4D v11 SQL データファイルキャッシュ

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

Technical Note 09-43

はじめに

データファイルキャッシュは 4D v11 SQL で劇的に改良されました。より大容量に、より効率的に、そしてキャッシュの動作を図るためのツールが追加されました。

このテクニカルノートではデータファイルキャッシュとは何か、どのように変更されたのか、そしてどのようなツールを使用できるのかについて説明します。

概要

4D v11 のデータファイルキャッシュ (あるいは単に“キャッシュ”) は劇的に改良されました。

64-bit OS のサポートおよびメモリ管理の向上により、より多くのキャッシュを利用できるようになりました (訳注：4D v11 は 32-bit アプリケーションです)。

キャッシュは 4D v11 SQL で、特にサイズの増大という点から見てより効率的になりました。

最後に、キャッシュの利用状況やトラブルシュートを手助けするための新しいツールが追加されました。

このテクニカルノートではキャッシュとは何か、どの点が変更されたか、どのようなツールが追加されたかについて説明します。

キャッシュとは

データキャッシュは 4D のとても重要な部分です。その重要度を理解するためには、キャッシュが何であるか、そして 4D がキャッシュをどのように利用するのかについて理解する必要があります。

この節で説明する内容は特に断らない限り 4D v11 SQL 特有のものではないことに留意してください。以前のバージョンの 4D について理解するためにも役立つはずです。

キャッシュは 4D が管理する大きなメモリの塊です。このような簡単な説明の中にも、重大な要素が含まれています。4D がキャッシュメモリを管理します。これは 4D が使用する他のメモリが OS に管理されているのとは異なる点です。つまりこのメモリに対して 4D が完全なコントロールを持っています。

これはハードウェアキャッシュとはだいぶ異なるものです。確かにハードウェアキャッシュはより遅いストレージからより速いストレージにコンテンツを格納するようデザインされています。4D のキャッシ

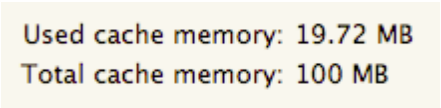
ュもディスク上のコンテンツを RAM に格納するようデザインされています。例えばレコードやインデックスはディスクからキャッシュにロードされます。しかしキャッシュはデータファイルオブジェクトだけのものではありません。すべての種類のデータエンジンオブジェクト (セレクション、トランザクション、データの並び替え等) に使用されます。以下はキャッシュに格納されるオブジェクトの一例です:

- テーブル、フィールド、リレーション、インデックスなどのストラクチャ定義
- 開かれたデータベースに関する一般的な情報 (ファイルパスやプロパティ等)
- データファイル内部構造のビットテーブル
- レコード、インデックス、Blob、その他追加のプロパティのアドレステーブル
- インデックスページ
- レコード
- BLOB (キャッシュに十分なスペースがない場合はメインメモリに置かれることもある)
- 追加のプロパティ
- シーケンス番号
- トランザクション
- セレクション
- セット
- 並び替えの一時的バッファ、先読み、ディスク書き込みのバッファ等

ご想像いただけるとおり、4D はこのメモリを常に管理し、可能な限り効率的に異なる種類のオブジェクトを保持できるよう入れ替えを行っています。

キャッシュの利用

4D Server v11 SQL では、キャッシュの利用を追跡する新しい統計があります。これは 4D Server の管理ウィンドウのアプリケーションサーバページに"使用キャッシュメモリ"として表示されます:



データベースが利用されるに伴い、使用キャッシュサイズは最大サイズから最小サイズの間を上下します。

もし 4D キャッシュをハードウェアキャッシュと同じように考えるとすると、この点は理解が難しい点となります; なぜサイズが減少することがあるのでしょう。一度オブジェクトがキャッシュにロードされる

