4D v11 SQL の SQL データタイプ

By Chris Visaya, Technical Services Team Member, 4D Inc.

Technical Note 09-22

概要

4D に SQL が導入され、開発者は SQL のデータ定義言語 (DDL) を使用してネイティブに、データベースストラクチャを変更することができるようになりました。 これを行う際に、開発者は SQL 互換の型名を使用してデータ型を指定する必要があります。 これらのタイプは 4D データ型に対応しており、マッピングは問題なく動作しますが、留意すべき点がいくつかあります。

はじめに

SQL を使用してテーブルそしてフィールドを作成するには、SQL 標準のデータタイプを使用する必要があります。ほとんどの場合、これらのタイプと 4D のタイプとの関連は妥当なものです。しかしいくつかの点で開発者が見落としてしまう点があります。このテクニカルノートでは SQL を使用してテーブルを作成する際に留意すべき点について述べます。

以下の表は SQL タイプと 4D のタイプとの対応を示しています:

SQL	4D	説明
Varchar	Text	英数字のテキスト
Real	Real	浮動小数点数 -/+ 3,4E38
Numeric	Int, 64 bits	整数 -/+ 2E64
Float	Real	浮動小数点数 (ほぼ無限)
Smallint	Integer	数値 -/+ 32,000
Int	Longint	数値 -/+ 20 億
Bit	Boolean	TRUE または FALSE のみを値にとる
Boolean	Boolean	TRUE または FALSE のみを値にとる
Blob	BLOB	<= 2GB; すべてのバイナリオブジェクト
Clob	Text	<= 2GB 文字; デフォルトでレコードには保存されない
Text	Text	<= 2GB 文字; デフォルトでレコードには保存されない
Timestamp	Date/Time	日付と時間は個別に処理される
Duration	Time	時間 Day:Hours:Minutes:Seconds:milliseconds
Interval	Time	時間 Day:Hours:Minutes:Seconds:milliseconds
Picture	Picture	<= 2GB

一度 4D のテーブルにフィールドが作成されると、ストラクチャエディタを使用してフィールドが作成されたのと同様に操作、使用できます。

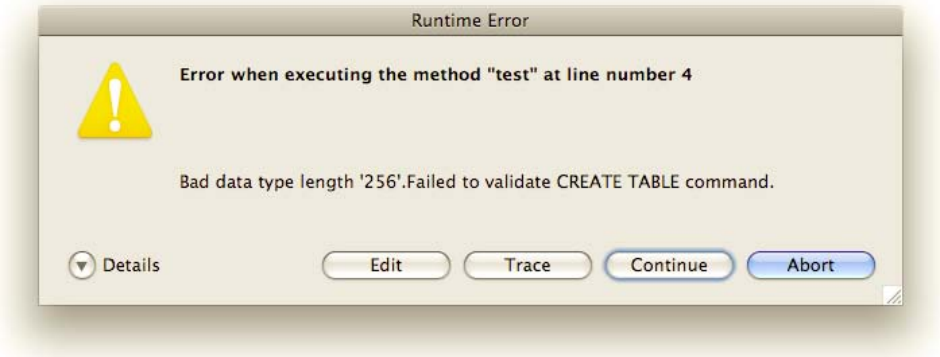
文字タイプ

文字フィールド

255 文字まで可変長の文字タイプを定義するには、VARCHAR を使用し、あとの括弧内に文字長を指定します。以下の例では 30 文字までを格納する文字フィールド"name"を作成しています:

```
Begin SQL
   CREATE TABLE Consumers
   (name VARCHAR (30))
End SQL
```

この方法で 255 文字以上の文字フィールドを作成しようとすると、以下のエラーが発生します。



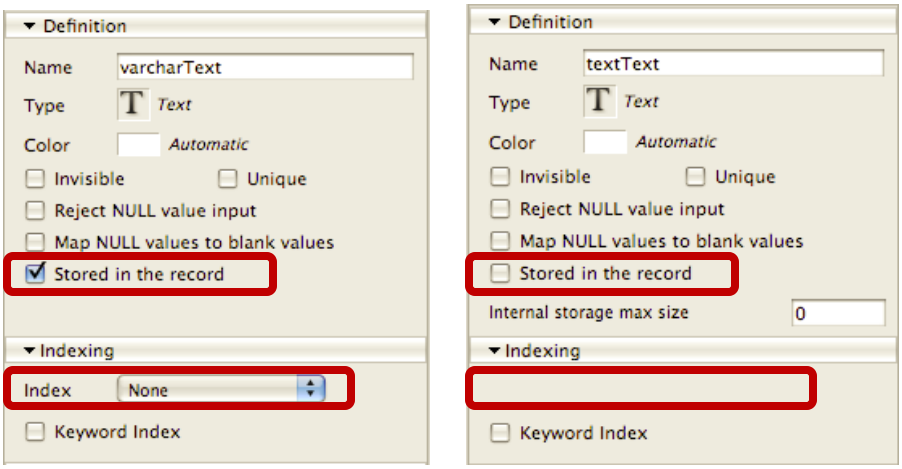
括弧を省略してサイズを指定しない場合、VARCHAR はテキストフィールドを作成します (後述)。

