



FLEXI-VIEW: A Multi-Dimensional Data Modeling System

Erik S. Friis
IBM Corporation
IS&SG Sterling Forest
P.O. Box 700
Suffern, N.Y. 10901
(914)-578-3584

Jay A. Goldberg
IBM Corporation
NCMD Branch Office JC8
150 State Street
Rochester, N.Y. 14614
(716)-726-8279

Abstract

This paper addresses the basic components of a system which facilitates the modeling of multi-dimensional data in terms of a relational database table. The name of the system is FLEXI-VIEW (XVU). FLEXI-VIEW is implemented in APL2 and runs on the IBM Virtual Machine/System Product (VM/SP) operating system.

FLEXI-VIEW interfaces, via the auxiliary processors AP126 and AP127, to the Graphical Data Display Manager (GDDM) and the Structured Query Language Data System (SQL/DS).

SQL is the centerpiece of the FLEXI-VIEW architecture. It was chosen in February of 1985 by the American National Standards Institute (ANSI) as the standard language for relational Data Base Management Systems (DBMS). In addition, IBM has chosen SQL as its strategic relational DBMS product.

The Interactive Chart Utility (ICU) is invoked by FLEXI-VIEW to deliver full-screen graphics and business charts. ICU is a component of GDDM which is responsible for creating, managing, and delivering the system's full-screen panels. Each panel is programmed function (PF) key and window driven.

VM/SP, APL2, SQL/DS, and GDDM/ICU are IBM Program Products.

Introduction

The Concept

The concept behind the FLEXI-VIEW system is to enable an individual who works primarily with numbers to define a natural and dynamic working environment. The working environment consists of a set of tools which facilitate the entry, manipulation, and representation of data.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

The Working Environment

During the initial design sessions of FLEXI-VIEW, a great emphasis was placed on *generality* and *user-friendliness*. APL2 was chosen as the language in which the system would be written. The concept of “general arrays” and the interface to SQL/DS were the main criteria upon which the final decision was based. The philosophy was to create a user-friendly system, designed for people who do not know what APL is and do not care. Some of the more elegant features of APL2, namely “event simulation and handling”, allow FLEXI-VIEW to maintain a full-screen user environment, even in the occurrence of an actual system error.

The n-Dimensional Model

Any individual who works with numbers in some organized form has, in general, a set of categories under which his data elements fall. The problem is that almost every individual's needs mandate a unique set of categories and data elements per category.

In order to achieve our first design goal of “generality”, we found it necessary to allow the user to define any categories he wishes to work with. Each category is referred to as a FLEXI-VIEW *label*. A FLEXI-VIEW *file* is defined by a set of “n” labels, with the constraint that “n” is greater than or equal to two. A file which is defined by “n” labels is, in essence, an n-dimensional array. This constitutes an extremely natural implementation for APL, being one of the only languages which facilitates the use and manipulation of multi-dimensional data.

Even as early as the initial design sessions it became obvious that we would need a vehicle which would be capable of storing and performing inquiries on potentially very large amounts of data. Even though APL2 offers elegant primitives for the manipulation of data, the physical size of the APL2 workspace simply cannot hold the amount of data that the system's design called for. Similarly, if the data were to be stored in CMS sequential files, these files would be too large and clumsy to serve as the database for general queries.

With the availability of AP127, using SQL/DS as the data storage medium was the only logical choice. SQL/DS not only offers the ability to store very large amounts of data, but also offers the flexibility to handle an almost limitless number of data queries.

As the system design progressed, two problems surfaced:

1. A SQL table is a two-dimensional object, the data may be expressed in n-dimensions.
2. As user-defined structures grow at a linear rate, the number of data elements contained in the structure grows exponentially.

The first problem is better stated by the following question:

“How does one take an n-dimensional array and reduce it to a two-dimensional structure which may then be stored as a table in a relational database?”

In order to reduce an n-dimensional array into an object that can be represented in terms of two dimensions, it is necessary to establish what type of data each dimension will represent. Given an n-dimensional array, each dimension is represented by a user-defined label, such as “Year”, “Color”, “Shape”, etc. If the same “n” dimensions are represented in two dimensions, one might select a single label for one of the two dimensions and a grouping of the remaining labels for the other.

In order to be consistent with the first design goal of generality, *all* of the labels are grouped, forming the first dimension. And the second dimension simply becomes, “Data”.

Since each data element in the n-dimensional structure may be expressed in terms of “n” indices, representing its position in the n-dimensional array, and its actual value, the structure may be modeled in an “n+1” column SQL table. There is a direct relationship between each element in the n-dimensional array and a row in the SQL table. In other words, the SQL table must contain at least

as many rows as there are data elements in the array. This leads us directly into the second problem stated above.

To circumvent the second problem, we simply discard each row in the SQL table value which has a datum value of zero, otherwise known as sparse representation of an array.

The sparse representation of an n-dimensional array not only yields a large savings in storage (assuming that most user-defined arrays will be somewhat sparse), but elegantly allows FLEXI-VIEW to perform simple SQL queries to store and retrieve data from any perspective.

Once the user enters and/or selects the labels which define the frame-work of the structure, the corresponding "n+1" column SQL table can be built:

Label 1	Label 2	...	Label n	Data
Col. 1	Col. 2	...	Col. n	Col. n+1

Figure 1. The Structure of the SQL Data Table

The name of the SQL table is identical to that which is selected by the user as a name for the corresponding FLEXI-VIEW file. The generic name for this table when related to FLEXI-VIEW is the "data table".

The SQL table will always consist of "n+1" columns, where each column name is identical to its corresponding label. Column "n+1" is simply named "Data".

The following SQL command is used to create the data table:

```
CREATE <userid.tablename> (Label-1 VARCHAR(10) NOT NULL,
                          Label-2 VARCHAR(10) NOT NULL,
                          :           :           :
                          :           :           :
                          Label-n VARCHAR(10) NOT NULL,
                          Data      FLOAT NOT NULL)
IN XVU
```

Figure 2. SQL CREATE Command for the Data Table

There are two upper bounds on the number of rows that the table may potentially hold, the "primary upper-bound" U_1 and the "secondary upper-bound" U_2 .

$$\begin{aligned}
 U_1 &= (A_n \times A_{n-1} \times \dots \times A_1) \\
 &\text{or} \\
 U_1 &= \prod_{i=1}^n A_i \\
 U_2 &= \infty \\
 &\text{where,} \\
 A_i &= \# \text{ of items} \in \text{Label}_i \ni (1 \leq i \leq n)
 \end{aligned}$$

The upper-bound U_2 is the limit set by SQL/DS as the absolute maximum number of rows that a SQL table may hold. For all practical purposes this number is infinite. The primary upper-bound may only be exceeded when the entire structure is filled with all non-zero data elements and “user-defined formulas” are created. User-defined formulas will be covered in the following section.

The lower-bound is of course zero, since all the data elements of a structure are initially set to zero when a FLEXI-VIEW file is created. According to the sparse array definition above, only the indices of the non-zero elements are stored along with the value of the element.

