ENERGY LABORATORY
INFORMATION CENTER

PRIMER FOR GMIS
Generalized Management Information System
Energy Laboratory in Association with the
Alfred P. Sloan School of Management
Working Paper No. MIT-EL-75-012WP
September 1975

This research was supported by the New England Regional Commission
August 1975
Acknowledgement

Abstract

1. Introduction
2. Overview System Architecture
3. Procedure for logging onto VM
4. Procedure for entry to TRANSACT
   a) Single user
   b) Multi user
5. Use of TRANSACT/SEQUEL
6. Use of Bulk Loader
7. Procedure for entry to APL
8. Use of APL
9. Use of EPLAN
10. Use of TSP
11. Log out procedure
12. References

Appendix (1) SEQUEL Syntax
(2) TRANSACT
   1. Syntax
   2. Explanation of commands
(3) Bulk Loader
ACKNOWLEDGEMENT

Many people have played a critical note in the development of GMIS; Faculty, Staff and Students of MIT, personnel of New England Regional Commission and IBM Staff members. This document was prepared with the assistance of George Berry, NEEMIS Project Coordinator, Bryan Mau, Louis Gutentag and John Donovan.

John J. Donovan,
Principal Investigator
Associate Professor,
MIT Sloan School
Cambridge, Massachusetts
ABSTRACT

GMIS (Generalized Management Information System) is a set of tools for building information systems. These tools include a flexible data management capability, various modeling and analytical capabilities. Facilities for on line queries and execution as well as facilities for multi-user access to a data base. These tools have been applied to produce a facility (NEEMIS, New England Energy Management Information System) for energy planning and analysis within New England.

This manual's purpose is to be a primer for using GMIS and the NEEMIS facility. For motivation and overview of the research involved in GMIS, see "GMIS - An Experimental System for Data Management and Analysis", by J. Donovan and H. Jacoby. For a complete description of the facility, see GMIS User's Guide.
1. **Introduction**

GMIS (Generalized Management Information System) is a software capability that facilitates the construction of management information systems, models, and user interfaces. The tools are particularly applicable to systems with the following characteristics:

- several classes of users, each of which has a different degree of sophistication
- complex and changing security requirements
- data that exhibits complex and changing interrelationships
- changing needs to be met by the information system
- need for quick and inexpensive implementation
- complex data validation requirements
- complex models to be built that access data

Information systems to assist in public and corporate policy making have the above characteristics. These policy making areas fall into three categories: (1) areas where the problems keep changing due to changing concerns, perceptions, and availability of data, or (2) which are one-time problems, or (3) where the problems addresses are ill-defined and where the system, therefore, after being built, usually needs major modifications.

Commercial systems, while adequate for well-defined applications, e.g., inventory control, accounting, etc., have focused on problems associated with such applications (e.g., low variable costs); these systems are not adequate for the demands of the market for systems for public and corporate policy makers, and have not focused on the problems associated with these systems (e.g., high fixed costs, interactive operations, etc.).
The major technologies which have made GMIS possible have been the application and extension of research in virtual machines, the hierarchical approach and relational implementation of data. The approach is hierarchical in implementation because this technique provides for ease of debugging, independence of hardware, and a basis for investigating properties of completeness, integrity, correctness, and performance. It is hierarchical in its presentation to the user to take cognizance of the fact that levels of user sophistication demand appropriate command environments. As such, the casual system user has powerful, high-level commands at his disposal, while the sophisticated (perhaps also the more mathematically inclined) user has more detailed and basic commands, but with a low tolerance for error.

Existing modeling facilities do not provide capabilities to meet the needs of systems with the above characteristics. Existing data management systems do not fulfill the requirements above; some are lacking the statistical and modeling packages, most are not interactive, and most do not allow multiple users to access the same data base and most important, none were designed for a changing environment. Using the GMIS facility, it is possible to construct an information system in a matter of days. For example, we have applied GMIS to building a New England Energy Management Information System, NEEMIS, (Donovan, Jacoby, 1975), for assisting in regional energy policy making. We reconstructed the entire data base of NEEMIS five times in one month during the summer of 1975 -- once to incorporate additional data of a data series, twice to restructure for efficiency reasons, and twice because the users wanted data and models for which we had not originally planned.
Currently GMIS is implemented on a IBM 370 computer. It uses the virtual machine concept extensively. The VM concept is presented in several places (Parmelee, 1972; Madnick and Donovan, 1974, and Goldberg 1973), and many of its advantages are articulated elsewhere (Madnick, 1974 and Buzen, 1973). The concept of "virtual machines" has been developed by IBM to the point of a production system release, VM/370 (IBM, 1972). A virtual machine may be defined as a replica of a real computer system simulated by a combination of a Virtual Machine Monitor (VMM) software program and appropriate hardware support. For example, the VM/370 system enables a single IBM System/370 to appear functionally as though it were multiple independent System/370's (i.e., multiple "virtual machines"). Thus, a VMM can make one computer system function as though it were multiple, physically isolated systems.

A configuration of virtual machines used in GMIS is depicted in Figure 1, where each box denotes a separate virtual machine. Those virtual machines across the top of the Figure are executing programs that provide user interfaces, whether they be analytical facilities, existing models, or data base systems. All these programs can access data managed by the general data management facility running on the virtual machine depicted in the center of the page. A scenario of a user of this architecture may be as follows. A user activates a model, say in the EPLAN/APL machine. That model requests data from the general data base machine (called the Transaction Virtual Machine, or TVM), which responds
Figure 1: Overview of the Architecture of GMIS
by passing back the requested data. Note that all the analytical facilities may be incompatible with each other, in that they may run under different operating systems. The communications facility between virtual machines in GMIS is described in (Gutentag 1975). Extensions to this architecture to allow interfaces to other data base systems and other computer systems are discussed in (Donovan and Jacoby 1975).

