Unorthodox IMAGE Accessing for Power

Joseph Berry
Kiryat Telz Stone 116A
D.N. Harei Yehuda, Israel

Noel Magee
Telephone Employees Credit Union
123 S. Marengo Avenue
Pasadena, CA 91101

Introduction

Traditionally, large databases are difficult to manage and manipulate. Datasets with millions of records cannot be unloaded and loaded whenever desired. Broken chains cannot, therefore, be easily repaired. Large numbers of records cannot easily be deleted.

For databases meeting certain logical and physical criteria, a technique is presented here for performing the almost instantaneous removal of millions of historical records from a detail dataset and moving them to an archive. This technique additionally allows access to as many of those archived records as the user wishes, subject only to the constraint of disc space. Modifications to application software is minimal. As many detail records as are present will be accessed by the application software.

Additional advantages to this technique include never having a fragmented detail data set and being able to produce historical reports in a very timely manner, even when millions of records need to be scanned.

Description of the Problem

At the Telephone Employees Credit Union we provide credit union services to communication industry employees in the Southern California area. We are, at present, the eleventh largest credit union in the United States. In addition to standard savings and withdrawal accounts, we provide access to over 10,000 Automated Teller Machines, lines of credit (both personal and equity based), Automated Voice Response, interbank/automatic funds transfer and other standard financial services. In order to provide these services in a timely fashion we use two Series 70s at a central site running Silhouette/3000 and Filepro from Carolian, MTS, RJE, NS/3000, DS/3000, and a variety of other third party packages. Our ten branches (located over the Southern

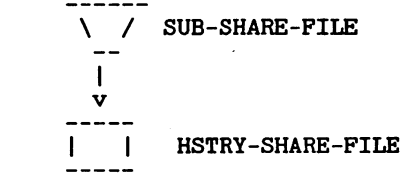
California area) communicate with the central site using 3270 protocol and remote 3274 look-alike controllers.

During the past seven years TECU has experienced a growth rate of 10-20% per year in transaction volume. In 1988 we averaged 40,000 online and 30,000 to 40,000 batch transactions per day. Analysis showed that the growth rate would problably not change in the near future. Additionally several new services needed to be implemented in the 1988 time-frame. These increases were reflected in two primary history data sets, HSTRY-SHARE-FILE and HSTRY-LOAN-FILE. From a base of 400,000 entries in 1979 the HSTRY-SHARE-FILE grew to over 4 million by September of 1987 and occupied some 1.5 million sectors of disc. This, of course, presented us with a file on the verge of all MPE and IMAGE limitations.

The principle difficulties with which TECU was attempting to deal were, first, the deletion time for inactive records and, secondly, the disc space required for so large a file. In the former case we had a deletion program which ran some 56 hours simply to delete the oldest thirty days of history. This amount of time was required due to the fact that the structure was a single large IMAGE detail. In fact, prior to Turbo-IMAGE, we could not do daily reporting from this due to the fact that a chain, keyed by date, would have been too long for IMAGE. Secondarily, the file was so large that, while one might have sufficient disc space in total, the extent sizes sometimes prevented a simple restore of the file. Often we ended up condensing volumes simply to acquire large enough blocks of free space to accomidate the extents; more often, we ran out of space. Also, since we updated the file on a daily basis, it was being stored on a nightly basis. This resulted in from one to three additional tapes (at 6250bpi) being generated per night and from 9 to 27 additional minutes of down time. Both of these problems became critical when, in 1985, we moved to 18 hours/day operation with the addition of proprietary and networked ATMs. In 1986/1987, the problems were compounded when we moved to a new headquarters building and started to look at full 24 hour, non-stop availability. Fumdamentally, we could no longer afford either the maintenance nor the disc space problems in our coming environment.

Proposed Solutions

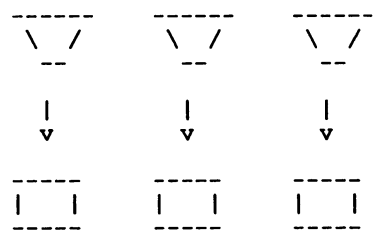
Prior to the changes to be described, share history data was kept in the database, TECUDB, in the HSTRY-SHARE-FILE detail data set. This data set was chained off the manual master, SUB-SHARE-FILE.



The decision was made to solve the problems associated with the large HSTRY-SHARE-FILE detail data set and a number of alternative solutions were considered. These included multiple KSAM files and multiple IMAGE data sets.

