The Use and Abuse of Non-hashing Keys in IMAGE

Fred White
Senior Research Scientist

Adager Corporation
Sun Valley, Idaho 83353-3000 U.S.A.
Tel. (208) 726-9100 *Fax* (208) 726-8191
fred@adager.com *http://www.adager.com*

Background

The first set of specifications for IMAGE/3000 proposed that all master dataset primary address assignment be accomplished by hashing the key of each master entry and reducing the result modulo the capacity of the master dataset.

In late November of 1971, some reviewers suggested that application designers might want the application to have control of this primary address assignment and requested that we modify our specifications to provide such a capability.

Our initial response was to require the key field of such masters to be a 16-bit (or 32-bit) integer whose value would be treated as invalid if it was less than 1 or greater than the capacity.

If we had left it at that, we would have provided the user with master datasets using this simple direct access method.

This was aesthetically displeasing in that it restricted both the length of keys and the values of keys.

We eliminated the former restriction by allowing any key length while using only the low order 31 bits as input in calculating the primary address.

We eliminated the latter restriction by reducing this 31-bit value modulo the capacity N (with a zero result mapping into N).

The user also had to have some method of specifying to IMAGE, via the database schema, which type of primary address calculation to apply for each master dataset.

It seemed only natural that IMAGE should apply hashing to keys of data types U and X and that the "integer" data types I, J and K were perfect for use in the generalized direct access

method of primary address assignment. It was less obvious which method to employ with data types P, Z and R.

Then one of us noticed that the HP3000 internal representations of the mantissas for IMAGE data types I, J, K and R were all in binary format and that P and Z were not. On this rather weak basis we decided that all keys of non-binary format would employ hashing and that the others would not.

Thus, hashing is employed for keys of data types U, X, P and Z but not for keys of data types I, J, K and R.

Synonyms

Two or more key values are said to be synonyms if they are assigned the same primary address.

Whenever a new entry is assigned a primary address which matches that of one or more existing entries, IMAGE locates it at an alternate address close by its synonyms.

It attempts to place it in the block containing the entry at the primary address. If this block is filled, IMAGE serially searches subsequent blocks until it finds an unoccupied location to place the new entry.

IMAGE links all synonyms for a given primary address together into what is known as a synonym chain.

If there is no severe clustering (see below) and if the dataset is not almost full and if the blocking factor is large relative to the average chain length, most synonym chains will reside in a single disc block and thus have little impact on performance since they can all be made present in memory with a single disc read.

For hashing keys, the average synonym chain length can be kept small by:

1. using keys at least 10 (preferably 12) characters long. (IMAGE's hashing algorithm does a better job with long keys than with short ones.)
2. using key values which are not excessively uniform in their content
3. using capacities 15 to 30 percent larger than the expected number of entries

Also, their negative impact on performance can be reduced by:

1. making the blocking factor large relative to the average synonym chain length.
2. not allowing the dataset to become nearly full.

For heavily accessed masters, try to have a blocking factor of at least 6 or 8. If your blocking factor is less than 6, consider replacing the manual master with an automatic master and a

related detail containing the data portion of the original manual master.

Clustering

It is entirely possible, particularly for non-hashing keys, that many of the records of a dataset will fill contiguous blocks in one or more portions of a dataset while other portions are empty, or nearly so. Such a phenomenon is referred to as clustering.

Clustering is harmless as long as there are no synonyms. Otherwise, clustering is typically dangerous, as we shall see.

The Use of Non-Hashing Keys

An excellent use of the direct access method provided by non-hashing keys would arise if the key, for example, were DAY-OF-YEAR.

In this situation, the master dataset would only require a capacity of 366 (any higher would be wasted disc space).

The data item DAY-OF-YEAR could be defined as type I and the key values would be the positive integers 1, 2,..., 366.

As long as the application prevented other key values from occurring, no synonyms would ever arise, there would be no waste space, and IMAGE performance would be optimal.

Note also that the record with key value of 1 would be record number 1, the one with key value of 2 would be record number 2, and so forth. This “natural” ordering might be of some advantage to your application.

You may find other such situations, perhaps involving badge numbers, building numbers, or whatever, where you might want to employ integer keys (i.e., data types I1 or I2) in this manner. Usually, however, this will involve some wasted disc space. Only you can decide if the wasted space is too exorbitant for the benefits offered.

The Clustering Pitfall

My first live encounter with a misuse of integer keys arose in 1978.

One Friday in 1978 I received a phone call from an insurance firm in the San Francisco Bay Area. I was told that their claims application was having serious performance problems and that, in an attempt to improve the situation, they had, on the previous Friday, performed a DBUNLOAD, changed some capacities and then started a DBLOAD which did not conclude until the early hours of Tuesday morning!

They were a $100,000,000-plus company which couldn't stand the on-line response they were getting and couldn't

afford losing another Monday in another vain attempt to resolve their problems.

Investigation revealed that claims information was stored in two detail datasets with paths to a shared automatic master. The search fields for these three datasets was a double integer key whose values were all of the form YYNNNNN (shown in decimal) where YY was the two-digit representation of the year (beginning with 71) and where each year NNNNN took on the values 00001, 00002, etc. up to 30,000.