The Structural Elements

The contents of the data table are dynamic, in that they are added, updated, and deleted as numerical data is entered into the system and changed. As stated previously, each record in the data table has a direct relationship with a non-zero data element in the array. Along the value of the element, each record contains “n” indices.

The indices are referred to as FLEXI-VIEW “items”, where each item is a sub-element of a label. Before any data elements may be entered, each label must be associated with at least one item.

Let’s look at an example:

A market research group wishes to determine which programming languages are most commonly used within several corporations in the marketplace. At the same time, a financial planning group, in the same organization, has been given the mission to build an industry model to help project the relative profitability of the various programming languages over time. The name of the model and corresponding FLEXI-VIEW file is “EXPERT”.

Following are the minimum of four categories or “labels” which are necessary to build such a model:

1. Line Item
2. Language
3. Vendor
4. Year

Given this 4-dimensional structure, the corresponding FLEXI-VIEW file will contain 4 labels and the corresponding SQL table will be composed of five columns:

Line Item	Language	Vendor	Year	Data

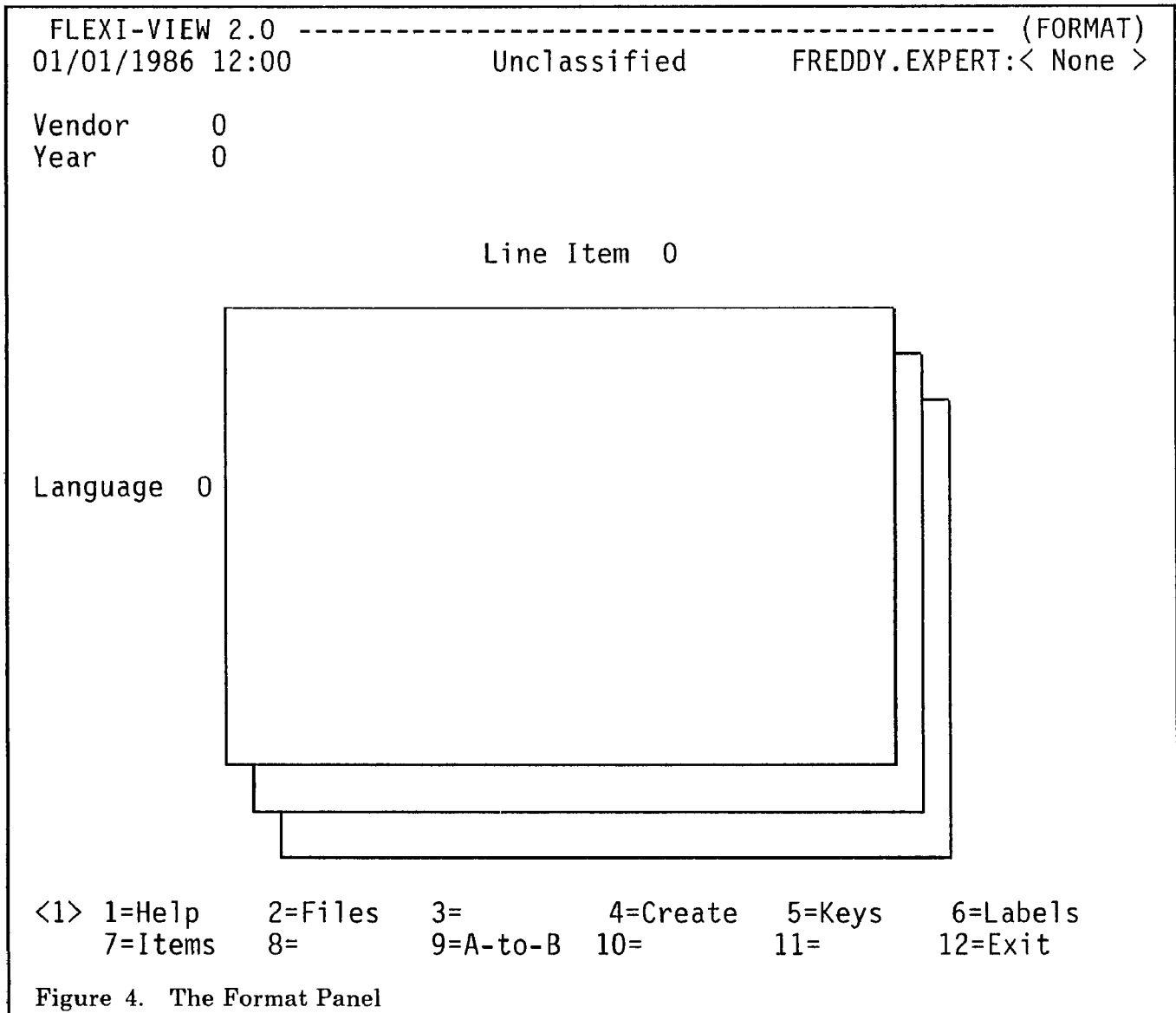
Figure 3. The Structure of the SQL Data Table

The ordering of the first four columns from left to right is determined by the order in which the labels were selected in FLEXI-VIEW. The rightmost column is always the data column. Since SQL is a relational database system, the particular ordering of the columns in the table is of little significance.

Format Mode

The Format Panel

Once the file has been created and given a name, the labels take their default positions on the *Format* panel:



The real power of FLEXI-VIEW is in the way an n-dimensional data structure may be manipulated by the user.

The icon in the center of the Format panel represents a series of parallel pages, and is intended to represent the multi-dimensional aspect of the Format mode.

The simplest method of working with multi-dimensional data is to think of its structure in terms of a series of planes or two-dimensional matrices. In FLEXI-VIEW, each matrix is called a "page".

While in the Format mode, the user may select a "page orientation" by positioning any one of the "n" labels on the X axis of the page, and any one of the remaining "n-1" labels on the Y axis. By swapping the label "Line Item" with "Language" and then "Vendor" with "Line Item", the following page orientation is selected:

```

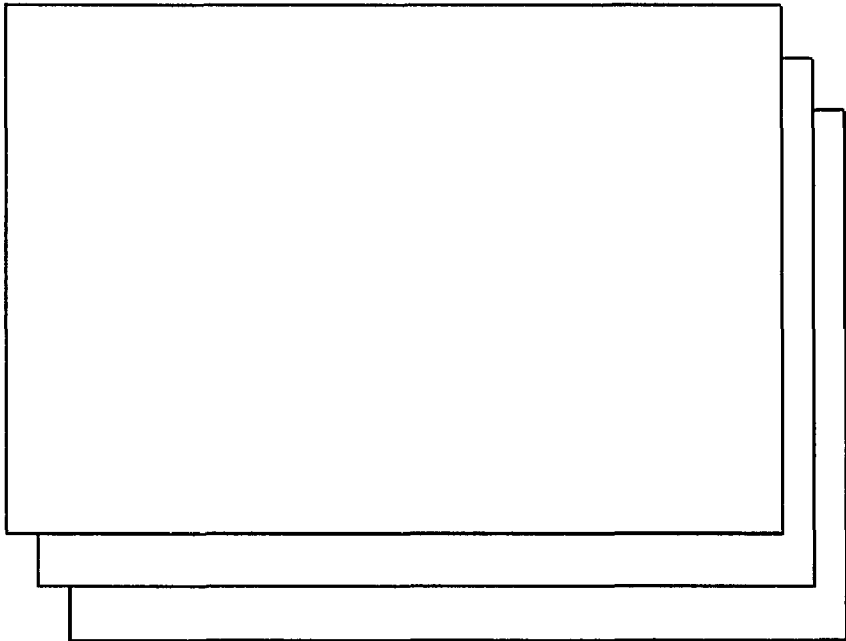
FLEXI-VIEW 2.0 ----- (FORMAT)
01/01/1986 12:00          Unclassified          FREDDY.EXPERT:< None >

Line Item  0
Year       0

                        Language  0

Vendor  0

