SQL Server Full Text Search Language Features, Part 2

04 July 2006
by Hilary Cotter

Query-time behavior

This is the second of a two-part article that explores the language features of SQL Full-text Search (SQL FTS), an component of SQL Server 7 and above that allows fast and efficient querying of large amounts of unstructured textual data. Part I dealt with **index time** language options, covering how words or tokens are broken from the text stream emitted from the iFilters and stored in the index.

Here, we move on to **query time** behavior. This refers to the manner in which search phrases supplied in a query are expanded by the parser before hitting the full-text index. This expansion process is performed by a language-specific component called a **stemmer** and the exact expansions that occur will depend on the language-specific rules that are in place. However, for example, expansion can be used to accommodate the following types of searches:

- Searches for plural forms of a word – a search on apple would also return apples
- Thesaurus searches for synonymous forms of word – a search on *IE* might return hits to *Internet Explorer* and *IE* (thesaurus based expansion); a search on *Bombay* might also return hits to *Mumbai* (thesaurus based replacement)
- Searches that will return all different linguistic forms of a word (called **generations**) – a search on bank would return hits to *banking*, *banked*, *banks*, *banks'* and *bank's*, etc. (all declensions and/or conjugations of the search term bank)
- Accent insensitive searches – a search on *café* would return hits to *cafe* and *café*

Before we investigate stemming in more detail, it's worth discussing the sort of expansions that will be performed depending on whether you issue a *strict* query (using the CONTAINS predicate) or a *fuzzy* query (using the FREETEXT predicate).

Strict querying using CONTAINS

By default, use of the CONTAINS predicate will entail minimal language specific query time expansion. For example, consider a search on the term *book*.

```
select * from TableName where CONTAINS(*,'Book')
```

The asterisk (*) is used to search all full-text indexed columns. In SQL 2000 you can search all full-text indexed columns or one named full-text index column:

```
select * from TableName where CONTAINS(col1,'Book'))
```

In SQL 2005 you can search all columns, a named column, or a subset of all full-text indexed columns:

```
select * from TableName where CONTAINS((col1,col2),'Book')
```

There may be text in the table where the term "book" is used both as a noun (*John threw the **book** at Sam*) and a verb (*John was **booked** for assault and battery when he threw the book at Sam*). However, a query using the basic CONTAINS predicate will only match with *book* (and alternative word forms for *book* – although in English there are none).

Fortunately, you can choose to have your CONTAINS search expanded to capture different generations of a word by using the FORMSOF term. This term accepts two arguments – INFLECTIONAL or THESAURUS. The INFLECTIONAL argument will expand the search phrase to search on all conjugations and declensions or each word in the search phrase, and the THESAURUS argument will enable a thesaurus expansion on the search phrase. Here are two examples of what this would look like:

```
Select * from TableName where CONTAINS(*,'FORMSOF(INFLECTIONAL,run)')
```

And

```
Select * from TableName where CONTAINS(*,'FORMSOF(THESAURUS,run)')
```

Furthermore, with some languages, additional breaking will occur by default at query time. For example, consider a search on *häuser* (German for houses) on a full-text indexed table containing the term *häuser* in one row and *haeuser*, the alternative word form, in another row. If the default full-text language on the Server was US English then the query would not be expanded and you would only get a hit on the row containing *häuser*:

```
select * from TableName where CONTAINS(*,'häuser')
```

However, if your server had a default full-text language of German, you would get hits on both rows because your search phrase would be automatically expanded to the following:

```
select * from TableName where CONTAINS(*,'"häuser" or "haeuser"')
```

Note that the search phrases are now wrapped in double quotes. When using CONTAINS, you must use double quotes to wrap a search phrase containing more than one word.

Fuzzy querying using FREETEXT

The term "fuzzy" is something of a misnomer as it implies use of a technology such as *Levenstein Edit Distance* which is tolerant of minor spelling variations or mistakes, which is not the case. In fact, use of FREETEXT means that, by default, the search is expanded to encompass all generations of the searched word, for the default full-text language setting of your server. So, to continue the previous "book" example, a FREETEXT search on book:

```
Select * from TableName
where FREETEXT
(*,'Book')
```

would be expanded to the equivalent of:

```
Select * from TableName
where CONTAINS
(*,'"Book" or "books" or "book''s" or books''" or "booked" or "booking")
```

The search would also return any relevant thesaurus expansions.

