HYPERION SYSTEM 9 FINANCIAL MANAGEMENT SUBCUBE ARCHITECTURE

APPLICATION DESIGN CONSIDERATIONS

03.23.2006

CONTENTS

PURPOSE

The "subcube" describes the structure of data storage and retrieval in Hyperion's Financial Management solution[1] (HFM). The approach to subcube management in HFM has changed significantly with this release to improve performance and to facilitate much larger subcube sizes than were previously possible.

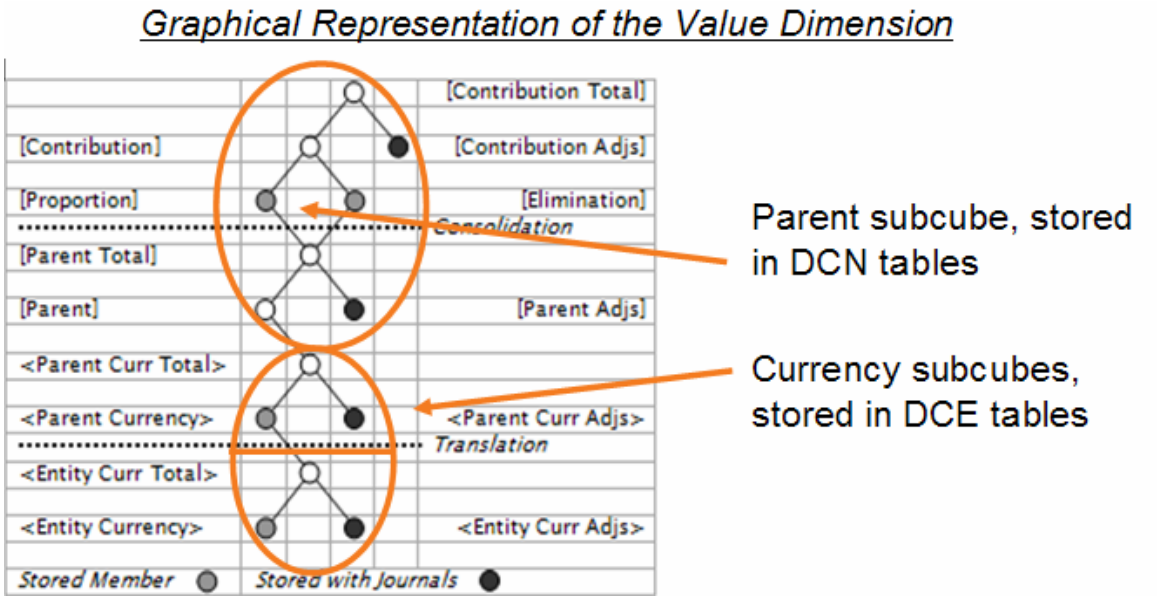
[1] Hyperion Financial Management is the product name for versions prior to System 9 Financial Management Release 4.1. With the introduction of Hyperion System 9 and Financial Management 4.1, the product is called *System 9 Financial Management.* The releases prior to System 9 are referred to in this document as *HFM*, and the System 9 releases are referred to as *Financial Management*. Unless otherwise specified, the structures and concepts that apply to HFM apply to both HFM and Financial Management.

DEFINING THE SUBCUBE

HFM stores data in one of three table types:

- **DCE (Currency subcube)**—Stores Entity Currency and Parent Currency values and their adjustments. These are often referred to as the currency triplets in the Value dimension:
  - The triplets are formed by the entity default currency, journal adjustments posted in the entity's default currency, and the total aggregated value of the two.
  - If the entity's parent has a different default currency, a second triplet of data is provided for the parent currency.
  - It is also possible for a user to force a translation into another currency in addition to the <Parent Currency>. The result of this forced translation is a set of stored data in the additional currency.
- **DCN (Parent subcube)**—Stores the remaining Value dimension members.
  - All data in the Parent cube is specific to a parent-child combination, and as such can only be stored or retrieved by specifying this relationship. DCN tables are structurally similar to DCE tables, with one exception: DCN tables contain an additional field for the parent entity ID (LPARENT).
- **DCT (Journal transactions)**—Stores the journal transactions, which when posted, transfer data values to DCE (for <Entity Currency Adjs> and <Parent Currency Adjs>) or DCN tables (for [Parent Adjs] and Contribution Adjs]).

