Discussion Paper No. 391
A Data Base for Operations Research Models
by
Edward A. Stohr
and
Mohan R. Tanniru

Northwestern University
Graduate School of Management
Evanston, Illinois 60201

July, 1979

*Graduate School of Business, University of Wisconsin, Madison, Wisconsin 53715
A DATA BASE FOR OPERATIONS RESEARCH MODELS

Edward A. Shore
Graduate School of Management
Northwestern University
Evaston, Illinois 60201

and

Robert W. Tanew
Graduate School of Business
University of Wisconsin
Madison, Wisconsin 53706

ABSTRACT

This paper develops the design of a data base system to support operational research models in the context of an integrated planning system involving a number of different users and computer programs. The requirements for such a system are described, a 'network' data base schema is developed and the schema and command language are illustrated through a specific example.

1. INTRODUCTION

Computerized planning systems and models have been used increasingly in recent years [11]. However, there are relatively few examples of truly integrated systems involving multiple corporate departments and several layers of decision-making from the most detailed planning at the production level to more strategic financial summaries and analysis for budgeting and strategic decision-making. Much research has been concerned with the complexity of such systems but the ways in which computer can provide information and analytic support to the decision-makers. The general considerations for a dedicated support system (DSS) have been outlined in [1], [2] and [9]. In this paper we describe how data base management system techniques can be used to keep track of the complex information flows between different decision-making units and computer programs. The objectives are: (1) to increase the integrity of the planning process by ensuring that the correct inputs are used by the various computer programs and by providing an 'audit trail'; and (2) to add the value by facilitating data retrieval and processing and by providing an environment in which it is easy to test and keep track of the effects of different model assumptions and data values during sensitivity analyses.

The advantages of data base management systems (DBMS) in assuring data integrity and security, reducing data redundancy, maintaining multiple relationships between data items and providing the basis for powerful retrieval languages are well-documented [10]. To date applications have been primarily in an information retrieval or data processing environment. The 'data base approach' is to design the logical structure (schema) for the data to allow for the estimated data requirements of different users and programs. For modeling and planning systems this might appear to be a difficult task since the data relationships to be maintained often differ from model to model and in the context of any one model may be abstracted only through extensive transformations aimed at finding the most important variables affecting the objective of success. What is needed here is a retrieval language and data base system which is capable of dynamically transforming logical data structures to the needs of a particular model of a particular time. In logical structures we must the data have 'salient' (executive administrator's view) and more particularly the class of 'knowledge' (programmer's view) which can be derived from the schema. A mapping language for this purpose has been outlined in (4).

In this paper we concentrate on a different aspect of the use of DB systems to support...
determination—usually its use to help manage
the data flow and models during sensitivity
analyses in both the simulation and production
use of a planning system. The computer pro-
gram themselves are represented by record
occurrences in the data base which can be re-
trieved and accessed by high level commands.
We describe a schema based on the CRISP Data
Base Task Group (CBTG) 'network' approach (3)
which is one problem-specific and can be used
in a wide variety of applications. Related
work systems in (2). The present paper is an
examination and illustration of previous work on
this subject (12).

3. REQUIREMENTS FOR A CONCEPTUAL DESIGN

Some of the problems which must be faced
in the construction of an integrated planning
system are:

(1) The necessity to transform data into the
specific format required by a particular soft-
ware package or computer program.

(2) Difficulties in data merging conventions
when different programs and packages are to be
coordinated.

(3) The fact that the planning system might re-
quire alternative mappings of the same data in-
puts into different values of the same output
variables. Common examples occur with the
planning system is to be capable of existing sit-
uations under different depreciation or inventory
management methods.

(4) The necessity to coordinate a large num-
ber of programs which interact with each other
through their inputs and outputs and must be
executed in certain sequences to ensure valid
results.

(5) A complex data integrity problem must be
solved in that data redundancy is a desired trait
rather than something to be avoided. The plan-
ning system should be capable of generating and
maintaining alternative values of output vari-
bles at each of several stages and levels of
the planning process. It should be possible
and simple to cross, display and, if necessary,
adjust the entire history of progress—on
assumptions and data values which underlie
any output. This is useful, for example, in reca-
tive budgeting processes involving negotiation
between corporate divisions and headquarters.