The primary advantage of KSAM as the file mechanism for storing the data was that since it was not inside an IMAGE structure, maintenance would be easier. It's disadvantage was that it did not have the nice item lists of IMAGE; thus entailing more "up front" work (i.e., more severe modifications to existing programs). This difficulty could be overcome by making the KSAM file self-describing and the appropiate user SL routines cognizant of that fact. The dynamic allocation of storage space which KSAM uses would save disc usage. The greatest disadvantage of using KSAM was the time required to build the key file whenever the data were moved from the TECUDB database.

Two IMAGE designs were also investigated. One design consisted of a number of data set pairs as shown here:



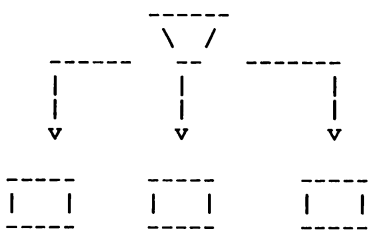
This technique required that a separate database be built containing pairs of data sets for each month (or time period) in the past that is desired. Each master/detail would have the same format as the current master/detail.

A number of advantages were presented by this design. Each period's history data is defined as an data set pair. If there was not enough room on the system for all the historical data, then as many data sets as would fit would be maintained. To delete a month's worth of history data, the

data set linkage for that month wouldn't really have to be erased. It is possible to physically purge the two data sets and rebuild them. This procedure would take a minimum amount of time. There is no reason to limit the number of time periods. Depending on disc space availability, many more months' worth of data can be stored. In all cases, the "active" number of data sets and which pair of data sets represents which time period would be determinable by inspecting a stand-alone control master data.

A major disadvantage of this technique is that the large master data set must be duplicated for each month. This would be a large waste of disc space. Furthermore, in order to profit from the purging and rebuilding of individual data sets, privileged mode (PM) access will probably have to be used.

An alternative variation to the above IMAGE design was to link all the detail data sets to one master data set. This solution eliminates all the excess disc space. However, similar to KSAM, an inordinate amount of time would be spent building (or updating) the pointer information in the master data set as historical data is added to it.

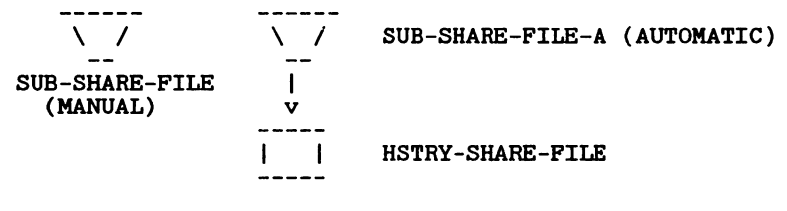


Proposed Solution

Given the problems associated with the database TECUDB, we will describe an efficient and effective solution for "splitting up" the HSTRY-SHARE-FILE data set of the TECUDB database, making it considerably more manageable.

Our minimum requirements were to continue maintaining more than three months' worth of history data in order to process statements and quarterly postings. An important point here is that except for the current month, no information is added or updated (the only exception being by the TECU Information Services staff when repairing bad data).

The proposed enhancement required a change in the relationship between these two data sets. We changed the data set structure to the following new structure:

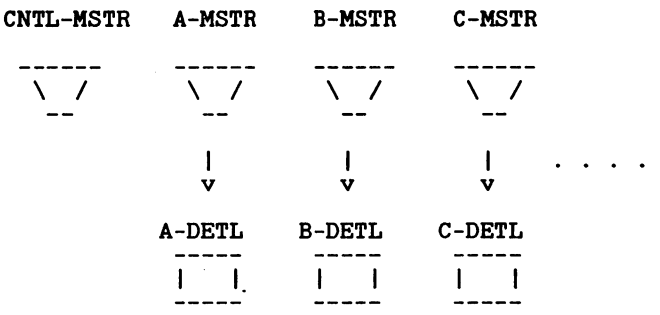


We took the SUB-SHARE-FILE data set and removed the connection to the detail data set (how this was accomplished is explained further). In addition, we added a new automatic master that essentially replaced one of the functions of the previous SUB-SHARE-FILE, i.e., chained access to the detail. The effect of this change on existing programs that access either of these data sets is interesting: no program changes were required! Note that if a chained read into HSTRY-SHARE-FILE is needed, the DBFIND references the name of the detail data set and not the master (therefore, no change). If a by-key access is required into the master to retrieve stored information, the DBGET references the master data set name, SUB-SHARE-FILE, which is now the name of our standalone master.