The following graphic shows the Value dimension and which members are grouped into each subcube:



DATA TABLES

The naming scheme for each data table follows this order (examples in parentheses):

1. Application prefix (COMMA)
2. Table function name (DCN, DCE, or DCT)
3. Scenario ID ("1")
4. Year

The members in the following DCN and DCE tables identify the remaining members of the dimensions (Entity, Value, Account, ICP, Custom1-4). The input column represents the data. The InputTransType represents a binary flag for the status of the data.

One Input and one InputTransType is available for each period defined in the application. If the application contains 12 periods, 12 Input fields are named dP0_Input through dP11_Input. If it contains 365 periods, 365 Input fields are named dP0_Input through dP364_Input. One dPX_InputTransType is available for each dPX_Input column. The dTimestamp field identifies the time of the last change.

Table 1: DCE Data Table

COMMA_DCE_1_2000	DATA TYPE	ALLOWS NULLS	DESCRIPTION
LENTITY	int	NOT NULL	Entity ID
LVALUE	int	NOT NULL	Value ID
LACCOUNT	int	NOT NULL	Account ID
LICP	int	NOT NULL	ICP ID
LCUSTOM1	int	NOT NULL	Custom1 ID
LCUSTOM2	int	NOT NULL	Custom2 ID
LCUSTOM3	int	NOT NULL	Custom3 ID
LCUSTOM4	int	NOT NULL	Custom4 ID
DPx_INPUT [2]	float	NOT NULL	Period 1 Input data2
DPx_INPUTTRANSTYPE [2]	tinyint	NOT NULL	Period 1 Status flag2
DTIMESTAMP	float	NOT NULL	Time stamp of last data change

[2] DPx_INPUT and DPx_INPUTTRANSTYPE are repeating fields. One of each of these fields is provided for each period in the largest frequency in the Application. If the Application contains 12 periods, then there will be 12 DPx_INPUT and 12 DPx_INPUTTRANSTYPE columns in the table. The numbers are zero-based, so for 12 periods, the column names will be DP0_INPUT to DP11_INPUT and DP0_INPUTTRANSTYPE to DP11_INPUTTRANSTYPE.

Table 2: DCN Data Table

COMMA_DCN_1_2000	DATA TYPE	ALLOWS NULLS	DESCRIPTION
LENTITY	int	NOT NULL	Entity ID
LPARENT	int	NOT NULL	Parent Entity ID
LVALUE	int	NOT NULL	Value ID
LACCOUNT	int	NOT NULL	Account ID
LICP	int	NOT NULL	ICP ID
LCUSTOM1	int	NOT NULL	Custom1 ID
LCUSTOM2	int	NOT NULL	Custom2 ID
LCUSTOM3	int	NOT NULL	Custom3 ID
LCUSTOM4	int	NOT NULL	Custom4 ID
DPx_INPUT 2	float	NOT NULL	Period 1 Input data2
DPx_INPUTTRANSTYPE2	tinyint	NOT NULL	Period 1 Status flag2
DTIMESTAMP	float	NOT NULL	Time stamp of last data change

IMPACT OF THE APPLICATION PROFILE

As previously described, each data record contains values for all periods within a given scenario_year combination. Therefore, the greater the number of periods defined in an application profile, the greater the number of fields in each record and consequently, the greater amount of memory required to store the record.

The most common profile contains twelve months as the base level. Data records for such a profile consume approximately 120 bytes of RAM. More specifically, a data record occupies (8 bytes per period for each double-byte data value + 1 byte per period for the cell status) + 4 bytes for overhead = 12 periods * (8 + 1) + 4 bytes = 112 bytes per record. Profiles designed for weekly data input contain 53 base periods, and consume approximately 53 periods * (8 + 1) + 4 bytes per record, or about 481 bytes. Daily profiles, containing 366 base periods, consume approximately 366 * (8 + 1) + 4 bytes per record or 3,298 bytes.

This record size is for data only. The cell index is defined as the unique intersection of Entity, Value, Account, ICP, and Custom1…4. As of 4.1 the index is not stored with the data, but in a separate space in memory. Although the index is not part of the data record size calculation, it does consume part of the total 2 GB of available memory. See the later section "New In-Memory Representation of the Subcube" for more details.