```



```

<1> 1=Help   2=Files   3=       4=Create   5=Keys   6=Labels
      7=Items   8=       9=A-to-B  10=       11=       12=Exit

```

Figure 5. The Format Panel

In total, there exist " $n \times (n-1)$ " possible page orientations for any n-dimensional structure.

Since the file, "EXPERT", contains the 4 labels "Line Item", "Language", "Vendor", and "Year", there are (4×3) or 12 possible page orientations:

- | | |
|--------------------------|------------------------|
| 1. Line Item by Language | 7. Vendor by Line Item |
| 2. Line Item by Vendor | 8. Vendor by Language |
| 3. Line Item by Year | 9. Vendor by Year |
| 4. Language by Line Item | 10. Year by Line Item |
| 5. Language by Vendor | 11. Year by Language |
| 6. Language by Year | 12. Year by Vendor |

By removing the inversions, we are left with 6 unique page orientations:

1. Line Item by Language or Language by Line Item
2. Line Item by Vendor or Vendor by Line Item
3. Line Item by Year or Year by Line Item
4. Language by Vendor or Vendor by Language
5. Language by Year or Year by Language
6. Vendor by Year or Year by Vendor

where the number of unique page orientations is defined by the equation:

$$(n \times (n-1)) / 2$$

The Page Size

The maximum size of a page, given a particular page orientation, is governed by the number of items associated with the labels that are part of the page orientation (the labels on the X and Y axes). Given that the number of items associated with the X and Y axis labels are A_1 and A_2 , respectively. The size of the page is A_1 by A_2 and the number of data elements on each page is equal to the product, $A_1 \times A_2$.

For example:

Let the label "Language" contain the following items:

- PAL2
- LIPS
- SNOLOG
- PROTRAN
- RASCAL
- Z

Let the label "Vendor" contain the following items:

- ABC
- BMI
- U.S. Jones
- GOLD

If the page orientation is defined by the labels "Vendor" and "Language", the number of elements on the page is equal to (4×6) or 24, since there are four items associated with the label "Vendor" and six items associated with the label "Language".

	PAL2	LIPS	SNOLOG	RASCAL	PROTRAN	Z
ABC	a(1,1)	a(1,2)	a(1,3)	a(1,4)	a(1,5)	a(1,6)
BMI	a(2,1)	a(2,2)	a(2,3)	a(2,4)	a(2,5)	a(2,6)
U.S. Jones	a(3,1)	a(3,2)	a(3,3)	a(3,4)	a(3,5)	a(3,6)
GOLD	a(4,1)	a(4,2)	a(4,3)	a(4,4)	a(4,5)	a(4,6)

Figure 6. The Page Layout

The Page Index

As previously stated, a page orientation is defined by two of the “n” labels in a file. One is placed on the X axis (on top of the Format panel icon) and the other is placed on the Y axis (to the left of the Format panel icon).

The remaining “n-2” labels are placed in the *index area* (upper left-hand corner) of the Format panel.

In the example, the labels “Year” and “Type” are positioned as the X and Y axes, respectively. The remaining labels “Line Item” and “Year” are positioned in the “index area” and constitute the “index” of the file “EXPERT”.

“Line Item” is the first and *most significant* label in the index. Following is the set of items which are associated with it:

- Price
- Quantity
- Fix Cost
- Var Cost

The label “Year” is the second, last, and *least significant* label in the index. The file contains data that spans six contiguous years:

- 1983
- 1984
- 1985
- 1986
- 1987
- 1988

Note: The significance of a label is proportional to the frequency at which page breaks occur, regarding the items associated with that label, in the report and graph functions. In the above example, page breaks will occur for the items associated with the label “Year” before those with the label “Line Item”.

The number of pages which are defined by any particular page orientation is equal to the product of the number of items associated with each label in the index area.

If the labels which are positioned on the X and Y axes are associated with A_1 and A_2 items, respectively, then the number of pages which have the identical page orientation is defined as follows:

$$\text{Pages} = (A_n \times A_{n-1} \times \dots \times A_3), \forall n > 2$$

or

$$\text{Pages} = \prod_{i=3}^n A_i, \forall n > 2$$

$$\text{Pages} = 1, n = 2$$

where,

$$A_i = \# \text{ of items} \in \text{Label}_i \ni (3 \leq i \leq n)$$

In the simple case when a file is defined by only two labels, there exists only a single page for each of the two page orientations.

For any file which contains four or more labels, each parallel page is defined by a unique selection of a single item from each label in the index.

In this example, the labels "Line Item" and "Year" comprise the index for the page orientation, "Vendor by Language". The labels "Line Item" and "Year" are associated with 4 and 6 items, respectively. Thus, there are 24 pages which have the page orientation, "Vendor by Language".

Since multiplication is commutative, the number of pages with a common orientation is unaffected by the ordering of the labels in the index area. Similarly, the number of pages which have a specific orientation is equivalent to the number of pages which have its inversion.

For example, the number of pages which have the orientation, "Vendor by Language" is equivalent to the number of pages which have an orientation of "Language by Vendor". In FLEXI-VIEW these pages also represent the same data, but in a different format.

The View

In order to work with numeric data in FLEXI-VIEW, the user must define a view. In fact, any FLEXI-VIEW file may contain a multitude of user-defined views.