Obviously, data sets SUB-SHARE-FILE and SUB-SHARE-FILE-A have the same key, ACCOUNT-SUFFIX. Due to changes in the blocking factor of the SUB-SHARE-FILE, the resulting structure with three data sets actually consumes less disc space than the old structure consisting of two data sets.

With respect to performance degradation (i.e., increasing numbers of I/Os due to the additional data set), there is a very slight increase in the number of total I/Os. Specifically, adding a record to the HSTRY-SHARE-FILE data set with the existing schema took approximately seven physical I/Os (not including logging). With the three data set structure, the number of I/Os increases to eight.

In addition to the changes in the structure of the TECUDB database, a new, additional database, called HISTDB, was added to the system having the following structure:



In an operational mode, the automatic master and detail data set from the TECUDB database will be moved into this new database when the data in the original detail data set fills beyond a certain limit. Each pair of master/detail data sets will represent one arbitrary length of history data. The master data set is a duplicate of SUB-SHARE-FILE-A (which is now much smaller as an automatic master data set) while the detail is a duplicate of HSTRY-SHARE-FILE. Database TECUDB will only contain data for the current period. The CNTL-MSTR data set, pictured above, contains various kinds of identification information.

A number of advantages result from this solution:

1. When deleting the history data, the data in HSTRY-SHARE-FILE doesn't really have to be erased. The two data sets are simply purged and rebuilt. This procedure takes almost no time. In reality, instead of purging this pair of data sets, we rename them into the database that stores the historical information. We then create a new pair of empty data sets for TECUDB. Therefore, there is never any deleting of the records in HSTRY-SHARE-FILE (more on this bit of magic later).

2. There is no reason to limit the number of time periods. Depending on disc space availability, many periods' worth of data can be stored. In all cases, the "active" number of data sets and which pair of data sets represents which period is determined by inspecting the stand-alone control master data data set.

3. If a broken chain is found in the master/detail data set pair of the active TECUDB database, the repair can be accomplished as soon as there is a period turnover. This data set pair is moved to our new database and is only read accessed. A duplicate of the data set pair can be repaired on one of the backup computers and when corrected, these two files would replace the original ones.

An interface procedure, HDBGET, was written to allow all the applications to access either the normal data set pair or one of the data set pairs from the new HISTDB database in a transparent manner. The interface procedure accepts almost the same calling sequence as DBGET. It is responsible for calling the real HSTRY-SHARE-FILE data set in the TECUDB database. If, during a backwards chained read we arrive at the beginning of the chain in this data set, we must then access the most recent data set pair in the new database and return its information to the user. We must then store information reminding us of where we were and in which database in order to continue our chained read. This information must also be stored for serial access reads.

All programs that perform DBGETs on the HSTRY-SHARE-FILE needed to be modified. Instead of calling DBGET directly, these programs called HDBGET. The size of the status array was slightly enlarged and two parameters were added.

Structure of the HISTDB Data Base

The HISTDB database, when in normal operation, consists of zero or more (up to 24) physical pairs of SUB-SHARE-FILE-A masters and HSTRY-SHARE-FILE details. These data set pairs are moved into HISTDB via "RENAME"s of the data sets (see the HISTUTIL program below). In addition, there may be zero to 24 pairs of data sets associated with loan information (this is not currently implemented except in HISTDB and procedure HDBGET).

In order to fool IMAGE into believing that this database actually has so many data sets (48 pairs x 2), the schema file was actually configured with the 96 data sets.

In addition, there is a stand-alone master data set that controls access to the various data set pairs. This data set marks which data set pairs actually exist and what their date ranges consist of.

Below we see the layout of the CNTL-MSTR manual master data set. This data set contains two records: one for transaction history information (the detail data set being HSTRY-SHARE-FILE) and the second being for loan information (the detail being HSTRY-LOAN-FILE). Within each record, the remaining fields are replicated (indexed) 24 times. This represents the 24 possible data set pairs that can be present in the HISTDB database.

Each index represents one data set pair. The indices are maintained in backwards chronological sequence (i.e., the most recent date is in index 1, the first index). Since the

data in the detail data set (as it came from the TECUDB database) was never modified, the data is always in physical chronological order. Therefore, by examining the first and last records of the data set, the date/time range can be quickly determined.

There is really no connection between the data set pairs and where they are located. The data sets of HISTDB were arbitrarily given the logical names HISTDB02, HISTDB03, ..., HISTDB97 (the master data set was HISTDB01). Thus, the logical and physical names of the data sets are identical. When a data set pair are moved from database TECUDB to HISTDB, information is inserted into the control data set according to the BEGIN-DATE date. The name of the detail data set that is assigned to this pair is inserted into variable DS-NAME. In this way, the assignment of the data set pairs can be managed, controlled, and accessed by the data contained in the CNTL-MSTR data set.

