Keyword Support in 4D v11 SQL

By Josh Fletcher, Technical Support Engineer, 4D Inc.

Technical Note 01-08

Abstract

This Technical Note covers the new keyword features in 4D v11 SQL. 4D v11 SQL adds support for keywords in the form of three new features: keyword indexes, the "contains keyword" comparison operator, and support for regular expressions (regex). These powerful tools allow 4D Developers to perform fast, efficient keyword-based searches.

Introduction

In previous versions of 4D it was possible to do a "contains" search in order to test if a set of text-based data contained the pattern being searched for. However the delineation of "words" was not really natively supported. It was up to the developer to further refine the search and/or "massage" the search results to get the desired data.

4D v11 SQL adds support for "keywords" for both Alpha and Text fields. With this new support for keywords in most cases the developer has no need to "tweak" the search when a keyword search is to be performed. Now that 4D is able to determine what constitutes a "word", the developer need only specify the pattern to search for. Furthermore, if 4D's interpretation of what constitutes a "word" is not acceptable, the new regex features can be used to specify a precise search pattern.

Whether 4D's native interpretation or a specific regex pattern is used, the end result is the same: faster, more efficient searches and cleaner code.

The keyword support in 4D v11 SQL is accomplished with three new features:

- Keyword indexes
- The "contains keyword" operator
- Support for regex

The new keyword index provides a performance benefit when performing keyword searches on 4D fields.

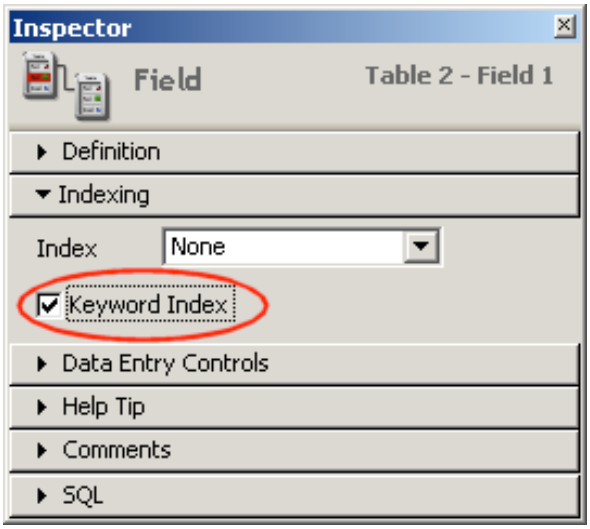
The "contains keyword" operator, of course, is used to perform the actual keyword searches.

Support for regex is provided with the new **Match regex** command. Note that the new regex features in 4D v11 SQL are only briefly covered in this Technical Note. For more information on regex in 4D see Technical Note 07-47, "Introduction to Regular Expressions in 4D v11 SQL".

Each of these features is explored in the following sections.

Keyword Index

In 4D v11 SQL it is possible to set a keyword index on any Text or Alpha field. This is done through the Inspector:



The "Keyword Index" check box appears under the "Indexing" section in the Inspector.

Once the keyword index is enabled, the text in any records that have the indexed field will be indexed, word-by-word. One of the major benefits of this type of index is that it drastically reduces the search time when doing a keyword search. Some other things to note:

- All words are indexed, even if they are only 1 or 2 characters long.
- It is possible to have **both** a keyword index and a "normal" index on the same field. 4D v11 SQL will automatically use the appropriate index depending on the operation performed.
- Text field data can now be stored outside the record. However, Text fields cannot have a "normal" index if the text data is not stored in the record. A keyword index can be specified regardless of the storage method.

The "contains keyword" Operator

4D v11 SQL introduces a new comparison operator in order to facilitate keyword searches. The new operator is symbolized the percent symbol, **%**. This operator is called the "contains keyword" operator. Here is a simple example:

Let's say we want to find all customers who live in cities that contain the word "Mar". Note that we are looking for the **word** "Mar", not the string of characters "Mar". Here is the query:

```
QUERY([Customers];[Customers]City%"Mar")
```

This simple query would return all customers who live in a city that contains the word "Mar". Note that this is **not** the same as the following:

```
QUERY([Customers];[Customers]City="@Mar@")
```

Or:

```
QUERY([Customers];[Customers]City="Mar@")
```

Here is a table that illustrates the difference:

Match?	%"Mar"	="@Mar@"	="Mar@"
"Mar Vista"	True	True	True
"Marina"	**False**	True	True

In this case the keyword search makes it possible to find more specialized results. Here we would not find erroneous results like "Marina" whereas in previous versions of 4D further processing would have been required to eliminate cities that only contain the string "Mar" but not the word.

Keyword Searches

4D v11 SQL defines a "word" in a keyword search as a string of letters or numbers delineated by "separators". Separators are defined as punctuation, symbols, and white space.

