CARDS: AN INVERTED LIST DATABASE CREATED USING SAS® SOFTWARE

Luke L. Diamond, Cedars-Sinai Medical Center

INTRODUCTION

Cedars-Sinai Medical Center, located in Los Angeles, California, is one of the largest private hospitals in the United States. With just over a thousand beds, our facility admits over 50,000 patients annually. Providing data analysis support for the management of the various CSMC departments has proven to be quite challenging. Both the volume and the complexity of the Medical Center's data have sometimes made it difficult to present clear and accurate reports. The SAS system has generally been the tool of choice for ad hoc statistical reporting at CSMC, primarily because of its ease of use and powerful number-crunching capability.

The hospital's operational data reside in various databases and systems maintained by different departments throughout the institution. Several years ago, a project was initiated to combine data from the major operational systems and create one centralized reporting database. The Cedars-Sinai Analytical Reporting Data System (CARDS) was built using information from three "feeder" systems:

1. A front-end patient management system.
2. An accounting system operated by an outside vendor, and
3. An online financial tracking system.

The first and third are CICSTM applications which access INSTM databases. Data from the second system comes to us as some other type of hierarchical database which has been unloaded to tape. Each month, a series of scheduled INSTM batch jobs is released, extracting data from these three sources and updating CARDS.

This paper concerns itself with the internal level of CARDS architecture, i.e. how the data is actually stored in SAS libraries. User "views" and their implementation are still under development. They therefore have been excluded as topics from this discussion, possibly to be addressed in future works. Within our scope, however, are the creation, update and accessing of CARDS using SAS software.

DATA OVERVIEW

The bulk of CARDS data is patient-based. Each patient visiting Cedars-Sinai is assigned a unique identifier, called his or her medical record number (MRN). No matter how many times a person is admitted, he or she retains this same reference number. Each "episode" of medical treatment for a given patient is further identified by a case number. A case is designated as inpatient, outpatient or emergency, based on how the patient was serviced. Certain outpatients, primarily people undergoing some type of therapy which requires repeated visits to the hospital, are classified as "unitized". These cases are subdivided into units, each unit representing one month of service. The combination of MRN, case and unit signifies one patient account or billing episode.

A variety of information is collected and stored concerning each patient, which can be classified as demographic, financial, clinical, physician and census data. CARDS links this data together, organizing it in a way that provides easy and efficient access.

Financial data takes the form of transactions which have been posted to patient accounts. Each transaction consists of a dollar amount, a service code and possibly a quantity of units (e.g. a number of days, bandages, etc.). These transactions are lumped together into convenient reporting "buckets" known as departments.

Census data only applies to inpatients. Each bed and nursing station the patient was assigned to is documented, along with the date and time the assignment took place. This data can be used to report occupancy levels of different parts of the hospital.

Clinical data includes diagnoses and procedures stored as ICD-9 (International Classification of Diseases, Ninth Revision) codes. Also recorded are doctor consultations and anesthesia administered.

Associated with each patient account are several kinds of physician information. Every case (or unit) generally identifies an admitting and an attending doctor of record. Furthermore, the clinical data includes, for each procedure, consultation or anesthesia, an associated Cedars-Sinai physician, known as the responsible party. Along with this patient-based data, a CSMC doctor directory is also maintained, which contains data elements such as age, office address and phone number.

SYSTEM DESIGN

An inverted list database is a collection of tables linked together by pointers. A table, for purposes of this discussion, is a file with a fixed number of fields contained by every record. An index is a special type of table consisting of precisely two fields: one which contains a data value and a second which contains a pointer to a record in another table. A pointer, by establishing addressability with another table, links the data in that table with the data in its own table.

The CARDS database consists of eight tables: PATIENT_ROOT, DEPARTMENT, DIAGNOSES & PROCEDURES, CENSUS, ATTENDING PHYSICIAN INDEX, PHYSICIAN MONTHLY STATISTICS, PHYSICIAN_ROOT and PHYSICIAN_INDEX (see figure 1). Most of these tables contain pointers. A pointer, as it has been implemented in CARDS, is an observation number from a related SAS data set. Tables have been implemented in two ways: as SAS data sets and as SAS informats. The PHYSICIAN_INDEX is an informat, while all the others are data sets, each occupying its own SAS data library.
The PATIENT ROOT data set contains demographic data, along with several summary variables concerning financial, clinical and census information. Also included on each PATIENT ROOT observation is the Attending Physician for that patient. Each observation in the DEPARTMENT data set contains financial data for a patient account in a particular department. Rather than having the MRN, case number and unit number for the account, however, it has a single variable ROOTPOSX, a pointer to the PATIENT ROOT data set. The DIAGNOSES & PROCEDURES data set contains one observation for each diagnosis, procedure, consultation and anesthesia associated with a patient account. The CENSUS data set contains an observation for each inpatient bed assignment. Like the DEPARTMENT data set, both the DIAGNOSES & PROCEDURES data set and the CENSUS data set are linked to the PATIENT ROOT data set by a pointer variable called ROOTPOSX. Similarly, the PATIENT ROOT data set is linked by pointers to these other three data sets.

