

Optimizing Record Selections - Part 2

Under the Hood of the 4D Data File

By Walt Nelson and Jean-Pierre Ribreau

4th Dimension Technical Note 97-12

Technical Notes for 97-05-May 1997

Introduction

As your data file becomes larger, and as you add more 4D Client or Web connections, optimization becomes even more important. The purpose of this two-part Technical Note is to give you some insight into how you can optimize the handling of your record selections in 4D. In Part 1, we discussed the organization of a 4D data file, and how it uses disk space and memory. In Part 2, we examine the concepts of the current record, the current selection, and named selections in 4D and discuss principles that will help you to optimize performance when you have large data files and/or large numbers of users.

Understanding Selections

A selection in 4D is an Address table that contains a list of the record numbers of the records in the selection. The size of a selection is 4 bytes for each record in the selection, plus an additional 4 bytes for the zero element of the selection. If there are 10,000 records in the current selection, then this selection uses 40,004 bytes or approximately 40k. A 100,000 record selection uses 400k, a 1,000,000 record selection use 4,000k (4MB).

This 4 bytes-per-record rule is valid for all types of selections except a selection that was created with the `ALL RECORDS` command. This command is optimized in 4D so that, when you call `ALL RECORDS ([aTable])`, 4D creates an Exception table that contains the addresses of the records that have been deleted. The optimization here is that 4D assumes that the number of deleted records will be less than the number of active records, so the size of the deleted record selection will be smaller than the size of the active record selection.

The Current Selection

For every process, 4D maintains a current selection for every 4D table per process (user environment, user processes and Web connection processes). For example, if you have four processes and ten tables, you will have 40 current selections. When you start 4D in Custom Menu mode, the current selection for every table is always an empty selection. Each time the user executes any command or action that creates a new selection, 4D does the following:

1. Builds a new Selection table (4 bytes per selected record). In Client/Server, this selection is maintained on the server. If there is not enough room in RAM to keep the entire selection in memory, 4D will store part of the selection on disk in a temporary file.
2. Moves the record pointer to the first physical record.
3. Loads the first record as the current record, including duplicate copies of certain fields.
4. Saves the address of the current record in the zero element of the Selection table array.

Exceptions to the Rules of the Current Selection

There are some cases in which 4D may not necessarily perform all four steps. They are:

¥ DISPLAY SELECTION, MODIFY SELECTION

When 4D displays a list of records in an output form or a subform, 4D draws the selection on the screen, and then it unloads the current record automatically. The purpose of this action is to avoid any accidental record-locking that might prevent other users from modifying a record. Therefore, in an output form or a subform, there is no current record unless the user clicks or double-clicks a record. This automatic unloading of the current record in a display is important in multi-process, multi-user applications.

¥ USE SET, USE NAMED SELECTION

When you use a set or a named selection, 4D remembers where the record pointer was at the time you created the selection or set, and it moves the record pointer to that same record. Instead of automatically loading the `first` record, 4D automatically loads the `remembered` record.

¥ POP RECORD

When you have pushed a record onto the stack in order to perform some operations and then you POP the record, the current selection is still the same as it was before you popped the record, but the current record pointer is at the popped record. In this case, the current record is actually outside the current selection!

¥ BEFORE SELECTION, END SELECTION

If you have procedurally executed an operation that moved the record pointer beyond the selection by using `PREVIOUS RECORD` or `NEXT RECORD`, the current record pointer is outside the selection, pointing to a record that doesn't exist; therefore, there is no current record.

Using the GOTO SELECTED RECORD Command

Just as `GOTO RECORD` is the fastest random access to a record in a table, `GOTO SELECTED RECORD` is the fastest random access to a record in a selection. If you `GOTO SELECTED RECORD ([aTable];35)`, 4D will move the record pointer to the 35th record in the current selection of [aTable]. This is much faster than a `QUERY` or any other method of finding a particular record.

Understanding Named Selections

A named selection is an ordered list of records that was, at some point, a current selection. When you create a named selection, 4D not only remembers the records in the selection, but also remembers the order of the records in the selection. A named selection can be at the process level or the inter-process level. So far, a named selection sounds very much like a set. The following are the similarities and differences between named selections and sets.