Layout of CNTL-MSTR file:

key

SET ID	BEGIN-DATE x24	END-DATE x24	BEGIN-HOUR x24	END-HOUR x24	DS-NAME x24

Elements in CNTL-MSTR:

Element	Type	Size	Remarks
SET-ID	I	1	Data set number of the data set from the TECUDB database (i.e., there will be one for the HSTRY-SHARE-FILE and one for the HSTRY-LOAN-FILE).
BEGIN-DATE	I	2	YYMMDD of beginning date of dat set pair. 24 subitems.
END-DATE	I	2	YYMMDD of ending date of data set pair. 24 subitems.
BEGIN-HOUR	I	2	HHMMSS of beginning time of data set pair. 24 subitems.
END-HOUR	I	2	HHMMSS of ending time of data set pair. 24 subitems.
DS-NAME	X	16	Name of the detail data set that contains the records for this date/time range. 24 subitems.

HISTUTIL Utility

HISTUTIL was designed to perform two basic functions: (1) To take HSTRY-SHARE-FILE data set pairs from the TECUDB database and move them to the HISTDB database, updating its CNTL-MSTR data set. (2) To store/restore portions of data from the HISTDB database to and from tape. We designed the syntax of HISTUTIL to always relate to the HISTDB database. This was necessary because we needed a consistent point of reference for all commands.

This program, written in PASCAL, performs privileged mode functions without actually going into privileged mode (PM). The main reason for requiring PM was to be able to rename the data set pairs from the TECUDB database into the HISTDB. TECU already owns a program that can perform this function: MPEX from VESOFT, Inc. A phone call to VESOFT gave us the technique for communicating program-to-program with MPEX. This is accomplished with the MPE MAIL intrinsics: SENDMAIL and RECEIVEMAIL. HISTUTIL, therefore, created the MPEX process as a son and transferred commands to it via SENDMAIL. The following two procedures demonstrate this (error branches have been removed and procedure calls simplified).

```
procedure create_mpex (var pin:smallint);
var
   progname : string[30];

   procedure  createprocess;  intrinsic;

    { beginning of procedure }
begin
progname := 'MPEX.PUB.VESOFT ';

createprocess(progname, pin);
end;



procedure talk_to_mpex(pin:smallint;var message:string);
var
   status, messlen : smallint;

   procedure activate;           intrinsic;
   function sendmail : smallint; intrinsic;

    { beginning of procedure }
begin
message := '!' + message + ' ';
messlen := strlen(message) div 2;
status := sendmail(pin, messlen, message, 1);
```

```
activate(pin, 3);
end;
```

Since all the PM code was isolated into a known program, MPEX, it was easier and certainly less "dangerous" to debug. The other function that MPEX performs is a copy function. When HISTUTIL renames the data set pair from being a part of TECUDB to HISTDB, an empty data set pair must be copied into that location. For this purpose, we created a special group called HOLD that held an empty version of those two data sets (i.e., two IMAGE data set files). The MPEX "FCOPY" command with the ",FAST" option was used to copy these two privileged mode files into the proper group. The copy is usually done automatically after the rename unless overridden with a HISTUTIL NOREPLACE control word.

The standard command for adding data to the HISTDB data base is as follows:

```
ADD HSTRY-SHARE-FILE FROM TECUDB
```

The second major function of HISTUTIL is to store (or archive) data from HISTDB to tape (particularly data that is older than a certain date). The user runs HISTUTIL and enters the command STORE. The program presents the user with a list of the current data set pairs (with their date ranges) which he/she then chooses. The program generates a job stream for storing the appropriate files. These files are first moved into a holding area (group HOLD) with a copy of CNTL-MSTR data set to be able to quickly identify the contents of the data set pairs being stored.

The data set pairs are then archived to tape. In a similar manner, HISTUTIL contains a control verb option that allows data set pairs to be reloaded into HISTDB from the archives. This is accomplished by restoring an archive tape into group HOLD and executing HISTUTIL with the following command:

```
ADD HSTRY-SHARE-FILE FROM HOLD
```

Two options that were added to HISTUTIL include displaying the contents of the current HISTDB database and initializing the CNTL-MSTR data set.

