Article #15868: dBASE Expression Indexes: A Primer

Technical Information Database

```
TI868D.txt  dBASE Expression Indexes: A Primer
Category   :Database Programming
Platform   :All
Product    :Delphi  All
```

Description:
Indexes for dBASE tables may be based on the values from a single field, unmodified, or on an expression. Index expressions, unique to dBASE indexes, may be composed of multiple fields, modifications of field values, or combinations of these. The expression for a dBASE expression index is created by using dBASE functions and syntax to concatenate multiple fields or to perform the modifications of field values for fields included in the index expressions.

Two section are included at the end of this technical article which describe the mechanics of creating dBASE expression indexes, one applicable to doing this in the Database Desktop utility and the other for including this capability in Delphi applications.

Expression Indexes Based On Multiple Fields
===

dBASE functions are available for use in Delphi or the Database Desktop for the express use in index expressions, and then only in conjunction with dBASE indexes. That is, you cannot use dBASE functions or syntax to build an index expression for a Paradox or Local InterBase Server (LIBS) table. Nor can dBASE functions be used in Delphi programming. They are only available for dBASE expression indexes. The dBASE functions and syntax that can be used for expression indexes are provided by the Borland Database Engine (BDE) Dynamic Linked Library (DLL) file IDDBAS01.DLL.

When creating a dBASE index that is to be based on the values from two or more fields in the table for which the index is being created, the two or more fields are concatenated (connected together) in a manner similar to how String type values are concatenated in Delphi syntax: the "+" operator. For example, the expression needed to create an index that orders first on a LastName field and then on a FirstName field would be:

```
  LastName + FirstName
```

Unlike in dBASE itself, such indexes that are based on multiple fields are limited to using just those fields in the one table. dBASE allows the creation of indexes based on multiple fields contained in different tables. This is possible only by having the other table open at the time the index is created or when the table containing the index is used.

With multi-field indexes for other table types (e.g., Paradox and InterBase), the multiple fields are delimited by the semi-colon (;), as in:

```
  LastName;FirstName
```

In dBASE expression indexes that concatenate multiple fields, an actual expression is used:

```
  LastName + FirstName
```

When creating index expressions that concatenate two or more fields, all of the fields included in the index expression must be of the same data type. Additionally, if they are to be concatenated instead of added together, the fields must all be of String type. For example, given two Integer type fields, Value1 and Value2, the index expression...

```
  Value1 + Value2
```

...would not cause an error. But then, neither would it concatenate the two field values; it would add them together. Thus, if Value1 for a given record contained 4 and Value2 5, the resulting index node would be an Integer value of 9, not a String concatenation "45".

If fields to be included in an expression index are not of String type, they must be converted. Here are some dBASE functions to convert various data types to String for purposes of creating index expressions:

```
  STR( [,  [, ]])
```

Converts from either Float or Numeric dBASE types to Character (String)

```
  DTOS()
```

Converts Date value to Character, format YYYYMMDD

```
  MLINE(, )
```

Extracts a single line from a memo field as a Character value

Another consideration in creating indexes based on the concatenation of multiple field is the maximum allowable length of the index value. The value returned by an index expression may not exceed 100 characters. This is a limit on the length of the value returned by the expression, not on the length of the expression itself. For example, you cannot index on the concatenation of two fields that both have a length of 255 characters.

Expression Indexes Based On Modifications Of Field Values

In addition to creating indexes based on the concatenation of two or more field values, it is also possible to construct an index that is based on a modification of a field value. Examples of this include indexing on just the first three characters of a String type field, on just the year and month from a Date field, indexing on a contantenation of a String and Date field such that the ordering of the String field is ascending but the Date descending, and even indexing on Boolean field values.

Creating indexes that are based on modifications of field values requires at least a working knowledge of dBASE functions and syntax -- because the process uses dBASE, and not Delphi, functions and syntax. The dBASE function SUBSTR() extracts a substring of a String value. The Delphi equivalent for this dBASE function is Copy. But, of these two functions that serve the same purpose, only SUBSTR() may be used in dBASE index expressions.

