The 4D v11 SQL Data File Cache

By Josh Fletcher, Technical Services Team Member, 4D Inc.

Technical Note 09-43

Table of Contents

Abstract

The data file cache in 4D v11 SQL has been dramatically revamped. It can be much larger, is more efficient, and there are more tools available to find out what the cache is doing.

This Technical Note explains what the data file cache is, how it has changed, and what tools are available to investigate it.

Introduction

The data file cache (or simply "the cache") in 4D v11 SQL has been dramatically revamped.

With support for 64-bit operating systems and improved memory management in 4D, the cache can be much larger.

The cache has been redesigned in 4D v11 SQL to be much more efficient, especially in light of the increase in size.

Finally new tools have been added to aid the developer in tracking the usage of, and troubleshooting the cache.

This Technical Note explains what the cache is, how it has changed, and what tools there are to investigate it.

What is the Cache?

The data cache is a critical part of 4D. To understand just how critical it is, it is important to understand exactly what the cache is and how 4D uses it.

Note that the information in this section is not unique to 4D v11 SQL, unless otherwise noted, so it is useful in understanding previous versions of 4D as well.

The cache is nothing more than a large chunk of memory that is managed by 4D. Notice an important distinction even in this simple description: 4D manages the cache memory, as opposed to it being managed by the OS like the rest of the memory 4D uses. The idea is that 4D is in full control of the content of this memory.

It is really not like a hardware cache at all. True, hardware caches are designed to store content from a slower medium inside a faster one. The cache in 4D is, partially, designed to store disk content in RAM. For example, records and indexes are loaded into the cache from disk. However the cache is not just for **data file** objects. It is also used for all kinds of **data engine** objects (selections,

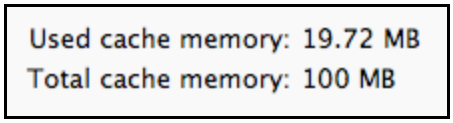
transactions, sorting data, etc). Here is a non-exhaustive list of the kind of objects that are kept in the cache:

- Structural definitions of tables, fields, relations, indexes, etc.
- General information about opened databases (file paths, properties, etc.)
- Data file allocation bit tables
- Address tables for records, indexes, blobs, extra properties, etc.
- Index pages
- Records
- BLOBs (may be allocated in main memory instead if not enough space in cache)
- Extra properties
- Sequence numbers
- Transactions
- Selections
- Sets
- Temporary buffers for sorting, read ahead, buffered disk write, etc.

As you can imagine, 4D is constantly managing this memory, juggling different kinds of objects in order to keep things as efficient as possible.

How is the Cache used?

In 4D Server v11 SQL there is a new statistic for tracking cache usage. This is reported in the 4D Server Administration dialog, under the Application Server page, as "Used cache memory":



Over the course of execution of a given database, the used cache value fluctuates up and down, sometimes going from the maximum value all the way down to the minimum.

If you think of the 4D cache as a hardware cache this is counter-intuitive; why would the value ever go down? Once objects are loaded in the cache they should either remain in the cache, or be replaced with newer objects, but never be removed, right?

Remember, 4D uses the cache for all kinds of objects! In order to make room for new objects, old objects must be removed. When you see the used cache go down, what this really means is 4D needed to remove some objects in order to make space for some data engine objects.

In fact this behavior is perfectly normal (meaning the cache is designed to work this way) and can also be used as a trouble-shooting tool. What is needed is a thorough understanding of how 4D manages the cache.

How Does 4D Manage the Cache?

If 4D needs to create some space in the cache for data engine objects, it must purge data file objects (and even other data engine objects). Note that a purge is not the same as a flush. Flushing modified objects to disk does not remove them from the cache, it only saves the changes to disk. On the other hand a purged object is no longer in the cache at all.

This is the algorithm that 4D uses to create space in the cache:

- Purge non-dirty data file objects. These are objects that can easily be restored because they simply need to be read from the data file again.
- If that does not create enough space, FLUSH BUFFERS.
- Purge non-dirty objects again. The flush operation should have created many more non-dirty objects, so it will allow 4D to purge more of them under normal circumstances.
- If that does not create enough space, copy other data engine objects to disk and purge them.
    - These stored objects will appear in the "temporary files" folder next to the data file.

