Data model processing

by MATTHEW B. KOLL, W. TERRY HARDGRAVE, and SANDRA B. SALAZAR

National Bureau of Standards*
Washington, D.C.

ABSTRACT

The Data Model Processor (DMP) is an interactive tool for defining and evaluating data models. It is based on Positional Set Notation, a formalism for uniform representation of data modeling objects. The DMP allows the user to enter a set-theoretic description of a data model’s structures and a definition of the model’s primitive operations based on positional set operations. Based on the data model definition, the DMP then emulates a database management system (DBMS) implementing that data model. It allows the user to play various roles associated with a DBMS, such as database definer and end user.

This paper gives an overview of the DMP and discusses its foundations, namely, Positional Set Notation and a Positional Set Processor. It traces an example showing how the DMP has been used to model the relational data model. (Hierarchical and network models have also been implemented on the DMP.) Future applications of the DMP are considered.

*This paper is a contribution of the National Bureau of Standards and is not subject to copyright.
INTRODUCTION

The study of “data models” is an important aspect of database management technology. A data model is defined here as a collection of data structures plus a collection of primitive operations used for database management. Each database management system (DBMS) may be viewed as an implementation of some underlying data model. While three data models are most widely discussed, many others have been and continue to be proposed.1

More rigorous definition of and comparisons among data models are needed. Better selection and use of DBMS’s could result from improved understanding of the various data models’ strengths and weaknesses, and from detailed specification of their structures and operations. A simple, general vehicle for formal analysis of data models could be a valuable tool. Such a tool may aid in database conversion and translation as well.

The Institute for Computer Science and Technology (ICST), within the National Bureau of Standards, is charged with establishing federal standards for database management systems. To support this effort, ICST is interested in developing a more structured approach to the analysis of data models. The Abstract Database Models project at ICST has developed and implemented the Data Model Processor (DMP), a software package for formal specification and analysis of data models and their implementations.

The first prototype DMP was recently completed and has been used to study the behavior of a relational, a tree-structured, and a network-structured data model. A significant feature of the DMP is that it not only provides for a common definition language for various data models, but that it allows each data model definition to be implemented by emulating a DBMS embodying that data model (as shown in Figure 1).

This paper describes the DMP and how it may be used. First we give an overview of the DMP. Then we discuss its foundations, namely, Positional Set Notation2 and the Positional Set Processor.3 Positional Set Notation (PSN) is a set-theoretic notation that enables uniform representation of the data structures for various data models. The Positional Set processor (PSP) is a sophisticated tool for manipulating (e.g., storing, retrieving, combining) positional sets (p-sets). Next, we show how the DMP can be used to define and emulate DBMS’s implementing the relational data model. Finally, we discuss planned and potential applications of the DMP.

THE DATA MODEL PROCESSOR

Overview

The Data Model Processor (DMP) is an interactive tool for defining, testing and evaluating data models. It has been implemented in the C programming language under UNIX and runs on either a PDP-11/45 or LSI-11/23. The DMP accepts formal definitions of the structures and operations of a data model from the user. It also allows the user to define and manipulate various features of an implementation of that data model (i.e., a DBMS). As described in Figure 1, after the DMP has been given the specifications for defining and implementing a DBMS for some data model, it then emulates that DBMS.

The DMP recognizes the following different human roles involved in the life cycle of a DBMS:

1. Data Model Definer (DMD)
2. DBMS Implemeneter (DI)
3. Database Definer (DBD)
4. Access Definer (AD)
5. Database Populator (DBP)
6. Query/Access Language Definer (QLD)
7. Data Transformation Definer (DTD)
8. Data Manipulator (DM)

The DMP first presents the user with the master menu of roles. The user may play any role, provided that the prerequisite roles have been fulfilled. The dependencies between roles are shown in Figure 2. Dotted lines indicate that the lower role may, but does not always, depend on the higher role.

User Roles in the DMP

The Data Model Definer (DMD) is the most important role in the DMP. The activities of the other roles are structured by

---

From the collection of the Computer History Museum (www.computerhistory.org)
information provided by the DMD. For example, the database definition and population phases consist entirely of filling in structures previously outlined by the DMD.