Procedure Hdbget

In order to minimize converting all the application software to the new HISTDB design, it was imperative that the new

interface procedure would be as transparent as possible in order to minimize coding changes. We started with the syntax to DBGET and were, in the end, forced to add two variables to the parameter list for reasons discussed below. Our new HDBGET supports the following syntax and modes:

```
          A      A     I      A      A      A        A
hdbget(pribase, dset, mode, status, list, buffer, argument,

          I         A
       secbase, secstatus);
```

mode 1: re-read; like normal IMAGE

mode 2: serial read, with data-set switch capability

mode 3: backward serial read, with data set switch capability

mode 5: chained read, based on secbase and secstatus relative record numbers. With data set switch capability

mode 6: backward chained read, based on secbase and secstatus relative record numbers. Actual directed dbget with data-set switch capability

Since this procedure was to reside in an SL, all variables used by the procedure were locally defined (Q-relative). That is, it is not possible to save information on previous calls within the procedure itself. It was seen that two variables had to be added to the procedure call. Variable SECBASE performs the same function as PRIBASE, i.e., to identify the previously opened secondary database (HISTDB).

Variable SECSTATUS is a 15 word array. The first ten words are used as the status array for the IMAGE calls made within HDBGET. The last five words contain the following information:

```
word 11:  database id of current database
     12:  data set number of current data set
     13:  unused
  14-15:  record number of current record
```

Let's take a look at how HDBGET processes some of the specific modes: Mode 2, or forward serial mode, assumes we start from the oldest records (the beginning of the data

base) to the newer records. Therefore this mode starts by accessing the HISTDB database. If the database doesn't exist, it immediately goes on to database TECUDB. HDBGET starts with the oldest data set pair in HISTDB (as determined by CNTL-MSTR). It is accessed in forward serial sequence and the data is returned to the user. If HDBGET reaches the end of data set, the next oldest detail data set is accessed. When no more data sets are to be found in HISTDB, HDBGET continues with the appropriate detail data set in TECUDB until the end of data set is reached. At this time, the user is given an "end of file" error message.

The more difficult transaction types to emulate were mode 6 and mode 5. Mode 6, for example, is the backward chained read. Here we start with the latest information in the chain and go backwards in time. Processing begins with the TECUDB database. A DBFIND is performed to the detail data set, HSTRY-SHARE-FILE. The status array returned contains the address of the last entry in the chain. DBGET, mode 4 is executed using this address, retrieving the information for the user. SECSTATUS words 11-14, are updated to contain the next address to be retrieved. Each subsequent call to HDBGET uses the information in SECSTATUS to do the DBGET, mode 4, read. When an end of chain error is encountered, HDBGET determines which data set pair in HISTDB is to be used next (the next most current). A DBFIND followed by a DBGET is executed (as above) and the information is updated in the SECSTATUS array. When all data is exhausted, an "end of chain" error is returned to the user.

To initialize the item list (for successive accesses via "*;"), HDBGET is called when the HISTDB database is still closed (i.e., when SECBASE is still zero). This will force HDBGET to open the HISTDB database and properly initialize all of its data set lists. If the user wants/needs to change the default item list in mid-program, then the HISTDB database must first be completely closed (DBCLOSE, mode=1) and SECBASE set to zero.

The return condition codes have been slightly altered to logically reflect the functionality of the HDBGET call. For example, it is assumed that a call to HDBGET will succeed and return data (more correctly, it is assumed that the key exists). Therefore, an invalid key will return a condition code of 14 (on a mode=3 access), which is a beginning of chain. It is as if we had gotten to the beginning of the chain and had not found any data. In such a case, portions of the DBEXPLAIN message may not be accurate. Specifically, the data set name and the mode may display erroneous results. This is due to the various other database accesses that HDBGET performs on the HISTDB database. The only item that is really examined is the condition code. Further information

can be gleaned (if one really wants to) by examining the SECSTATUS array.

Implementation of Solution

The implementation of the HISTDB database into an operational reality required careful and meticulous planning. The following steps describe the overall process that lead to the successful implementation the reorganization of the HSTRY-SHARE-FILE data set. Refer to the accompanying chart below.

[A] Test the process of converting the current TECUDB structure to the new TECUDB structure. Use the procedure described below.

[B] Design and build the HISTDB and place a working copy on the secondary computer system.

[C] Store a copy of SUB-SHARE-FILE and HSTRY-SHARE-FILE onto the secondary system. Using a contributed library program, SELCOPY, begin extracting historical data (sorted by date) from HSTRY-SHARE-FILE and store into data base HISTDB as a number of data set pairs.