If, after this process, 4D is still unable to allocate sufficient memory for data engine objects there are a few things to be aware of:

- First you will get an error message letting you know 4D could not allocate enough memory.
- If all objects were successfully purged from the cache (used cache dropped to 0), then your cache is too small. This is most common with tiny cache sizes (under 100 MB). More on this later.
- In theory this algorithm should be able to remove **all** objects from the cache. If it cannot there can be three problems:
    - Memory leaks: objects in the cache that are lost. Obviously this situation is not normal and should be reported as a bug.
    - Fragmentation: partially used blocks that prevent the free space from being accessible. This is an unfortunate reality of memory access in computers.
        - Some objects simply cannot be purged because they are locked (e.g. objects being used for an active sequential sort). This is "normal" but it is important to understand and manage this (see SET DATABASE PARAMETER selector 61).

Cache Changes

This section covers the changes that have been made to the cache in 4D v11 SQL.

Cache Size

4D v11 SQL features new memory management that allows it to access more memory, in general. This in turn allows larger caches to be allocated.

Although 4D v11 SQL is still a 32-bit application, and thus has the memory limits inherent to any 32-bit application, this limitation has doubled: 2GB was the maximum for 2004; the maximum for version 11 up to 4GB (depending on the configuration).

Even on a 32-bit OS, 4D v11 SQL will be able to access more memory than previous versions, though not that much more (10's of MBs at the most). However no one should be using 32-bit OS's for 4D, moving forward.

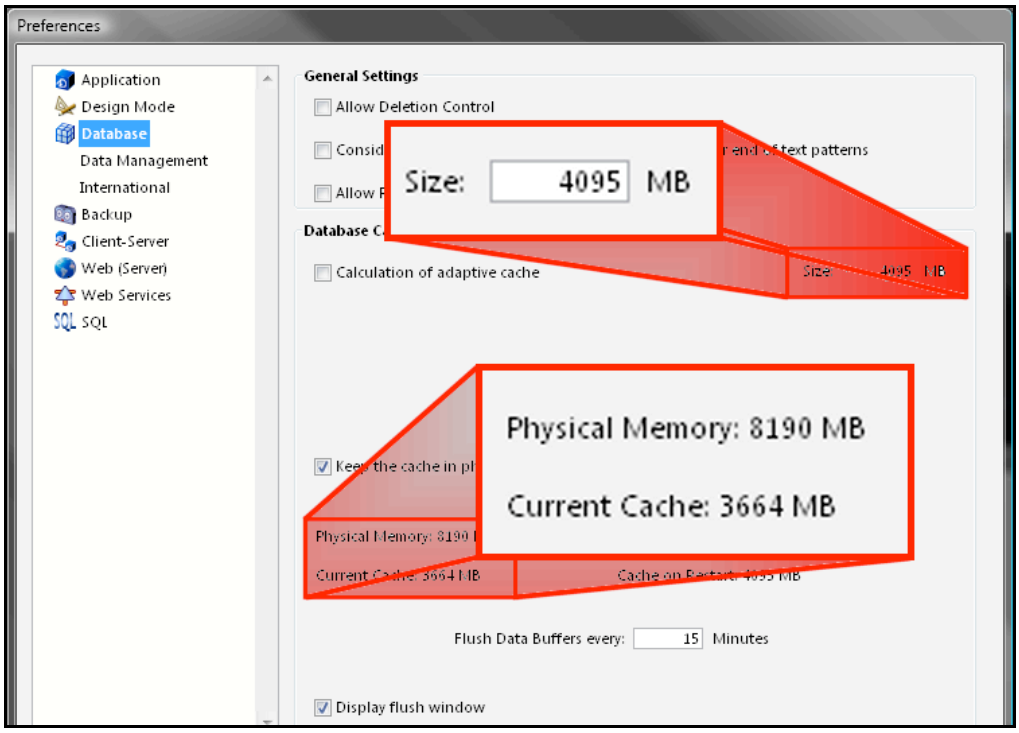
The 64-bit versions of Mac OS X and Windows Vista support up to 4GB per 32-bit application. To take advantage of all that memory, at least 8GB of RAM should be present in a machine dedicated to a 4D app. The reason for this is two-fold:

- A 64-bit OS will allow 4D to use the maximum amount of memory possible. To reach this maximum, it is necessary to have at least 8GB of RAM because of the way memory allocation works for 32-bit applications in 64-bit OS's.
- Again because of the nature of memory allocation in both Windows and Mac OS X, large amounts of RAM allow for the largest cache to be allocated. Both OS's tend to load DLLs/libraries into various areas of the system memory (e.g. not at the front or the back) so when 4D requests memory for the cache it is easier to run into limitations with less than 8GB of RAM.

Of course if the machine is not dedicated to running 4D it will be important to include more RAM to accommodate other tasks. On the other hand there is little benefit to adding RAM beyond the 8GB threshold if the machine is dedicated to 4D because 4D will still be limited to 3-4GB of memory for itself.

Here is an example of 4D Server v11 SQL 11.4 running on a 64-bit Vista machine with 8GB of RAM:

Cache Settings:



Note that the maximum possible value (4095) was specified for the cache in this test simply to ensure that the largest cache possible would be allocated. It is **not** recommended to use this value in practice. If the system really was able to allocate 4095 MB for the cache, then 4D would have only 1 MB left for non-cache memory (for things like processes).

Thus, to get the largest cache possible for 4D v11 SQL:

- Use a 64-bit OS.
- Install at least 8GB of RAM in the machine.

Redesign

The cache in 4D v11 SQL has been dramatically revamped to make it more efficient.

As previously mentioned, for one thing the cache can be much larger. In previous versions of 4D it was recommended never to set the cache too large. This may seem counter-intuitive because the cache should allow quicker access to the data, so if it is bigger it can hold more data. The problem in previous versions was that there were circumstances where 4D might need to search the cache and also the overhead of maintaining a large cache could adversely affect performance. The redesigned cache in 4D v11 SQL does not suffer from this issue.

- The new cache is 3-way associative. If this makes sense to you, great! If not, don't worry about it. The fundamental advantage is that there is never a sequential search any longer to find cached objects or free space. The addresses are computed directly.
  - This improves performance especially for large databases.
- The cache scoring system uses two metrics:
  - Number of accesses (a la 2004).
  - Most recently accessed (new).
- The cache has two indexes for finding objects:
  - One sorted by size.
  - One sorted by number of accesses.
- Objects are also sorted by disk address so flushing is faster.
- Records are loaded in "blocks", e.g. 1000-2000 at a time. Thus regular compacting is recommended, when practical.
  - Records are also flushed in blocks instead of individual records, when possible.
- To get the largest possible cache, run 4D on a 64-bit operating system. This will give you the highest potential for the largest chunk of contiguous memory (remember, cache must be contiguous).
  - Keep in mind that setting the cache very large takes away from the memory 4D can access for other resources (processes, etc.).

