フィールド形式の変更について

By Jean-Yves Fock-Hoon, QA Manager
Technical note 05-10

（原題: Data formating）

概要

フィールドタイプを途中で変更する際の注意点を考慮する。

4Dの優れた点

4Dは、いつでもフィールドが追加できるばかりか、フィールドのタイプまで変更できる。他のDBの場合、まずエクスポートしてから、新規ストラクチャにインポートする必要があるだろう。たとえば、テキスト/文字列フィールドのタイプをREALに変更してデータを保存すると、データが数値に変換される。フィールドを元のタイプに戻すと、今度は文字としての数字に変換され、データは失われてしまう。

4Dの場合、フィールドを更新してから保存した時にはじめてデータの変換が実行される。変更なしで保存した場合は、変換されない。

したがって、フィールドタイプを変更し、そのレコードの他のフィールドを修正することによってレコードを更新しても、変更したフィールドはタイプを戻せば元のデータが得られる。

デモDB

証拠に、デモDBを起動してもらいたい。デザインモードでみると、[Table3]には文字列フィールドが2つある。ユーザモードでみると、それぞれ"a text value"というデータが収めれている。デザインモードでField2を実数タイプに変更する。ユーザモードでみると、"a text value"は0になっているのが確認できる。

しかしながら実際にデータが変換されたのではない。0というのは表示上の都合であるに過ぎない。

メソッドModtable3を実行すると、Field1のデータが変更され、レコードは更新される。Field2は0のままであり、レコードは更新されたことが分かる。

最後にField2のタイプを実数から再び文字列に戻すと、ユーザモードで元のデータ"a text value"が復元されている。

この仕組みの利点は、フィールドタイプを変更したときに元のデータを復元できること、そしてフィールド全体のデータ変換に消費される時間を節約できることの2点である。

注意点

気をつけるべきなのは、4Dはデータを参照し、タイプが異なることを認識しつつも、変更が加えられなかったため、保存をしなかったという点である。この点をはっきりさせるため、メソッドM_Selection2Arrayを実行してみよう。SELECTION TO ARRAYが両フィールドに対して実行され、実行に要した時間が表示される。

Field1のほうが時間が長くかかっているのは、いろいろなタイプのデータが混在しているので、4Dに余分な処理が発生しているためである。具体的には、最適化された高速メソッドから、変換を扱う以前のメソッドに切り替えて処理をしている。

メソッドM_ResaveAll1あるいはM_ResaveAll2を実行して、レコードを更新する。再びSELECTION TO ARRAYを実行すると、時間の差はほとんどない。

以前のバージョンからコンバートにコンバートを重ねてきたDBの場合、昔作られたレコードが新しいバージョンのフィールド定義に厳密にあてはまらない可能性がある。入力がされなかったか、フィールドが途中から追加されたために、一部のレコードには特定のフィールドがない可能性もある。こうした状況は、4Dの問題、コードの問題ゆえに不具合を生じさせる危険があり、そうでなくても前述のデモでわかるようにパフォーマンスの低下を招く。したがって、レコードはすべて正しいタイプで保存するのが大原則なのである。

フィールドタイプを揃える方法

簡単な方法はメソッドM_ResaveAll1の手法を踏襲することである。すべてのテーブル情報を取得し、メソッドm_UpdateFields methodで更新をする。

```
NbTables:=Count tables
For (iTable;1;NbTables)
pTable:=Table(iTable)
NbFields:=Count  fields(iTable)
ALL RECORDS(pTable->)
While  (Not(End selection(pTable->)))
m_UpdateFields
SAVE  RECORD(pTable->)
NEXT  RECORD(pTable->)
End while
End for  ` iTable
```

メソッドm_UpdateFieldsはフィールド情報を取得した後、Case文によって、各タイプごとに更新を強制するような処理を実行している。

テキスト/文字列フィールドに対しては、ヌルストリングを追加している。文字列フィールドの長さを超過するものについては余剰分をカットしている。

数値フィールドの場合、0を加算するか、StringとNumを重ねて処理する。（例: MyField:=Num(String(MyField))）

日付、時間は、4Dの内部では数値であるので、0を追加する。あるいはAdd to date、Date、Num、String、Time stringを使用してもいいだろう。

BLOBはサイズによって扱いが異なるが、この例では別のBLOB変数に値をリロードすることによって処理している。

ピクチャには0やヌルストリングを加算することはできないので、BLOB同様変数を使用するか、この例のように1を乗算するか、空のピクチャを加算する（後者は保証されない動作）。

ブールは簡単である。OR FALSEをかければ値は元のままである。

サブテーブルの場合はどうだろうか。
公開されていない4Dの機能を使用することができる。 GET FIELD TITLESコマンドとFieldコマンドを使用して、サブフィールド名と、サブフィールドに対するポインタを取得することができる。

```
GET FIELD TITLES(pField->;SubfieldTitles;SubfieldNums)
pSubField:=Field(iTable;iField;SubfieldNums{$i_SubField})
```

GET FIELD TITLESコマンドは、引数として渡されたサブテーブルのサブフィールドのフィールド名とフィールド番号を返す。それを利用してFieldコマンドを使用すればサブフィールドに対するポインタが取得できる。この機能はマニュアルには記載されていないので動作保証外であるが、4D2004では機能している。将来のバージョンでどうなるかは不明であるので注意が必要だが。

あとは前述の処理を繰り返すわけだが、GET FIELD PROPERTIESはポインタを引数として使用できないので、文字列サブフィールドの長さを取得することができない。さらにサブテーブル内のサブテーブルの処理もできないが、そのような構造はそもそも現在の4Dでサポートしていない。

まとめ

SELECTION TO ARRAYコマンドだけでなく、最適化されたコマンドは、フィールド形式とデータが一致しているときに最大のパフォーマンスを発揮する。一致しない場合は、以前のルーチンに切り替わるので高速化の恩恵にあずかることができない。加えて、不一致データの扱いには潜在的な危険が存在する。4D Toolsはデータの復元を行なうことができるが、再定義されたフィールド形式に合わせて、データを再保存すルトは限らないので、データを形式にあわせて保存しなおうのが賢明であるといえよう。