(6) Because many users may be involved in the
planning process adequate security measures
must be provided. Many users will have spe-
rified rights to access and modify data and
to initiate progress runs.

(7) Because planning systems must be capable
of evolution and extension they should be built
in a modular manner and easy to modify and
maintain.

(8) Adequate programmer and user documentation
should be available.

(9) Since many users will be involved with
different levels of programming expertise, the
commands language must be easily understandable
and forgiving of mistakes. Users should feel
'comfortable' with the system in the sense that
they know where they are at all stages and can
have confidence that the data supplied when
is up-to-date and correct. In this connection
the planning system should facilitate the com-
munication process so that a given user can
readily determine the relevant actions of other
users—a form of electronic mail.

In the following section we outline the
design of a system which attempts to meet many
of these requirements.

3. BACKGROUND AND TECHNOLOGY

The data base design will be illustrated
using the network approach (1) primarily be-
cause such systems are more readily available
commercially. An equivalent system design could
readily be implemented in a hierarchical (8)
or relational (6) view.

In the network approach the schema or logical
description of the data involves data items,
record types and set types. A data item is the
smallest unit of data which has a meaning. Sub-
sets of data items can be associated with each
other by specifying that they belong to the
same record type. Record types are in turn re-
lated to each other via the 'sets' construct
which constitutes a one-to-many relationship
between an 'owner' record type and a 'member'
record type. These constraints are sufficient
to represent data relationships of arbitrary
completeness (see the graphical representation of the proposed data base scheme in Figure 3). As may be seen from the stored database (or "realization") of the scheme will consist of an arbitrary number of "instances" of each record-type corresponding to the data items (see Figure 6 and 7). After the schema has been designed and the data loaded, retrieval according to the DBMS specification involves the use of a data manipulation language (DML). The DML allows the user the ability to traverse the data base network in a record-by-record manner. Since this is a fairly "low level" language, unsuitable for non-expert users, the proposed command language will contain many "pre-programmed" retrieval programs.

The planning system is being implemented in the APL programming language using the EBRS Data Base Management System. 113. The schema is described using a "Data Definition Language" (DDL). A partial definition of the proposed schema using the EBRS DDL is illustrated in Figure 6.

In the implementation advantage has been taken of some of the features available in APL which are not duplicated in other languages:

(1) The language possesses an "execute" operator which allows the construction of commands as character strings which can subsequently be executed. Using this feature character strings consisting of sequences of commands to open and read files and execute programs ("subroutines") in APL can be stored in the database and retrieved and executed as required.

(2) Functions can be converted to character matrices, stored in the data base system in this form and subsequently retrieved. A group of such functions can be stored and retrieved in a similar manner. In the following "Function Group" will be equivalent to a program (with its 'main' and sub-programs) in another programming language.

(3) Data arrays of any dimension can be stored as "components" of a random access file. Furthermore it is a simple matter to encode character and numeric arrays into a character string which can be stored as a single file component.

The above capabilities of APL have been used to simplify the design and will simplify the operation which follows. However the logical design could be implemented in other programming languages with only minor changes.

Before describing the scheme and the processing commands which operate on the database it is necessary to define some of the terms we will be using. A 'Function Group' consists of a number of Nodes. Each Node in turn consists of a number of Processes. A Process involves the use of a Function Group in the context of a model. A Function Group ("program") executes a sequence of operations on the data and represents a computational step. A Function Group may consist of one or more cooperating APL functions. A Process is associated with only one Function Group however the same Function Group may be used in a number of different Processes in the same or different models. The specification for a Process includes a definition of the environment in which the associated Function Group is to operate including the naming of the inputs and outputs (with renaming of variables where necessary)—see Figure 4. Thus a Function Group might execute a linear programming algorithm and might be used as a Process in both a production planning model and a cash management model.

To maintain data integrity and to ensure that all inputs which affect the results of a Process are recorded all data is stored in direct access files which are entered by the user. This includes data entered in response to program prompts if that data affects the values of output variables. It is necessary to distinguish between different kinds of data inputs to Processes: 'Input Parameters' to a Process is data previously entered by another process. Since this data entry functions (e.g. loading of data from a magnetic tape) are performed by Processes almost all data is exogenous. The most important exception to this occurs with processes accepting on-line input from users. Such data is referred to as 'On-line Input.'