Because more than one record may exist in the three "repeating group" data sets which correspond to a single record in the PATIENT ROOT data set, two pointers to each of these data sets are required in the PATIENT ROOT: beginning position and ending position. The six variables DEPT_BEG, DEPT_END, DIAG_BEG, DIAG_END, CENC_BEG and CENC_END perform this function. Obviously, all four data sets must be sorted (or at least grouped) by Account Number (MRN, Case Number, Unit Number) for this scheme to work. This method of storing patient information minimizes the amount of wasted direct access storage by eliminating redundancy (the same data value in more than one data set) and empty fields (unnecessary missing values, e.g. the provision for a dollar amount for a particular department when that department had no dollars posted to it).
The PHYSICIAN ROOT data set and the PHYSICIAN MONTHLY STATISTICS data set are linked in much the same way as the four just discussed, with the PHYSICIAN MONTHLY STATISTICS table containing repeating groups. Each observation in the PHYSICIAN ROOT data set represents one doctor associated with CSMC. Several indicators are calculated on a monthly basis for each of these doctors, such as Number of Patients Discharged, Total Patient Charges, etc. Each observation in the PHYSICIAN MONTHLY STATISTICS data set corresponds to one month for which a physician had activity. The PHYSICIAN ROOT data set, in addition to containing information about each doctor, contains two pointers to the PHYSICIAN MONTHLY STATISTICS data set: DARS_BEG and DARS_END. Other variables in this data set include the doctor's unique identification number (four digits, stored as a character variable) and his or her area of specialization (e.g. cardiology). The PHYSICIAN ROOT data set is sorted by Physician I.D. Number. The PHYSICIAN MONTHLY STATISTICS data set also has Physician I.D. as one of its variables. It too is sorted by Physician I.D. (within Physician I.D. it is further sorted by Discharge Month).

As we can see, two-way links between tables with the same sort order (or grouping) can be established by having pointer fields right in the tables themselves. Another type of link, however, can be implemented when the target table (also called the indexed table) contains a small number of records. Using this technique, the source table (also called the indexing table) contains the actual data value, rather than a pointer. With Version 5.18 of the SAS system came the ability in the FORMAT procedure to create user-written informats using the INVALUE statement. An informat can thus be created which maps data values of a variable to its record in the indexed table. This is the method used to link the PHYSICIAN MONTHLY STATISTICS data set with the PHYSICIAN ROOT data set. In this case, addressability is being established from one table to another via an index.

Indices implemented as informats can be used to link tables together regardless of how they are sorted. Another type of link is possible between the PHYSICIAN ROOT table and the PHYSICIAN INDEX table based on Attend ing Physician and links the DIAGNOSES & PROCEDURES table to the PHYSICIAN ROOT table based on Responsible Party. This is possible because both Attending Physician and Responsible Party are stored in their respective tables as a Physician I.D. Number. The three fields Physician I.D. (in the PHYSICIAN ROOT table), Attending Physician (in the PATIENT ROOT table) and Responsible Party (in the DIAGNOSES & PROCEDURES table) can be said to have the same domain, or set of allowable values (in this case, the set of valid doctor numbers).

Whenever different fields have the same domain, a link can be established. These fields can then be described as keys for that link. When each record in a table has a unique value for a particular key, that key can also be said to be unique. Please note that sometimes a link can be established between fields in the same table. Although CARDS has no examples of this yet, one can imagine an employee file in which the employee's supervisor (also an employee) is one of the fields.

Because there is a limit to the number of ranges that can be specified in an INVALUE statement, it is not always possible to implement indices as SAS informats. When the indexed table contains more records than can be listed as ranges in the INVALUE statement of the FORMAT procedure, another linking method must be used. Informats are also inadequate when one record in the indexing table must be linked to more than one record in the indexed table. In these cases, CARDS uses SAS data sets designated as "index data sets".