The DMD role is divided into the following four sections:

1. Declaring the basic CONCEPTS of the data model
2. Outlining the P-SETS used to represent the concepts
3. Identifying the SOURCES of values to populate the sets
4. Defining the primitive OPERATIONS allowed on the p-sets

In order to define a data model, the DMD must use PSN to define the structures (i.e., p-sets) that will be manipulated by the data model. The DMD outlines p-sets using the TEMPLATE command, discussed below (see Figure 3). These p-sets represent the basic concepts of the model (e.g., relations, trees). These structures are usually partitioned into two classes: data definition structures and occurrence structures. However, the DMD has substantial flexibility in these definitions. Note that the DMD is the only role requiring knowledge of PSN.

In addition to the stored structures, the DMD must define the primitive operations to manipulate these structures. These operations become available to the Data Manipulator (DM), Query Language Definer (QLD) and Access Definer (AD). Currently, operations are written in the C programming language. As seen in the example below, the operations consist mainly of calls to PSP operations.

The DBMS Implementor (DI) completes those aspects of the data model that were not completely specified by the DMD. The term “implementation” has a substantially different meaning in the DMP context than in its traditional context. Implementation, in its traditional sense, is automatic because structures are completely specified in PSN, and most or all operations are completely specified by the DMD. Except for defining the elementary sets (e.g., integers, character-strings), the DI is allowed to define only the sets and procedures which have been explicitly designated to the DI by the DMD. Such designated sets and procedures would be those needed for implementation but not considered part of the data model definition (e.g., valid names).

The Database Definer (DBD) enters the data definition for some databases. For each concept designated in the SOURCE section to be populated by the DBD, the DMP prompts the DBD to provide values to populate the p-set(s) representing that concept. The DMD provides the “outline” for the p-set with the TEMPLATE statement; the DBD provides the values to fill in the p-set.

The Access Definer (AD) proceeds in the same manner as the DBD. The AD populates p-sets which were outlined in the P-SETS section and designated (in the SOURCE section) for population by the AD. He assigns users to user classes, and defines a perspective (e.g., view, subschema) of the database for each class. Access control mechanisms and granularity vary with the data model and implementation.

The Database Populator (DBP) proceeds in the same manner as the DBD and AD. Traversing the appropriate template, the DMP prompts the DBP to populate each p-set designated for DBP population in the SOURCE section. These filled-in p-sets (usually the actual database) are stored
as files and are manipulable by the DMP primitive operations (through the PSP).

The Query/Access Language Definer (QLD) enters tables to define the syntax and semantics of a query or access language. These tables are inputs to the Query Processor (QP), an augmented macro processor. QP accepts the specification of the syntax of the language in a tabular form that is reminiscent of SNOBOL-like programming languages. This technique is different in one simple but significant way: a mechanism is provided to pass information from the syntactic grammar to the primitive operations specified in semantic tables. This allows the study of a single syntax that has multiple semantic interpretations.

The Data Transformation Definer (DTD) is the only role that has not yet been implemented. We expect that role to be similar to the QLD. The DTD will enter tables and/or programs based on PSP operations in order to define mappings between different implementations or data models.

The Data Manipulator (DM) may use the primitive operations or a language defined by a QLD to manipulate the data in a populated database. The nature of the session with the DM depends to a large extent on the language of the data model that has been defined. The DMP asks the DM to enter his user-id, the model, implementation and database, and the language he will use so it can link to the appropriate parser.

Commands issued by the DM are then passed to that parser (probably QP), which then executes the designated primitive operation(s), which, in turn, execute PSP commands, returning the response (edited if necessary) back to the DM.

Foundation

The DMP uses PSN for representing the various data models' data structures and the PSP for manipulating those structures. We provide here a brief description of PSN; more detail is available elsewhere. The essence of PSN is the recursive definition of the p-set:

\[ s = [x_i@p_i \ldots] \]

where the \( x_i \) are either atoms (i.e., numbers or character-strings), or p-sets; and the \( p_i \) (Position IDentifiers—PIDs) are either atoms, or # (the null PID).

The \( x_i \) are the elements of the p-set; the \( p_i \) are the positions of their membership. The \( p_i \) occurring in a p-set need not be unique. A pair, \( x_i@p_i \), is called a duplex. The duplexes within a p-set are unordered; a p-set may be thought of as a set of ordered pairs.