Note: *Keyword searches can be performed regardless of whether or not the keyword index is enabled for a field. The benefit of using the keyword index is performance.*

Dealing with Separators

4D's definition of what constitutes a "word" brings up some behaviors to be aware of. For example say the [Invoices] table contains two invoices in the amounts of "123.34" and "123.00". This query will return both records:

```
QUERY([Invoices];[Invoices]Amount%"123")
```

Why? Because all punctuation is treated as separators when determining what constitutes a "word". In the above example there are actually two "words"; the dollar amount, e.g. "123" and the cents, e.g. "34". The query returns both records because, obviously, "123" matches one of the words in the string being tested. Thus a keyword search is probably not the ideal tool for matching currency.

Similarly say you want to find all customers with the last name "Smith-Doe":

```
QUERY([Customers];[Customers]Lastname%"Smith")
```

This query would find all of the customers whose names are "Smith-Doe" since the hyphen is a separator. But, it would also find any customer whose last name was "Smith", "Smith Van Helsing", etc. How can this be avoided?

Multi-Keyword Search

You cannot search for more than one keyword at a time with the % operator. For example, given a record that contains "The quick brown fox" the following query is performed:

```
QUERY([Some Table];[Some Table]theRecord%"brown fox")
```

No records will be found on a keyword-indexed field; the % operator does not support multiple words in the search pattern. In this case a compound query must be used:

```
QUERY([Some Table];[Some Table]theRecord%"brown";*)
QUERY([Some Table];&;[Some Table]theRecord%"fox")
```

Also the new regex features in 4D v11 SQL could be used to perform the match in one statement (there is an example of this in the regex section).

Sensitivity

The % operator is neither case-sensitive nor diacritic-sensitive. Here are some examples:

Data	Pattern	Match?
"BROWN FOX"	"brown"	True
"Brown Fox"	"BROWN"	True
"Piñata"	"pinata"	True

Wildcards

The % operator supports the 4D wildcard character "@" at the end of the keyword. For example:

Data	Pattern	Match?
"Brown Fox"	"brown@"	True
"Brown Fox"	"bro@"	True
"Brown Fox"	"own@"	**False**

Using the @ operator at the end of the word says, in essence, match any word that begins with the pattern specified.

Support for Regex

4D v11 SQL adds support for regular expressions via the Match regex command. With regards to keyword searching, regex allows the developer to strictly specify what constitutes a "word".

Here is one of the previous examples that could be improved with regex:

```
QUERY([Some Table];[Some Table]theRecord%"brown";*)
QUERY([Some Table];&;[Some Table]theRecord%"fox")
```

A similar statement, using Match regex, would be:

```
$match:=Match regex(".* brown fox";[Some Table]theRecord))
```

Using regex, the pattern ".* brown fox" is used to try and find a match in the text "The quick brown fox". This regex pattern means, "any number of characters, followed by a space, and ending with 'brown fox'."

The tricky part of regex is, of course, coming up with the correct pattern. For more information on regex in 4D see Technical Note 07-47, "Introduction to Regular Expressions in 4D v11 SQL".

Obviously the above examples are not strictly equivalent. The first example performs a query while the second example is only performing a comparison. The regex example would need to be used in a loop on all of the records in order to act like a query. The advantage here is obviously not performance but, instead, strict control over the search.

Performance

Here is a simple performance test:

Setup

Created a database with 1 table and 2 Text fields. One of the text fields has a keyword index, the other does not.

Created 10 records. The text data came from 10 random Wikipedia pages (using the "Random article" link). Both fields contain the same data. Then duplicated those records at random so that there were a total of 50,000 records in the table. This worked out to a 1.5GB data file.

The test machine was a MacBook Pro, 2.33 GHz Intel Core 2 Duo, 2GB RAM running Mac OS X 10.4.10.

Tests

The following tests were performed:

Test 1: Keyword search on the keyword indexed field

```
QUERY([Table_1];[Table_1]IndexedText%"maya")
```

Test 2: Keyword search on the non-indexed field

```
QUERY([Table_1];[Table_1]NonIndexedText%"maya")
```

Test 3: Wildcard search on the keyword indexed field

```
QUERY([Table_1];[Table_1]IndexedText="@maya@")
```

Test 4: Wildcard search on the non-indexed field

```
QUERY([Table_1];[Table_1]NonIndexedText="@maya@")
```

The execution time of each query was tracked, as well as the number of records found. Finally, each test was repeated 10 times.

Results

Execution times are in milliseconds and are the average of the 10 repetitions.

Test	Records Found	Time (ms)
Keyword w/ Index	2443	0.3
Keyword w/o Index	2443	9,640.6
Wildcard w/ Index	2443	19,255.0
Wildcard w/o Index	2443	18,765.2

As can be seen there, is tremendous benefit to both using the keyword search (instead of wildcards) and enabling the keyword index.

Conclusion

This Technical Note described the new keyword features in 4D v11 SQL. With the new keyword index and keyword search operator, tremendous performance gains can be made when searching text data. If more granular search control is required, the regex functionality can be used. These improvements to the 4D product line provide powerful tools for any 4D developer.

Related Resources

Technical Note 07-47, "Introduction to Regular Expressions in 4D v11 SQL"
http://www.4d.com/support/technotes.html