To use dBASE functions in dBASE index expressions, simply include the function wherever an index expression is called for, using the appropriate dBASE syntax for the function, along with a reference to the name(s) of the field(s) used in the function. For example, an index expression based on only the last three characters of a String type field called Code, that is 20 characters long, would be:

```
  RIGHT(Code, 3)
```

Caution should be used in constructing dBASE index expressions that modify field values to ensure that the resulting expression would return a value of a consistent length for every record in the table. For instance, the dBASE TRIM() function removes the trailing blanks (ASCII decimal 32) from a String type value. If this were used in conjunction with concatenating two String fields where the field does not contain values of the same length for all records, the value resulting from the expression will not be the same for all records. Case in point, an index expression based on the concatenation of a LastName and a FirstName field, where the TRIM() function is applied to the LastName field:

```
  TRIM(LastName) + FirstName
```

This expression would not return values of a consistent length for all records. If the LastName and FirstName fields contained the values...

```
  LastName FirstName
  -------- ---------
  Smith    Jonas
  Wesson   Nancy
```

...the result of applying the index expression above would be:

```
  SmithJonas
  WessonNancy
```

As can be seen, the length of the value for the first record would be 10 characters, while that for the second 11 characters. The index nodes created for this index expression would be based on the field values for the first record encountered. This would result in an index node 10 characters long being applied to the field values for all record. In this example, that would result in the truncation of the expression value for the second record to "WessonNanc". This would subsequently cause searches based on the full field value to fail.

The solution to this dilemma would be to not use the TRIM() function so that the full length of the LastName field, including padding from the trailing spaces, is used. In indexes that use the IIF() function to order by one field or another, based on the evaluation of a logical expression in the IIF(), if the two fields are of different lengths, the shorter field would need to be padded with spaces to make it the same length as the longer field. For example, assuming an index that uses the IIF() function to index either on a Company or a Name field, based on the contents of Category field, and where the Company field is 40 characters long but the Name field is 25 characters long, the Name field would need to be padded with 15 spaces; accomplished with the dBASE function SPACE(). That index expression would then be:

```
  IIF(Category = "B", Company, Name + SPACE(15))
```

Searches And dBASE Expression Indexes

dBASE expression indexes are exceptions to the norm in how they are handled by Delphi and the BDE as opposed to how multiple field indexes for other table types are handled.

This puts such dBASE indexes into a separate class. Handling of such indexes by Delphi and the BDE is different than those for other table types. One of these differences is that not all index-based searching using Delphi syntax can be used with these dBASE expression indexes. The FindKey, FindNearest, and GotoKey methods of the TTable component cannot be used with expression indexes. If an attempt to use FindKey is made, this will result in the error message: "Field index out of range." If the GotoKey method is tried, this error message may occur or the record pointer may just not move (indicating the search value was not found). Only the GotoNearest method may be used with expression indexes. Even then, the GotoNearest method may not work with some index expressions. Experimentation will be needed to see whether the GotoNearest method will work with a given index expression.

Filtering With dBASE Expression Indexes

As with index-based searches, dBASE expression indexes present some exceptions when using Delphi filtering.

With an expression index active, the SetRange method of the TTable component will produce the error: "Field index out of range." However, with the same expression index active, the SetRangeStart and SetRangeEnd methods will successfully filter the data set.

For example, with an expression index concatenating a LastName and a FirstName field active, the code below using the FindKey method (intended to filter to just those records where the first character of the LastName field is "S") will fail with an error:

```
  begin
    Table1.SetRange(['S'], ['Szzz'])
  end;
```

Whereas, the code below, with the same expression index active and filtering on the same LastName field, will successfully filter the data and not incur an error:

```
  begin
    with Table1 do begin
      SetRangeStart;
      FieldByName('LastName').AsString := 'S';
      SetRangeEnd;
      FieldByName('LastName').AsString := 'Szzz';
      ApplyRange;
    end;
  end;
```

And, as is the case with index-based searches, with filtering, success of a filtering attempt will also be dependent on the index expression. The use of the SetRangeStart and SetRangeEnd methods in the preceeding example worked with an index that simply concatenated two String type fields. But if the expression for the index was instead based conditionally on one or the other fields through use of the IIF() function, the same filtering routine would fail (although without an error).

Some Handy dBASE Index Expressions

Here are some handy dBASE index expressions. Some are readily apparent in the intended purpose, others are more arcane.