There is a mapping from p-sets, \( s \), to mathematical sets, \( s' \), such that:

\[ s = [x_i@p_i \ldots] \Rightarrow s' = \{< x, p > \ldots\}. \]

That is, for each p-set there exists a corresponding set of ordered pairs.

P-sets are used to model data modeling objects. The three-level relationship among data modeling objects, p-sets, and mathematical sets is shown in Table I.

<table>
<thead>
<tr>
<th>Data Modeling</th>
<th>PSN</th>
<th>Mathematics</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Jones, 30)</td>
<td>[Jones@#, 30@#]</td>
<td>{&lt;Jones, # &gt;, &lt;30, # &gt;}</td>
</tr>
<tr>
<td>&lt;Jones, 30&gt;</td>
<td>[Jones@1,30@2]</td>
<td>{&lt;Jones, 1 &gt;, &lt;30, 2 &gt;}</td>
</tr>
<tr>
<td>Name, Age</td>
<td>[Jones@Name, 30@Age]</td>
<td>{&lt;Jones, Name &gt;, &lt;30, Age &gt;}</td>
</tr>
</tbody>
</table>

The PSP is an access mechanism used to manipulate these objects. The PSP is actually a collection of about 40 operators for manipulating p-sets, each of which may be called independently at the UNIX shell level or as a subroutine. While the PSP provides many of the features of a DBMS, it is NOT a DBMS, lacking such important features as a data definition facility. Unlike most access methods, however, it has a very powerful query facility. Also, while most access methods exist for performance improvement and convenience, the PSP exists to allow precise specification and manipulation of mathematical objects.

PSP operations can be broken into four functional groups. The Classical Set Operations include union, intersection, cardinality, etc. When applied to positional sets, they are analogous to the traditional set operations.

The Positional Set Operations provide the user with the following basic database functions: retrieving, updating, adding, and deleting. Some of these operations resemble those available in relational query languages. In particular, there is a RANGE command for linking range variables with p-sets and a CREATE command which performs functions analogous to the SELECT, PROJECT and JOIN found in relational algebra. There are additional operations to return the (classical) set of elements or the (classical) set of PIDs for a p-set and to distribute a PID over a classical set (that is, to un-nest a nested set).

Three other operations of this group are noteworthy: (1) TEMPLATE, (2) POPULATE, and (3) CONFORM. As seen in Figure 3, they play key parts in the DMP. TEMPLATE allows the user to specify a template and a set of constraints that defines a class of p-sets. That is, templates are a kind of metalanguage for p-sets. POPULATE traverses a template and prompts the user to enter values at the appropriate PIDs. CONFORM is a predicate that compares a p-set to a template and set of constraints. It returns true if the p-set is structured according to the rules given in the template and if the values within the p-set conform to the specified constraints.

Some data models (e.g., CODASYL) require the manipulation of sequences. To take advantage of the ordering within sequences, the PSP has Sequence Operations that allow manipulation of sequences as a special form of positional sets. For example, special insert and delete operations renumber sequences after changing their contents.

The Utility Operations (e.g., copy, print) provide additional capabilities needed for use in an interactive environment.
SAMPLE APPLICATION

This section shows annotated excerpts from application of the DMP to the relational data model. The full definition and exercising of an implementation of the relational model, which has been recently completed, is too long to include here. We also omit details of data model processing for hierarchical and network models for lack of space. We hope that the segments shown will help explain how the DMP works.

The relational definition shown is not meant to represent THE definition, but rather, is designed to show one possible definition of the relational model that is generally consistent with common understanding.5, 6, 7 Where possible, system output is given in lower case and user input in upper case, with comments enclosed by "/*" and "*/". Wavy lines indicate that a portion of the interaction IS omitted. In some cases, additional comments are inserted for clarification.

******************* data model processor*******************

commit option: DMD
enter name of model: RELATIONAL

concept definition section
declare concepts ('$END' to terminate):

> REP_DEF DOMAINS WITH DOM-DEF; /*
> REPRESENT DOMAIN DEFINITIONS WITH
> THE P-SET "DOM_DEF" */
> REP_DEF RELATIONS WITH REL-DEF; /*
> REPRESENT RELATION DEFNS WITH "REL-DEF" */
> REP_OCC RELATIONS WITH REL-OCC; /*
> REPRESENT RELATION OCCURRENCES WITH
> "REL-OCC" */
> $END

p-set definition section
enter p-sets:

The next section defines templates and constraints for relation definitions and occurrences.