The user views all data stored in the general data base system in simple constructs called relations. A relation can be thought of as a two dimensional table with rows and columns. Entries in a column come from sets called domains. An example of a relation named CARSALES is depicted in Figure 2 where names of the columns are: MODEL, DATE, SALES and MPG.

<table>
<thead>
<tr>
<th>MODEL</th>
<th>DATE</th>
<th>SALES</th>
<th>MPG</th>
</tr>
</thead>
<tbody>
<tr>
<td>VEGA</td>
<td>7401</td>
<td>38455</td>
<td>30.2</td>
</tr>
<tr>
<td>PINTO</td>
<td>7401</td>
<td>38000</td>
<td>30.0</td>
</tr>
<tr>
<td>COLT</td>
<td>7401</td>
<td>24000</td>
<td>30.1</td>
</tr>
</tbody>
</table>

FIGURE 2 - EXAMPLE OF A RELATION
2.2 SEQUEL

We needed an interactive query data definition language, and we proceeded with an implementation of an MIT relational system (Smith, 1974). The current version of GMIS, whose data management capability is based on a language known as SEQUEL, evolved through the vehicle of an IBM/MIT Joint Study Agreement. SEQUEL (Structured English Query Language) is an experimental relational query and data definition language developed at the IBM San Jose Research Laboratory (Chamberlain, 1974). In cooperation with the IBM Cambridge Scientific Center, we have extended this experimental system by easing restrictions of numeric data types to small integers, and relaxing contraints on the number of columns allowed in a table, and by increasing the allowable lengths of identifiers and character strings. We also designed mechanisms for security and for handling missing data, expanded the bulk loading facilities, and added additional syntax, and made several changes to improve performance.

2.3 Multi-User Transaction Interface

Two requirements of GMIS are that multiple users be able to access the same data base and that different analytical and modeling facilities be able to access the data base all at the same time. For example, one user may want to build an econometric model using TSP while another user will request the system to generate a standard report. Still a third user may want to query the data base from an APL (A Programming Language) environment. These requirements have been met with the design and implementation of the Multi-User Transaction Interface (Gutentag 1975). Each GMIS user operates in his own virtual machine with a copy of the user interface he requires. Each user transaction to the data base is written into a transaction file, and that user's request for processing is sent to the data base machine (Transaction Virtual Machine). The Multi-User Interface processes each request in a first-in/first-out (FIFO) order, by reading the selected user's transaction file, and
writing the results to a reply file that belongs to the user. Each user interface reads the reply file as if the reply had been passed directly from the data base management system. For some user interfaces e.g., APL, EPLAN, TSP, we have written routines callable from the user interface which makes this communication transparent to the user.

2.4 **User Interfaces**

GMIS provides the capability for users to write their own interfaces to communicate with the data base system. TRANSACT is a general user interface that is designed to process transactions from most teletypewriters and CRT terminals. It allows the user to direct transaction output to any virtual device on the VM/370.

Interfaces to APL, TSP, and PL/I are operational and enable users to communicate with the Transaction Virtual Machine (Figure 1) simultaneously with all other users. An interface to the TROLL econometric modeling facility is in the design stage.

The architecture depicted in Figure 1 also allows the use of any of these modeling or analytical facilities independent of the transaction virtual machine. For example, functions may be written in APL to operate on data stored in the APL's work space. TSP modeling and reporting capabilities can operate on data stored in TSP's data base. FORTRAN or PL/I can operate on data stored in their own machine. It should be noted, however, that not using the general data base facility seriously inhibits flexibility and makes the algorithms dependent on the physical organization of the data but more importantly inhibits the community of users as they cannot conveniently access the common data base.
2.5 A User Interface - TRANSACT

As depicted in Figure 1, one of the user interfaces to GMIS is called TRANSACT. TRANSACT is primarily a data base editing and retrieval interface. It performs the following functions:

1. It accepts input from the terminal.
2. It operates directly on commands which specify output formats.
3. It passes data management commands to SEQUEL, which is the primary data management program running in the Transaction Virtual Machine.
4. It transmits returned data to the device selected by the operator in the format specified by the operator through TRANSACT commands.
3. PROCEDURE FOR LOGGING ONTO VM

In the remainder of this primer, items that the user should type in at his terminal are shaded, and computer responses are capitalized. Shaded items in quotations indicate places when names furnished by the user are to be inserted -- the user should replace the quoted item (and the quotes) by the furnished name.

The following steps should be executed in order to use GMIS from a 2741 or CDI terminal:

1. Turn on power at the terminal.
2. With a CDI terminal a normal telephone can be used. Insert the telephone handset into the coupler on the terminal, and be sure that the terminal is set for 15 cps half-duplex. With the 2741, a data set telephone must be used. Push the "talk" button on the data set and pick up the handset.
3. Dial the telephone number for the computer: (617) 868-2981.
4. Within thirty seconds after dialing, the establishment of telephone contact with the computer is indicated as follows:
   - with the CDI: the "online" light comes on
   - with the 2741: the handset will emit a high-pitched tone -- the user should now press the "data" button on the data set.
5. If telephone contact is not established within one minute, hang up, then redial.
6. A few seconds after contact is established, the computer will respond by printing:
   VM/370 ONLINE
7. The user should now press the ATTN key. The user is now ready to begin entering commands to the computer. He should keep in mind the following conventions when typing on the console:
   - Each command line is ended with a carriage return.
   - A character may be changed before keying carriage return by keying @ to erase one character at a time or @ to erase a line at a time, then retyping the proper characters.