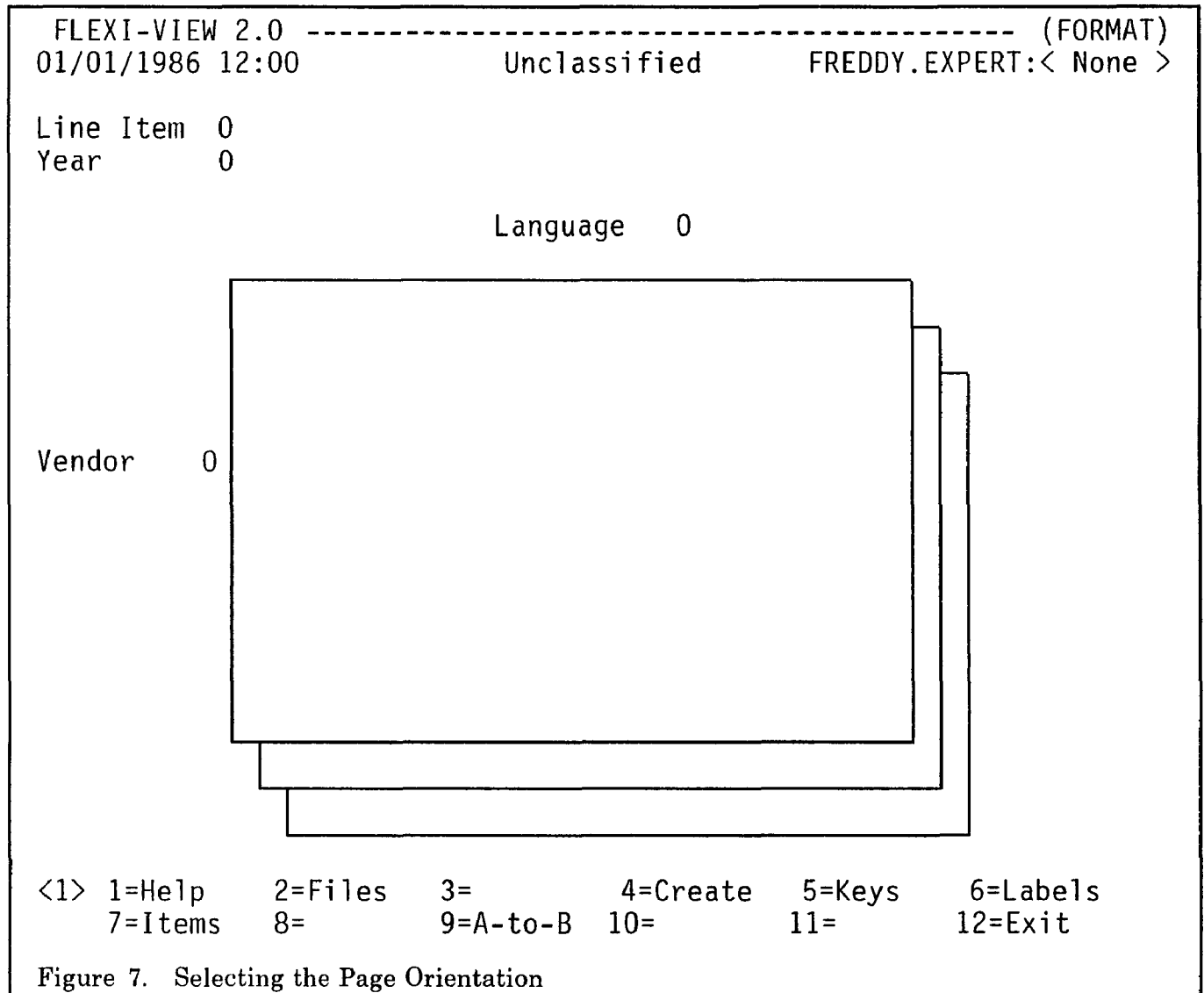
A view is determined by three factors:

1. A page orientation defined by two labels.
2. An ordering of the remaining "n-2" labels in the index area.
3. A selection of items associated with each label.

Just as every FLEXI-VIEW file has a corresponding SQL table, every view in FLEXI-VIEW has a corresponding SQL view.

Let's assume that an individual who is working with the file "EXPERT" wishes to analyze historical accounting data for sales of languages by several vendors. In order to work with the specified data he may perform the following steps:

1. Recall the file "EXPERT" and position the labels "Vendor" and "Language" on the X and Y axes, respectively.



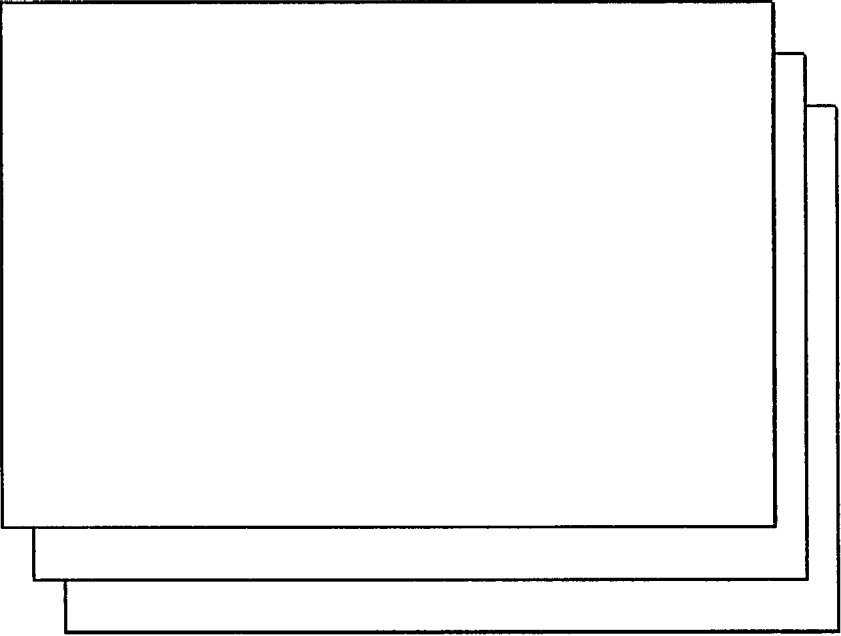
2. Select each item under the labels "Vendor" and "Language".

FLEXI-VIEW 2.0 ----- (FORMAT)
01/01/1986 12:00 Unclassified FREDDY.EXPERT:< None >

Line Item 0
Year 0

Language 6

Vendor 4



<1> 1=Help 2=Files 3= 4=Create 5=Keys 6=Labels
 7=Items 8= 9=A-to-B 10= 11= 12=Exit

Figure 8. Selecting the Contents of the Page

3. Select the historical years, "1983", "1984", and "1985".
4. Select the line items "Quantity" and "Price".

```

FLEXI-VIEW 2.0 ----- (FORMAT)
01/01/1986 12:00          Unclassified      FREDDY:EXPERT.< None >

Line Item  2
Year       3

                                Language    6

Vendor     4

<1> 1=Help   2=Files   3=          4=Create   5=Keys   6=Labels
      7=Items  8=          9=A-to-B  10=          11=          12=Exit

```

Figure 9. Selecting the Pages

The definition of the view is complete. If the user decides to save the view permanently, the following SQL command is executed:

```

CREATE VIEW <viewname> AS
  SELECT * FROM <userid.tablename> WHERE
    "LANGUAGE"   IN ('PAL2','LIPS','SNOLOG','RASCAL','PROTRAN','Z') AND
    "VENDOR"     IN ('ABC','BMI','GOLD','U.S. Jones') AND
    "LINE ITEM"  IN ('Quantity','Price') AND
    "YEAR"       IN ('1983','1984','1985')

```

Figure 10. Creating the SQL View

The view has the page orientation, "Vendor by Language", where each page consists of "4 x 6" or 24 element matrix:

	PAL2	LIPS	SNOLOG	RASCAL	PROTRAN	Z
ABC						
BMI						
U.S. Jones						
GOLD						

Figure 11. A Page of the View

And a total of six pages:

Line Item: Quantity
Year: 1983

Line Item: Quantity
Year: 1984

Line Item: Quantity
Year: 1985

Line Item: Price
Year: 1983

Line Item: Price
Year: 1984

Line Item: Price
Year: 1985

The Sharing of Data and Security

A powerful feature using SQL/DS is the ability to share data among users. In FLEXI-VIEW full advantage is taken of this feature, in that any FLEXI-VIEW user may grant other users, access to all or part of any of his files. In order to specify exactly which data elements are to be shared, a FLEXI-VIEW view is created and granted to other individuals. The grantor may specify whether the grantee will receive "read/only" or "read/write" access to the data delineated by the view.