と、それらはキャッシュの中に保持されるか、より新しいオブジェクトに置きかえられるかです。取り除かれるなんて普通はありません。

4D はあらゆる種類のオブジェクトに対しキャッシュを使用することを思い出してください。新しいオブジェクトのために空き空間を作るためには、古いオブジェクトを取り除かなければなりません。使用キャッシュ量が減少するのは、実際データエンジンオブジェクトのために空きスペースを作るため、オブジェクトを削除する必要があった場合なのです。

実際この振る舞いは普通のものであり (つまりキャッシュがそのように動作するようデザインされています)、トラブルシュートツールとして使用することができます。必要なことは 4D がどのようにキャッシュを管理するかを理解することです。

4D はどのようにキャッシュを管理するのか

データエンジンオブジェクトを格納するために空きスペースを確保する必要があるとき、4D はデータファイルオブジェクト (さらには他のデータエンジンオブジェクト) を消去しなければなりません。パージはフラッシュと異なる点に留意してください。フラッシュはオブジェクトをディスクに書き出しますが、キャッシュから取り除くことはしません。変更がディスクに保存されるだけです。他方パージされたオブジェクトはキャッシュ中に存在しなくなります。

以下は 4D がキャッシュ内に空きスペースを作るときのアルゴリズムです:

- 汚染されていないデータファイルオブジェクトをパージします。これはデータファイルから再度読み込むだけで容易に元の状態に戻すことのできるオブジェクトです。
- 十分な空きができない場合、FLUSH BUFFERS を行います。
- 再度汚染されていないオブジェクトをパージします。フラッシュ処理により多くの汚染されていないオブジェクトが作成されているはずなので、4D はより多くのオブジェクトをパージできます。
- まだ十分な空きができない場合、他のデータエンジンオブジェクトをディスクにコピーして、それらをパージします。
    - これらのオブジェクトはデータファイルと同階層の“temporary files” フォルダに置かれます。

この処理ののち、まだ 4D がデータエンジンオブジェクトを格納するための十分なメモリを確保できない場合、いくつか知っておくべきことがあります:

- まず 4D が十分なメモリを確保できなかったことを知らせるエラーメッセージが表示されます。
- キャッシュからすべてのオブジェクトをパージできたら (使用キャッシュが 0 になる)、キャッシュが少なすぎることを意味します。これは (100MB 以下の) 少量キャッシュのときのもっとも一般的な状況です。後ほど議論します。
- 理論上このアルゴリズムでキャッシュからすべてのオブジェクトを取り除けるはずです。そうならなかった場合、3 つの問題があり得ます:
    - メモリリーク: キャッシュ中に参照を失ったオブジェクトが存在するケースです。これは異常な状態であり、バグとして報告されるべきです。
    - フラグメント: 空きスペースとしてアクセスされるのを防げる、部分的な使用ブロック。これはコンピュータのメモリアクセスの悲しい現実です。
        - オブジェクトによってはロックされているためにパージできないことがあります (例. シーケンシャルソートのために使用中のオブジェクトなど)。これは"通常の"振る舞いですが、このことを理解して管理することが重要です(SET DATABASE PARAMETER のセレクタ 61 参照)。

キャッシュの変更点

この節では 4D v11 SQL でキャッシュに対して行われた変更点について説明します。

キャッシュサイズ

4D v11 SQL ではより多くのメモリにアクセスできるよう新しいメモリ管理が導入されました。これにより、より多くのキャッシュを割り当てることができるようになりました。

4D v11 SQL はいまだ 32-bit アプリケーションであり、32-bit アプリケーションのメモリの制限を継承していますが、この制限は 2 倍になりました: 2004 では 2GB が上限でしたが、v11 では 4GB が上限です (構成により異なります)。

32-bit OS 上であっても、4D v11 SQL は以前のバージョンより多くのメモリにアクセスできますが、さほど大きな違いはありません。何 10MB 程度増えるくらいです。4D を使用する場合 32-bit OS を使用すべきではありません。