Note that when using FREETEXT in SQL 2000 you have to wrap your search phase in double quotes if a search phrase contains more than one word or token (as when using CONTAINS). However, in SQL 2005 you no longer need to do this when using the FREETEXT predicate. In fact if you wrap your search phrase in double quotes stemming and thesaurus expansion and replacements are disabled and your search phrase is treated as a strict match – in other words the functional equivalent of a CONTAINS predicate (although the ranking algorithm is different).

The LANGUAGE argument

In SQL 2005, you can override the default full-text language settings for your server by using the LANGUAGE argument with the CONTAINS and FREETEXT predicates and have your searches conducted in different languages. For example, on a SQL 2005 Server with a default full-text language setting of US_ENGLISH, you could search on German phrases by using the LANGUAGE argument, and have your German search terms stemmed in a FREETEXT predicate according to German language rules.

Script 3 provides a full example of this (see the "Scripts" link at the top right of this page). In this script we are loading HTML documents into the varbinary data type column of a table called German. In each case we specify that word boundaries are identified using the US English Word breaker, as denoted by the LANGUAGE 1033 clause:

```
CREATE FULLTEXT INDEX ON German
   (varbinarycol TYPE COLUMN documentExtension
      LANGUAGE 1033 )
KEY INDEX GermanPK
```

However, when we load some of these documents into the database, we include a German language MetaTag:

```
INSERT INTO German
 (varbinarycol, documentExtension,word, language )
VALUES
 (CONVERT(VARBINARY(256),
  '<HTML><HEAD><META NAME="MS.LOCALE" CONTENT="DE">
     </HEAD><BODY>wanderlust</BODY></HTML>'),
  '.htm','wanderlust','GERMAN')
```

And when we load others we use the US English language MetaTag:

```
INSERT INTO German
 (varbinarycol, documentExtension,word, language )
VALUES
 (CONVERT(VARBINARY(256),
  '<HTML><HEAD><META NAME="MS.LOCALE" CONTENT="EN-US">
     </HEAD><BODY>wanderlust</BODY></HTML>'),
  '.htm','wanderlust','ENGLISH')
```

So, a query on *wanderlust* returns two documents – the English and German language documents contain the word *wanderlust*.

```
select word, language from german where freetext(*,'wanderlust')
```

However, a search on *lust* returns three documents – two exact matches for *lust* and one further match with *wanderlust* in the German language document. The reason for this is that the German word breaker was launched by the HTML language tag and this overrides the word breaker setting in our full-text index creation statement. As a result, *wanderlust* is broken by the German word breaker into *wanderlust*, *wandern*, and *lust*. Hence a search on *lust* returns the German HTML document with *wanderlust* in it.

If we override the default full-text language settings for our server (US English) and set the language of our search to German:

```
select word, language from german
where freetext(*,'wanderlust', language 1031)
```

Then we get six rows returned, since now our search phrase is automatically expanded into *wanderlust*, *wandern* and *lust*, and so we get matches for all three of these in both German and English documents.

Stemming

By **stemming**, Microsoft means:

- Generating declensions of a word – possessive and gender (although gender is not supported in SQL FTS)
- Generating conjugations of a word

Stemming reduces a search term to its linguistic root (for example *careers* would be reduced to *career*), and then expands from that root to encompass all linguistic forms of that root (for example *career, career's, careers', careers', careered*, and *careering*).

Most stemmers implement a form of the *Porter Stemming Algorithm* with dictionaries to handle irregularly stemmed words such as goose/geese, mouse/mice, fit/fat, and so on. Some of the more sophisticated stemmers implement an inference algorithm which when encountering a freshly minted word (like blog for instance) can conjugate, or generate declensions of, this new word based on its similarity to other older words.

Note: It is interesting to search on terms such as "careers" on various web sites to see what matches are returned. It is frequently incorrectly stemmed as careful.

Declensions

Generating declensions is best illustrated through a few examples:

- A FREETEXT or CONTAINS search on the noun *apple* would return hits to *apple* and *apples*
- A FREETEXT or CONTAINS inflectional search on the advjective *pretty* would match to *pretty*, *prettier* and *prettiest*.

Please see script 4 for examples of declination in action.

Conjugation

Again, this is best illustrated by example. If you search on the verb *book* the stemmer will generate all conjugations of this term and row matches will be returned for *book, booked, booking,* and *books*. Script 4 has an example of this.