The data record size is always determined by the lowest level of input, even though a given scenario may not use that size. That is, if an application profile contains 53 base periods, all scenarios will have 53 input periods in the data tables even if they do not use them. For example, an application using a weekly profile will contain 53 input fields for all scenarios, regardless of the frequency used for each scenario.

MICROSOFT WINDOWS 32-BIT MEMORY ADDRESS SPACE LIMIT

HFM runs on several versions of Microsoft Windows 32-bit operating systems. Such systems currently provide a single process only 2 GB of addressable virtual memory. Regardless of how much physical memory the application server hardware and bios can support, 32-bit Windows cannot provide more than 2 GB of RAM to a single process.

HFM uses two primary processes running on the application server:

- **HsxServer—**Manages user authentication and HFM system messages. One HsxServer process is launched per HFM application server. This action makes the initial connection between the HFM application server and the database server.
- **HsvDataSource—**Manages all data functions for HFM, and is the main process for HFM. This component does the majority of work on the application server. One HsvDataSource process is spawned for each application opened on a given application server.

WORKING WITHIN 2 GB OF RAM

HFM must constantly monitor how much memory the HsvDataSource process consumes, and regularly release memory. It uses a Least Recently Used (LRU) algorithm to purge or discard records from memory, to make room for new subcubes being loaded. This means that when a process such as a report, data load, or consolidation is executed, HFM counts the total number of subcubes in memory for an application every 15 minutes [by default] or every 500 additional cubes loaded into memory. If the total number of cubes in RAM meets or exceeds the registry value MaxNumCubesInRAM, HFM purges up to 100 percent of the cubes in RAM.[3]

HFM determines which cubes to purge based on the ones that have been in memory the longest without being used by a process. This purging is recorded in the system messages by a record beginning with "FreeLRUCachesIfMoreRAMIsNeeded released data cubes." It follows this comment with a total number of cubes in RAM and an aggregate number of records for those cubes on a "before" and "after" basis of purging. It also records the record volume for the largest cube in memory at that moment. HFM purges all its cubes in RAM unless a currently running process requires some of them. If a process requires them, some cubes remain in memory after purging.

If the total number of cubes in RAM is fewer than MaxNumCubesInRAM, HFM counts the total number of records in RAM. If this meets or exceeds the MaxNumDataRecordsInRAM value, HFM runs the FreeLRU process previously described.

If neither condition causes HFM to run FreeLRU, it continues processing until another 500 cubes are loaded, or 15 minutes pass. Possibly, during this time, processing may exceed the recommended limit of 1,500,000 records, which may result in a heap allocation failure when HFM runs out of virtual memory. This threshold is based on an analysis of customer applications in the field using HFM versions 3.4 through 4.0.2, inclusive.

[3] HFM 4.1 and later no longer use the MaxNumcubesInRAM setting. The purging mechanism is now based entirely on the MaxNumDataRecordsInRAM as described here.

OPERATING SYSTEM SOLUTIONS FOR EXCEEDING THE 2 GB LIMIT

Certain versions of Windows operating systems can enable a process to see up to 3 GB of addressable memory space. Microsoft calls this capability *4GT RAM Tuning,* or generally the *3 GB switch.* Regarding HFM memory management, the remaining discussion in this white paper still applies to the previous discussion. In all cases, however, the ceiling for RAM available to HFM is 3 GB.

List of supporting server operating systems:

- Windows 2000 Advanced Server
- Windows 2000 Datacenter Server
- Windows Server 2003
- Windows Server 2003, Enterprise Edition
- Windows Server 2003, Datacenter Edition

Hyperion has tested and supports the 3 GB switch for HFM Release 3.5 and later.