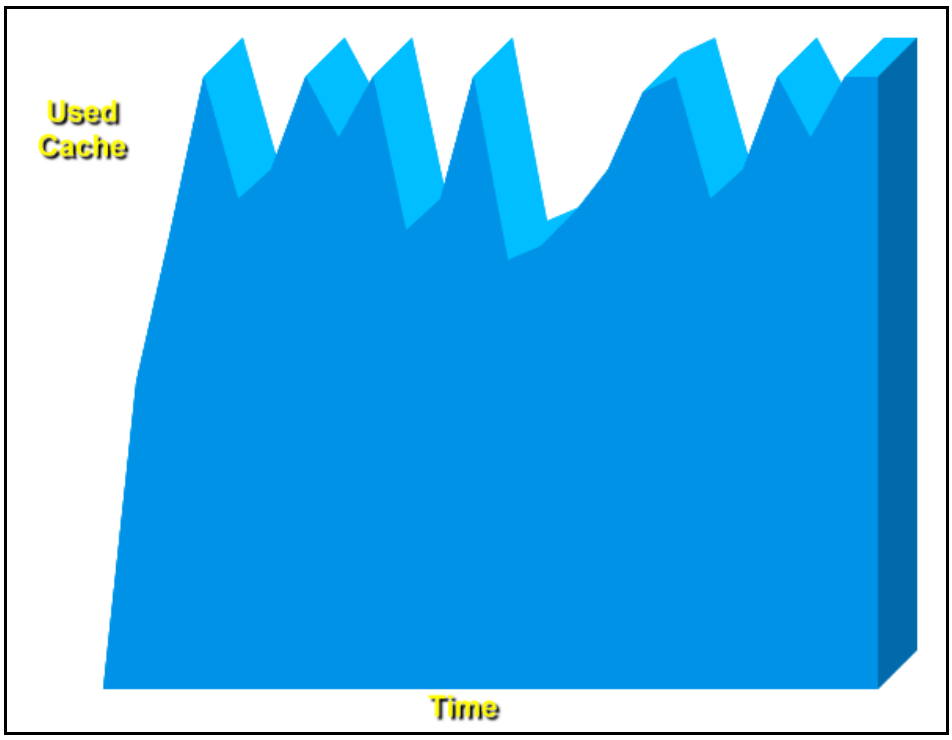
Observing Used Cache

With this information in mind, observing patterns in the used cache metric can help the 4D developer manage cache size and even troubleshoot problems.

It was mentioned previously that the used cache value can fluctuate up and down. In fact there are several "typical" fluctuation scenarios that will be presented in the next sections.

PLEASE NOTE: these scenarios are meant to help convey a better understanding of how 4D uses the cache. It is important to not take these examples as rules. For every 4D database there may be a different pattern in behavior in how the cache is used. The most important thing any 4D developer can do when analyzing cache usage is understand their own database.

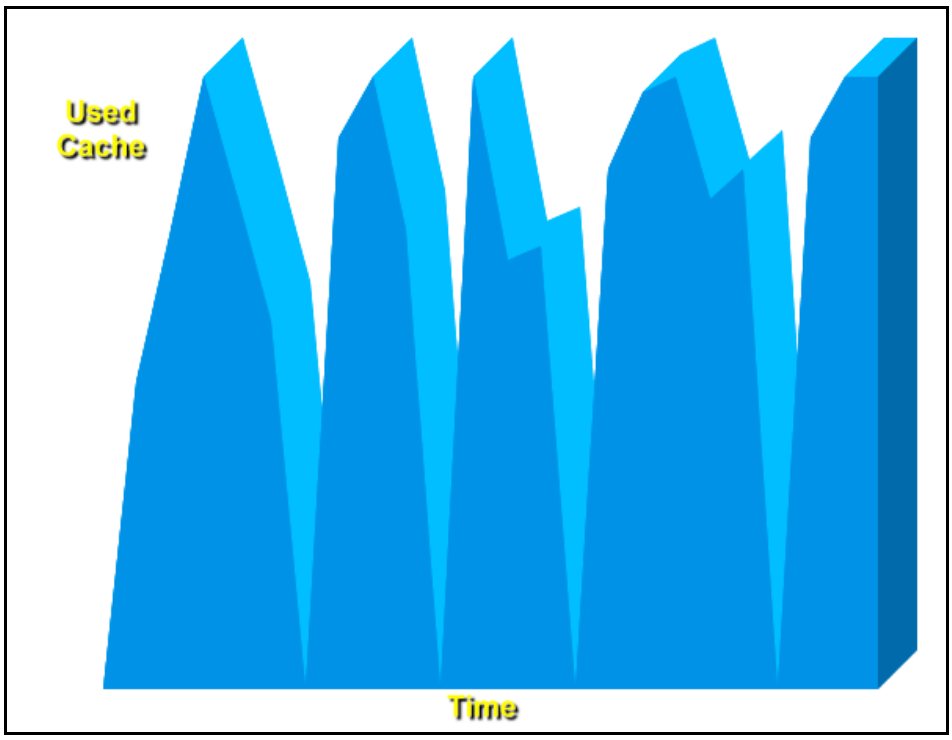
Normal Cache Usage



This chart represents the expected behavior of the cache. As the cache fills to maximum there should be no fluctuation (because there is still free space). Once the maximum is reached*, the used cache value will fluctuate but it should still be possible for it to always go back to the maximum value.