When the grantee presses the PF key to display the names of all of his files, any granted views will show up as files, but with the grantor's user id prefixing the file name instead of his own. The grantee may work with the granted file just as if it were his own, including creating views on it. If the grantee has been granted "read/only" access, he may perform any operation on the file, except permanently saving the data.

All data security administration is controlled by the sophisticated mechanisms which are built into SQL/DS, thus freeing the kernel APL2 workspace from these duties. When a new user is added to FLEXI-VIEW by a designated administrator, a *private* SQL DBSPACE named "XVU" is acquired for him. The privilege of *connect authority* to SQL/DS is also granted to the user. All FLEXI-VIEW "data tables" which are owned by the user reside there.

A master FLEXI-VIEW DBSPACE is used to keep track of all information in the database which is attributed to FLEXI-VIEW, including files, views, labels, items, and grantees.

Update Mode

The Update Panel

The update portion of FLEXI-VIEW allows the user to enter, change, and analyze data as organized in Format mode.

Using the example described in the previous section, pressing PF10 from Format will display the following screen:

```

FLEXI-VIEW 2.0 ----- (UPDATE)
01/01/1986 12:00          Unclassified          FREDDY:EXPERT.< None >
Line Item  4 Price
Year       6 1983

Vendor
4:1-4      PAL2    LIPS    SNOLOG  RASCAL  PROTRAN Z    Total
ABC        22      13      9        57      102      133      336
BMI        100     11      7        66      122      102      408
U.S. Jones 125     14      10       64      115      149      477
GOLD       27      17      12       72      127      167      422
Total     274     55      38      259     466     551     1643

<1> 1=Help   2=Mode   3=Protect   4=Undo   5=Keys   6=Set Point
    7=       8=       9=       10=      11=      12=Exit

```

Figure 12. The Update Panel

The salient features of Update mode are as follows:

- | | |
|---------------------|---|
| Balancing | Simple matrix balancing may be performed on the current matrix. If the user changes any row, column, or grand total, the entire matrix will be balanced accordingly, with each individual cell adjusted based on its proportional weight in the original matrix. See the next section for the details on balancing. |
| Percent Mode | Display each element of the matrix as a percentage of the grand total. Balancing is available in this mode. |
| Row Percent | Display each element of the matrix as a percentage of the row total. Balancing is available in this mode. |
| Col Percent | Display each element of the matrix as a percentage of the column total. Balancing is available in this mode. |
| Protect | Hold any cell or block of cells in the matrix constant during matrix balancing. |
| Undo | Return the values of the matrix before the last change. |
| Set Point | Delineate a block of cells in the current matrix such that a function may operate on each element within the block. Functions that work in conjunction with Set Point are Protect, Copy, Interpolate, Preview, Graph, and Report. |

Convert	Convert a percentage matrix to an absolute matrix by specifying the absolute value of one percentage cell.
Utilities	<p>Various utilities are available:</p> <ul style="list-style-type: none"> ● Profile — Various display and output options ● Preview — Decision support graphics ● Notepad — Notes and information on files ● Import — Import a CMS file of numbers into the current page ● Calculate — Simple calculator function
Interpolate	Linear or exponential interpolation may be performed on an entire row or column of data by specifying a start point and end point, or by specifying a start point and a slope or growth constant.
Copy	Copy a block of cells to any part of the current matrix or to any part of another page of data.
Graphics	The Interactive Chart Utility (ICU) is used to deliver elaborate graphics which may be tailored to the user's specifications. Pre-defined business charts including bar, pie, and line charts are available. User defined graph formats may also be used. In addition, an APGS plotfile may be created.

A new page of data may be selected by directly entering a new item name next to a label in the index area. If the item is spelled incorrectly, a window of possible values will be displayed. Alternatively, a select key is available to list all the items which are associated with any label.

The reporting capability in FLEXI-VIEW has been kept simple. Since FLEXI-VIEW stores data in a readable format in SQL/DS, any application that interfaces with SQL/DS may use FLEXI-VIEW data.

For example, a user may apply FLEXI-VIEW's multi-dimensional capability to enter and analyze data, but use QMF, ISQL, VMAS, or IC/1 to generate reports on the data.

Balancing

Introduction

A simple non-iterative balancing scheme has been implemented in FLEXI-VIEW. It is based on a modified version of a method described in an article by D. Frieland in the Journal of Royal Statistical Society (volume 124, pages 412-420).

In the FLEXI-VIEW update module, we have a two dimensional contingency table, i.e., a two dimensional distribution of numbers with row and column totals (see figure below).

	1981	1982	1983	1984	1985	Total
Jan-Apr	5	4	10	12	32	63
May-Aug	13	11	12	22	33	91
Sep-Dec	11	12	13	14	15	65
Total	29	27	35	48	80	219

Figure 13. A Two-Dimensional Contingency Table

The simplest case exists when the internal distribution of the matrix is altered, i.e., a value is altered which is not a row or column total. In this case, the row and column totals are recalculated and the resultant matrix displayed.

The more complex case involves altering a total. If any row or column totals are changed, the matrix will be balanced accordingly. Each cell is adjusted based on its proportional weight in the original matrix.

The user may hold cells constant by applying the protect function. This causes a cell or any group of cells to remain fixed after a balancing operation has taken place. In addition, any zero cells, any altered cells, and any cells in the intersection of adjusted row and column totals are implicitly fixed.

An other important mechanism which is a component of the balancing implementation is a "trimming" method. Cells are first balanced, then the results are rounded. After rounding, it is very probable that the cells no longer add up to the corresponding row or column total. For example:

Cell 1	Cell 2	Cell 3	Cell 4	Total
1	1	1	1	4

Suppose we change the total to be 5 so that we get:

Cell 1	Cell 2	Cell 3	Cell 4	Total
1.25	1.25	1.25	1.25	5

Now if the user has only one decimal place of precision set, we would see:

Cell 1	Cell 2	Cell 3	Cell 4	Total
1.3	1.3	1.3	1.3	5

which is incorrect!