テキストフィールド

2GB までの大きな文字データを処理するには、4D のテキストタイプを使用します。SQL では複数の方法で作成できます:

```
Begin SQL
    CREATE TABLE Consumers
    (firstname VARCHAR,
    lastname TEXT,
    title CLOB)
End SQL
```

この例で作成されたフィールドはすべてテキストタイプとなります。違いはなんでしょうか。前のセクションで示した通り、サイズが指定されていない場合 VARCHAR は 4D の制限である 2GB までの可変長文字フィールドを作成します。Character Large Objects または CLOB は特に他のデータベース管理システムで使用され、大量のテキストをレコードとは別な場所に格納するために使用されます。そのためインデックスを付けることはできません。これはフィールドプロパティに反映されます。



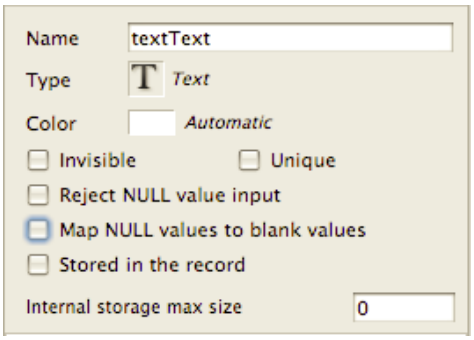
(a) (b)

(a)ではフィールドに"レコードに格納"オプションがチェックされ、標準のインデックスを設定することが可能です。このフィールドは VARCHAR を使用して作成されています。(b)では"レコードに格納"オプションにチェックがされておらず、標準のインデックスを設定することできません。このフィールドは CLOB または SQL の Text を使用して作成されています。

SQL の Text や CLOB タイプを使用してテキストフィールドを作成した後、ストラクチャエディタを開いて、"レコードに格納"オプションとフィールドのインデックスをチェックしてみてください。ただしキーワードインデックスは 3 タイプに適用できます。

フィールドプロパティについていえば、フィールドを作成するときに SQL を使用する場合とストラクチャエディタを使用する場合との微妙な違いは、"NULL 値を空値にマップ"オプションが

デフォルトでチェックされていないことです。このオプションはストラクチャエディタでフィールドを作成する場合デフォルトでチェックされています。



NULL 値は SQL 中でネイティブに扱われます。そこでフィールドの作成に SQL を使用した場合、NULL 値をデフォルト値に自動でマップすることは、作成時には強制しないようになっています。

4D における NULL 値とその影響に関する検討は、テクニカルノート#51416 "NULLs in 4D v11 SQL"を参照してください。

フィールドが作成されると、レコードに格納や NULL 値のデフォルト値へのマップ、インデックスにかかわらず、通常の 4D のテキストフィールドのように振る舞います。言い換えれば、SQL で作成されたテキストフィールドはどれも、デフォルトのフィールドプロパティに違いはありますが、4D のテキストフィールドとして扱われます。

時間

Timestamp

SQL の Timestamp データタイプには特段の注意が必要です。ストラクチャエディタで日付のように見えるにもかかわらず、Timestamp は時間や日付ではありません。このタイプには日付と時間両方の情報が含まれていて、自動で対応する 4D タイプにマップされます。以下の例で見てみましょう:

```
Begin SQL
   CREATE TABLE Invoices
   (Order_Date TIMESTAMP)
End SQL
```

ストラクチャエディタ中でフィールドは日付のように見えます:

以下のコードは INSERT コマンドを使用してこのテーブルにレコードを作成しています。レコードが作成されると、次の SQL 文でテーブルをクエリしています。Order_Date が 3 回 SELECT 文中に現われ、結果が 3 つの変数に格納される点に留意してください:

```
C_TIME (vTime)
C_DATE (vDate)
C_LONGINT (vMillis)

Begin SQL
   INSERT INTO Invoices (Order_Date)
   VALUES (CURRENT DATE());

   SELECT Order_Date, Order_Date, MILLISECOND(Order_Date)
   FROM Invoices
   INTO :vDate, :vTime, :vMillis
End SQL
```

クエリの結果...