* *Because of memory fragmentation, it is nearly impossible to reach a true, full cache. Fragmentation is an unfortunate reality of memory access in computers since data structures are rarely perfectly block aligned. Thus as small objects occupy bigger blocks, there will be portions of "free" space that are not accessible.*

Cache Too Small



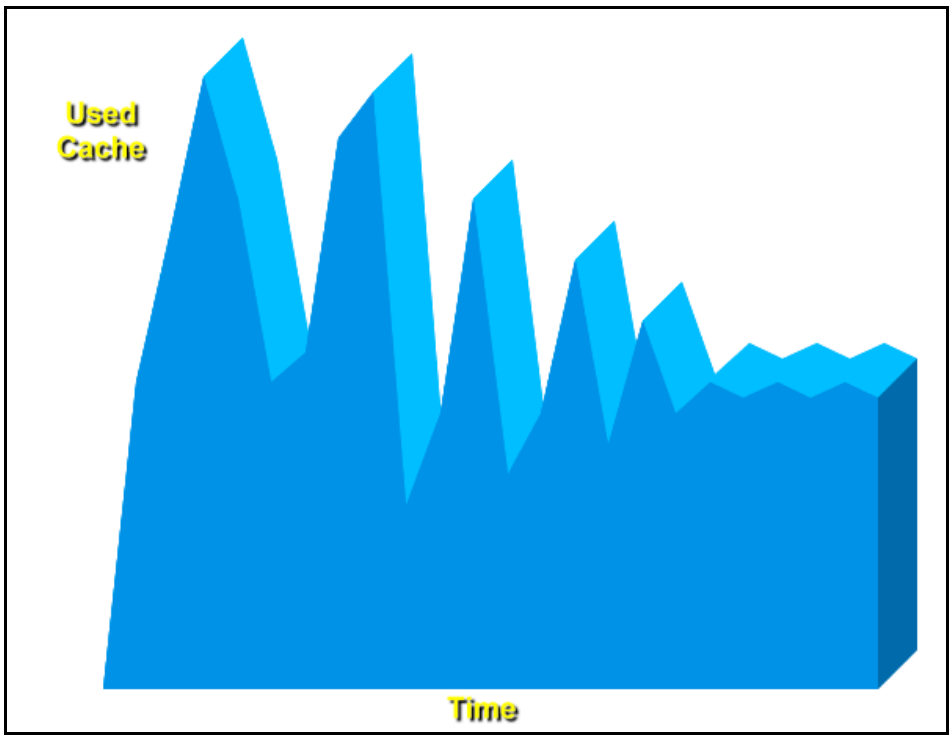
If the cache is fluctuating from maximum to 0 over and over this is a pretty good indication that the cache is too small. In this case 4D repeatedly needs to purge the entire contents of the cache in order to make enough room for data engine objects.

Other symptoms of the cache being too small are the presence of purged object files in the "temporary files" folder. Again this can indicate that 4D is purging excessively.

Of course it is possible, depending on the design of the database, for it to be necessary to purge the entire cache. Purging the whole cache is not, in and of itself, abnormal. In fact the ability to purge the entire cache is a sign of a healthy cache (see next section). The issue here is the pattern, not the event. If it occurs over and over it is something worth investigating.

As mentioned previously, aside from increasing the size of the cache, another feature that can be used to mitigate this problem is database parameter 61 (to limit the amount of memory for sequential sorts).