Character field ascending and Date field descending

With a Character field called Name and a Date field OrdDate:

```
  Name + STR(OrdDate - {12/31/3099}, 10, 0)
```

Character field ascending and Numeric (or Float) field descending

With a Character field called Company and a Numeric field Amount (the Amount field being 9 digits wide with two decimal places):

```
  Company + STR(Amount - 999999.99, 9, 2)
```

Ordering by a Logical field

To have True values appear before False values for a Logical field called Paid:

```
  IIF(Paid, "A", "Z")
```

Two Numeric (or Float) fields

Assuming two Numeric fields with widths of five and two decimal places, the first field named Price and the second Quantity:

```
  STR(Price, 5, 2) + STR(Quantity, 5, 2)
```

Ordering by one field of two, depending on a logical condition

Assuming that if the Company field is empty, the record should be included in the sort order by the Name field (instead of an empty Company field).

```
  IIF(Company = " ", Name, Company)
```

Ordering by the names of months in a Character field

Assuming a field containing the names of the months ("Jan," "Feb" etc.) to put the records in proper month order (field named M):

```
  IIF(M="Jan", 1, IIF(M="Feb", 2, IIF(M="Mar", 3, IIF(M="Apr", 4,
  IIF(M="May", 5, IIF(M="Jun", 6, IIF(M="Jul", 7, IIF(M="Aug", 8,
  IIF(M="Sep", 9, IIF(M="Oct", 10, IIF(M="Nov", 11, 12)))))))))))
```

(The above is a single expression line, broken into multiple lines here due to page width.)

Ordering by the first line of a memo field

For a memo field named Notes:

```
  MLINE(Notes, 1)
```

Ordering by the middle three characters in a nine character long field

For a nine character long field called StockNo:

```
  SUBSTR(StockNo, 4, 3)
```

Creating dBASE Expression Indexes In Database Desktop

In the Database Desktop utility, indexes may be created for a table either duting the process of creating a new table or by restructuring an existing table. In both cases, the Define Index dialog is used to create one or more indexes for the table used.

To get to the Create Index dialog while creating a new table, in the Create dBASE Table dialog (showing the structure), from the Table Properties listbox select "Indexes" and click the Define button.

To get to the Create Index dialog to create an index for an existing table, select Utilities|Restructure, select the table file from the Select File dialog, and in the Restructure dBASE Table dialog (showing the table structure) from the Table Properties listbox select "Indexes" and click the Define button.

Once in the Create Index dialog, expression indexes can be created by clicking the Expression Index button and entering the expression to be used in the Expression Index entry field. To assist in this process, you can double-click on a field name in the Field List listbox and that field name will be inserted into the Index Expression entry field at the current insertion point (caret position).

Once the index expression has been entered, click the OK button. Enter the name of the new index tag in the Index Tag Name entry field on the Save Index As dialog and click OK. (Remember, dBASE index tag names cannot exceed ten characters in length and must abide by the normal dBASE naming conventions.)

Creating dBASE Expression Indexes In Delphi Applications

dBASE indexes can be created programmatically in Delphi applications, either as a new table is being created (CreateTable method of the TTable component) or by adding an index to an existing table.

Another peculiarity of the dBASE expression index and the BDE is that the table must exist prior to creating an expression index. Thus, while single-field indexes may be created as the table is created by populating TIndexDef objects, this cannot be done with expression indexes. Expression indexes can only be added to a newly created table after the call is made to the CreateTable method, using the AddIndex method. The Options parameter of the AddIndex method must include the index option value ixExpression. This index option is unique to dBASE indexes, and should only be used with dBASE expression indexes. For example:

```
  with Table1 do begin
    Active := False;
    DatabaseName := 'Delphi_Demos';
    TableName := 'CustInfo';
    TableType := ttdBASE;
    with FieldDefs do begin
      Clear;
      Add('LastName', ftString, 30, False);
      Add('FirstName', ftString, 20, False);
    end;
    CreateTable;
    AddIndex('FullName', 'LastName + FirstName', [ixExpression]);
  end;
```

Learning More About dBASE Functions And Syntax

Only dBASE functions and syntax that apply to data manipulation can be used to construct a dBASE expression index. However, it is beyond the scope of this technical article to fully list and describe all of these functions. To learn more about dBASE data manipulation functions, the user is advised to consult the dBASE Language Reference manual or one of the many third-party dBASE books.

Reference:

7/16/98 4:33:55 PM

Last Modified: 01-SEP-99