Similarities between named selections and sets:

¥ Both exist in memory.

¥ Both store references to records that existed with specific field values at a specific point in time. If the records are modified or deleted, the named selection or the set can become invalid.

¥ Both `remember` the current record as of the time they were created.

¥ Both automatically Load the remembered record and make it the current record.

Differences between named selections and sets:

¥ A named selection is an ordered list of records; a set is not an ordered list.

¥ A named selection requires 4 bytes for each record in the selection; a set requires one bit for each record in the file.

¥ A set can be saved to disk; a named selection cannot.

¥ Sets support the Intersection, Union, and Difference operations; named selections do not support those operations.

Commands Used to Manipulate Named Selections

4D has four commands that allow you to manipulate Named Selections:

¥ COPY NAMED SELECTION

¥ CUT NAMED SELECTION

¥ USE NAMED SELECTION

¥ CLEAR NAMED SELECTION

COPY NAMED SELECTION

When you copy a current selection into a named selection, 4D makes a duplicate of the Selection Address table. The original selection still exists; it is still the current selection. The new named selection also exists. Like a current selection, a named selection resides on the server. Also, like a current selection, the named selection takes up 4 bytes per record, plus 4 additional bytes.

However, unlike the current selection, a named selection must fit completely in memory. If there is not enough memory to hold the named selection, 4D will NOT create the selection. 4D will generate an error, but the user will not get an error message on-screen; as the developer, you must trap for this error using an `ON ERR CALL` procedure.

Since it must copy the entire Address table, the performance of `COPY NAMED SELECTION` will vary. If the selection is small, the operation will be fast. However, if the selection is large, the operation may be slow. 4D may have to move things around in memory in order to make room for the selection.

CUT NAMED SELECTION

When you cut a current selection and put it into a named selection, 4D does not make a duplicate of the selection. Instead, 4D creates a reference to the current selection. This reference uses only 4 bytes, so it is always possible to cut a named selection, and the performance of this command is very fast. The trade-off when using `CUT NAMED SELECTION` instead of `COPY NAMED SELECTION` is that when you cut the current selection, it is no longer the current selection for that table in that process. The current selection becomes empty. Remember the following when using `CUT NAMED SELECTION`:

¥ It is fast.

¥ It only uses 4 bytes, so you can always be sure that 4D has enough memory to complete the operation.

¥ Since it empties the current selection and the current record, do not use `CUT NAMED SELECTION` when a record has been modified but not saved.

USE NAMED SELECTION (ÒSelectionNameÓ)

When you have cut or copied a named selection, you can call the command `USE NAMED SELECTION` to make the named selection the current selection. When you use the named selection, 4D moves the record pointer to the record that was the current record as of the time the named selection was created, and then it loads that record.

Remember the following when using a named selection:

¥ It is a reference to records that existed with particular field values at a particular point in time. If the records are modified or deleted, the named selection can become invalid.

¥ If you are in the process of modifying a record, remember to save the record before executing `USE NAMED SELECTION`.

¥ If the original named selection had been created with `COPY NAMED SELECTION`, the original named selection still exists after the `USE NAMED SELECTION` operation. So you now have two identical copies of the selection: one is the named selection, and the other is the current selection.

¥ If the original named selection had been created with `CUT NAMED SELECTION`, the original named selection no longer exists after the `USE NAMED SELECTION` operation.

CLEAR NAMED SELECTION

A named selection uses memory; a large named selection uses a large amount of memory. Therefore, if you want to optimize your applications, you should always clear named selections as soon as they are no longer needed. To be efficient with this, you should always remember whether a named selection was created with `COPY NAMED SELECTION` or with `CUT NAMED SELECTION`. If it was created with a copy operation, then you need to clear it. However, if it was created with a cut operation, then when you use (`USE NAMED SELECTION`) the named selection, you are also clearing it. So in this case, `USE NAMED SELECTION` performs a use operation plus a clear operation. If you try to clear a named selection that was automatically cleared when it was used, you will get an internal 4D error. You should use `ON ERR CALL` to trap for those errors. This will help to "clean up" your application before delivering it to users.