[D] Identify all source code that accesses the HSTRY-SHARE-FILE with a view towards identifying the types of source code changes that need to be made with the new structure.

[E] Write and debug the procedure that accesses the new HSTRY-SHARE-FILE and the HISTDB database (procedure HDBGET). Once debugged, install into production system. (It should function in a pass-through mode since the HISTDB database won't exist yet.)

[F] Make any changes to existing programs (as identified in [D]) that access HSTRY-SHARE-FILE to also access the new HISTDB database. Use the procedure(s) written in [E].

[G] Define and setup, in detail, the procedure for switching the HSTRY-SHARE-FILE from TECUDB to the new database (on a history set interval changeover). This will result in a utility that will perform or invoke all the functionality automatically (HISTUTIL).

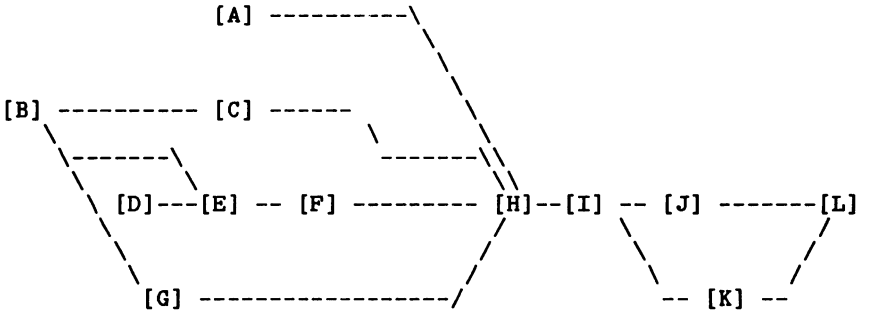
[H] Perform full system test of the online system and other programs that access the HSTRY-SHARE-FILE.

[I] When ready to make the changeover to the new database structure, store the latest copy of SUB-SHARE-FILE and HSTRY-SHARE-FILE from the primary system onto the secondary system. If there are space constraints, store the historical data temporarily to tape.

[J] On the secondary system, use SELCOPY to extract the remaining data from HSTRY-SHARE-FILE that have not yet been extracted, sort the data by date, and write it to HISTDB. Delete the SUB-SHARE-FILE and HSTRY-SHARE-FILE from the secondary system.

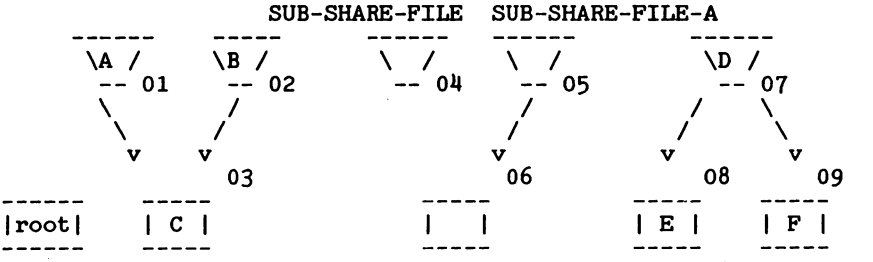
[K] Convert the production TECUDB database to the new structure

[L] Move HISTDB to the primary system.



Prior to Conversion Day

(1) Build a new database using TECUDB's new structure.

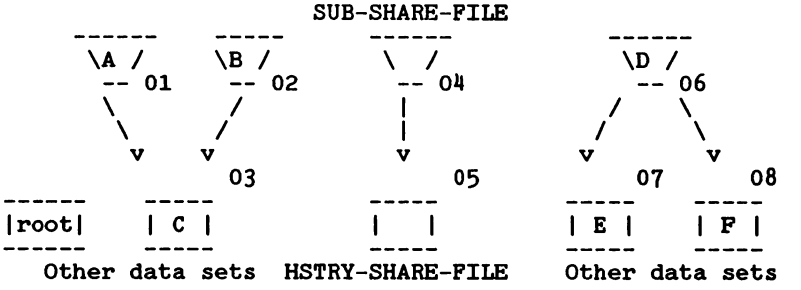


(2) Store the database root file and data sets SUB-SHARE-FILE (04), SUB-SHARE-FILE-A (05), and HSTRY-SHARE-FILE (06) to tape. Remember, that these data sets are empty.