Thesaurus-based searches

The thesaurus option allows expansion of your search term to include preconfigured terms, and replacement of your search term with another search term. To configure your thesaurus options, edit the files found in C:\Program Files\Microsoft SQL Server\MSSQL.1 \MSSQL\FTData. The thesaurus file for your language will follow the naming convention **tsXXX.xml** where **XXX** represents your language code, i.e. ENU for US-English, ENG for UK-English, and so on.

If you edit tsXXX.xml in a text editor you will find that there are two sections, or nodes, to the thesaurus file: an expansion node, and a replacement node. The expansion node is used to extend your search argument from one term to another – known as **thesaurus expansion**. For example in the thesaurus file you will find the following expansion:

```
<expansion>
   <sub>Internet Explorer</sub>
   <sub>IE</sub>
   <sub>IE5</sub>
</expansion>
```

This will convert any searches on *IE* to search on *IE, IE5,* and *Internet Explorer.*

Note that you will need to remove the comment lines from your thesaurus file in order to activate this.

The replacement node is used to replace one search argument with another – known as **thesaurus replacement**. For example, you may want a search on *sex* to be replaced by a search on *gender*. By configuring your thesaurus file as follows, any search on *sex* will only get hits to rows containing the term *gender* and will miss rows that do not contain *gender*, but contain the word *sex*:

```
<replacement>
   <pat>sex</pat>
   <sub>gender</sub>
</replacement>
```

The pat element (*sex*) indicates the pattern you want substituted by the sub element (*gender*).

A FREETEXT query will automatically use the thesaurus file for the language type, and a CONTAINS query will do so when you use FORMSOF with the thesaurus option:

```
SELECT * FROM TableName
WHERE CONTAINS(*,'FORMSOF(Thesaurus,"IE")')
```

This would return matches to rows containing *IE, IE5*, and *Internet Explorer*.

Diacritics and accents

A diacritical mark, or diacritic (sometimes called an accent mark), is a mark added to a letter to alter a word's pronunciation or to distinguish it from other similar words. Previous versions of SQL Server were unable to handle accents so a search on *café* would not match with rows containing *cafe* – even when using the French word breaker or stemmer.

SQL Server 2005 has the option of creating accent insensitive catalogs. This means that a search on *café* will match with *café, cafe, cafè, cafê,* and *cafë*. In other words it does a very unintelligent match, but one that will solve many of the problems that users have when handling diacritics. Script 5 illustrates this functionality.

False friends or false conjugates

When you are searching content that is written in a variety of languages, you must be careful to account for a linguistic phenomenon called *false friends* or *false conjugates*.

A false friend is a word that has the same spelling in different languages, but different meanings. An example is the word *gift* which means a present in English, but in German means poison. You need to be careful to avoid a search on the English word returning rows with German content.False friends are not to be confused with wanderworts – words which occur with the same spelling in different languages and also have the same meaning.

The best approach to problems like this is to segregate your content into language specific columns.

UK vs. US English at query time

There are very slight variations between US and UK English stemmers, mostly arising from spelling variations between American and International English. For example in America we spell *color* and *center, analog, apologize, flavor*; whereas in Britain these words are spelt *colour, centre, analogue, apologise,* and *flavour*. There is no conformity in the rest of the English speaking world, however most of the English speaking countries use the International version of English; although it may vary from one region to another – for example in Western Canada – the spelling *center* predominates, in Eastern Canada *centre* predominates.

If you are querying in International English *color* would be stemmed as *color* , whereas *colour* would be stemmed as *colour, colour's, colours, colours', coloured* and *colouring*. Likewise if you searched on *colour* using American English you would not get hits to rows containing *colour, colour's, colours, colours', coloured* and *colouring.*

The key to avoiding idiomatic spelling variations is to scrupulously clean your content or use the thesaurus option to expand your search to alternate spellings.

Summary

This two-part article covered language features of SQL full text search. Key to keep in mind is that the language features occur at index time and query time. Different content types may control which language resources are used and, depending on the language resources used,

the content may be broken at intervals other than white space and punctuation. A word or token may be indexed as more than one word, so when you query on a word a match may be made to one of the alternate word forms of that word. At query time the search is expanded according to language rules to provide a more natural way of searching that incorporates language features.

These language features make SQL full-text search a very valuable tool for the developer and DBA.