The user now logs on as follows:
ENTER PASSWORD

"password"

where the user replaced the name "acctname" with the name of his account and the name "password" with his user password; the 'm' requests the computer to mask the password.

Now the user has a bare machine assigned to him. He must now load an operating system. To load the interactive operating system CMS, the user types:

```
logon "acctname" m

ENTER PASSWORD

"password"
```

This command loads and activates the CMS operating system. The computer terminology for this action is sometimes called IPL or initial program load. CMS is the operating system that many of our user interfaces run under. If you were executing a model in a language which requires a different operating system, you would activate that other operating system at this point, rather than CMS (See IBM 1972).

8. Enter a carriage return. This will initialize your operating system and print several lines of messages.
4. PROCEDURE FOR ENTRY TO TRANSACT

Once a user has logged into VM and has loaded the CMS operating system he may now go into the TRANSACT environment. There are two different modes in which the user can use TRANSACT: The single-user mode and the multi-user mode. The logon procedure is slightly different for each case.

3.1 Single User

The user types the command 'TRANSACT', which loads into his computer the program TRANSACT and the SEQUEL processor:

```
TRANSACT VERSION xx (DATE)
```

Your virtual computer is now ready to accept and act on TRANSACT/SEQUEL commands to work on the data base in your computer.

3.2 Multi-user

The user types the command 'transact', followed by an argument which is the name of the virtual computer where the database he wishes to use and a SEQUEL processor are located (the Transaction Virtual Machine - TVM).

```
transact "acctname2"
```

```
TRANSACT VERSION xx (DATE)
```

'acctname2' is the ID for the second virtual computer, which is furnished by the operations branch of the NEEMIS Project (617-253-5945). A common transaction virtual machine which contains much of the NEEMIS data base is named NEEMIS.

Your virtual computer, 'acctname', is now ready to accept and execute TRANSACT commands; the virtual computer 'acctname2' will execute the SEQUEL commands which your virtual computer will send it.

Note: If your virtual computer does not receive any response from the TVM, you should check with the Operations Branch Manager of the NEEMIS Project at MIT -- (617) 253-5945
5. USE OF TRANSACT/SEQUEL

After logging into VM and establishing a TRANSACT environment, the user, by using SEQUEL, can create tables, input data and access data. Appendix (1) gives a complete syntax of all SEQUEL commands. Here we give a sample session, using the MIT/IBM version of SEQUEL.

In this example, the user wishes to construct a table similar to that depicted in Figure 2. He must first define the domains he wishes to use for his columns. A domain is a set of data which all have the same attributes - for example, the data items may all be character strings representing fuel types, or all numbers representing population totals.

Entries in a column in a table are drawn from a domain. Note - entries in several columns in a table may come from the same domain. In fact, columns from different tables may come from the same domain (one table, for example, might have a column of population totals for the United States, and another might have a column of population totals for Canada, where entries in both columns come from the same domain consisting of a set of numbers).

The user wishes to create four domains, one to hold sales volume data, one for miles per gallon figures, one to represent dates and one to hold the names of car models. Note: Three of these domain contain numbers and one, character strings - denoted by "num" and "char", respectively. He types:

```
create domain vol (num);
DOMAIN DEFINITION WAS SUCCESSFUL
READY;
create domain model (char);
DOMAIN DEFINITION WAS SUCCESSFUL
READY;
create domain mpg (num);
DOMAIN DEFINITION WAS SUCCESSFUL
READY;
create domain date (num);
DOMAIN DEFINITION WAS SUCCESSFUL
READY;
```
Now the user is ready to define the table. He wishes to create a table with four columns: model, date, sales and mpg. Note that entries in these columns come from the four domains model, date, vol and mpg. He also informs the system that he will most likely access the table by denoting entries in the columns model or date. He types:

```
create table cars
    model (model),
    date (date),
    sales (vol),
    mpg (mpg)
key is (model, date);
```

TABLE DEFINITION WAS SUCCESSFUL
READY;

Now the table has been created but it does not contain any data yet. One way the user may enter data, is by using the "insert" command. For example, the following command enters one entire row into the table "cars".

```
insert into cars (model, date, sales, mpg):
    ('vega', 7401, 38455, 32.3)
```

INSERTION WAS SUCCESSFUL
READY;

The user may query the data by using the "select" command, which has the following general form:

```
SELECT "column-list" FROM "table-name";
```

The complete format of the SELECT command is found in Appendix I.

*-------------------*  
* Note to CDI terminal users: The CDI does not have the '<' and '>' characters as part of the regular character set. To use the "insert" command, the TRANSACT user should follow these two steps:

1) Enter the following TRANSACT command:
   `cma set input $ 4G;`
   This tells the system to translate the $ character into a '<' character.

2) Use a '+' character (upshift 1) in place of the '>' character, and a '$' character in place of a '<' character.
To query all of the data in the table just created, the user types:

```
SELECT * FROM cars
```

```
MODEL   DATE   SALES   MPG
VEGA    7401   38455  32
READY;
```

(The "*" means "all columns")

A user can change an entry in a table using the "update" command. For example, to change the volume of sales for the entry 'vega', the user types:

```
UPDATE cars SET sales = 33600
WHERE model = 'vega';
```

UPDATE WAS SUCCESSFUL
READY;

For more examples of SELECT commands, see 'GMIS: An Experimental System for Data Management and Analysis' or 'NEEMIS Applications'. The complete Syntax of SEQUEL is found in Appendix I.

GMIS users may use one of several commands which report information about the structure of the data base. These are illustrated below.