A number of algorithms (including commercial planning simulation packages) operate by
interpreting a logical specification of the user's problem. Often, data values (parameters) are embedded in this logical statement. However, it is better practice for many reasons to make the logical statement independent as possible of the data by expressing parameters as variables. To explore the best model to represent the logical documenting capabilities of the systems this form of 'logic' is treated separately and is referred to as "Problem Statement."

During the execution phase of a model the data inputs and outputs of each process are recorded in the data base. The initial set of data for a Process will be referred to collectively as a Data Base. An adjustment of the values of a base case for sensitivity testing or other purposes is a Case. To save data, a process is represented only by the APL instructions which transform values of the base case data. A Run is the record of a computation performed for a specific Case of a Process. There is a 1:1 correspondence between a Run and the associated output data.

A DATA BASE SCHEMA FOR DECISION SUPPORT

The database design includes four APL files. The Data Dictionary File contains definitions of: (1) the Models, (2) the Processes including their input and output variables, and (3) the Data Base Definitions (format) for input variables. The Data Base Dictionary contains the logical relationships between Models, Processes and the Data Bases, Cases and Runs which constitute the computational history of each Process. The term "dictionary" is used because the data base records primarily contain 'pointers' to the Data Dictionary and Model Data Files in which the data values themselves are stored. The Data File for a Model contains the actual input and output data values and character matrices describing the nature and purpose of each variable, Cases and Runs. The Function Library contains the code for all functions used by the Planning System; either individual functions or complete Function Groups may be accessed. All functions must be assembled in the Function Library before they can be used by a Process. This process of registration also enhances programmer documentation. The Function Library is part of another 'system development' system and will not be further described here.

Figure 1 shows a partial schematic representation for the Data Base Directory in the DBS while Figure 2 below is a graphical representation of the schema.

![Figure 1: Partial Schema Definition](image-url)

Declared data item names in Figure 1 are enclosed 'brackets' containing of user-given names for the required instances. The suffix 'PN' indicates that the data items in a 'pointer' (value of a component number in another file). Data item names ending with 'DESCPT-PN' are pointers to five-character character matrices describing the associated record features. The meanings of the other data items are:

MODEL-NAME = name of model Data File
PROCESS-NUMBER = name of APL workspace name of Function Group
PROCESS-PN-PN = pointer to character matrix representing the definition of APL commands required to execute the Function Group.
CASE-C-Desc-PN = pointer to character matrix giving the APL commands representing the changes to the database associated with this particular Case.
CASE-PN-NAME = the name of the Problem Statement associated with this case.
MODEL-PN-PTN = pointer to the data output from the Run.
FUNCTION-PN-PN = pointer to the character matrix containing the Problem Statement.
The Planning System is designed to facilitate automated running of all or some of the processes within a model at the option of the user. The processes recorded within the P-F-P set are stored in their 'natural' precedence order (the first record in the set is the first process to be run, etc.). The Planning System has the information to ensure this (see Section 3). The records in the P-F and P-C sets are maintained by the Planning System in LIFO order under the assumption that the latest beware and case are the most likely to be used. If this is not the case the user can indicate the particular beware and cases to be used prior to running the model (for any valid subsequence of processes).

The various cases relate the input and output data generated during sensitivity analysis. The C-EXEC-PFT for the last CASE record in the B-C set (which is loaded at the same time as its associated B-DEC-PFT record) actually points to a character array of ALL instructions which loads the entire set of data for the RUNCAST. The C-EXEC-PFT's for other CASE records point to

![Diagram](image_url)
character matrices of the APL instructions which convert the original raw data values to
those for the Case. Thus the last record in the P-O set is usually accessed first and success-
ive 'Delete' records are obtained until the desired one is reached. The PS-ROXX field in the
Case record cross-references the PROJ-STATEMENT record associated with the version of the Pro-
blem Statement (if any) used by the Case. The P-O set associates a Case of a Process with the
Exogenous Input used (these inputs are by definition the outputs of Runs of other Pro-
cesses). The 0-00 record type and D-O set were introduced to facilitate retrieval of information
pertinent to specific involving data. For example, one might wish to display yesterday's RUN records
or some last month's processing history.