For memory usage and optimization, it would be ideal to always create a named selection by using `CUT NAMED SELECTION`. This is the fastest method, and it uses only 4 additional bytes of memory. If you have a named selection of several hundred thousand records or several million records, this can make quite a difference. However, there are times when you need to keep the original selection and the copy of the selection. In those cases, use `COPY NAMED SELECTION`. In order to be sure about the origin of a named selection, may we suggest that you add a suffix `_Cut` or `_Copy` at the end of the name of the selection when you create it. For example:

```
COPY NAMED SELECTION([aTable];ÓMySelection_CopyÓ)
```

```
CUT NAMED SELECTION([aTable];ÓMySelection_CutÓ)
```

If you do this, it will help you to prevent mistakes such as:

```
CLEAR NAMED SELECTION ([aTable];ÓMySelection_CutÓ)
```

An Optimization Technique for Named Selections

A frequent use of named selections is to have two ordered selections, and then go back and forth between them. Typically, to accomplish this, you would do the following:

```
` Create the first named selection
QUERY ([aTable];SearchCriterion1) ` 1,000 records found, 4K selection
ORDER BY ([aTable];SortOrder1)
COPY NAMED SELECTION ([aTable];ÓSelection1_CopyÓ)
  ` Current Selection is 1,000 records

  ` Create the second named selection
QUERY ([aTable];SearchCriterion2) ` 50,000 records found, 200K selection
ORDER BY ([aTable];SortOrder2)
COPY NAMED SELECTION ([aTable];ÓSelection2_CopyÓ)
  ` Current Selection is 50,000 records

USE NAMED SELECTION (ÓSelection1_CopyÓ) ` 4k
  ` Perform some operations

USE NAMED SELECTION (ÓSelection2_Copy) ` 200k
  ` Perform some operations
```

When you do this, you have three selections in memory at all times. Their approximate sizes are:

```
¥ Selection1_Copy (4k)
¥ Selection2_Copy (200k)
¥ËCurrent Selection (sometimes 4k, sometimes 200k).
```

This one routine, in this one process, on this one workstation, can use up to 400k of server memory! If you have several users or several processes running this routine, along with their other normal operations, you can cause the server to run out of memory.

Here is a way to optimize this routine:

```
` Create the first named selection
QUERY ([aTable];SearchCriterion1) ` 1,000 records found, 4K selection
ORDER BY ([aTable];SortOrder1)
CUT NAMED SELECTION ([aTable];ÓSelection1_CutÓ)
  ` Current Selection is empty, Selection1_Cut exists.
  ` Total memory in use: 4k

  ` Create the second named selection
QUERY ([aTable];SearchCriterion2) ` 50,000 records found, 200K selection
ORDER BY ([aTable];SortOrder2)
CUT NAMED SELECTION ([aTable];ÓSelection2_CutÓ)
  ` Current Selection is empty, Selection2_Cut exists.
  ` Total memory in use: 204k

USE NAMED SELECTION (ÓSelection1_CutÓ)
  ` Current Selection is 1,000 records, Selection1_Cut no longer exists
  ` Total memory in use: 204k
```

` Perform some operations

CUT NAMED SELECTION ([aTable];ÓSelection1_CutÓ)

- ` Current Selection is empty, Selection1_Cut exists
- ` Total memory in use: 204k

USE NAMED SELECTION ([aTable];ÓSelection2_CutÓ)

- ` Current Selection is 50,000 records, Selection2_Cut no longer exists
- ` Total memory in use: 204k
- ` Perform some operations

CUT NAMED SELECTION ([aTable];ÓSelection2_CutÓ)

- ` Current Selection is empty, Selection2_Cut exists
- ` Total memory in use: 204k

USE NAMED SELECTION (ÓSelection1_Cut)

- ` Current Selection is 50,000 records, Selection1_Cut no longer exists
- ` Total memory in use: 204k
- ` Perform some operations

` ...

- ` At the end of the routine

REDUCE SELECTION ([aTable];0)

You can continue this cycle indefinitely, and your memory usage will be a constant 204k, instead of 404k as in the first method. This is a savings of 200k. If you multiply this savings by the number of processes and users running this routine, this can be quite a savings in server memory. Even if the savings are only 200k, it will be worth the effort if you have a large application, with a large number of users, running 24 hours a day. Also, keep in mind that the memory savings can make the difference between the operation succeeding with enough memory, or failing due to lack of memory!