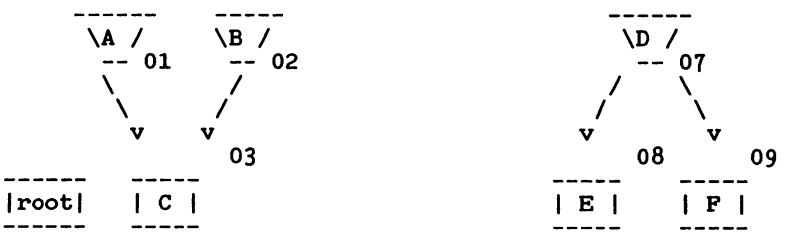
Conversion Day

(3) Save the manual master data by performing a data set unload of SUB-SHARE-FILE from the live TECUDB database using program DICTDBU. This takes about 30 minutes on a stand-alone series 70.

(4) Store SUB-SHARE-FILE and HSTRY-SHARE-FILE from the live TECUDB to tape and RESTORE onto the secondary system where the HISTDB will be updated to the present date.



(5) Working with the live TECUDB database, physically rename data set 08 to 09, 07 to 08, and 06 to 07 (i.e., all the data sets after HSTRY-SHARE-FILE). Then delete data set 04 and 05.



(6) RESTORE the root file and data sets 04, 05, 06 that were created above in step (2). We now have a database in the new structure.

(7) Load the SUB-SHARE-FILE data set with the data that was previously unloaded above in step (3). Using DICTDBL, this takes approximately 70 minutes.

(8) Save the new database to tape.

(9) During the testing stage, STORE the database to tape (define the database with small capacities). This will give an additional verification that IMAGE accepts this newly built data base.

Problems Encountered with the Implementation

Our original HSTRY-SHARE-FILE data set contained approximately 4 million records. We couldn't delete all this data when we converted to the new structure since credit union members inquire on their recent transaction history. Since this data represented approximately three to four months' worth of transaction history, we decided to extract the data on monthly bounderies, creating data set pairs (see step [C] above). We knew that the data needed to be sorted by date and time (since we had to do serial reads and extracts through the detail data set to find the appropriate date range) and, therefore, used SELCOPY's sort feature to sort this data (SELCOPY uses HP's SORT intrinsics). Unfortunately, we had had no idea that many of the transactions that had been processed by the online system were being completed in less than 0.1 seconds. This was the time resolution that we maintained in the data. In other words, it frequently happened that two transactions for a particular member had identical time stamps. It wasn't until our first day live with the new system that this problem was discovered. A look at the transaction history of a member showed the transactions to be out of order (imagine shuffling your checkbook entries -- the total is correct but the intermediate results are wrong). It required some fast hacking to repair the historical data.

Related to the problem of breaking the detail data set data into manageable chunks was the problem of disc space. While we had a second series 70 with six 7933 disc drives, this system was being used to silhouette the primary, production system and therefore already had one full copy of the database on it. Much data set shuffling was done via magnetic tape and a 7978B tape drive.

The disc space problem was aggravated when we tried to build the HISTDB database. The schema specified 24 data set pairs for the HSTRY-SHARE-FILE information plus an additional 24 data set pairs for the loan information (to be implemented in

the future). These data sets were defined with "reasonable" capacities. We ran DBSCHEMA and error-terminated with the infamous file system error 46 (out of disc space). It was then that our back of the envelope calculations showed that we needed at least 14 million sectors of disc space to build HISTDB (we had nothing near that much disc space available). The real problem was that we didn't really need all those data sets, just the definitions in the root file so that IMAGE would recognize them when present. Our solution was simple: we created the database root file using DBSCHEMA. Using DISKED5, we changed the flag in the root file from a "virgin" database to a functional database. IMAGE was happy.

We encountered one problem that almost spelled the end to the entire system. After we had created one of the data set pairs (due to an extraction with SELCOPY), we inserted the data into the framework of the new HISTDB database. We then tested the structure of the new database by trying to access various data with QUERY and comparing it against the original TECUDB database. Much to our shock, QUERY's FIND did not work! In the original database, FIND retrieved the correct data. Using HISTDB, FIND did not find anything. What was wrong? Careful examination of the structures of the two databases eventually revealed up the difference: the blocking factors of the respective data sets. While we had known that the capacities of the master data sets must be identical in order for the hashing algorithm to work, we had forgotten our IMAGE internals knowledge that the actual record address is a function of the block number and record within the block. We had to build our data sets with the correct blocking factors. Unfortunately, there is no utility that sets the blocking factor to that which we wanted. Nevertheless, a technique was found. We built the data set with a BLOCKMAX specification slightly larger than necessary and with a capacity of one too large. The credit union owns ADAGER; this program was then used to reduce the capacity by one and to reblock the data set. The new blocking factor then became identical to the original one in TECUDB.