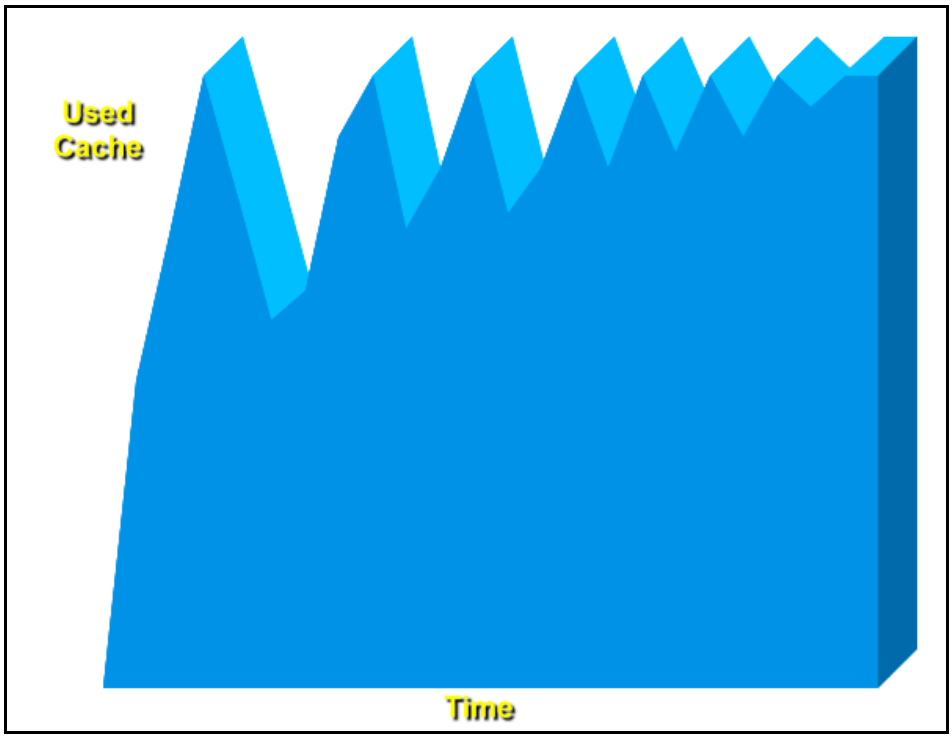
Cache Malfunction



The pattern that indicates a problem in the above chart is the "narrowing" of the used cache range. The idea is that, when 4D tries to purge, the number of objects purged gets smaller and smaller. Furthermore, when 4D tries to load new objects into the free space, the amount that can be loaded also gets smaller and smaller. This pattern can indicate the following:

- Normal behavior: In fact, this pattern may represent normal database behavior depending on what the database is doing; it may simply represent the database "settling in" to a common pattern of access. Perhaps the most atypical symptom here is that, not only does the used cache narrow, but the value it narrows to is quite low. This pattern is not typically normal.
- Memory leak: Objects in the cache cannot be purged because they are lost.
- Fragmentation: The "free space" is not truly accessible because small objects are occupying large blocks.
- Locking: Objects in the cache cannot be purged because they are locked. It would be important to determine why they are locked over time, of course.

This next chart is a variation of the same scenario with one change: if there is not a large problem with fragmentation, you might still see the used cache range "narrowing" but observe that the used cache is still high:



This is perhaps a better example of the database "settling in" to a pattern of usage. It could be that the database only needs to purge a little (and allocate a little) based on what the users are doing. This is where the developer's understanding of their own database is important. Still it is not typical to see this pattern so it is something to keep an eye out for.

Why is Cache Usage in 11 Different Than 2004?

There are a couple of significant changes that lead to cache usage in 4D v11 SQL being quite different that 4D 2004.

First, with regards to purging objects:

- In 2004 the purge was roughly 1/3 of the cache at a time (larger cache means larger purge!).
- In 11 it's smaller and fixed size, so the purge operations can be more efficient.

The second difference is with regards to large objects (LOBs, aka BLOBs and Pictures). To understand this difference it is important to understand what happens when a record is loaded into the cache:

- When a record is loaded into the cache, it is not in a language-friendly format (but the LOB data is included).

    - If the record is not needed for the language this is very optimized. I.e. if 4D is just doing a sort, it need not convert the record for your code to act on it.
- If the record is needed for the language, a copy is made.
- In both 2004 and 11, all scalar types (numbers, text, Boolean, etc.) are copied into a single object.