We therefore apply a trim operation to get:

Cell 1	Cell 2	Cell 3	Cell 4	Total
1.2	1.2	1.3	1.3	5

This is now a true equality. Basically, we have reduced the first two cells by 0.1. In general, the trimming mechanism will adjust cells uniformly that carry the most weight. Since the entire row distribution was uniform in the above example, the trimming process simply works from left to right until the cells add up correctly.

Another feature that is included in the balancing scheme is the ability to enter a value in a row or column total which is initially all zero. This will create a uniform distribution in the matrix. For example, if we start out with the following row:

Cell 1	Cell 2	Cell 3	Cell 4	Total
0	0	0	0	0

and we change the total:

Cell 1	Cell 2	Cell 3	Cell 4	Total
0	0	0	0	100

the result will be:

Cell 1	Cell 2	Cell 3	Cell 4	Total
25	25	25	25	100

Here are a few examples which illustrate some of the balancing techniques:

Original Matrix	Changed Matrix	Balanced Matrix
<pre> 1 2 3 4 10 2 3 4 5 14 ===== 3 6 7 9 24 </pre>	<pre> 1 2 3 4 20 2 3 4 5 14 ===== 3 6 7 9 24 </pre>	<pre> 2 4 6 8 20 2 3 4 5 14 ===== 4 7 10 13 34 </pre>

Figure 14. Example of Simple Row Balancing.

Original Matrix	Changed Matrix	Balanced Matrix
<pre> 1 <2> 3 4 10 2 3 4 5 14 ===== 3 6 7 9 24 </pre>	<pre> 1 <2> 3 4 20 2 3 4 5 14 ===== 3 6 7 9 24 </pre>	<pre> 2 2 7 9 20 2 3 4 5 14 ===== 4 5 11 14 34 </pre>

Figure 15. Example of Simple Row Balancing with One Element Fixed.

Note: <*> indicates a fixed element.

1	2	3	4		10
2	3	4	5		14
=====					
3	6	7	9		24

1	2	3	<4>		20
2	3	4	5		14
=====					
3	6	7	12		24

3	5	8	4		20
2	3	4	8		17
=====					
5	8	12	12		37

Figure 16. Example of Balancing Both Row and Column Totals.

Note: (4) lies in the intersection of the altered row and column and is therefore implicitly fixed.

1	2	3	4		10
2	3	4	5		14
=====					
3	6	7	9		24

1	2	3	4		10
2	3	4	5		14
=====					
3	6	7	9		85

4	7	11	13		35
7	11	14	18		50
=====					
11	18	25	31		85

Figure 17. Example of Top/Down Balancing.

Mathematical Background

The general matrix we wish to consider is as follows:

a(1,1)	a(1,2)	a(1,3)	...	a(1,n)	r(1)
a(2,1)	a(2,2)	a(2,3)	...	a(2,n)	r(2)
a(3,1)	a(3,2)	a(3,3)	...	a(3,n)	r(3)
.
.
.
a(m,1)	a(m,2)	a(m,3)	...	a(m,n)	r(m)
c(1)	c(2)	c(3)		c(4)	T

where:

$$\sum_{j=1}^n a(i,j) = r(i) \quad \text{for } i = 1,2,3,\dots,m$$

$$\sum_{i=1}^m a(i,j) = c(j) \quad \text{for } j = 1,2,3,\dots,n$$

$$\sum_{j=1}^m r(i) = \sum_{i=1}^n c(j) = T$$

Now define a set F to be the set of all (i,j) such that $a(i,j)$ is fixed. Let $\neg F$ be the complement of F , $F(j)$ be the projection onto the first component of (i,j) and $F(i)$ be the projection onto the second component of (i,j) . Recall that an element $a(i,j)$ in the matrix is *fixed* if:

1. It is zero
2. It has changed
3. It lies in the intersection of a row and column total that has been changed
4. It is protected

Calculations of the new row elements are then given by the formula:

$$a(i,j) = a(i,j) \times [r(i) - \sum a(i,j)] / \sum a(i,j) j \in F(i) j \in \neg F(i) i = 1, 2, 3, \dots, m$$

Similarly, the row elements are given by the formula:

$$a(i,j) = a(i,j) \times [c(j) - \sum a(i,j)] / \sum a(i,j) i \in F(j) i \in \neg F(j) j = 1, 2, 3, \dots, n$$

Note: Since we have fixed the intersection of altered row and column totals, it is irrelevant which direction is balanced first, i.e., first balancing with respect to the rows and then with respect to the columns will yield the same result if done in the opposite order. Furthermore, there is no need to iterate since, as long as there is sufficient degrees of freedom, a solution is guaranteed.

This scheme is elegantly implemented in APL2. Two stochastic matrices are assembled containing the weights of the multipliers for the original matrix.

The following APL2 variables are defined:

ROW_TOTALS — Row total elements $r(1), r(2), \dots, r(m)$.
COL_TOTALS — Column total elements $c(1), c(2), \dots, c(n)$.
MATRIX — The current matrix elements $a(i,j)$, $1 \leq i \leq m$, $1 \leq j \leq n$.
FIX — m by n Boolean matrix containing $f(i,j)$ where
 $f(i,j) = 0$ if $a(i,j)$ is not fixed
 $f(i,j) = 1$ if $a(i,j)$ is fixed.

The row and column stochastic matrices are built as follows:

```
S_ROW←FIX×(¬FIX)×[1](ROW_TOTALS-÷/[2]FIX×MATRIX) DIV ÷/[2](¬FIX)×MATRIX
S_COL←FIX×(¬FIX)×[2](COL_TOTALS-÷/[1]FIX×MATRIX) DIV ÷/[1](¬FIX)×MATRIX
```

where “DIV” is regular division, “÷”, with division by zero defined to be zero, i.e., $A \text{ DIV } 0 \longleftrightarrow 0$, where A is anything.

The resultant balanced matrix is then given by

$$NEW_MATRIX \leftarrow S_COL \times S_ROW \times MATRIX$$

The above calculation represents the core of the balancing algorithm implemented in FLEXI-VIEW.

Note: The above scheme may be easily modified for iteration. Remove the third definition of a “fixed” cell (mentioned above) and iterate until convergence to a solution matrix.

Some other features are:

- A uniform distribution of numbers may be generated when the matrix is initially zero
- Top/Down and Bottoms/Up allocation
- Degree of freedom check and error messages
- Blank cells mapped to zero values
- Protect ability to fix cells
- Zero to nine decimal places of precision
- Totals may be shut off to disable the balancing mechanism

Formulas

Introduction

As previously stated, each label is associated with a set of items. There are two types of items that may fall under a label, namely *simple* and *non-simple* items. *Simple items* are self-contained and do not depend on other items. *Non-simple items* are associated with an algebraic expression. The algebraic expression may be composed of simple items, other non-simple items, and constants.

For example, a user may have a label called "MONTHS" which is associated with the items "JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC", "Q1", "Q2", "Q3", "Q4".