- vDate には現在の日付が year/month/day フォーマットで返されます。
- vTime には現在の時刻が hour:minute:second フォーマットで返されます。
- vMillis にはタイムスタンプのミリ秒精度です。

4D バージョンの CURRENT DATE の使用は、単に日付を埋めます。時間は返しません。他方、先のコードをストラクチャエディタで作成した日付フィールドに使用すると、ここで説明したように動作します。言い換えれば、フィールドがストラクチャエディタで作成されたか SQL で作成されたかにかかわらず、一つのフィールドに日付とミリ秒精度までの時刻を格納し、SQL を使用して適切な 4D 変数に値を取得できます。

Duration/Interval

時間フィールドを作成するには、Duration や Interval を使用します:

```
Begin SQL
   CREATE TABLE Invoices
   (tInterval INTERVAL,
   tDuration DURATION)
End SQL
```

対応するテーブルと 2 つの時間型フィールドが作成されます。

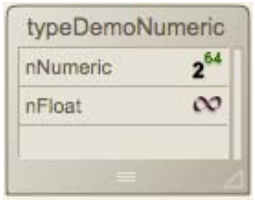
この時点で、4D の CURRENT TIME コマンドを使用して 4D 時間をフィールドに格納できます。

数値

Numeric と Float

4D v11 SQL は新しい数値タイプ、64-bit Integer と Float をサポートしています。このタイプのフィールドは SQL を使用しているときにアクセスできます。4D ランゲージを使用しているとき、これらのフィールドの値は実数に変換され、データを失う可能性があります。

SQL を使用してストラクチャエディタにこれらのタイプのフィールドを作成できます:



このテーブルを作成するコードは以下です:

```
Begin SQL
   CREATE TABLE typeDemoNumeric
   (nNumeric NUMERIC,
   nFloat FLOAT)
End SQL
```

SQL の他の数値タイプ (Smallint, Int, そして Real) は 4D の整数、倍長整数、そして実数にマップされ、ランゲージ中でもシームレスに振る舞います。

Boolean

ブールフィールドを作成する方法は 2 つあり、BIT と BOOLEAN を使用します。

```
Begin SQL
   CREATE TABLE bCheck
   (bBit BIT,
   bBool BOOLEAN)
End SQL
```

両フィールドはブールフィールドとして作成されます。True や False 条件で、4D ランゲージを使用してこれらのフィールドをクエリできます。SQL によるクエリは若干異なり、若干自由度があります。まず SQL は CAST を使用することで、**True/False** と **1/0** 値両方を認識します。例えば bBit フィールドに False を含むレコードをクエリするとします。以下の文は同じ結果になります:

```
ARRAY BOOLEAN($aBool;0)

Begin SQL
   SELECT bBit FROM bCHeck WHERE bBoolean = CAST ('FALSE' AS BOOLEAN)
   INTO :$aBool
End SQL

Begin SQL
   SELECT bBit FROM bCHeck WHERE bBoolean = CAST (0 AS BOOLEAN)
   INTO :$aBool
End SQL
```

まとめれば、BIT や BOOLEAN で作成されたフィールドはお互い同様に動作します。なので 4D ランゲージで使う場合も SQL で使用する場合も、混乱することはないでしょう。

ラージオブジェクト

Picture と BLOB

ピクチャフィールドと BLOB フィールドを SQL で作成できます:

```
Begin SQL
   CREATE TABLE typeDemoLOB
   (bBlob BLOB,
   bPict PICTURE)
End SQL
```

フィールドはそのままマップされ、4D 内で期待通りに動作します。SQL や 4D を使用したこれらのフィールドへのデータ作成とアクセスはシームレスです。

まとめ

プログラムでフィールドを作成するために SQL を使用するとき、命名規則が SQL 標準に互換であるように注意してください。そしてそれぞれのタイプが 4D でどのようにマップされるかを知る必要があります。フィールド作成後、若干の調整が必要ですが、デベロッパは 4D と同様これらのフィールドにデータを入力できます。SQL を使用して作成されたフィールドとそのタイプは 4D にネイティブに理解され、ストラクチャエディタで作成したのと同様に操作できます。