but cannot contain other INGRES commands). Lastly, statements are inserted at the end of the block to pass control back to the routine which reads a tuple from pipe B. This mechanism provides the "piped" mode of tuple return mentioned in [7].

The precompiler lastly supports the ability for a programmer to use C-variables in the qualification and the right hand side of a target list expression in any places where a constant would be allowed. This feature is often useful for iteration.

One C-QUEL program supported is an interactive query formulation aid which supports extensive interactive editing of user interactions from a terminal. This interface is described in [18,19]. Other special purpose interfaces being implemented include CUPID [14], a QUEL subset with no tuple variables and an inventory control system front end with special purpose prompts and error messages.

GEO-QUEL is an augmentation of the above terminal interface to support the additional commands of the previous section and is written in C-QUEL. The implementation philosophy is to support the additional features by macro expansion whenever possible. When not possible, C code is used to perform required functions (mostly in the SHADE routine). The following macro expansion of CENTER and POINTGRAPH illustrate this approach.

The command

CENTER relname AT x,y WITH MAGNITUDE xmag, ymag

can be easily expanded to a single QUEL command as follows:

```
RANGE OF M IS MAPRELATION
REPLACE M(XCENTER = x, MYCENTER = y, XMAG = xmag, YMAG = ymag)
    WHERE MRELNAME = relname AND MRELOWNER = this user
```

A more complex set of interactions is required for POINTGRAPH. The expansion shown considers only one YDOMAIN for simplicity. Moreover, the axes are drawn passing through the point (0,0). The reader should note that the QUEL statements involve the special graphics relations considered earlier.

