IMAGE's COMING OF AGE: Breaking free from restrictions to Data-Base transformations.

F. ALFREDO REGO

Chairman, Software Department
Instituto de Informática y Ciencias de Computación
Universidad Francisco Marroquín
6a. Avenida 0-28, zona 10
Guatemala, GUATEMALA.

ABSTRACT

A computerized data base should reflect an organization's way of behaving. As real-world circumstancies change, forcing the organization to adopt new ways and abandon old ones, the data base should also adapt itself.

Hewlett-Packard provides tools, such as DBUNLOAD and DBLOAD, which allow a limited set of transformations to IMAGE/3000 data bases. But these tools do not lend themselves to the easy implementation of the radical transformations that are sometimes necessary. The restrictions of these tools, we feel, are analogous to the do's and don'ts imposed on children by loving parents.

Taking into consideration that children (just like computer users) eventually come of age and will do their own thing despite formidable restrictions, we have developed a software system called "DATABASE.UTILITY" to help IMAGE/3000 users out of their data-base transformation predicaments.

"DATABASE.UTILITY" is an MPE 'group.account' that contains a set of software modules designed specifically to allow a large selection of transformations to IMAGE/3000 data bases without having to mess around with magnetic tapes or schema recompilations. And, in good parental fashion, this system also keeps a watchful eye for any possible difficulties that might potentially upset the health, consistency and integrity of the "adolescent" data base.

KEYWORDS AND PHRASES

Concurrent data-base operation and evolution, data-base adaptability, data-base consistency, data-base conversion, data-base design, data-base integrity, data-base redesign, data-base restructuring, data-base management systems, DBMS's, data-base transformation, Hewlett-Packard's IMAGE/3000 Data-Base Management System, root file transformation, schema changes.

MOTIVATION FOR THE DEVELOPMENT OF "DATABASE.UTILITY"

How can I be ABSOLUTELY sure that my data-base design is perfect? How can I GUARANTEE that I will NEVER have to change it to meet unexpected shifts in my organization's way of doing things?

If I can not answer these questions to my satisfaction, then what type of tuning (and fine-tuning) tools do I need to facilitate the constant and inevitable evolution of my data base?

What type of questions worry me about the tools I have currently available to me? And what type of questions linger in my mind as I dream of better and more effective ways to do what I have to do anyway?

- Why do I have to COMPLETELY STOP the operation of my live data base, even when I only want to make very slight changes like password reassignments? Could I maintain concurrent data-base access while I do certain non-radical transformations or while I radically transform data sets that are not being currently accessed? ("DATABASE.UTILITY" ANSWERS: yes.)
- Why do I have to spend (a sometimes very long) time to DBUNLOAD my WHOLE data base to magnetic tape before I transform my schema (assuming, of course, that I do not want to lose the live data I presently have!)? Could I skip the whole DBUNLOAD trip? ("DATABASE.UTILITY" ANSWERS: yes.)
- Why do I have to spend (a sometimes even longer) time to DBLOAD my previous data base, even though I merely want to optimize the storage locations of a primary path's entries? Could I simply reshuffle these entries without having to think and worry about the consequences of having to reshuffle the whole data base as well? ("DATABASE.UTILITY" ANSWERS: yes.)
- Why do I have to PURGE my entire data base, when all I want is to change the name of a data item? Could I simply make changes such as this without having to kill (and then re-issue life to) my data base? ("DATABASE.UTILITY" ANSWERS: yes.)

- Why do I have to EDIT and recompile my schema, when I simply want to change the read/write capabilities of a user class? Could I dinamically do this while the data base continues to earn its living? ("DATABASE.UTILITY" ANSWERS: yes.)
- Why do I have to CREATE, from the newly produced root file, a brand-new data base, if the old one was just fine except for the capacity of a data set? Could I change the capacity of a data set without having to go through this process once more? ("DATABASE.UTILITY" ANSWERS: yes.)
- Why am I at the mercy of subtle schema changes that CAN cause very unpleasant surprises, even after my previous data base has apparently been successfully DBLOADed to my new data base? Could I have some 'editor' which would make sure I do not clobber my schema? Could I know, before I ruin anything, that my data-base transformation request is illegal? Could I have a dialogue with the system to "discuss" the possible consequences of subtle changes in transformation requests? ("DATABASE.UTILITY" ANSWERS: yes.)
- Why do I have to write special application programs whenever I need to transform my data base in ways that are not supported by IMAGE/3000's transformation utilities? Could I have a flexible, non-procedural system that would even assemble data entries from bits and pieces taken from other data entries from the same data base, or from other data bases, or even from good old MPE files? Could I do data-type conversions (from integer to byte, from integer to double-integer, from byte to real, from integer to logical, from floating-point to byte with decimal-point suppression and decimal-place right-justification, etc...) if the source data type does not match the destination data type? ("DATABASE.UTILITY" ANSWERS: yes.)