> /* SETTING UP RANGE VARIABLES FOR USE
> IN CONSTRAINTS */
> RG RD IS REL-DEF /* RANGE OF RD IS REL-DEF */
> RG RDA IS RD.A-D-PAIRS /* RDA RANGES
> OVER THE P-SET STORED AT THE PID
> "A-D-PAIRS" NESTED WITHIN THE P-SET CURRENTLY POINTED TO BY RD */
> RG RO IS REL-OCC;

The next section shows the definition of one primitive operation.

primitive operations definition section
enter operations:

> /* GLOBAL PROCEDURE REMOVE_R
> */ REMOVE_R
> ** REMOVES THE RELATION REL FROM
> ** REL-OCC AND REL-DEF
> ** PARAMETERS:
> ** REL- > RELATION NAME
> */
> # include "primops.h"
> REMOVE_R(REL)
> char *REL;
> {
The next sections show the population of relation definitions and occurrences. The p-sets produced are shown in tabular form in Figures 4 and 5.

At this point, the DM can manipulate the database directly via the primitive operations.

select option: DM
enter user-id: MATT
enter model: RELATIONAL
enter implementation: ICST
enter database: TOY
enter name of query language (primops if using primitive operations): PRIMOPS
enter command ($END to terminate):
c> PRINT R EMP E E.ALL "E.SALARY > 25000" EMP_FMT;

From the collection of the Computer History Museum (www.computerhistory.org)
PHANS

Until very recently, the chief concern of this research had been the construction of the DMP. Now that it is operational, we can turn our attention to applications and experiments. Work will continue on the DMP to incorporate the DTD role, improve the DMP’s performance (the current version is quite inefficient), and modify it as necessary for future applications and experiments.

The three data models that have already been implemented on the DMP served as targets during the DMP’s development. The relational model used was a synthesis of the features of the major academic and commercial relational DBMS’s. The network model definition is fairly consistent with the specifications of ANSC/X3H2. The tree-structured model used is TDMS, a forerunner of System 2000, widely used in the federal government. Being able to model the structures and operations of these data models is a good indication of the DMP’s power and generality as a data model modeling tool (or meta-modeler). It may be insightful to attempt to model some other existing and proposed data models.

Query processing is an area in which we have done some experimentation and plan to do more. The completeness of the set of primitive operations defined for a data model can only really be tested by trying to map query languages onto the operations. Future experiments may include mapping several relational query languages (e.g., SQL, QUEL, QBE) onto one set of primitive operations, implementing different semantic approaches for TDMS, and mapping a relational query language onto CODASYL primitive operations.

Other work on mapping may be conducted within the scope of the DTD role. One idea is to use the DMP to develop database conversion strategies. While large conversions would not actually use the DMP, the DMP could provide a framework for design, formal definition, and preliminary testing of transformations.

Another use of the DMP will be implementing and evaluating data model specifications. Such work would be along the lines of that mentioned above using the ANSC/X3H2 specification of a network-structured data model. In attempting to formally define and implement an abstract specification, anomalies can often be discovered at an early stage.

The DMP may be of some practical value in the process of selecting a DBMS. A potential buyer may be able to use the DMP to help define his requirements and to evaluate prospective systems.

Finally, the DMP has pedagogical value. It allows one to explore a wide variety of data models and query languages in an experimental environment. A library of data model specifications available for teaching and modification on an experimental basis could be maintained.

ACKNOWLEDGMENTS

The products of the Abstract Database Models project, including this paper, represent a group effort. We would like to thank several individuals for their contributions: Steve Norman, Jerry Herstein, Tony Marriott, and Chuck Haas for implementing the DMP; Gary Sockut, Sabrina Saunders and T. C. Ting for their work in applying (and debugging) the DMP; and Joe Naputi, George Skillman, Ed Beller, and Tammy Kirkendall for giving the PSP the strength to carry on.

REFERENCES