Database Genetics and Cosmetics: The Adager Approach

F. Alfredo Rego
Adager Corporation
Sun Valley, Idaho 83353-3000 • *USA*

www.adager.com

Root files and dataset files

In my Christmas card of 1978, I mentioned that a database transformation involves two distinct (equally delicate) operations: a genetic change in the root file and a cosmetic adjustment in the datasets. Little did I know that the next three decades of my life would be dedicated to learning and implementing the painstaking technologies required by such a simple concept.

To appreciate the issues involved, let's look at a familiar example. As unbelievable as it may seem, some people with straight hair would prefer wavy hair (and some people with wavy hair would prefer straight hair). If you doubt the validity of this observation, just go to your neighborhood beauty parlor! The beauty salon provides cosmetic changes to hair that already exists. Unfortunately, the genetics remain unchanged and any new hair will stubbornly come out looking as it always did (a little more damaged, perhaps, but still basically the same).

In human terms, cosmetic engineering is "mature", whereas genetic engineering is a new (and somewhat frightening) area of concern. The opposite is true in database genetics and cosmetics. The database genes, kept in the root file, are trivial to modify (the obvious thing to do is to edit a schema and run DBSCHEMA to create a new root file). The true challenge involves the cosmetics: How do you synchronize the existing datasets to the new root file? How do you map millions of entries from the old datasets to the newly transformed datasets in an efficient way? You cannot afford to have your database "down for maintenance" for too long. Therefore, you ***must*** do the required cosmetic changes as fast as possible.

State transitions

Cosmetic changes in a database require the mapping of datasets. There are many ways to map datasets within the constraints of the

HP3000. How does Adager transform a database? By means of two internal data structures which model the current and original states of the database. At the beginning, the "current" state is identical to the "original" state. As you specify changes, the current state evolves.

Applying changes

At any time during the specification phase, you may ask Adager to apply the changes or you may continue specifying new changes. If and when you decide to apply the changes, Adager will use the "original genetics" to read your data from the "original datasets" and will transform your data in such a way that the new datasets will reflect your accumulated specifications.

Consolidating changes

Adager will do its best to minimize wasteful operations. In most cases, this means that Adager will consolidate as many changes as possible while it scans a transformed dataset. After lots of trials (and errors), we found that certain operations were not very sociable and did not interact well with their fellows. Instead of spending (even more) sleepless nights trying to figure out a way to civilize them, we decided to segregate them, since they were like hermits anyway. They performed best by themselves and the other transformations performed best by themselves. The two segregated kinds of Adager functions are SOME types of "fixing" and MOST types of "transforming" operations. For instance, we found that it was a better idea to repack datasets either before or after transforming them (repacking is a certain type of "fixing", since it fixes inefficiencies in the access of entries within datasets). However, we also found that fixing broken free-entry lists in detail datasets could be trivially done while transforming the datasets. So, as frustrating as it is, there are no absolute rules!

Sequence of events

Adager uses a critical-path method (CPM) to decide which sequence of operations has the highest likelihood of minimizing the total elapsed time. If you prefer to use your own sequence, you can certainly ask Adager to do its functions one-at-a-time.

Adager's syntax and DBSCHEMA's syntax

The syntax of your specifications to Adager is remarkably similar to DBSCHEMA's syntax. This means that you do not have to learn anything new: If you can specify a database through DBSCHEMA, you can maintain and modify that database through Adager. The similarity stops there, though, since Adager's response to illegal specifications is more forgiving than DBSCHEMA's and will allow

you to correct illegal specifications on the spot (if you are in session mode).

Some computer-science jargon: Adager's stacks and recursive operations

Since Adager's internal data structures are implemented as *stacks*, you can do some really neat things that would be impossible otherwise. For instance, you can simultaneously rename and relocate items and datasets without causing any confusion in Adager's logic and you can easily specify changes in a recursive manner, such as requesting a path to a non-existing dataset. In this case, Adager will let you specify the new dataset which, in turn, may have new fields that are not defined as items yet. Adager knows all of this and guides you through the required recursive maze. Once you clear up all these pending pieces of business, Adager returns you automatically to the task that began the whole recursive trip and you can continue on your merry way!

Different versions of IMAGE databases

The genetics and cosmetics of IMAGE/3000 are slightly different from the genetics and cosmetics of TurboIMAGE and IMAGE/SQL. But the fundamental Adager approach remains the same: Adager handles all mutations with equal ease.