DESCRIPTION OF "DATABASE.UTILITY"

"DATABASE.UTILITY" is an MPE 'group.account' with privileged capabilities assigned to it by the computer installation's system manager.

All our design trade-offs have one main objective: to preserve data-base consistency and integrity. We strongly feel the same way about preserving other user's domains and, of course, about preserving the operating system itself! Therefore, all privileged instructions in "DATABASE.UTILITY" are executed in bracketed fashion (that is to say, the programs execute in user, non-privileged mode 99% of the time; whenever it is imperative that privileged instructions be executed, a dynamic call to the

GETPRIVMODE system intrinsic is made immediately BEFORE the privileged instruction; then, a dynamic call to the GETUSERMODE system intrinsic is made immediately AFTER the privileged instruction.)

A good 90% of all module execution times is spent in making reasonably sure that the requested transformations are legal and will not produce unpleasant results. Complete log-on subsystems, analogous to MPE's, check to see that only authorized users access the programs. An IMAGE/3000 data base (of course!) is kept for all programs, users and transformations as applied to the various data bases in an installation.

At the least sigh of trouble, the target data base or data set is purged and the old one can be salvaged.

When necessary, the root file is appropriately "updated"; MPE files are created or purged as needed; data sets are re-organized to include or exclude structural information; data sets and data items are re-numbered if any intermediate elements have been eliminated, etc.

The Data Base Administrator (DBA) can easily obtain an up-to-date picture of the transformed data base by means of QUERY's "FORM" command and our own "PASSES" program. "FORM" lists data sets, data items and paths as defined within the data base's structure. "PASSES" lists passwords and user read/write classes.

DESCRIPTION OF "DATABASE.UTILITY" FUNCTIONS

1) NON-TRANSFORMATIONAL

ASSEMBLE: Assembles data entries ("records") for master or detail data sets from one or more data sets or MPE files. The source data sets may be mixed from several data bases and may be either details or masters. The source MPE files may be accessed sequentially or directly by key.

The program asks all relevant questions, such as data-item types (integer, byte, logical, etc.), beginning byte or word in the source entry/record, etc. If it detects inconsistencies (for instance, if the source data-item type is byte and the target data-item type is double-integer), it explains them and requests instructions to perform one of several possible actions, depending on the particular circumstances: ignore and re-try? do data conversion? ...

2) GLOBAL DATA-BASE TRANSFORMATIONS

COPY: Copies a data base from a source 'group.account' to a destination 'group.account'. The data-base creator MUST RELEASE the data base beforehand.

RELEASE: Releases the root file and all MPE privileged files assigned to a data base, so it may be copied from another 'group.account' (or accessed through QUERY or other application programs when running from other 'group.account')

SECURE: Reverses the effect of "RELEASE", securing the data base from access through other "group.account'.

RENAME: Assigns a new name to a data base. Changes MPE file names as well as internally-kept IMAGE names.

PASSES: Reports, lists, modifies, assigns, re-assigns, takes away, etc., maintenance passwords and read/write passwords and class numbers.

PURGE: Purges the root file and all MPE files assigned to a data base. Loops around asking for other data bases to be purged, instead of ending after having purged a data base as "DBUTIL, PURGE" currently does.

AUDIT: Produces a report of the usage of "DATABASE.UTILITY" programs by user, program, data base, etc.

3) TRANSFORMATIONS OF DETAIL DATA SETS

NEWDTAIL: Adds new detail data sets to the data base (with the appropriate new data items, if needed.)

CAPDTAIL: Modifies the capacity of a detail data set, preserving all current chains and making sure, in the case of a decrease in capacity, that the target capacity is at lease equal to the lowest permissible capacity for the given set's status.