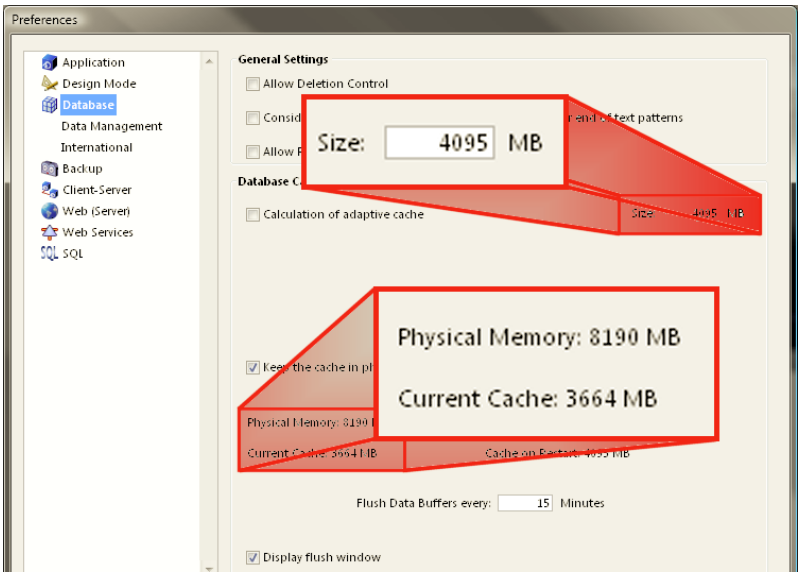
64-bit バージョンの Mac OS X や Windows Vista は 32-bit アプリケーションごとに 4GB をサポートします。これを最大限生かすためには 4D アプリケーションを動作させるマシンに最低 8GB の RAM が必要です。これは以下の理由によります:

- 64-bit OS では 4D は可能な最大メモリを利用できます。この最大に達するには、64-bit OS 上で 32-bit アプリケーションにメモリを割り当てる動作の仕様上、最低 8GB の RAM が必要です。
- Windows と Mac OS X 両方でメモリを割り当てる仕様上、大容量の RAM により最大量のキャッシュ割り当てが可能になります。両 OS とも DLL やライブラリを (メモリの先頭や最後ではなく) システムメモリのさまざまな領域にロードしようとします。4D がキャッシュ用にメモリをリクエストするとき、RAM が 8GB より少ないと、制限にかかりやすくなるのです。

もちろんマシンが 4D 専用でない場合、他のタスクをサポートするためにより多くの RAM を積むことが重要になります。他方、4D 専用のマシンに 8GB より多くのメモリを積むことにはあまり意味がありません。4D 自身は 3-4GB のメモリしか使用することができないからです。

8GB RAM を積んだ 64-bit Vista マシンで動作する 4D Server v11 SQL 11.4 の例を見てみましょう。

キャッシュ設定:



可能な最大値である 4095 は、可能な最大値を割り当て可能であることを確かめるためにのみ指定しました。実際にこの値を使用することはお勧めしません。もしシステムが実際に 4095MB をキャッシュに割り当てることができた場合、4D に残された非キャッシュ用のメモリは 1MB しかありません。

結局 4D v11 SQL で最大限キャッシュを得るためには:

- 64-bit OS を使用する
- 最低 8GB RAM をマシンにインストールする

デザインの変更

4D v11 SQL のキャッシュは劇的に改良され、より効率的になりました。

先ほど示した通り、より大きなキャッシュを使用できるようになりました。以前のバージョンの 4D ではキャッシュを大きくとりすぎないことが推奨されていました。通常キャッシュが大きければより多くのデータを格納でき、よりアクセスが速くなると思われるため、不思議に思われる方もいらっしゃったでしょう。以前のバージョンの問題は、4D がキャッシュを検索しなければならない状況があったり、大きなキャッシュを保守するときのオーバーヘッドがパフォーマンスに影響を与えたりする状況があったことです。4D v11 SQL で再設計されたキャッシュではこれらの状況が改善されています。