Out of every Problem Statement are asso-
ciated with a Process through the P-O set.
Note also that the APL instructions pointed to
by C-BLK-PAI may contain editing instructions
to allow experimentation with various changes to a Problem Statement. The P-O set contains one
or more INUIT records. The I-NAME LIST for
Exogenous INPUT data is a character string pre-
fixed by the model name (if from a different model) and Process name of the Process which gener-
ates the data. Array variable names are followed by the array dimensions in parentheses to allow data validation (see Figure 1).

The P-O set will contain only one mem-
ber record. However, the set of output variables
can be partitioned for reporting purposes. The
D-O set associates one or more alternative re-
port definitions with the variable list of the
owner record of the set occurrence. A general-
ized report generator system accepts a report
definition input interactively by the user and
produces it as a dedicated setting. This is included
into the data Dictionary during the "registration" process (see below) and is pointed to by the P-O set data item. Standardized input or
output variable data displays can be obtained
easily just by revising the necessary dictionary
for storing a report definition.

5. THE MODEL DEFINITION PHASE

During this phase the data dictionary is
loaded with data representing definitions of all components of the model. At the same time in-
stances of data records of type 1 through 5 are
loaded into the Data Bank Directory and any
user-defined functions and Function Groups are
tested into the Function Library. The loading
of this data is accomplished by "registering" the
Model, its Processes, Function Groups, Re-
port Definitions and (optionally) client Var-
iables using an interactive "REGISTRATION" function as outlined below. A FUNCTION NAME
INPUT VARIABLE LIST: 
\( X_1, X_2, X_3 \)

OUTPUT VARIABLE LIST: 
\( P(A), 700 \)

MODEL MOTOR PROBLEM

MODEL DESCRIPTION:
The model is used to determine the production mix for the following year. The model is a linear program (see Principles Definition). Processes, responsibilities, and data flow are as follows:

I. PROCESS MEASURES:

- Accounting, weekly cycle. Process reads values of variables: \( M_0, N_0, P_0 \) from a master tape and stores them in model data file.

II. PROCESS VARIABLES:

- Accounting, monthly cycle. Process computes production quantities, \( Q_1, Q_2, Q_3 \). (Keep these variables for data file allocation purposes.)

- Inputs: \( M_0, N_0, P_0 \). Outputs: \( Q_1, Q_2, Q_3 \).

III. PROCESS PROCEDURES:

- Production, weekly cycle. Process inputs a linear program to determine production plan.

- Inputs: \( X_1, X_2, X_3 \).

- Outputs: \( Q_1, Q_2, Q_3 \).

IV. DEFINITIONS OF VARIOUS VARIABLES:

- \( X_1 \): Number of resources consumed.
- \( X_2 \): Number of products.
- \( X_3 \): Number of difficult to produce products.
- \( X_4 \): Number of expensive to produce products.

V. INPUTS:

- \( X_1 \): Resources constraint limits.
- \( X_2 \): Prices of products.
- \( X_3 \): Direct costs.
- \( X_4 \): Costs of difficult to produce products.
- \( X_5 \): Costs of expensive to produce products.

DEFINITION OF PRODUCTION QUANTITIES:

- \( Q_1 \): Value of optimum production plan.
- \( Q_2 \): Value of optimum quantity.
- \( Q_3 \): Value of optimum production quantity.

DEFINITION OF OBJECTIVE FUNCTION:

- Minimize \( Q_1 \) subject to constraints: \( X_1, X_2, X_3, X_4, X_5 \).

This model is used to provide user- and programmer-level documentation of the model.

The Model Definition Phase will be illustrated by a simple application of linear programming to a production planning problem. The data flows involved are shown in Figure 1. The model itself is explained during the Regression Process which is illustrated in Figure 2. In step 3 the Planning System is linked into the user's workspace. In step 3 the Model is 'activated'.

Figure 3 shows the registration of the model, PROROBP, which runs the linear program. The LPGRAPH Function Module referred to accepts a Problem Statement in the form of a character matrix PROBINFO which defines the problem in 'symbolic notation' format (see Figure 5) (11).