An index data set contains just one variable, named after the indexed table. The data set itself is named after the field in the indexed table being linked to. For example, the ATTENDING PHYSICIAN INDEX links Physician I.D. in the PHYSICIAN ROOT table with Attending Physician in the PATIENT ROOT table. In other words, data for each physician is linked with the data for his or her patients, if he or she has any. The variable in the ATTENDING PHYSICIAN INDEX data set is called (you guessed it) ROOTPOSX, a pointer to the PATIENT ROOT data set. Although it has only one variable, it has as many observations as the PATIENT ROOT data set. What then are the two fields in this so-called index? The "data value" in this case is merely a record's position in the table, i.e. its observation number. The PHYSICIAN ROOT data set has two variables, PATI_BEG and PATI_END, which point to the first and last observations for which that physician was the attending physician. If the physician had no patients at the hospital, these two variables are set to missing. The ATTENDING PHYSICIAN INDEX data set is sorted by Attending Physician, while the data set to which it points (PATIENT ROOT) is not.

This last type of table is particularly useful for subsetting large tables based on some sort of selection criteria. Our plan is to create additional indices of this type and store them as data sets in the same index data library. Another useful field on which to index the PATIENT ROOT table might be Discharge Date, because all inquiries generally are limited to a particular date range. To make such an index useful, another table would have to be built that contains, at the very least, Discharge Date and two pointers.

The chart in figure 2 illustrates the four types of links we have discussed and shows when each should be used, depending on the characteristics of the indexed table.

<table>
<thead>
<tr>
<th>INDEXED TABLE</th>
<th>UNIQUE KEY</th>
<th>NON-UNIQUE KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALL NUMBER OF RECORDS</td>
<td>LARGE NUMBER OF RECORDS</td>
<td>SORTED BY KEY</td>
</tr>
<tr>
<td>TYPE OF LINK</td>
<td>Informat Index</td>
<td>One Pointer</td>
</tr>
</tbody>
</table>

figure 2

918
How then, can such a database be created? More specifically, how does one go about building the four types of links just described? Let us assume that all of the tables pictured in Figure 1 already exist, with the exception of the two indices. Furthermore, let us say that these six SAS data sets are called PATIENT.ROOT, DEPT.DEPT, DIAG.DIAG, CENSUS.CENSUS, PHYS.ROOT and MONTHLY.STATS. For the following four programming examples, we will also assume that none of the pointers exist on the files. Instead, PATIENT.ROOT, DEPT.DEPT, DIAG.DIAG and CENSUS.CENSUS contain variables MEDREC_X, CASENUMX and UNITNUMX, which together uniquely identify a patient account. The variable which uniquely identifies each physician in the PHYS.ROOT and MONTHLY.STATS data sets is called DR_HUM_X.

1. To build an informat index, two steps are required - one to write the "source" code and one to compile the informat.

```sas
DATA NULL;
FILE TEMP;
SET PHYS.ROOT END:ENDING;
IF N=1 THEN PUT "VALUE OF DOCINDEX=";
PUT 07 ** 
88 DR_HUM_X 
01 ** 
04 N;
IF ENDING THEN PUT 07 ";";
PROC FORMAT LIBRARY=SASLIB;
`include TEMPL;`.
```

2. The creation of the variable ROOTPOSX on the DEPT.DEPT data set illustrates how to build a one pointer link.

```sas
DATA DEPT.DEPT;
DROP MEDREC_X CASENUMX UNITNUMX;
MERGE DEPT.DEPT (IN=INDEPT) 
PATIENT.ROOT (KEEP=MEDREC_X CASENUMX UNITNUMX) 
BY MEDREC_X CASENUMX UNITNUMX;
RETAIN ROOTPOSX @;
IF FIRST.UNITNUMX THEN ROOTPOSX = ROOTPOSX+1;
IF INDEPT;
```

3. The building of a two pointer link resembles that of a one pointer link, as exemplified by the creation of the variables DEPT_BEG and DEPT_END on the PATIENT.ROOT data set. In this case, however, two steps are required instead of one.

```sas
DATA POINTERS;
DROP COUNT;
RETAIN DEPT_BEG COUNT 8;
MERGE DEPT.DEPT (IN=INDEPT KEEP=MEDREC_X CASENUMX UNITNUMX) 
PATIENT.ROOT (IN=INROOT KEEP=MEDREC_X CASENUMX UNITNUMX) 
BY MEDREC_X CASENUMX UNITNUMX;
IF INDEPT THEN COUNT=COUNT+1;
IF INDEPT AND INROOT;
IF FIRST.UNITNUMX THEN DEPT_BEG = COUNT;
IF LAST.UNITNUMX THEN DO;
   DEPT_END = COUNT;
   OUTPUT;
END;
```