The graphics command
POINTGRAPH relname ON xdomain WITH ydomain = graphchar, expands to the following QUEL statements:
(Here comments are delimited by /* and */)

```
RANGE OF T IS DSPTEMP
RANGE OF U IS MENU
RANGE OF M IS MAPRELATION
RANGE OF R IS RELNAME

/*
** clear DSPTEMP of all tuples
*/
DELETE T

/*
** delete all tuples in relation MENU whose MURELNAME is equal to "DSPTEMP"
** and whose MUOWNER is equal to the id of the present user
*/
DELETE U WHERE U.MURELNAME IS "DSPTEMP" AND U.MUOWNER IS this user.

/*
** set up in DSPTEMP tuples of points with x and y values that
** are the values of the XDOMAIN and YDOMAIN attributes of RELNAME
** and whose graph character is equal to graphchar
*/
APPEND TO DSPTEMP(DSPX1 = R.XDOMAIN, DSPY1 = R.YDOMAIN,
DSPX2 = R.XDOMAIN, DSPY2 = R.YDOMAIN, DSPINTEN = 4,
DSPMUID = graphchar, DSPPLZTYPE = "point")

/*
** add a tuple to the relation whose x and y values are zero
** so there is a point at the origin
*/
APPEND TO DSPTEMP(DSPX1 = 0.0, DSPY1 = 0.0, DSPX2 = 0.0, DSPY2 = 0.0,
DSPINTEN = 4, DSPMUID = 0, DSPPLZTYPE = "point")

/*
** set up a relation that contains all minimum and maximum values
** of the x's and y's
*/

RETRIEVE INTO MINMAX(PXMIN = MIN(T.DSPX1), PYMIN = MIN(T.DSPY1),
PXMAX = MAX(T.DSPX1), PYMAX = MAX(T.DSPY1))

/*
** set up tuples for the x and y axes
*/
RANGE OF MNX IS MINMAX
APPEND TO DSPTEMP(DSPX1 = MNX.PXMIN, DSPY1 = 0.0, DSPX2 = MNX.PXMAX,
DSPY2 = 0.0, DSPINTEN = 4, DSPMUID = 0, DSPPLZTYPE = "line")
APPEND TO DSPTEMP(DSPX1 = 0.0, DSPY1 = MNX.PYMIN, DSPX2 = 0.0,
DSPY2 = MNX.PYMAX, DSPINTEN = 4, DSPMUID = 0, DSPPLZTYPE = "line")

/*
** assume that a relation MISC1 exists, which contains ten
** tuples of one domain NUM having values from 1 to 10
*/


/*
** set up all necessary tuples that will aid in the marking and
** labelling of the x and y axes. assume for the moment that
** the axes will be marked off a total of only 10 times
*/
```

```
RETRIEVE INTO MISC2(WIDTH = MNX.PXMAX - MNX.PXMIN,
LENGTH = MNX.PYMAX - MNX.PYMIN, XINC = (MNX.PXMAX -
MNX.PXMIN)/10.0, YINC = (MNX.PYMAX - MNX.PYMIN)/10.0)

/*
** set up relations XINCREL and YINCREL each containing tuples
** which are incremental values for the X and Y axes respectively
*/
RANGE OF M1 IS MISC1
RANGE OF M2 IS MISC2
RETRIEVE INTO XINCREL(XINC = M1.NUM * M2.XINC) WHERE
M1.NUM * M2.XINC < MNX.PXMAX
APPEND TO XINCREL(XINC = M1.NUM * -M2.XINC) WHERE
M1.NUM * -M2.XINC > MNX.PXMIN
RETRIEVE INTO YINCREL(YINC = M1.NUM * M2.YINC) WHERE
M1.NUM * M2.YINC < MNX.PYMAX
APPEND TO YINCREL(YINC = M1.NUM * -M2.YINC) WHERE
M1.NUM * -M2.YINC > MNX.PYMIN

/*
** set up in DSPTEMP tuples for the marks on the X and Y axes
*/
RANGE OF X IS XINCREL
RANGE OF Y IS YINCREL
APPEND TO DSPTEMP(DSPX1 = X.XINC, DSPY1 = 0.0, DSPX2 = X.XINC,
DSPY2 = 0.0, DSPINTEN = 4, DSPMUID = "1", DSPPLZTYPE = "point")
APPEND TO DSPTEMP(DSPX1 = 0.0, DSPY1 = Y.YINC, DSPX2 = 0.0,
DSPY2 = Y.YINC, DSPINTEN = 4, DSPMUID = "-", DYPPLZTYPE = "point")

/*
** set up tuples in MENU and DSPTEMP for the label of each mark
** on the X and Y axes
*/
APPEND TO MENU(MUID = MAX(MUID)+1, MURELNAME = "DSPTEMP", MUOWNER = this user,
MUSTRING = X.XINC)
APPEND TO MENU(MUID = MAX(MUID)+1, MURELNAME = "DSPTEMP", MUOWNER = this user,
MUSTRING = Y.YINC)
APPEND TO DSPTEMP(DSPX1 = X.XINC, DSPY1 = -30.0, DSPX2 = X.XINC,
DSPY2 = -30.0, DSPINTEN = 4, DSPMUID = U.MUID, DSPPLZTYPE = "point")
WHERE U.MURELNAME = "DSPTEMP" AND U.MUOWNER = this user
AND U.MUSTRING = X.XINC
APPEND TO DSPTEMP(DSPX1 = -70.0, DSPY1 = Y.YINC, DSPX2 = -70.0,
DSPY2 = Y.YINC, DSPINTEN = 4, DSPMUID = U.MUID, DSPPLZTYPE = "point")
WHERE U.MURELNAME = "DSPTEMP" AND U.MUOIWNER = this user AND
U.MUSTRING = Y.YINC

/*
** update the tuple for DSPTEMP in MAPRELATION
*/
DELETE M WHERE M.MRELNAME = "DSPTEMP" AND
M.MRELOWNER = this device
APPEND TO MAPRELATION(MRELNAME = "DSPTEMP", MRELOWNER = this device,
XMAG = M2.WIDTH, YMAG = M2.LENGTH, XCENTER = MNX.PXMIN +
M2.WIDTH/2.0, YCENTER = MNX.PYMIN + M2.LENGTH/2.0, SHADEK = default)
```

The above QUEL statements replace more than 500 lines of code in the general purpose programming language "C" [16]. Note that a simple macro facility can expand the POINTGRAPH command to this code. Note furthermore that the axes, crosshatches and labels have been inserted using QUEL to illustrate its power. In reality we are using "C" to do the portions of these chores that are not more easily expressed in QUEL. Not all commands can be completely expanded to QUEL code alone but all can be expanded PRIMARILY to QUEL code. As a result the effort necessary to implement the facility indicated in the previous section is substantially reduced.

There are several other operations which can be easily done by a user with the query language. Eventually, these may also be built into the graphics facility.

- Overlaying of two or more maps(assuming they have all domain names the same):

```
RANGE OF M1 IS MAP1
APPEND TO MAP2(M1.ALL)
```

- Obtaining a subset of a map any of whose data domains satisfies certain qualifications:

```
RANGE OF M IS MAP
```

```
RETRIEVE INTO NEWMAP(M.ALL) WHERE M.datadomain = something

- Simple minded windowing:
RANGE OF M IS MAP
RETRIEVE INTO NEWMAP(M.ALL) WHERE M.X1<upperlimit AND M.X2< upperlimit
      and M.Y1<upperlimit AND M.Y2< upperlimit

- Adding data domains to a map:
RANGE OF M IS MAP
RANGE OF D IS DATA

/*
** Here, we assume that there exists a connecting domain (called "connecting
** id" in each relation which identifies which data items are
** associated with which map entries.
*/
RETRIEVE INTO NEWMAP(M.ALL, D.desired-data) WHERE D.connecting-id
            = M.connecting-id
```