Figure 4 illustrates the Process PROCES1P. Program PROCES1P illustrates the model 1's substitutions for 'C' in algebraic notation.

Note that the model uses a variable, \( u \), for the desired volume limits for production whereas the linear programming uses the variable \( l \). This difference in naming convention is achieved by the "equivalencing" phase MODER (a) to the input variable list for the process PROCES1P (Figure 4).

A realization of the data base Directory schema at the end of the Model Definition Phase is shown in Figure 5.
**Figure 6.** The four processes shown in the figure are called "READTAP," "CHARG," "PLANTS" and "PRODCU." They have been stored in the I-O-F set in a feasible execution sequence.

**Figure 7.** Running Phase of Model.

(*Key Command are Undated, Computer Prompts are Underlined)*
Because of space limitations only a few comments will be illustrated. Considering the previous interaction, Figure 7 illustrates the computer interaction of the Accounting Department running processes SAMPLED and COMMAND which uses the functions LOADFILE and LOADFILE respectively. The Planning system is run in 'manual' mode, i.e. each process is loaded separately - the user's workspace and the associated function is ended by the user. Each function uses Planning System Command language to read the data and access the results for the run.

Figure 8 shows the sequence of steps from step 7 to 10 after this interaction. The following is an explanation of the steps in Figure 8:

1. LOAD PLAN loads 'core' planning system functions and 'opened' the ProcFile database.
2. READ/FILE opens the database directory, the Data Dictionary and the PROCESS Data File for the PROCESS model.
3. LOAD/PROCESS accesses the Process record for LOAD/PROCESS. It does the PROCESS field to load the correct Function Group into the workspace and the SSIC-TH field to access and print the execution instructions for the process which were stored in the Data Dictionary. Since the P-1 is not in empty LOAD/PROCESS runs.
4. The NOW function can be used at any time to retrieve and print previously registered definitions and explanations of the Model, Processes, Function Groups and Global Variables.
5. The READFILE user function triggers commands to the computer operator to load the relevant magnetic tape and read the data into the next available "slot" in the Data File using the 'process' given in the $USER-FILENAME field of the OUTPUT record corresponding to the current PROCESS record. The Command Number in which the data has been placed is stored in a global variable for use in any subsequent Process command.
6. The SOFTWARE prompt asks if a DTW record is to be $-used in the Data Base Directory. The $USER-DTW-TH field gets the value of the Component Number from the DTW Variable of the previous step. The user's description is stored as the name available component in the Data File and the associated Component Number, in the $USER-FILENAME of the DTW Record.
The ENCAPES command clears the functions and global variables related to the RESERVED process from the workspace.

The CAPTURE Function Group (CNPC) is loaded into a component. In this case, there is an urgent input. However, since the CNPC field equals "NOCAP" the CAPTURE function does not itself read the data into the workspace as global variables...this will be done during the execution of the function CNPC in step 9.

The user-defined function CNPC is executed. Prior to reading the input data the function checks that it is being run under the "Process" CAPTURE and that the required compendium data were derived by the Process RESERVED. Since the user has previously specified the compendium cases to be used the latest ANSI record in the C-B set for RESERVED is accessed and the user is asked if the data for the box with HAND-AIDDATA should be used. In this case the user responds in the affirmative. If the user had said 'no' other ANSI records in the current C-B set would have been accessed in turn until the desired version of the data was obtained. The function then asks for the Open Input (OPEN) which is input from the terminal. Having located and read the input data CNPC then asks the user if a BASECASE record is to be stored and the user always answers 'yes'. Since this is a new BASECASE a BASE record is automatically generated and the B-A and B-C sets are connected for Process CAPTURE.

The API instruction in Component 6 of the Data File in Figure 8 is generated and stored automatically by the system. Then executed this instruction will retrieve the Base Case data from Component 3. At this stage the system records the source of the compendium data used by this run of the CAPTURE process. This is done by connecting (in the D-A-B set) the ANSI record of the RESERVED Process to the CASE record just established for the CAPTURE Process. After performing the computations the function displays the results and stores the corresponding ANSI and output data as shown.