Although the application was built on IMAGE in late 1976, the earlier claims information (from 1971 thru 1976) was loaded to be available for current access. I do not recall the exact capacity of the master dataset but, for purposes of displaying the nature of the problem (especially the fact that it didn't surface until 1978) I will assume a capacity of 370,000.

Although the number of claims per year varied the illustration will also assume that each year had 30,000.

The first claim of 1971 was claim number 7100001 to which, using a capacity of 370,000, IMAGE would assign a primary address of 70,001. This is because 7,100,001 is congruent to 70,001 modulo 370,000.

The 30,000 claims of 1971 were thus assigned the successive addresses 70,001 through 100,000.

Similar calculations show that the claims for each year were stored in clusters of successive addresses as follows:

Year	Claim numbers	Assigned addresses
1971	7100001-7130000	70,001-100,000
1972	7200001-7230000	170,001-200,000
1973	7300001-7330000	270,001-300,000
1974	7400001-7430000	1-30,000
1975	7500001-7530000	100,001-130,000
1976	7600001-7630000	200,001-230,000
1977	7700001-7730000	300,001-330,000

Note that no two records had the same assigned address and thus that there were no synonyms and that all DBPUTs, DBFINDs and keyed DBGETs were very fast indeed!

Along came 1978! Unfortunately 7,800,001 is congruent to 70,001 so that the first DBPUT for 1978 creates the very first synonym of the dataset. It is, in fact, a synonym of claim 7100001.

DBPUT attempts to place this synonym in the block occupied by claim 7100001 but that block is full so DBPUT performs a serial search of the succeeding blocks to find an

unused location. In this case, it searches the next 60,000 records before it finds an unused address at location 130,001! Even with a blocking factor of 50, this required 1200 additional disc reads making each DBPUT approximately 200 times as slow as those of all previous years!

Note that the next claim of 1978 (with claim number 7800002) is congruent to 70,002 so is a synonym of 7100002 and also leads to a serial search which ends at location 130,002! Thus each successive DBPUT results in a search of 60,000 records 59,999 of which it had inspected during the preceding DBPUT!

Clustering had claimed another victim! The designer of this system had unknowingly laid a trap which would snap at a mathematically predictable time, in this case 1978. After struggling with this problem for months, the user escaped the clustering pitfall by converting to "hashed keys" (in both the database and the application modules); a very expensive conversion!

Note that the problem was not a synonym problem in the sense that synonym chains were long nor was it a "fullness" problem since the master dataset was less than 57% full when disaster struck.

The problem was due to the fact that the records were severely clustered when the first synonym occurred and DBPUT's space searching algorithm is efficient only in the absence of severe clustering.

Note that the performance of DBFIND and DBGET was excellent.

A similar, more modest pitfall would have been encountered if, in the above example, the claim numbers had been of the form NNNNNYY with the same capacity of 370000. In this case, the performance of DBPUTs, DBFINDs and keyed DBGETs would all degrade over time but would never reach the disastrous level of the DBPUTs of the example. In this case, the degradation would arise due to the length of synonym chains and due to local clustering.

Note that this modest pitfall can be eliminated simply by changing the capacity, for example, to 370010.

Note however that this problem would still arise if the capacity were merely changed, for example, to 370001.

The Synonym Pitfall

An even worse case would arise if the designer elected to use a key whose data type was R4 and whose key values were greater than zero and less than 10 million.

To understand why, one must be knowledgeable about the format of 64-bit reals as represented on the HP3000 family of computers.

The leading bit is the sign bit, the next 9 bits are the exponent, and the remaining 54 bits are the mantissa excluding the leading bit.

As a consequence, the floating point format of all integers less than 8,388,609 (2**23+1) is such that the low order 31 bits are all zeroes. Therefore ALL entries would be in a single synonym chain having the dataset capacity as its primary address!

In adding a new entry, DBPUT would have to traverse the entire synonym chain to ensure that the key value of the new entry was not a duplicate before adding it to the chain. This would have an impact on performance proportional to the number of entries and inversely proportional to the blocking factor.

Also, each DBFIND or mode 7 DBGET would, on average, be forced to traverse half of the chain to locate the desired entry!

I hope that you will NEVER use fields of data type R as key fields.

Summary

Remember that, in electing to use non-hashing keys, the designer has taken the responsibility for primary address assignment out of the hands of IMAGE and placed it in the hands of the application.

This should be done only if:

1. some benefit will be derived by their use,
2. the application has absolute control over key value assignments,
3. the values so assigned, together with the assigned dataset capacity, assure the designer that the application will never encounter the clustering or synonym pitfalls.

Footnote

Avid readers of IMAGE articles might be surprised at the absence of any reference to primary numbers as capacities for master data sets.

The reason for this is that I consider any argument for or against their use as, at best, an academic exercise in futility and, at worst, a “red herring”. Application designers and database administrators can realize far greater performance improvements by dealing with other, more significant, issues such as those addressed in this paper.