The items JAN, ..., DEC are simple items and Q1, ..., Q4 are non-simple, i.e., they are formulas:

$$\begin{aligned} Q1 &= \{JAN\} + \{FEB\} + \{MAR\} \\ Q2 &= \{APR\} + \{MAY\} + \{JUN\} \\ Q3 &= \{JUL\} + \{AUG\} + \{SEP\} \\ Q4 &= \{OCT\} + \{NOV\} + \{DEC\} \end{aligned}$$

Once a non-simple item is defined (done in FORMAT mode) it may be used by other non-simple items. For example, once Q1, Q2, Q3, and Q4 are defined, a user may define HALF1, HALF2 where:

$$\begin{aligned} HALF1 &= \{Q1\} + \{Q2\} \\ HALF2 &= \{Q3\} + \{Q4\} \end{aligned}$$

Any combination of simple and non-simple items constitute an algebraically valid statement:

$$\begin{aligned} \text{Formula1} &= 100 * ((JAN) + (FEB)) / Q1 \\ \text{Formula2} &= (((JAN) + (FEB) - (MAR)) * (Q2)) * 0.45 * (\text{Formula1}) \end{aligned}$$

Formulas are defined in Format mode. After a formula is defined, it is treated just as if it were a simple item (from a user's perspective).

Implementation of Formulas

In the FLEXI-VIEW workspace, a variable named "*CURRENT_SELECT*" contains the item names which are chosen for the current page. This variable is used to issue SQL/DS queries via AP127 to retrieve the data for the current page.

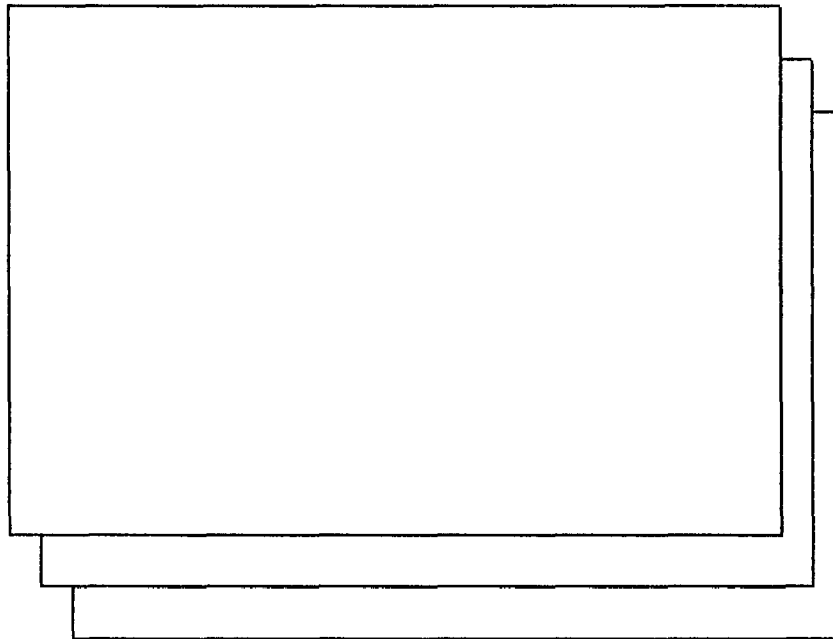
From the example in the Format section, suppose we have the following view defined for user "FREDDY", file "EXPERT", and view "VIEW001":

FLEXI-VIEW 2.0 ----- (FORMAT)
11/18/1986 08:45 Unclassified FREDDY:EXPERT.VIEW001

Vendor 5
Language 8

Line Item 7

Year 6



<1> 1=Help 2=Files 3= 4=Create 5=Keys 6=Labels
 7=Items 8= 9=A-to-B 10= 11= 12=Exit

Figure 18. The Format Panel

Suppose we have the following simple items under each label:

Language	Line Item	Vendor	Year
PAL2	Quantity	ABC	1983
LIPS	Price	BMI	1984
SNOLOG	Var Cost	US JONES	1985
RASCAL		GOLD	1986
PROTRAN			1987
Z			1988

Figure 19. Simple Items

And the following formula definitions:

<i>Language</i>	
Interpret	= (PAL2) + (LIPS) + (SNOLOG)
Compile	= (RASCAL) + (PROTRAN) + (Z)
<i>Line Item</i>	
Tot Cost	= (Var Cost) * (Quantity) + (Fix Cost)
Revenue	= (Quantity) * (Price)
Profit	= (Revenue) - (Tot Cost)
<i>Vendor</i>	
All	= (ABC) + (BMI) + (US JONES) + (GOLD)

Figure 20. Formulas Defined Under Each Item

When Update mode is re-entered, the following panel is displayed:

```

FLEXI-VIEW 2.0 ----- (UPDATE)
01/01/1986 12:00          Unclassified          FREDDY:EXPERT.< None >

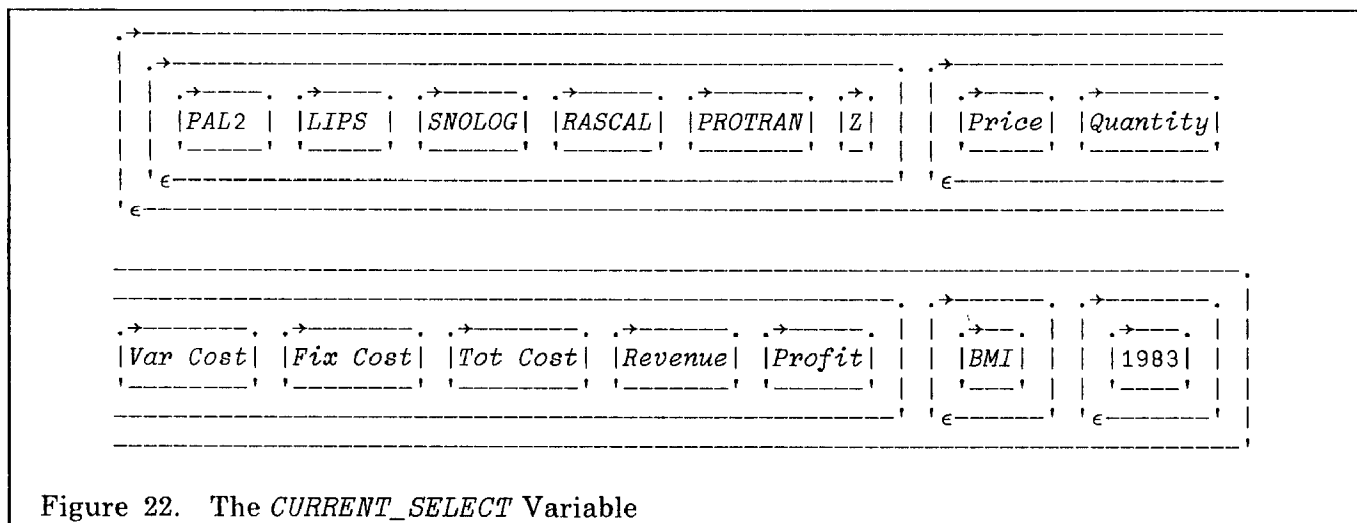
Vendor      4 BMI
Language    3 PAL2

Line Item
8:1-8      Year 6:1-6
1983      1984      1985      1986      1987      1988      Total
Quantity   100      120      124      129      143      165      781
Price      12       12       13       14       16       17       84
Var Cost   9        9        10       10       11       11       60
Fix Cost   60       60       63       64       69       84       400
Tot Cost   960      1140     1303     1354     1642     1899     8298
Revenue    1200     1440     1612     1806     2288     2805     11151
Profit     240      300      309      452      646      906      2853
Total      181      201      210      217      239      277      1325

<1> 1=Help    2=Mode    3=Protect  4=Undo    5=Keys    6=Set Point
    7=        8=        9=        10=       11=       12=Exit

```

In the above example, the variable *CURRENT_SELECT* would be constructed as follows:



The simple data is retrieved from the SQL/DS table named "FREDDY.EXPERT". Next, all non-simple items are decomposed into the simple items that they are composed of.