We tested the integrity of the data after moving a data set pair from TECUDB to HISTDB. While the FIND command within QUERY worked (after fixing the above problem) the FORM SETS command did not. The display of the current number of entries was incorrect, garbage. A little more internals' knowledge reminded us that the current number of records is a calculated value, based on the capacity. Our HISTDB database detail data set had been built with a capacity of one million entries. We had transferred HSTRY-SHARE-FILE from TECUDB with 4+ million entries. While the output from QUERY's FORM SETS was incorrect, no processing was affected by it so long as no attempt was made to add any data.

During the testing of the HDBGET procedure, we uncovered one further interesting problem. The online application software, as part of its start up procedures, automatically accesses every data set needed in the database in order to initialize the IMAGE list parameter (for later use via the "*;" construct). With the addition of the HISTDB database, this wasn't properly extended. We had to add code to HDBGET to perform the same functionality within the procedure.

General Conclusions for Future Power Users

We once told one of the analysts here that, "Anyone who says he doesn't need to know the machine code for the machine he's working on is blowing smoke ... somewhere." Unfortunately we still believe this. In order to accomplish the HSTRY-SHARE-FILE split we needed an understanding of the application, IMAGE, and MPE. One of the most critical elements was a knowledge of the dependancies between the root file, the master file, and the detail data sets. The isolation of details from root and master files provided us with the first of two bases for the entire project and almost trashed the project three quarters of the way through. A late-night discovery of the dependance between the root file and the master data set, an unintentional Adager reblock, and the problems associated with recovering these misfortunes certainly convinced us that an exact understanding of IMAGE internals was critical.

The second basis for the HSTRY-SHARE-FILE split is that of 'SL isolation.' By that we simply mean isolating the actual data structure behind an SL routine. The critical point to this is that you are able to isolate the data structure and debug existing code BEFORE the actual structual changes are ever made. This gives you a staged implementation rather than simply going for broke; essentially the old CYA principle. Over the past four years TECU has used SL isolation in three (now four) major cases with dramatically successful results. One such implementation saved 24 hours of run time per quarter and 10 hours per month, eliminated the down time caused in both those periods, and was accomplished with only one minor problem.

The driving idea behind this exercise is that of a 'logical data structure.' Certainly we have heard much of this with regard to relational DBMSs but very little has been said in the non-relational field. Logically speaking, TECU had a single data structure, HSTRY-SHARE-FILE, which contained multiple periodic entities. We split the periodic entities into separate physical parts while retaining the logical continuity; thus providing physical flexibility while retaining logical support for the extant data structure.

Perhaps we've been living right (dubious, at best) or perhaps we got lucky (highly probably) but we did obtain some unexpected advantages from the conversion. First, we are now able to size the details arbitrarily (so long as we retain the auto master and detail blocking factors) and we may have an arbitrary number of details. Secondly, because of the nature of TECU's applications, the physical sequence of the data now corresponds to the chronological sequence of the records. This allows for such minor items as a chronological binary search of a detail to find all records inside a given period. This in turn allows us to quickly find and access records in a given period where we previouly would have had to search the entirety of the detail data set. Of course, sometimes one neither lives right nor gets lucky and then structural damage to one's database may occur. Prior to the HSTRY-SHARE-FILE split there was no way to recover from something as simple as a broken chain. Now we can simply wait until the pair with the break rolls out of the HISTDB. This is made even more impressive when one considers the fact that TECU has not done an unload/reload on their primary database since 1980!

Finally, there is one overriding consideration to each and every step of this procedure; we are doing this for intermediate and end users. Thus we must look at the external interfaces from the users' perspective; in this respect we failed in one sense and succeeded in another. In the first case we failed to realize that operations does not care about data set pairs but, rather, about HSTRY-SHARE-FILE date ranges. Therefore we used the pair IDs for moves into and out of HSTRY-SHARE-FILE as well as HISTDB. Additionally, we overlooked the loss of access to the structure through Query, Inform, Report, etc. In the latter case we succeeded rather admirably with our programmers in that they noticed little or no coding/performance differences between the old and new calls.

Traditional techniques offered no solution to our problems of managing a large data set. Due to the particular characteristics of the TECU database, we were able to design a technique utilizing safe privileged mode access that surmounted these obstacles. While these techniques cannot always be used in every environment, recognizing the existence of such techniques can be helpful for other companies designing sophisticated applications. This design became reality in August, 1987, when the Telephone Employees Credit Union went live with the new HISTDB database.