KILLDET: Deletes a detail data set, making sure that it is safe to do so. It optionally dumps the entire set to an MPE file in the format specified by the user (or dumps only those data entries within the data set that meet the boolean specifications given by the user.)

4) TRANSFORMATIONS OF MASTER DATA SETS

NEWMASTR: Adds new automatic or manual master data sets to the data base (with the appropriate new data items, if needed.)

CAPMASTR: Modifies the capacity of a master data set, preserving hashing properties for calculated access and chain-head structural information. Reduces synonym occurrences by suggesting program-calculated prime-number capacities in the neighborhood of the user-specified capacity.

KILLMAST: Deletes a master data set, making sure that it is safe to do so and that no chains will be left hanging without chain heads. It optionally dumps the entire set to an MPE file in the format specified by the user (or dumps only those data entries within the data set that meet the boolean specifications given by the user.)

5) TRANSFORMATIONS OF DATA ITEMS

NEWITEM: Adds new items to existing data sets.

KILLITEM: Deletes a data item from an existing data set. If all data-set references to a given data item have been deleted, the item is also deleted from the root's item table.

RDEFITEM: Re-defines the type of a data item (from integer to byte, for instance), and does all the appropriate data conversions if necessary. If the new data type has a different word-length count, all data sets that reference the given item are re-organized to reflect the new structure.

6) TRANSFORMATIONS OF ELEMENT REFERENCES

NEWNAME: Assigns a new name to a data item or a data set. Checks non-duplicity and legality of new name.

7) TRANSFORMATIONS OF ACCESS PATHS

NEWPATH: Defines a new path connecting an existing master data set to an existing detail data set by means of an existing data item (i.e., it upgrades non-key data items to the status of key data items or SEARCH ITEMS.)

CLOSPATH: Deletes a path between a master data set and a detail data set. (i.e., it downgrades key data-items or SEARCH ITEMS to the status of non-key data items.)

SORT: Upgrades non-sort data items to the status of sort data items for a given path.

UNSORT: Downgrades sort data items for a given path to the status of non-sort data items.

PRIMARY: The path most frequently accessed in chained mode should be specified by the Data Base Administrator as the primary path for a detail data set. Should this state of affairs change, the DBA can specify that another path become the primary path for the detail data set by means of this program.

PAVEPATH: Reshuffles the entries of a detail data set so that the entries of each chain within the primary path will be in contiguous storage locations (for efficiency's sake in chained retrieval.)

-- CONCLUSIONS --

A) PARTICULAR:

"DATABASE.UTILITY" worries and keeps track of all IMAGE/3000 internal housekeeping, while the data base evolves.

The user is, then, free to concentrate his/her energies on the ONLY housekeeping task that really matters to him/her: THE DATA BASE'S ACCURATE REFLECTION OF THE ORGANIZATION'S WAY OF DOING BUSINESS.

The user can, now, specify WHAT he wants to have in his data base, knowing that tomorrow he can easily re-specify his requirements without having to fear lack of compatibility. HOW this is accomplished is the responsibility of "DATABASE.UTILITY".

B) GENERAL:

Our research and development team has concentrated its efforts on helping users of Data Base Management Systems realize the tremendous potential of this emerging computer technology.

In this report we mention only some of the projects that keep us busy and happy in the Land of Eternal Spring, Guatemala. Currently, we have software systems in various stages of development. The wide range of status is illustrated by the fact that some have been successfully installed at customer sites after alpha, beta and gamma tests and some others have just been dreamed up (such as our data dictionary project.)

To widen the scope of our activities, we would appreciate hearing from those researchers, developers, users and friends of Data-Base Management technology who share our enthusiasm. We will carefully examine and consider all suggestions and criticisms as well as any proposals for cooperation.

ACKNOWLEDGMENTS

We wish to express our appreciation to all our associates and friends who, directly or indirectly, made our software projects possible. In particular, we are indebted to: Jon Bale, Gabriel Buzzetti, Enrique Castillo, Alex Dengo, Pablo Gutierrez, Lissa Hanckel, Max Holzheu, Jerry Johnson, Einar Klanderud, Orly Larson, José Mirón, Félix Montes, Manuel Ponciano, Clara María Ramírez, Patricia Springmühl de Rego, Simón Sibony, Sergio Tenenbaum, John R. Trudeau, Fred White, René Woc.