The difference is in how LOBs are handled:

- In 2004, BLOBs and Pictures are copied **outside** the cache.
- In 11, when the copy is made, the BLOBs and Pictures are stored **inside** the cache.

Thus there is potential for a lot more cache usage in 11 since the large objects are stored in the cache.

Further Tips

Here are some more interesting features that can be used in the context of tuning the cache in 4D v11 SQL.

FLUSH BUFFERS(*)

The optional * parameter has been added to the FLUSH BUFFERS command. This causes FLUSH BUFFERS to not only perform its normal flush operation but also to perform a purge of the cache.

In the ideal, normal situations this command is totally unnecessary as 4D manages the purging automatically. However if problems are suspected this command provides some extra insight:

- If, after the command executes, the used cache is 0 (or near 0) this indicates that the cache is healthy since all objects could be purged.
- If, after the command executes, the used cache is not near 0 (and no other operations are occurring that might be refilling the cache) this indicates that some objects could not be purged from the cache (for the possible reasons already mentioned).

This command should be executed on the server.

Note: *This feature will officially appear in 4D v11 SQL Release 5 (11.5) but is available in the current 11.4 HOTFIXs as well.*

GET CACHE STATISTICS

This command was added to provide detailed information about the memory usage of the cache. This information can, for example, be used to create the

charts used in the previous section (by executing it at regular intervals over time).

```
GET CACHE STATISTICS(  info_type ; arrNames ; arrValues ; arrCount)
```

info_type	Longint	input	addition of constants to specify the type of info desired
arrNames	Text array	output	info titles
arrValues	Real array	output	info values
arrCount	Real array	output	count of objects for the info considered (when available)

The possible values to combine for info_type are:

1	general memory info such as that displayed in the Runtime explorer (physical, virtual, free and used memory space, etc.)
2	summary of database cache occupancy statistics

Note that type 1 info can be obtained quickly whereas type 2 requires a complete scan of the cache contents and may therefore require a considerable amount of time to be calculated.

Example for getting all available info:

```
GET CACHE STATISTICS( 1+2 ; arrNames ; arrValues ; arrCount)
```

This will return, for example, the total size of the cache, the used size of the cache, the number of blocks in the cache, etc.

This command should be executed on the server if cache information is required. On the other hand note that this command returns information about System memory as well (memory used by 4D, for example) so it can also be useful for tracking memory usage over time and, therefore, can also be run on a client. Just ignore the cache data on the client as it is not accurate or relevant.

Note: *This feature will officially appear in 4D v11 SQL Release 5 (11.5) but is available in the current 11.4 HOTFIXs as well.*

Keep Cache in Physical Memory

This feature forces the cache to be allocated in the physical RAM of the machine (rather than allowing portions of the cache paged out to disk). Forcing the cache to be allocated in RAM makes for better performance in general; however it is important to understand that this comes at the cost of free memory on the machine. If the machine running 4D is low on physical memory then it makes it more likely for paging to occur. Even if the cache can't be paged, other parts of 4D might still be paged.

_USER_IND_COLUMNS

4D v11 SQL features System Tables that can be accessed using SQL. In particular there are system tables to track everything related to indexes (including index type, which table the index belongs to, which field the index belongs to, etc.).

The System Table _USER_IND_COLUMNS can be used to obtain table and field information for every index in the database. This information can be used, in turn, to load every index into the cache, with a simple loop over each indexed field, for example.

The goal of this technique would be to determine the minimum cache that might be needed to fit all indexes into memory (or perhaps certain important indexes).

This information could also be used to "warm" the cache with all (or certain) indexes.

The advantage to this technique is, of course, that it is a programmatic solution that can automatically adapt as new indexes are added to the database.

Conclusion

The 4D data file cache is a critical component of 4D's architecture and no where is this more apparent than in 4D v11 SQL. By providing a deeper understanding of what the cache is, how it has changed, and how it can be observed, this Technical Note aids the 4D developer in building better applications.