4. This program segment builds a data set index link by creating the index data set INDEX.ATN_MD_I.

```sas
DATA ATN_MD_I (KEEP = ROOTPOSX DR_HUM_X);
RENAME ATN_MD_X = DR_HUM_X;
SET PATIENT.ROOT;
ROOTPOSX=N;
PROC FORMAT LIBRARY=SASLIB;
`include TEMPL;`
```

Each month, the PHYSICIAN ROOT and PHYSICIAN MONTHLY STATISTICS data sets are completely rebuilt from scratch. The PHYSICIAN INDEX informat and the ATTENDING PHYSICIAN INDEX data set are then rebuilt as well, using the algorithms discussed above in examples one and four. Of course, the PATIENT.ROOT data set must be updated before the ATTENDING PHYSICIAN index can be recreated.

The PATIENT.ROOT, DEPARTMENT, CENSUS, and DIAGNOSES & PROCEDURES data sets, however, are updated by applying monthly transactions. This is because of the enormous size of the SAS files (at last count, the PATIENT.ROOT data set had about 250,000 observations, the DEPARTMENT data set had about 2,500,000 observations, the DIAGNOSES & PROCEDURES data set had about 1,000,000 observations, and the CENSUS data set had about 300,000 observations) and, more importantly, the feeder files.

A monthly batch program first reads the feeder files, identifying patient accounts with new information. Four transaction files are then created, which are to be applied to the database master files. The program that creates the transaction files establishes links between them in the same fashion illustrated in examples two and three. We then have, essentially, two databases which must be merged.
5. This program segment illustrates how the transactions are applied to the master CARDS database tables. In this example, the four transaction data sets are called PATIENT.ROOT, DEPT. DEPT, DIAG. DIAG and CENSUS. CENSUS. Transaction files as well as master files are assumed to already contain the pointers mentioned earlier in this paper.

```
DATA ROOTM;
UPDATE PATIENT.ROOT
BY MEDREC_X CASENUMX UNITNUMX;
LENGTH UP 3;
UP = INROOT;
DATA DEPTM (KEEP = MEDREC X CASENUMX UNITNUMX CHARGE QUANT DEPT);
SET ROOTM (KEEP = MEDREC X CASENUMX UNITNUMX DEPT Beg DEPT_End UP);
IF NOT UP THEN DO I = DEPT_BEG TO DEPT_END;
                   SET DEPT.DEPT POINT=I;
                   OUTPUT;
                   END;
ELSE DO I = DEPT Beg TO DEPT EN>
                   SET DEPTT.DEPT POINT=I;
                   OUTPUT;
                   END;
DATA DIAGM (KEEP = MEDREC X CASENUMX UNITNUMX CLASCATX COED TX TYPEX CODEDATX RESPARTX);
SET ROOTM (KEEP = MEDREC X CASENUMX UNITNUMX DIAG_BEG DIAG_END UP);
IF NOT UP THEN DO I = DIAG_BEG TO DIAG_END;
                   SET DIAG.DIAG POINT=I;
                   OUTPUT;
                   END;
ELSE DO I = DIAG_BEG TO DIAG_END;
                   SET DIAGT.DIAG POINT=I;
                   OUTPUT;
                   END;
DATA CENSUSM (KEEP = MEDREC X CASENUMX UNITNUMX CH DATEX CHVALUEX CH-TYPEX CH_TIMEX);
SET ROOTM (KEEP = MEDREC X CASENUMX UNITNUMX CENC Beg CENC_End UP);
IF NOT UP THEN DO I = CENC_BEG TO CENC.End;
                   SET CENSUS.CENSUS POINT=I;
                   OUTPUT;
                   END;
ELSE DO I = CENC.BEG TO CENC.END;
                   SET CENSUST.CENSUS POINT=I;
                   OUTPUT;
                   END;
```

Note the use of the POINT option in some of the SET statements of example five. This is how one can use SAS software to access observations directly in a data set stored on disk. The SAS system, then, is the database management system (DBMS) of CARDS. It provides the DATA Step language as the language for building new tables, establishing links between tables, updating tables and, finally, accessing data from the tables. The following are programming examples of how to use the links to make programs efficient by eliminating unnecessary I/O.