ENABLING THE 3 GB MEMORY FOR HFM

To enable the 3 GB switch for HFM using Windows 2000 Advanced Server:

1. Install one of the previously listed operating systems.
2. Change `boot.ini` to provide the /3GB switch.

   For example

   - `[operating systems]`
   - `multi(0)disk(0)rdisk(0)partition(1)\WINNT="Microsoft Windows 2000 Server" /fastdetect /3GB`
   - `reboot`

To enable the 3 GB switch for HFM using Windows 2003:

1. Install one of the previously listed operating systems.
2. Change `boot.ini` to provide the /3GB switch.

   For example

   - `[operating systems]`
   - `multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows Server 2003" /fastdetect /3GB`
   - `reboot`

To test the proper application of the setting:

1. Ensure that HFM is not running. (`HsvDataSource.exe` should not be running as viewed by Task Manager.)
2. In a command window, run `ImageCfg.exe -l memtest.exe`. (`ImageCfg.exe` is in the Windows Advanced Server CD)
3. Run `memtest.exe`.[4]
4. The `memtest.exe` utility will leak memory up to the amount available. That is, if 3 GB switch is properly enabled on the operating system, `memtest.exe` will leak memory past the 2 GB limit and will stop at 3 GB. The `memtest.exe` utility will display a leak counting in 1 Mb increments from 1 Mb to 2047 Mb, then proceed to count from -2047 Mb to -1031 Mb if the 3 GB switch has been properly enabled.

To enable HFM to make use of the 3 GB switch, run `ImageCfg.exe -1 hsvdatasource.exe`.

Links to further details from Microsoft on this topic:

http://support.microsoft.com/default.aspx?scid=kb;en-us;297812

http://www.microsoft.com/whdc/system/platform/server/PAE/PAEmem.mspx

PHYSICAL ADDRESS EXTENSIONS

Microsoft Windows offers another option for advanced memory management called *Physical Address Extension* (PAE). Applications (HFM in this case) can be written using AWE (the Microsoft application programming interface) to take advantage of PAE, as long as the applications are run on the operating systems previously listed in this paper. HFM, however, does not support PAE as of this writing. For more information on PAE, visit http://support.microsoft.com/default.aspx?scid=kb;en-us;283037 .

SUBCUBE SIZE LIMITATIONS

In releases prior to 4.1, Hyperion recommends a limit of 100,000 records for any one subcube in an application. This value is based on a standard 12-month profile. Larger profiles allow for smaller recommended data volumes (see the effect of the application profile discussed earlier).

This limit is not a hard limit, and it is possible to have subcubes much larger than 100,000 records without failing. It is much more important to keep MaxNumDataRecordsInRAM below the 1,500,000 threshold. The larger the individual subcubes, the greater the risk of running into this record limit.

EFFECT OF AGGREGATION ON THE SUBCUBE SIZE

As described earlier, HFM stores only base-level records in the database for Accounts, Customs, ICP, and Value dimension members. When loading the subcube, you calculate numbers for all

[4] Memtest.exe is a Hyperion-developed utility that tests the server memory configuration and displays the proper configuration of the 3 GB switch.

parent accounts (using only base members for the other five dimensions). The performance of this part is mostly dependent on the number of base numbers in the subcube as well as the number of ancestors in the account dimension (that is, the depth). In a worst case, for every base number, a new number is calculated for every parent account. For example, if the subcube contains 1,000 base numbers and every base account is 10 levels deep, you could end up with 10,000 numbers in RAM. The number of records in memory recorded in system messages reflects the aggregated subcube size.

Later, when a user asks for a number for a parent ICP member or Custom members, the system calculates only what is necessary to determine the requested number. This requires that you inspect each number for the specified account and then determine if the number contributes to the total. In general, this process goes fast because fewer numbers need inspecting (all numbers for just one account), and the result is a single number. Therefore, the depth of Custom dimensions does not make much difference.

In all cases, calculations are data-driven, not metadata-driven. That is, you never calculate by asking a member for all its children and then adding up thousands of NODATA results. You always look at the base data that exists, and then push results up to the parents. This approach works best for sparse subcubes.

You should distinguish between the type of calculations that HFM performs simply to generate parent members in Account, Custom, ICP, and the Value dimensions and the calculations that are controlled by Rules. The former calculations are *aggregations*, because they are controlled solely by the dimensional hierarchy and metadata attributes (aggregation weight for custom dimension members, for example). The latter *calculations* are entirely controlled through HFM Rules.