REDUCE SELECTION Command

Note that at the end of our code example, the last line of code was `REDUCE SELECTION ([aTable];0)`. Just as you can clear a named selection with the command `CLEAR SELECTION`; you can clear a current selection with the command `REDUCE SELECTION`. This is especially important after a routine that manipulated large selections of records.

Optimizing the Use of the Current Record

As you learned from the sections on named selections, the consequences of not controlling memory usage are cumulative—small mistakes combine to become big mistakes. The same is true for the current record.

Calculating the Size of the Current Record

Suppose, for example, that you have a file that contains Alpha fields, fixed-length fields, and one picture field. You calculate the average size of a record and find that the data in the average record is 2,000 bytes, and the average picture size is 25,000 bytes. You know that internally, the data and the picture are stored separately. Therefore, you calculate the actual size of the record on disk as follows:

Size of the Alpha and fixed-length data:

$2,000 / 128 = 15.625$, must round up to 16.

$16 * 128 = 2,048$ bytes

Size of the picture:

$25,000 / 128 = 195.3125$, must round up to 196.

$196 * 128 = 25,088$ bytes

The total size on disk is 27,136 bytes. However, you know that when this record is loaded as the current record, 4D will make two copies of the portion of the record that contains the Alpha and fixed-length data. Therefore, the approximate size of this record in memory will be:

$2,048 \text{ bytes} + 2,048 \text{ bytes} + 25,088 \text{ bytes} = 29,184 \text{ bytes}$, or approximately 29k.

To load the current record in a client/server application, you will need 29k of contiguous memory on the server, plus 29k of contiguous memory on the client. This may not seem to be a large requirement. However, remember that you have many tables in your structure, you have several clients accessing the server, and you have several processes running on each client.

Let's look at some hypothetical numbers in a large client/server application:

¥ 75 tables

¥ 4 processes per user

¥ 50 users

For the server, that is a potential of $(75 * 4 * 50 = 15,000)$ current records! Of course, not all of these users will be loading a new current record at the same time. However, you can see that the potential for a memory problem exists.

For each client, the potential is $(75 * 4 = 300)$ current records. Unless something is done to avoid it, each client might have 300 current records loaded at any one time. Even at an average size of only 2k per record, that is a total of 600k of RAM, constantly in use to keep current records in memory! As a serious 4D developer, you MUST pay attention to the loading and unloading of the current record.

Commands that Change the Current Selection and the Current Record

In order to understand the side effects of your code, you need to know which commands affect the status of the current selection and the current record. First let's look at the commands that change the current selection and load the current record, if any.

ALL RECORDS	ADD RECORD	ARRAY TO SELECTION
CREATE RECORD	CUT NAMED SELECTION	DELETE RECORD
DELETE SELECTION	DUPLICATE RECORD	GOTO RECORD
IMPORT (all)	JOIN	ONE RECORD SELECT
PROJECT SELECTION	QUERY (all - v6)	RECEIVE RECORD
REDUCE SELECTION	RELATE MANY	RELATE ONE
SEARCH (all - v3)	USE NAMED SELECTION	

The following commands do not change the current selection, but they load the current record:

LOAD RECORD
POP RECORD (Current record is outside the current selection)

This command does not change the current selection or the current record, but it unloads the current record. Any time you are finished with a record, you should either move the record pointer to another record, or change the selection, or call this command to unload the current record:

UNLOAD RECORD

This command allows you to do a QUERY without changing the current selection or the current record:

SET QUERY DESTINATION

Summary

In Part 1 of this technical note, you have learned more about the 4D data file and what happens within the data file when a user saves a new record, modifies a record, or deletes a record.

In Part 2, you learned how to calculate the size of a selection, and how to minimize the memory requirements for your current selections and named selections. You learned that the cumulative effect of maintaining current records in memory can be very taxing on the client and on the server. In future Technical Notes, we will provide you with generic optimization routines that demonstrate how to minimize the memory used by selections and current records.

See Also

Optimizing Record Selections - Part 1.