The user may type:

```
list tables;
```

**LIST OF TABLES**

<table>
<thead>
<tr>
<th>INTEGRITY</th>
<th>DOMCAT</th>
<th>CATALOG</th>
<th>CARS</th>
</tr>
</thead>
</table>

This command lists the names of all the tables that have been created. Note that the system creates three tables (INTEGRITY, DOMCAT and CATALOG) for its own use, although the user may query these three tables as if they were user created.

To obtain a description of a particular table (e.g., CARS), the user types:

```
describe table cars;
```

**DESCRIPTION OF TABLE CARS**

<table>
<thead>
<tr>
<th>NAME</th>
<th>DOMAIN</th>
<th>TYPE</th>
<th>C</th>
<th>KEY</th>
<th>INV</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL</td>
<td>MODEL</td>
<td>CHAR</td>
<td>1</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>NUM</td>
<td>0</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>SALES</td>
<td>VOL</td>
<td>NUM</td>
<td>0</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>MPG</td>
<td>MPG</td>
<td>NUM</td>
<td>0</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>
The response of the system lists the name of each column, the domain from which are taken the entries for that column, the data type of each column, (either "char" or "num"), whether a column is part of the "key", and whether or not it is inverted for performance.

The user may also request a list of all the domains that have been created, with some information about each of them, by typing:

```
list domains;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>USE</th>
<th>LLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELNAME</td>
<td>CHAR</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>CNAME</td>
<td>CHAR</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>COLNAME</td>
<td>CHAR</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>DOMNAME</td>
<td>CHAR</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>SYSCHAR</td>
<td>CHAR</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SYSNUM</td>
<td>NUM</td>
<td>4</td>
<td>---</td>
</tr>
<tr>
<td>DATE</td>
<td>NUM</td>
<td>1</td>
<td>---</td>
</tr>
<tr>
<td>MODEL</td>
<td>CHAR</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>VOL</td>
<td>NUM</td>
<td>1</td>
<td>---</td>
</tr>
<tr>
<td>MPG</td>
<td>NUM</td>
<td>1</td>
<td>---</td>
</tr>
</tbody>
</table>

The response of the system lists the names of all domains that have been created, the data type of each domain (either "num" or "char"), the number of columns in the data base that are drawn from each domain (indicated by the "use" entry), and if the domain is of type "char", the system indicates under "LLE" -- Largest Length Entry -- The numbers of characters in the largest entry present in the domain.

Note that the first six domains are created by the system, for use in the three tables mentioned above; the last four were user-created.
6. USE OF THE BULK LOADER

For all but small amounts of data, entering data using the "insert" command is extremely time consuming. GMIS has a facility for easily entering large volumes of data which are in machine-readable form into a relation data base: The Bulk Loader.

A user wishing to use the bulk loader facility must first create a file containing the data he wishes to load. This can be done in several ways; for example, the data can first be punched on cards, then loaded from the cards into the file. This involves giving the cards to a representative of the NEEMIS Operations Branch, with an indication of which account the user wants the cards loaded onto. (For further assistance in loading the cards, contact the Operations Manager of NEEMIS). Alternatively, the file may be created using the CMS editor (refer to the IBM Virtual Machine Facility/370; EDIT Guide -- available from the Operations Branch.)

After the data has been put into a file, the user must place as a preamble to the data certain control commands to tell the bulk loader what the data looks like, and where to put it. The user can do this using the CMS editor. If he had a data file named, "sample data", for example, and he wanted to insert the character string "LOADTAB TABLENAME" into the file, he would type:

```
edit sample data
EDIT:
insert $LOADTAB TABLENAME
```

The bulk loader facility, in addition to loading data, can interpret the CREATE TABLE and CREATE DOMAIN commands. These commands may be inserted into the bulk loader file the user wishes to load.

A bulk loader file for defining domains, creating a table, describing the formats of the ensuing data and the data itself is depicted in the Figure 3. Note this is a CMS file created by edit commands. The file was given the name "sample data." The bulk loader will read the file, interpret the SEQUEL commands and load the data into the newly created tables. The following command would activate the bulk loader to do this -- The user types:

```
loaddb sample data
```
$DEFDOM MODEL CHAR
$DEFDOM VOL NUM
$DEFDOM MPG NUM
$DEFDOM DATE NUM
$DEFTAB CARSALES
  MODEL MODEL
  DATE DATE
  VOLUME VOL
  MPG MPG
$PRIKEY MODEL DATE
$LOADTAB CARSALES
  MODEL  1 1 1 15
  DATE  1 20 1 23
  VOLUME 1 28 1 34
  MPG  1 17 1 19

SENDCOL
CHEVROLET 1247401 33108
CORVETTE 1547401 2078
CHEVELLE 1797401 21175
CHEVY NOVA 1977401 21464
SPORTVAN 1527401 1370
MONTE CARLO 1497401 15668
CAMARO 1797401 8787
VEGA 3027401 38455
PONTIAC 1387401 10170
GRAND PRIX 1037401 4042
FIREFIRD 1797401 3666
VENTURA 1217401 4890
OLDSMOBILE 1107401 10533

SENDLOAD
SENDINP

FIGURE 3
AN EXAMPLE BULK LOADER FILE
The above command will cause the bulk loader to process the file "sample data", and to load the data in it into the data base. Note that the bulk loading facility can only be used to load data onto a user's personal account; that is, he cannot load data onto a remote TVM by using the bulk loader. To load large quantities of data into a TVM to which he does not have direct (i.e., logon) access, the user should contact the Operations Manager.

For a more complete description of the bulk loader, and the syntax of the various loader control commands that can be inserted into a bulk loader file, see Appendix 3.
7. PROCEDURE FOR ENTERING APL

After a user has logged on to VM as described in Section 3, he may enter an APL environment as follows:

The user types:

```
vmapl
APL / CMS
CLEAR WS
```

He is now in the APL environment. In order to change his terminal to use the APL character set, he should:

-- with the CDI: Set "mode" switch (under the top cover) to "APL"
-- with the 2741: Change the typeball to an APL ball
After a user has entered the APL environment, he may wish to access a data base from APL. He may access a data base that is stored in another virtual computer (a TVM) - provided that the TVM has been previously prepared (as with multi-user TRANSACT).

Accessing the data base machine

The following is the procedure a user will follow to access the disconnected data base machine (The TVM), to retrieve or store data. The user types:

```
(copy 201 sequel sequelgrp)
```

This loads into his workspace certain APL functions we have written that are necessary to establish the proper link to the TVM.

Now to identify the particular virtual computer which has the data base he intends to access, the user types:

```
(seqopen 'acctname2')
```

where 'acctname2' is the name of that virtual computer (supplied by Operations Branch).

'NEEMIS' is the name of one account that contains many of the important NEEMIS data bases.

Note: With this command, the name must be actually surrounded by quotes. That is, an example of an actual seqopen command would be:

```
SEQOPEN 'NEEMIS'.
```
To access the data base, the user should use the 'query' function, which has the general form:

QUERY 'sequel-command'

This function will send the sequel command to the TVM. If the command is a SELECT command, the TVM will return to the user's virtual computer a set of vectors, one for each column that was asked for in the SELECT command. The user's machine will print out the names of these vectors. For example, in the following APL statement, the user asks for 1975 miles per gallon figures from the table MILEAGE:

```
QUERY 'SELECT MODEL, MPGCITY, MPGHWY FROM MILEAGE WHERE YEAR = 1975;'
```

MODEL  
MPGCITY  
MPGHWY  

RETURNED VECTORS
CONTAINING INFORMATION
ON MILES PER GALLON
The user may now handle the selected data (in the APL vectors), using any of the facilities available in APL.

**EXAMPLE:**

This example session is intended for execution by an operator who is familiar with the APL computer language. The session begins with consideration of two tables CARSALES and MILEAGE, both of which have been preloaded into a TVM. It will be seen that the table MILEAGE contains a finer breakdown of mileage data than table CARSALES. An APL program is used to aggregate the data in MILEAGE so that it can be compared with the data in CARSALES. The data in these tables are depicted in Figure 4.

**MILEAGE:**

<table>
<thead>
<tr>
<th>MODEL</th>
<th>YEAR</th>
<th>MPGCITY</th>
<th>MPGHWY</th>
<th>MPGAVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORONET</td>
<td>1975</td>
<td>18</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>MATADOR</td>
<td>1975</td>
<td>14</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>APOLLO</td>
<td>1975</td>
<td>16</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>SKYHAWK</td>
<td>1975</td>
<td>19</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>CENTURY</td>
<td>1975</td>
<td>16</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>LESABRE</td>
<td>1975</td>
<td>12</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>ELECTRA</td>
<td>1975</td>
<td>11</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>RIVIERA</td>
<td>1975</td>
<td>12</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>CADILLAC</td>
<td>1975</td>
<td>11</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>ELDRADO</td>
<td>1975</td>
<td>11</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>VEGA</td>
<td>1975</td>
<td>19</td>
<td>22</td>
<td>0</td>
</tr>
</tbody>
</table>

**CARSALES:**

<table>
<thead>
<tr>
<th>MODEL</th>
<th>DATE</th>
<th>SALES</th>
<th>MPG</th>
</tr>
</thead>
<tbody>
<tr>
<td>VEGA</td>
<td>7401</td>
<td>33600</td>
<td>30</td>
</tr>
<tr>
<td>CHEVROLET</td>
<td>7401</td>
<td>33103</td>
<td>12</td>
</tr>
<tr>
<td>CORVETTE</td>
<td>7401</td>
<td>2078</td>
<td>15</td>
</tr>
<tr>
<td>CHEVY</td>
<td>7401</td>
<td>21175</td>
<td>17</td>
</tr>
<tr>
<td>CHEVY NOVA</td>
<td>7401</td>
<td>21464</td>
<td>18</td>
</tr>
<tr>
<td>SPORTVAN</td>
<td>7401</td>
<td>1370</td>
<td>15</td>
</tr>
<tr>
<td>MONTE CARLO</td>
<td>7401</td>
<td>15666</td>
<td>14</td>
</tr>
<tr>
<td>CAMARO</td>
<td>7401</td>
<td>8767</td>
<td>17</td>
</tr>
<tr>
<td>PONTIAC</td>
<td>7401</td>
<td>10170</td>
<td>13</td>
</tr>
<tr>
<td>GRAND PRIX</td>
<td>7401</td>
<td>4042</td>
<td>10</td>
</tr>
</tbody>
</table>

**FIGURE 4**

DATA IN TABLES MILEAGE AND CARSALES
First, the user enters the APL environment (as outlined in Section 7) and accesses
the appropriate data base as shown above - for assistance in accessing the data base
used in this example, contact the Operations Manager. Then, the user must ex-
tract the data he wishes to use from the SEQUEL tables where they are stored. To
do this, he types (as above):

```
QUERY 'SELECT MODEL, MPGCITY, MPGHWY
FROM MILEAGE WHERE YEAR = 1975:'
MODEL
MPGCITY
MPGHWY
```

The user now has, in the three APL vectors MODEL, MPGCITY and MPGHWY
the 1975 data from the table MILEAGE.

Now the user wishes to calculate, using the data in those vectors, a single miles-
per-gallon figure for each car model, so that he may compare it with the figures
found in the table CARSALES.

To do this, he wishes to use the following formula:

\[
\text{AVG MPG} = \frac{1}{\frac{.45}{\text{MPGHWY}} + \frac{.55}{\text{MPGCITY}}}
\]
The following APL function, called CHANGE, implements this formula; it is used to update the column MPGAVG in the table MILEAGE. To do this, the function creates a SEQUEL "update" command, and passes it to the TVM using the QUERY function. The user should enter the following:

```
[1] CHANGE APLCODE
[2]   D<UPDATE MILEAGE SET MPGAVG = I
[3]   D<WHERE MODEL = 1
[5]   END = 1 + 0.5 * MPGAVG / (1000 - 10)
[6]   MODEL = N + 1
[7]   N<1 (N<1 0.5)
[8]   END = N
[9]   D<END (TRUNC MODEL/1.1)
[10]  QUERY APLCODE
[12] [13]
[14] [15]
```

To execute the function, the user types:

```
CHANGE
```

The function will then update the MILEAGE table. The user can then use SEQUEL commands to compare the MPGAVG column in the table CARSales, either by staying in the APL environment (using the QUERY function) or by returning to CMS (as outlined in Section 1.2) and entering TRANSACT.

(Note: TRUNC is an APL function that eliminates leading and trailing blanks from its character string argument. A simple version of the function is available from the Operations Manager at NEEMIS (617-253-5945).
9. USE OF EPLAN

EPLAN is a collection of APL functions that can be accessed from the APL environment. The functions are useful for performing econometric modelling and statistical analysis of data, as well as plotting functions. For a description of the capabilities and use of EPLAN functions, refer to EPLAN-an-APL - Based Language for Econometric Modelling and Forecasting by F. Schober. (Available from the Operations Manager)

A user may use these functions to operate on data from a SEQUEL table by loading the data into APL vectors (See section 8), and then applying these functions to those vectors. To use EPLAN, the user should:

-- log on to VM, following the procedures outlined in Section 3
-- enter the APL environment (Section 7).
-- type in the following command (to load the EPLAN functions):

\[ \text{copy 201 eplan eplangrp} \]

The EPLAN functions are now loaded into the user's workspace.
10. USE OF TSP

TSP (Time Series Processor) is an econometric modeling facility (Hall 1975). A GMIS user may use this facility by using the following procedure:

-- log on to VM, following the procedure outlined in Section 3.
-- create a CMS file containing the TSP program you wish to run. This can be done using the CMS Editor (IBM, 1975). To invoke the editor, the user types:

```
edit "filename" tsp
```

where "filename" is replaced by an arbitrary name, which the user wishes to call his TSP program. To begin inputting his program, he should type:

```
input
```

This command puts the CMS Editor into the INPUT mode; so that whatever the user types in goes directly into his file. He may now begin entering his program. When he has finished entering the program, he should store his file by doing two things: first, entering a null line (i.e., just entering a carriage return) and second, using the "file" command as follows -

The user types:

```
file
```

The file is now ready for processing by the TSP processor. In Figure 5, which follows, is an example session where the user is creating a TSP file called test.

-- invoke the TSP processor, as follows -

The user types:

```
tsp "filename"
```

where "filename" is replaced by the name of the user's TSP file. The TSP processor will now begin processing the file, printing out results (or error messages) at the user's terminal.
FIGURE 5

EXAMPLE SESSION WITH CMS EDITOR TO CREATE TSP FILE
LOGOUT PROCEDURE

From TRANSACT

If the user is using TRANSACT and wishes to log off, he should first return to the CMS environment by typing the TRANSACT "QUIT" command:

```
quit;
R;
```

His virtual computer is now in the same environment (the CMS environment) in which it was immediately after the logon procedure outlined in Section 3.

To log off, the user types:

```
log
```

The computer will print a final message and log off.

If the keyboard is locked, that is, if the user cannot enter the "quit" command, there is an alternate way of returning to the CMS environment: The user should strike the ATTN key once, and then type

```
hx
```

This will interrupt the TRANSACT program and return to the CMS environment.

The user may then type the "log" command as above.

The CMS Environment

Instead of entering the "log" command, the user may, when he has returned to the CMS environment, use any of the GMIS facilities that run under the CMS operating system, such as VMAPL, EPLAN, the bulk loader or the CMS Editor. He may also enter TRANSACT again, perhaps to use multiuser TRANSACT with another TVM.

From APL

If the user is in the APL environment and wishes to log off, he should type

```
)OFF
```

The virtual computer will print some final messages and log out.

If the user wishes only to return to the CMS environment (if, for example, he wishes to start using TRANSACT -- see the above section on the CMS Environment), he types:

```
)OFF HOLD
R;
```

The user's virtual computer is returned to the CMS environment.
REFERENCES


Donovan, J. and Jacoby, H., "GMIS - An Experimental System for Data Management and Analysis",


"NEEMIS Applications", NERCOM/MIT Workshop, MIT Energy Laboratory, September 1975.


APPENDIX I: SEQUEL SYNTAX
SQL DEL SYNTAX : RELEASE 2

\[
t-query ::= \text{SELECT} \{ \text{name}, \ldots \} \text{ FROM} \{ \text{var-name IN} \} \[ \star \] \text{table-name [WHERE sel-exp]} \\
\text{[COMPUTE var-name = [val-exp], \ldots ; query]}
\]

\[
s-query ::= \text{SELECT} \{ \text{COUNT} \} \{ \{ \text{UNIQUE col-name} \} \text{ AVG} \{ \star \} \text{ MAX} \text{ MIN} \text{ TOT} \text{ FROM} \{ \text{var-name IN} \} \text{table-name [WHERE sel-exp]} ;
\]

\[
sel-exp ::= \text{expression of sel-terms using AND OR ( )}
\]

\[
\text{sel-term ::= [val-exp \& val-exp] [val-exp \& s-query] [col-name IN } \{ t-query \} \{ (\text{literal,} \ldots ) \} [<\text{col-name,} \ldots > \text{ IN } \{ \text{t-query} \} \{ (\text{<literal,} \ldots ), \ldots ) \} [\text{val-exp BETWEEN val-exp AND val-exp}]
\]

\[
\text{val-exp ::= [col-name] [var.col-name] [\text{literal}]}
\]

\[
\alpha ::= \{ \& \& \& < \& \& \& \& \}
\]
CREATE \{var-name IN\} table-name \{SET \{col-name, \ldots\} \{\{val-exp\}, \ldots\}\} \{WHILE \{sel-exp\}\};

INSERT INTO table-name \{(col-name, \ldots)\};\{\{literal, \ldots\}\}; \{\{t-query\}\};

DELETE \{var-name IN\} table-name \{WHEN \{sel-exp\}\};

ENTITY TABLE table-name;

CREATE TABLE table-name \{(col-name, \ldots)\};\{(com-name)\}; \{\{KEY IS \{col-name\}\}\}; \{\{KEYS ARE \{col-name, \ldots\}\}\};

DROP TABLE table-name;

CREATE DOMAIN \{com-name\};\{NUM\}; \{CHAR\};

DROP DOMAIN \{com-name\};

\{CREATE\} INVERSIONS ON table-name \{(col-name, \ldots)\}; \{DROP\};

ASSESS asst-name ON \{var-name IN\} table-name; \{sel-term\};
DROP

DISPLAY

USER-NAME;
APPENDIX II: TRANSACT COMMANDS

1. Syntax
2. Explanation of Commands
6.
   S3 <command> ;

7. Any SE2ULL transaction.

8. QUIT;
EXPLANATION OF TRANSACT COMMANDS

1. RUN

Executes the transactions in the file specified by in it
in. Only the transactions described on the pages labeled
SQLDBL SYNTAX should be included in a run file. Transactions
will be processed sequentially and the system will respond
with the message TRANSACTION COMPLETED after the last
transaction has been processed. Errors may or may not cause
processing to terminate, depending on the nature of the
particular error involved. Run files must consist of
variable length records (in CM5 EDIT type RECFM V).

2. OUTPUT

Directs all output except for messages to the device
specified.

OPTIONS

LINESIZE

The user may indicate a linesize for terminal, disk, or
printer as long as it conforms to the limitations of the
output device. Unless otherwise set the following LINESIZE
values will be in effect:

for terminal - LINESIZE 80
for disk - LINESIZE 60
for printer - LINESIZE 130

TERMINAL

All output will be displayed at the terminal. If the CRT
option is not specified output tables will be folded.

CRT

This option will enable the resultant tables from the SELECT
command and the catalog queries LIST DOMAINS, DESCRIBE TABLE
to be displayed in "window" format.
commands which may be used with the window are:
up [ i  j ]
|down [ i ]
|next [ i ]
left [ i ]
nright [ i ]
continue

UP, DOWN, LEFT, RIGHT will cause a displacement of a row or column in the corresponding direction. CONTINUE will allow the system to accept the next TRANSACTION command.

DISK

Output will be directed to the file indicated by IN IT IN.

RPLNT:

Output will be directed to the printer. Any arguments from the CP SPOOL PRT command may be used to modify the spooling control options. Standard CP SPOOL defaults will be used.

3. SET

HEIGHT

If HEIGHT is ON, the system types out on the terminal a diagnostic trace of all routines entered.

STATISTICS

If STATISTICS is ON, the system keeps statistics on time spent in processing. The statistics are printed, together with the current SEQUEL statement on the STATISTICS listing file.

CASE

If CASE is SEQUEL, all quoted strings will be passed directly to SEQUEL without conversion to upper case. Thus, SET CASE LOW will enable all literal character strings entered at the terminal to remain as they are typed in. SET CASE UPPER will cause character strings to be converted to upper case.

ROWNUM

If ROWNUM is ON, the rows generated by a SELECT command will be numbered for both folded and "window" output table format.
TITLE

This option provides an optional titling facility for reports. When a title is specified it will be centered at the top of all output tables (except those output to a CRT). \textsc{set title off} stops the printing of the current title but allows the title to be retained in the system. \textsc{set title on} causes the current title to be printed with the output tables as described above. Lines of a title may be specified by using \textsc{set title} \\
/\langle title1\rangle/\langle title2\rangle/...\). A maximum of 5 lines is allowed. Each line must be enclosed in single quotes and separated by slashes. A single quote may be used in the text of a title by entering it as two single quotes.

4. \textsc{query}

Displays the options which are currently in effect for the \textsc{output} and \textsc{set} commands.

5. \textsc{cp}

Allows \textsc{cp} commands to be executed in the \textsc{transact} environment.

6. \textsc{cms}

Allows the following subset of \textsc{cms} commands to be executed in the \textsc{transact} environment:

\begin{verbatim}
ACCESS LISTFILE RENAME
CP PAINT ALTUN
DISK PUNCH SET
ERASE QUERY STATE
EXEC READCARD TYPE
\end{verbatim}

If \textsc{cms} commands other than those listed are issued, an error message will be typed and the command will be rejected.

7. See \textsc{sequel inp} for valid queries.

8. \textsc{cal}

Ends the session.

\textbf{NOTES:}

1) All commands must be followed by a semicolon (nested
transactions may require more than one input line. A transaction may be deleted by typing (%). If you wish to use (%) at a percent sign, enter (%%).

b) TRANSACT senses the end of a transaction if the last character entered on a line is a semicolon. Therefore, to avoid premature termination by TRANSACT or transaction input, make sure that a semicolon within a command (e.g. a COMPUTE clause) is not the last character entered on a line.

c) When TERMINAL LINESIZE is changed in TRANSACT, or TERMINAL LINESIZE is also changed.

d) To enter the single-user TRANSACT environment from CMS type:

    transact