An important distinction must be made, because Rules can be run against base-level members of the Account, Custom, ICP, and Period dimensions only, certain members of the Value dimension, and any member of the Entity dimension, and are stored in the database. Aggregated members are generated dynamically in memory and are not stored in the database. Ultimately both types of calculations contribute to the subcube size, although usage determines the generated size of the subcube.

LIMITATIONS ON THE NUMBER OF CHILDREN FOR A GIVEN ENTITY

The recommended limit for any parent entity is 500 children, although this is a function of the total number of records in RAM at one time. During consolidation, HFM keeps the [Contribution Total] subcube for all its children in memory concurrently while it calculates and then writes the resulting records to the parent entity. You must, therefore, keep the total number of records in RAM for all these entities below about 1,500,000.

For example, if you have a parent entity composed of 500 children, each having 5,000 records in their respective subcubes, you can expect to have at least 500 * 5,000 = 2,500,000 total records. Additionally, the parent entity subcube in RAM is well beyond the limit of 1,500,000 total records. In this application, the parent must have about half the number of children to avoid a heap allocation failure during the consolidation of this level. Applications where the average cube size is much smaller may have more children for a given parent entity.

NEW FEATURES IN SYSTEM 9 FINANCIAL MANAGEMENT 4.1

Financial Management 4.1 overcomes the 2 GB memory limit by working more efficiently within that limit, and by introducing a secondary process that enables it to page data in and out, theoretically eliminating the subcube data volume ceiling.

IMPROVED MEMORY MANAGEMENT

Financial Management 4.1 adds several new features that reduce the amount of memory required by certain operations. This approach provides for more memory allocation to data records than in previous releases.

Earlier releases of HFM require that data for a given subcube be stored in memory all at one time. This process increases the memory requirements for HFM as a whole when you need to process many large subcubes. For example, in a report that lists a single account of Net Profit for 50 entities, you must load all records for those entities, not just Net Profit. If the chart of accounts had 100 members, we would see potentially all 100 in memory, not just Net Profit.

Prior to release 4.1, HFM memory management was structured to release subcubes in their entirety. Financial Management 4.1 instead purges on a per-record basis. It does this by tracking each data record with a "usage" index, enabling Financial Management to purge only those records within a subcube that have been least recently used. This more granular approach provides for much more effective memory management since the records set in memory has a much higher usage ratio. Using the earlier example, Financial Management 4.1 requires only Net Profit to stay in memory, rather than all 100 accounts.

Financial Management 4.1 can benefit from the 3 GB switch discussed earlier, and still leverage the new subcube management technology.

LAZY COPY

A "lazy copy" feature provides for fewer copies of subcubes in memory. Without this feature, if two users request data from the same subcube, a copy of the cube in memory is made for each user. In Financial Management 4.1, both users are pointed back to the same cube and no copy is made of the subcube unless one user must modify the subcube. Financial Management creates a copy of the subcube in memory only in cases where you modify records within the cube, such as during a data load or consolidation process. Consequently, more memory is available for additional data, compared to earlier versions of HFM. A further benefit of this approach is speed in processing, because it avoids making and managing extra copies.

NEW IN-MEMORY REPRESENTATION OF THE SUBCUBE

Another way to reduce memory requirements for Financial Management is to alter the way that data is stored in memory. Financial Management 4.1 splits the subcube into "index" and "data" portions. The index stores coordinates, or the location in memory for a certain record. This includes the record point of view in Financial Management terms. The data stores only data values for a record, typically twelve data points in a monthly frequency. This release of

Financial Management reduces the memory requirement and facilitates more efficient data access by rules, data queries, and aggregations.

PAGING

Probably the most "visible" new feature for Financial Management is the introduction of a paging mechanism for data records. Paging involves copying collections of data records from one cache to another, typically from a fast performing cache to a slower performing cache. Financial Management now provides an embedded database, BerkeleyDB. It facilitates paging and provides for a secondary data store for records that are not in active use although still residing in RAM.

Financial Management 4.1 introduces a Data Cache allocator specifically for managing application data record caches. The administrator can control the amount of memory reserved for data records allowing memory to be used for other tasks in HFM or other applications. For example, the MaxDataCacheSizeInMB setting controls how much of the 2 GB is used for data as opposed to VBScript (for rules and member lists), calculation status, or COM objects.