6. Suppose one wants to list the names, office addresses and phone numbers of all doctors who attended more than ten inpatients during March of 1989. This program segment demonstrates how to use an informat link. DISMONX is a SAS date indicating the discharge month, and IDISCHGX is the number of inpatient discharges. Both of these variables are on the PHYSICIAN MONTHLY STATISTICS data set. DRNAME, OFFADDRX, OFFCITYX, OFFZIPX and OFFPHONX represent Physician Name, Address, City, Zip Code and Phone Number, respectively, on the PHYSICIAN ROOT file.

```
TITLE 'DOCTORS WITH MORE THAN TEN DISCHARGES';
TITLE2 'DURING MARCH, 1989';
DATA DOCTORS;
SET MONTHLY.STATS;
IF PUT(OISMON_X,YYMMDD4.)='8903' AND IDISCHGX>UJ
   THEN DO;
      POINTER =$
      INPUT(DR_NUM_X,DOCINDX.);
      SET PHYS.ROOT POINT=POINTER;
      OUTPUT;
   END;
PROC PRINT;
VAR DR NUM_X DRNAME X OFFADDRX OFFCITYX OFFZIPX OFFPHONX;
```

7. The next example shows how this temporary SAS data set WORK.DOCTORS can be used to list the patients each of these doctors had for that month. This program segment illustrates the use of a data set index. PATNAMEX is the variable on the PATIENT ROOT data set representing the patient's name. DISDATEX is the patient's discharge date.

```
TITLE1 'SELECTED INPATIENTS';
TITLE2 'DISCHARGED IN MARCH, 1989';
DATA PATIENTS;
DROP PATI BEG PATI END;
SET DOCTORS;
   DO I=PATI_BEG TO PATI_End;
      SET INDEX ....
      TN_hI>_1
      POl:I;
      SET PATIENT.ROOT POINT=ROOTPOSXi
      IF PUT(DISDATEX,YYMMDD4.)='8903' THEN OUTPUT;
   END;
PROC PRINT;
VAR MEDREC X CASENUMX UNI~
      PTNAME X DISDATEx DRNAME_X;
```

Examples six and seven further demonstrate the use of the POINT option in some of the SET statements of example five. This is how one can use SAS software to access observations directly in a data set stored on disk. The SAS system, then, is the database management system (DBMS) of CARDS. It provides the DATA Step language as the language for building new tables, establishing links between tables, updating tables and, finally, accessing data from the tables. The following are programming examples of how to use the links to make programs efficient by eliminating unnecessary I/O.
When in a DATA step, all of the SET statements have the POINT option, it is necessary to end the DATA step with a STOP statement. Otherwise, the program enters an endless loop. The next example illustrates this.

8. This program segment extracts all of the patients of doctor number 0123.

```
DATA PATS0123;
  DROP PATI_BEG PATI_END;
  POINTER=INPUT('0123'.DOCINDEX.);
  SET PHYS.ROOT POINT=POINTER;
  DO I=PATI_BEG TO PATI_END;
    SET IN>EX.ATN MD I=POINT;
    SET PATIENT.ROOT=ROOTPOSX;
    OUTPUT;
  END;
  STOP;
```

Linking large files together saves disk storage space by reducing both redundancy and empty fields. Also it allows one to take advantage of the SAS system's ability to access data set observations directly. Using SAS software in this way makes programs more efficient by drastically reducing I/O. This is especially helpful in improving the response time of online applications, such as those written using SAS/AF®, SAS/FSP® and/or SAS/IML®.

CONCLUSION

The techniques outlined in this paper represent the views of the author and are not in any way endorsed by Cedars-Sinai Medical Center. The programming examples are for illustrative purposes only and do not necessarily reflect actual code in CSMC computer systems.

SAS, SAS/AF, SAS/FSP, and SAS/IML are registered trademarks of SAS Institute Inc., Cary, NC, USA.

CICS, IMS and MVS are trademarks of International Business Machines, Inc.

The author may be contacted at:

Cedars-Sinai Medical Center
Information Systems Services
8700 Beverly Blvd. ASB3
Los Angeles, CA 90048
Telephone (213) 855-5205

ACKNOWLEDGEMENTS

The author gratefully acknowledges the efforts of all those who participated in the design, development, testing and implementation of the CARDS database. Warranting specific mention are Roger E. Bolus, Ph.D., Thomas J. Eunice, Daniel C. Walker, Katherine M. Kerns, Peggy D. Rosenow, David Kuhn, Ph.D., and Debbie M. Gibson.

REFERENCES