- 新しいキャッシュは 3-way associative です。この意味がわからなくても心配なさらないでください。要はキャッシュされたオブジェクトや空き領域を探すときにシーケンシャルサーチを行わなくなったということです。アドレスは直接計算されます。
    - これは特に大きなデータベースでのパフォーマンスを向上させます。
- キャッシュのスコア付与に 2 つの基準が使用されます:
    - アクセス数 (2004 モード)
    - 最新のアクセス (新規)
- オブジェクトを見つけるために 2 つのインデックスがキャッシュで使用されます:
    - サイズによるソート
    - アクセス数によるソート
- オブジェクトはさらにディスクアドレスでもソートされ、これによりフラッシュが高速になります
- レコードは “ブロック”でロードされます。例えば一度に 1000-2000 レコードがロードされます。そのため可能であれば定期的に圧縮を行うことが推奨されます。
    - またレコードは個々のレコードごとではなく、可能であればブロック単位でフラッシュされます。
- 可能な最大のキャッシュを取得するためには、64-bit OS1 上で 4D を実行します。これにより連続したメモリの塊を最大限大きくとることができる可能性が高くなります (キャッシュは連続したメモリ領域でなければならないことを思い出してください)。
    - 大きすぎるキャッシュを取得すると、4D が他のリソース (プロセス等) のために使用できる分が少なくなることに注意してください。

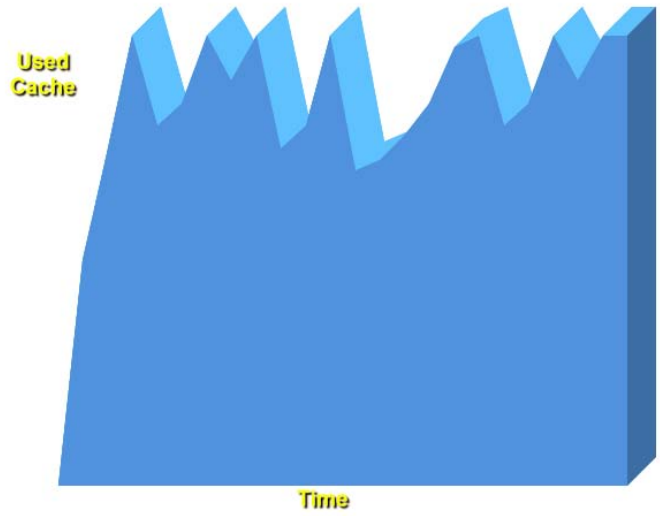
使用キャッシュを観察する

ここまでの情報をもとに、使用キャッシュを測量的に観察することで、4D 開発者はキャッシュサイズを管理したり問題を解決したりできます。

先に使用キャッシュサイズは増減することがあると書きました。実際次の節で示すような典型的な値上下のシナリオがあります。

PLEASE NOTE: これらのシナリオはどのように 4D がキャッシュを使用するのかについてより理解を深めていただくために例示するものです。これらを規則とは理解しないでください。すべての 4D データベースはキャッシュの使用において異なる動作をすることがあります。キャッシュ利用を解析する際に 4D 開発者が行うことのできる最も重要なことは、自身のデータベースを理解することです。

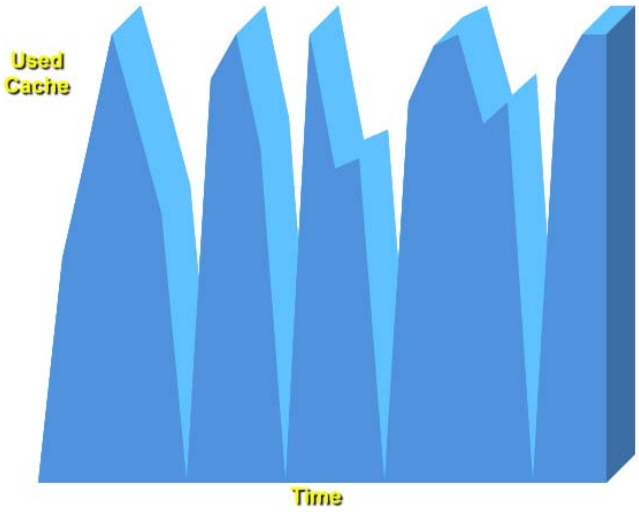
通常のキャッシュ利用



この図はキャッシュの期待される動作を示しています。キャッシュが最大に利用されるまでは、使用量の上下はありません (まだ空きがあるため)。一度最大値に達すると*、使用キャッシュ量は最大量から上下します。