When Financial Management exceeds MaxDataCacheSizeInMB, it begins paging out to a temporary file on the hard disk. Reads from and writes to a disk are considerably slower than reads from and writes to memory; therefore, you must balance all activities to minimize paging to the disk. If performance of a consolidation operation decreases because of paging, you can increase MaxDataCacheSizeInMB to minimize paging.

As data is read into a cache, the allocated memory pool must expand. In Financial Management 4.1, a growth mechanism has been added, which minimizes the time required to expand the pool while minimizing fragmentation.

PERFORMANCE TUNING CONSIDERATIONS

- For small and medium-sized applications (that is, applications that do not require changes to MaxNumDataRecordsInRAM to increase the LRU size), you need not change the default settings.
- Applications that require large subcubes may require a larger MaxNumDataRecordsInRAM setting. (Either because a subcube contains many records, or because the size of a record is larger, as in weekly applications)
- You want to achieve a balance in the size of the LRU and the amount of memory assigned to DataCache. A large LRU size means that you hold too many records, which fill up the data cache faster, putting the system under more memory pressure. Too low a MaxDataCacheSizeInMB setting means that you will run out of memory to store data records and begin paging, which reduces system performance.
- Financial Management writes a data record to the disk in a file located in the application folder of the ServerWorkingFolder. For large consolidation operations, the size of this file may be close to 500 MB or bigger.
- Informational diagnostic messages about the paging subsystem are written out to the event log during a consolidation operation. A message looks like *"PagedNodes=0 SingleRefNodes=64649 PageOutOps=0 CleanPageOutOps=0 PageInOps=0 PageInThreadOps=0 NAllocs=371787 NFrees=307138"*.

This message provides the following information:

Field Name	Description and interpretation
PagedNodes	The number of records paged out to the disk. A nonzero value indicates that the system is paging records to the disk.
SingleRefNodes	The number of records in memory
PageOutOps	The number of page-out operations. A nonzero value indicates that the system is paging records to the disk.
CleanPageOutOps	The number of unmodified records paged out. A nonzero value indicates that the system is paging unmodified records to the disk.
PageInOps	The number of page-in operations performed. A nonzero value indicates that the system is paging in records from the disk.
PageInThreadOps	The number of page-in operations performed by the background page in the thread. A nonzero value indicates that the background pager is active and bringing in records from the disk.
NAllocs	The number of records allocated
NFrees	The number of records freed

Financial Management 4.1 Performance Tuning Settings

The following registry settings are defined as REG_DWORD values under **HKEY_LOCAL_MACHINE\SOFTWARE\Hyperion Solutions\Hyperion Financial Management\Server:**

Registry Setting	Default Value	Value Range	Description
MaxNumDataRecordsInRAM	1000000	5000-5000000	When total number of records in RAM goes above this value, FreeLRU is called to release records from memory to provide memory for the Financial Management server. The Financial Management server allocates this much space upfront upon startup to store the cell values and cell status.
MinDataCacheSizeInMB	100	10-100	Note: By setting this value at a higher number, you can reduce the number of DataCache growth attempts and hence reduce memory fragmentation. Typically DataCache is grown on a per need basis, and it will grow 25 MB maximum at a time.
MaxDataCacheSizeInMB	250	50-600	The maximum amount of memory that the Financial Management server allocates to store the cell values and cell status. If more memory is required by the system, the cell value and cell status are paged out to the disk, based on LRU logic.

			Note: Typically, you set this value to minimize "page out," because the system in "paged out" mode can reduce performance. You set this value for more than the total memory usage allowed by "MaxNumDataRecordsInRAM," so that the system does not page out cells to the disk unnecessarily. For example, for a 12-period application, if MaxNumDataRecordsInRAM = 2,000,000, MaxDataCacheSizeInMB should be at least 240. When the Financial Management server starts, a warning is generated in the event log if MaxDataCacheSizeInMB is not large enough.
ENABLEBACKGROUNDPAGEIN	1	0-1	Setting this value to "1" enables the Background Page In feature, which reads the paged out cells from the disk in the background and minimizes the disk I/O wait for the system. Keep this setting turned on to improve system performance. When set to "0," Financial Management pages a single record at a time, which degrades performance.