It should be clearly noted that the entire power of QUEL is available to form the DATA relation in which the user is interested. He need not be limited to "precanned" or "pre-extracted" data.

```
- Reversing an overlay:

/*
** assume that MAP is an overlay of a point map and a line map
*/
RANGE OF M IS MAP
DELETE M WHERE M.plztype = "point"
```

VI SUMMARY

This paper has indicated a mechanism for implementation of a geo-data system. Basically, it is a very "thin" layer on top of a general purpose relational data base management system. Everything possible is turned into commands to this general purpose system (primarily by macro expansion). This has the following advantages:

1) A geographic facility can be brought up readily and involves little special purpose code.

2) New capabilities are readily added (or old ones changed).

3) The implementation is straightforward.

A possible disadvantage to this approach is that INGRES may not be able to perform certain operations as fast as they could be done by special purpose code in a lower level language. However, INGRES supports a variety of storage structures for relations and secondary indices [15] for augmented performance. Moreover, extensive effort has gone into (and will continue to go into) the optimization of processing interactions. Hence, the performance penalty (at least for the commands discussed in this paper) should be small. The issue of performance is further discussed in [20] in a more general context.

VII REFERENCES

[1] Macri, P., "BUDS: The Berkeley Urban Data System," Electronics Research Laboratory, University of California, Berkeley, Memoranoum M412, November, 1973.

[2] Christiani, E. J., et al., "An Interactive System for Aiding Evaluation of Local Government Policies," IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-3, no. 2, Mar. 1973, pp. 141-146.

[3] Mantey, P. E., et al., "Information for Problem Solving: The Development of an Interactive Geographic Information System," IEEE Conference on Communication, vol. II, Seattle, Washington, June 1973.

[4] Parker, J. L., "Information Retrieval with Large Scale Geographic Data Bases," Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, Ca., Nov. 1971.

[5] Depta, D. J., and Irwin, G. M., "FIRS II - Design Requirements Component," Weyerhauser Company, Woods Product Information Systems, Tacoma, Wash., March 1974.

[6] Williams, R., "On the Application of Relational Data Structures in Computer Graphics," IBM Research Laboratory, San Jose, Ca., Jan. 1974.

[7] Codd, E., "A Data Base Sublanguage Founded on the Relational Calculus," Proc. 1971 ACM-SIGFIDET

Workshop on Data Description, Access and Control, San Diego, Ca., Nov. 1971.

[8] Boyce, R., et al., "Specifying Queries as Relational Expressions: SQUARE," Proc. ACM SIGPLAN-SIGIR Interface Meeting, Gaithersberg, Md., Nov. 1973.

[9] Chamberlin, D. and Boyce, R., "SEQUEL: A Structured English Query Language," Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1974.

[10] Codd, E., "Relational Completeness of Data Base Sublanguages," Courant Computer Sciences Symposium, New York, N.Y., May, 1971.

[11] Stonebraker, M., "A Functional View of Data Independence," Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1974.

[12] Codd, E., "A Relational Model of Data for Large Shared Data Banks," CACM 16, (June 1970).

[13] Held, G., et al., "INGRES - A Relational Data Base System," Proc. 1975 National Computer Conference, Anaheim, Ca., May 1975.

[14] McDonald, N. and Stonebraker, M., "CUPID - The Friendly Query Language," Electronics Research Laboratory, University of California, Berkeley, Memorandum M487, October, 1974.

[15] Held, G. and Stonebraker, M., "Storage Structures and Access Methods in the Relational Data Base Management System, INGRES," Proc. ACM-PACIFIC-75, San Francisco, Ca., April 1975.

[16] Ritchie, D. M., "C Reference Manual," Bell Telephone Laboratories, Murray Hill, New Jersey, Jan. 1974.

[17] Ritchie, D. M. and Thompson, K., "The UNIX Time Sharing System," CACM, Vol 17, No. 7, pp. 365-375, July, 1974.

[18] Stonebraker, M., "Getting Started in INGRES- A Tutorial," Electronics Research Laboratory, University of California, Berkeley, Memorandum M518, April, 1975.

[19] Zook, W., et al., "INGRES Reference Manual," Electronics Research Laboratory, University of California, Berkeley, Memorandum M519, April, 1975.

[20] Stonebraker, M., "Networks, Hierarchies and Relations in Data Base Management Systems," Proc. ACM-PACIFIC, San Francisco, Ca., April, 1975.