* *メモリの断片化のため、完全にキャッシュがいっぱいになることはほとんど不可能です。断片化はコンピュータにおいて、データ構造が完全にブロックにそろえられるわけではないことから、メモリアクセス上不幸にも発生してしまうものです。つまり小さなオブジェクトが大きなブロックに格納されることにより、アクセスできない空きスペースが発生してしまうのです。*

キャッシュが小さすぎる場合



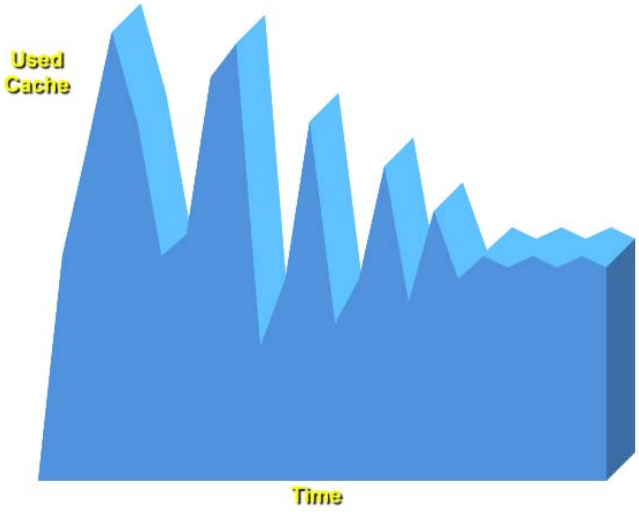
キャッシュが最大値と 0 の間を上下している場合、これはキャッシュが小さすぎることを意味しています。この場合、4D は繰り返しキャッシュ全体をパージし、データエンジンオブジェクトを格納するための空きを作成しなければなりません。

キャッシュが小さい場合の他の症状としては、パージされたオブジェクトが “temporary files”フォルダにファイルとしてあらわれることです。これは 4D が過剰にパージを行っていることを示しています。

もちろんアプリケーションのデザインによっては、キャッシュ全体をパージする必要があることもあります。キャッシュ全体のパージ自体は異常なことではありません。実際キャッシュ全体のパージが行えるということは、キャッシュが健康であるということです (次節参照)。ここでの問題はそれが発生することではなく、パターンです。キャッシュの全クリアが繰り返し発生する場合、調査を行うべきです。

先に示した通り、この症状を緩和させる方法としては、キャッシュサイズを増やすことのほかに、データベースパラメタのセレクタ 61 (シーケンシャルソートで使用するメモリ量を制限する) を使用することもできます。

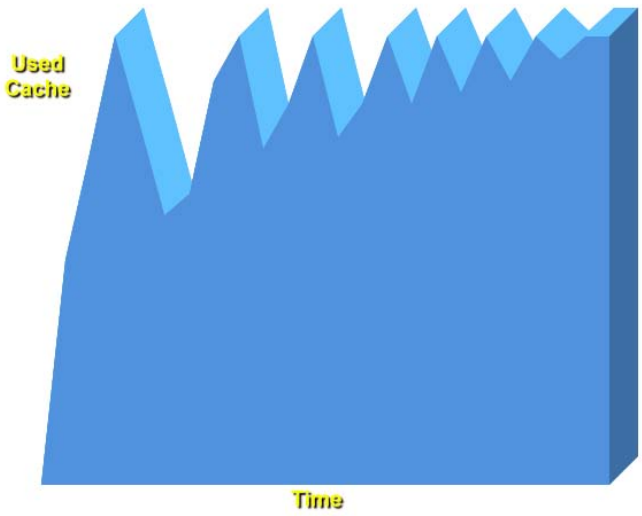
キャッシュの異常な振る舞い



この図の例で示される問題は、使用キャッシュサイズがだんだんと減少していることです。考えられることは、4D はパージを試行しますが、パージできたオブジェクトの数がだんだんと少なくなります。その上、4D が新しいオブジェクトを空きスペースにロードしようとすると、ロード可能な量が段々と小さくなっていきます。このパターンでは以下のことが考えられます:

- 通常の振る舞い: 実際データベースが行っていることによっては、この状況が普通に発生しえます。データベースが一般的なアクセスパターンに落ち着いている状態を示しています。おそらくここでの最も典型的な症状は、使用キャッシュが減るだけでなく、減少幅がとても小さくなります。このパターンを見ることはあまりないでしょう。
- メモリリーク: キャッシュ中のオブジェクトの参照が失われているため、パージができない状態です。
- 断片化: 小さなオブジェクトが大きなブロック内に格納されているため、空きスペースに実際アクセスできない状態です。
- ロック: キャッシュ内でオブジェクトがロックされているためパージできない状態です。もちろんオブジェクトが長時間ロックされている理由を調べる必要があります。

次の図は同じシナリオで発生するものですが、一つが異なっています。断片化に起因する大きな問題がない場合、使用されるキャッシュの範囲が徐々に少なくなる現象はありますが、使用キャッシュ量は依然として高いままです:



恐らくこれはデータベースの利用が落ち着いた状態にあることのよりよい例です。ユーザが行うことに基づき、データベースは少量のパージと少量の割り当てを行うだけで十分な状態と考えられます。ここが、開発者が自身のデータベースについての理解が重要であるという部分です。このパターンを見ることは一般的ではありませんので、他の要素も見る必要があります。

キャッシュの利用法が 11 と 2004 で異なる点

4D 2004 と 4D v11 SQL ではキャッシュの利用法が大きく変更されています。

最初はオブジェクトのパージに関する点です:

- 2004 ではパージは一度にキャッシュの約 1/3 に対して行われていました (キャッシュが大量に使用されている場合、パージも比例して大きくなるということです)。
- v11 では、サイズはより小さな固定サイズとなり、パージ処理はより効率的になりました。

2 つ目の違いは大きなオブジェクト (BLOB やピクチャなどの LOB) に関するものです。この違いについて理解するためには、レコードがキャッシュにロードされるときに何が起こるかを知らなければなりません:

- レコードは LOB データが含まれた、ランゲージフレンドリでないフォーマットでキャッシュにロードされます。
    - レコードがランゲージから必要とされない場合、これはとても最適化されています。例えばもし 4D が単に並び替えを行う場合、それを行うためにレコードを変換する必要はありません。

- ランゲージがレコードを必要とするとき、コピーが作成されます。
- 2004 と v11 両方ともすべてのスカラタイプ (数値、文字列、ブール等) は一つのオブジェクトとしてコピーされます。

LOB が処理されるときの違いは以下の通りです:

- 2004 で、BLOB とピクチャはキャッシュの外側にコピーされます。
- v11 でコピーが行われるとき、BLOB とピクチャはキャッシュ内に格納されます。

つまり、v11 ではキャッシュに大きなオブジェクトが格納されることから、より多くのキャッシュが使用される可能性があります。

Tips

4D v11 SQL にはキャッシュのチューニングに使用できる、興味深い機能がほかにもあります。

FLUSH BUFFERS(*)

オプションの * 引数が **FLUSH BUFFERS** コマンドに追加されました。この引数を使用すると FLUSH BUFFERS 通常のフラッシュ処理だけでなく、キャッシュのパージも行います。

通常この引数付きでコマンドを呼び出すことは、まったくもって必要ありません。4D が自動でパージを管理します。この用法はデバッグのためにのみ利用できます:

- コマンド実行後使用キャッシュが 0、あるいはほとんど 0 であれば、すべてのオブジェクトがパージでいており、キャッシュが正常であることを示しています。
- コマンド実行後、キャッシュの使用量が 0 に近くなく、キャッシュに再びオブジェクトを格納するような処理がさなれていない場合、(先に示した理由により) パージできなかったオブジェクトがキャッシュ内に存在することを意味します。

このコマンドはサーバ上で実行しなければなりません。

Note: *この機能は 4D v11 SQL Release 5 (11.5) の追加修正情報で説明されていません。このコマンドは一般に使用されることを想定したものではなく、あくまでデバッグ目的であることに留意してください。最終的なアプリケーションからはアスタリスク付きの FLUSH BUFFERS は取り除かれるべきです。*

GET CACHE STATISTICS

このコマンドはキャッシュの利用に関する詳細な情報を提供する目的で追加されました。コマンドを定期的に実行することで、この情報は例えば先の節で使用した図表を作成するために使用できます。

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

info_type	倍長整数	->	取得する情報を指定するセレクタ
arrNames	テキスト配列	<-	情報タイトル
arrValues	実数配列	<-	情報の値
arrCount	実数配列	<-	情報に関連するオブジェクトの数 (利用可能な場合)

info_type には以下の値を指定できます (加算可):

1	ランタイムエクスプローラに表示される一般的なメモリ情報 (物理、仮想、空き、そして使用メモリ等)
2	データベースキャッシュの利用統計に関するサマリ

タイプ 1 の情報は素早く取得できますが、タイプ 2 についてはキャッシュの内容を完全にスキャンする必要があり、計算に時間がかかる点に留意してください。

すべての情報を取得する例:

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

このコードからはキャッシュの総サイズ、使用キャッシュサイズ、キャッシュ中のブロック数などが返されます。

キャッシュ情報が必要な場合、このコマンドはサーバ上で実行しなければなりません。他方、このコマンドはシステムメモリに関する情報も返します (例えば 4D が使用するメモリ)。時間経過に伴うサーバー上のメモリの利用状況を調査したり、クライアント側のメモリを調べたりすることができます。クライアント側ではキャッシュに関する情報は無視してください。

キャッシュを物理メモリに置く

この機能はキャッシュを強制的にマシンの物理 RAM に割り当てます (キャッシュの一部をディスクにページアウトさせないようにします)。キャッシュを強制的に RAM に割り当てるようにすると、一般的に

よりよいパフォーマンスが得られます。しかしこれにはマシンの空きメモリのコストが必要である点を理解することが重要です。4D が実行されているマシンの物理メモリが少ない場合、より多くのページングが発生します。キャッシュのページングが禁止されていても、4D の他の部分はページされます。

_USER_IND_COLUMNS

4D v11 SQL には SQL を通じてアクセスできるシステムテーブルがあります。特にインデックスに関連するすべてを追跡するシステムテーブルがあります (インデックスタイプ、インデックスが属するテーブルインデックスが属するフィールド等)。

システムテーブル _USER_IND_COLUMNS を使用して、データベースのすべてのインデックスに関するテーブルとフィールドの情報を取得できます。この情報は例えば、インデックス付きフィールドをループすることで、すべてのインデックスをロードするために使用できます。

このテクニックの目的は、すべてのインデックス (あるいは重要なインデックス) をメモリに読み込むために必要な最小のキャッシュサイズを得ることにあります。

この情報はまたすべて (あるいは一部) のインデックスをあらかじめキャッシュに入れておくためにも使用できます。

このテクニックの利点は、データベースに追加された新しいインデックスとして自動で適用できる、プログラム的なソリューションである点です。

まとめ

4D データファイルキャッシュは 4D アーキテクチャのとても重要なコンポーネントであり、とくに 4D V11 SQL ではそのとおりです。キャッシュやその変更点、どのように観察するかに関する理解を深めることで、このテクニカルノートでは開発者がよりよいアプリケーションの構築できるよう手助けします。