    and wait for the READY; message.

e) If you want to use TRANSACT to communicate with a disconnected Transaction Virtual Machine with an ID or TVM, for example, type:

    transact tvm

    and wait for the READY; message.
APPENDIX III: BULK LOADING FACILITY
BULKLOADING OF A DATABASE

The programs LDBS and LDBB load a database from a CMS file, using free-form file loader commands and user-specified format for the data records. The file may have an arbitrary filename, filetype, and filename. The programs are in the form of stand-alone modules. They can be executed at any time (but not during a TRANSACT or SEQUEL session) after LDBS has been run. The programs add the new tables to those already in the database, leaving the existing tables unaffected. It one tries to load a table with the same name as an existing table, an error condition results.

The difference between LDBS and LDBB is only one of implementation—that is, the syntax for the loader files is the same for both. LDBS is the raster of the two; LDBB, however, enters data using SEQUEL, which allows the user to apply SequeL integrity assertions to the data as it is loaded (the assertions must be set independently, using SEQUEL).

The loaders know the input file as COMPILE in the program. Hence, the person using one of the loaders to load from his input file called, for example, "INCLUDATA Integrity", in his h-disk, must link "COMPILE" to "$INCLUDATA" by means of a FILEDR:

FILENAME COMPILE DISK INCLUDATA INTTYPE 1 (FLRD)

(It is not necessary that the input file be on the h-disk.)

An easy way to invoke one of the loaders without entering the FILEDR is to use the one of the EXEC commands LOADB or LOADB as follows:

LOADB1 in it in

or

LOADB2 in it in

These EXECs will issue the FILEDR for the arbitrary file "in it in", and will then invoke LDBS automatically.

The INCLUDATA file is a regular CMS file of 32-character records (ASCII). The loader commands consist of one or more records in the INCLUDATA file. Each command control card is read in a free format—each argument must be separated by one or more blanks. Note that this is not true of the data records themselves—these must be in a format specified in the LOADB/CMD card. There are five major commands:

1. EXEC FOREIGN Command
format: `DEFINE <domain name> <domain type>

<domain type> can be NUL (for numeric) or CHAR (for character).

Each command defines one domain.

2. Define Table Command

This command defines the table name, and for each column in the table, it identifies both the column name and domain name.

format:
`DEFINE <table name> <column name> <domain name> ...

The order of the columns (left to right) in the table is the same as the order of the <column name> fields in the format description. The Define Table command must immediately be followed by a Define Primary Key command.

3. Define Primary Key Command

This command must immediately follow a Define Table command. The format is:

`DEFINE <column name> ... SENDKEY

Note that though it is not necessary for a table to have a primary key, it is necessary to include a Define Primary Key command immediately after a Define Table command in INPUTDATA. If there is no primary key, there must be at least 1 space between DEFINE and SENDKEY.

(Also note: the SENDKEY marker is the only syntax requirement that the loaders impose that makes a loader file designed for the older versions of the loader incompatible with these (newer) loaders—the older loaders did not require SENDKEY.)

4. Load Table Command

This command tells the system what table is being loaded, and in what format the data to be loaded is found; it also supplies the data in that format. Note that the table indicated in the command must have been previously defined.

The command has two parts: the format description and
the data cards (records).

FORMAT DESCRIPTION

LOADTAB <table name> <column name> <1st card #> <1st position> <last card #> <last position> ...

Each column to be loaded has a group of five tokens associated with it in the format description:

column name
1st card # -- the index (relative to the next data card the loader will process) of the first card with data to be loaded into this column (this is normally 1—that is, the loader starts reading data off the first data card it finds—it the next one).
1st position -- the leftmost card column (in the <1st card #>th card) where the data to be inserted into this column is to be found.
last card # -- the index, relative to the next data card (that is, the same data card to which <1st card #> is relative) of the last card where data for this column is located
last position -- the rightmost card column on this last card where data for this column is located.
The effect of each set of 5 tokens is that the loader inserts into the column <column name> all the data between <1st position> on <1st card> and <last position> on <last card>.
The order of the columns is immaterial.

If some columns in the table being loaded do not appear in the format description the system assumes a default value of zero for that column. This is a potential problem since some columns have underlying domains of type CHAR. So, it is advised that all columns in the table have an entry in the format description.
The records are processed one by one in the order as they appear in LOADTAB. If a column name appears more than once in the format description, an error condition will result and the program will terminate.
DATA RECORDS

Data records are 60-byte card image records which follow their respective format description. The format of a data record is completely determined by the format description. A set of data records has the data for one row of a table (one tuple). Since one row on a table may require more than a single 60-byte record of the input file, the loaders allow each input row of a table to extend over n data records. These data records are assigned relative numbers 1, 2, ..., n. The relative numbers are used in the <1st card> and <last card> fields of the format description to indicate the starting and ending points of a given column. It is therefore possible to have the data for a single column extend over more than one record.

If the data for a column, after being read from data card(s), contains nothing but blanks, the loaders assume a default of 0 for columns of data type NUM, and UNKNOWN for columns of data type CHIH.

End-Load Marker:

The last data card must be followed by

ENDLOAD

5. END-OF-INPUT COMMAND

The last record of INPUTDATA must be an end-of-input card whose format is:

ENDINP