Three matrices are constructed from the data which is retrieved from the SQL/DS table:

1. F_X_DATA — Values of the items composing any x-axis formulas
2. F_Y_DATA — Values of the items composing any y-axis formulas
3. F_O_DATA — Values of the items lying in the intersection of any x-axis and y-axis formulas.

The user-defined formulas are then converted into executable APL2 expressions, e.g., “ $X/5-2+3$ ” is converted to “ $(X\div 5) + (-2) + 3$ ”. The formulas are stored in nested variables F_X and F_Y .

In the above example, only the label on the y-axis contains a formula. Therefore, F_X and the corresponding variable F_X_DATA are null. If either of F_X and F_Y are null, then so is F_O_DATA .

We have the following for F_Y :

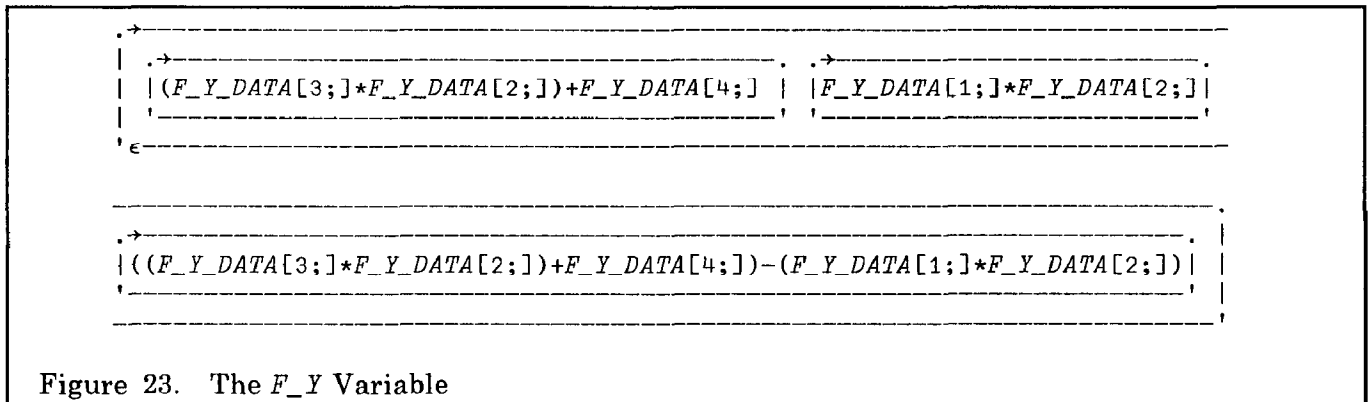


Figure 23. The F_Y Variable

The variable F_Y_DATA contains a matrix corresponding to the language “PAL2”, and the vendor “BMI”. The rows represent the line items “Quantity”, “Price”, “Var Cost”, “Fix Cost” and the columns represent the years “1983”, “1984”, “1985”, “1986”, “1987”, “1988”.

	1983	1984	1985	1986	1987	1988
Quantity	a(1,1)	a(1,2)	a(1,3)	a(1,4)	a(1,5)	a(1,6)
Price	a(2,1)	a(2,2)	a(2,3)	a(2,4)	a(2,5)	a(2,6)
Var Cost	a(3,1)	a(3,2)	a(3,3)	a(3,4)	a(3,5)	a(3,6)
Fix Cost	a(4,1)	a(4,2)	a(4,3)	a(4,4)	a(4,5)	a(4,6)

Figure 24. The F_Y_DATA Matrix

The length of F_Y would equal the number of formulas on the y-axis. Likewise, F_X would have the same length as the number of formulas defined in the x-axis.

The overlap cells are calculated via the equations:

$$\begin{aligned}
 F_X_DATA &\leftarrow F_O_DATA \\
 F_X_DATA &\leftarrow \Phi \rightarrow \& F_X \\
 F_O_DATA &\leftarrow \& F_Y
 \end{aligned}$$

Note: For non-commutative expressions, different results will occur when interchanging the x and y axes. Usually, the overlapping cells are meaningless in this situation.

The final step is to insert the formula calculations into the matrix. This is easily accomplished by performing an “ $\mathbf{\Phi}$ ” on F_X and F_Y . then indexing properly into the displayed matrix. This step is performed each time the matrix changes so that the formula cells are updated when the items they depend on change.

The user may also select a formula for any of the index area labels. To facilitate this, all matrices that the formula depends on is retrieved from SQL/DS. The appropriate formula is applied to the planes. The resultant matrix is stored as the current matrix and the above scheme remains the same.

Summary

Only through the powerful aspects of APL2, namely:

- general arrays
- the each operator
- selective specification
- recursion
- event simulation

were we able to devise and implement an elegant and non-traditional approach to solving some traditional real world problems.

The multi-faceted functionality of ICU/GDDM provides the user of FLEXI-VIEW with a commandless and “user-friendly” environment. Through the use of *windows*, we were able to take a relatively complex application concept, which conceivably would have consisted of dozens of full-screen panels, and implement it with just *four* dynamic full-screen panels.

The ICU provides both the interactive decision-support and presentation business graphics, which are fast, effective, and attractive.

SQL/DS gives FLEXI-VIEW the ability to store vast amounts of data and the capability to organize the data in an almost limitless number of perspectives. SQL/DS has also contributed greatly to the organization and simplicity of the FLEXI-VIEW workspace by undertaking the complex tasks of security administration and concurrent data access. In addition, using SQL/DS for data storage gives the user the ability to use any system that interfaces with SQL/DS (QMF, VM/AS, IC/1, etc.) to access FLEXI-VIEW data. QMF, VM/AS, and IC/1 are IBM Program Products.

Acknowledgments

We would like to offer special thanks to Jonah Atlas, Wei-Tih Cheng, and Thomas K. Lee for their invaluable contributions to the FLEXI-VIEW project as part of the original design team.

We also greatly appreciate the contributions made by Daniel Berndt, Thomas Byrne, Edward Eusebi, Kenneth Fordyce, Kenneth Halbrecht, Gilbert Laganne, and Ron Wilks.