**Appendix I: The Search Engine Syntax**

CONTAINS

      ( { column\_name | ( column\_list ) | \* }

          , '< contains\_search\_condition >'

   [ , LANGUAGE language\_term ]

      )

< contains\_search\_condition > ::=

    { < simple\_term >

    | < prefix\_term >

    | < generation\_term >

    | < proximity\_term >

    | < weighted\_term >

    }

    | { ( < contains\_search\_condition > )

    [ { < AND > | < AND NOT > | < OR > } ]

< contains\_search\_condition > [ ...n ]

    }

< simple\_term > ::=

          word | " phrase "

< prefix term > ::=

     { "word \* " | "phrase \*" }

< generation\_term > ::=

     FORMSOF ( { INFLECTIONAL | THESAURUS } , < simple\_term > [ ,...n ] )

< proximity\_term > ::=

     { < simple\_term > | < prefix\_term > }

     { { NEAR | ~ }

     { < simple\_term > | < prefix\_term > }

     } [ ...n ]

< weighted\_term > ::=

     ISABOUT

        ( { {

  < simple\_term >

  | < prefix\_term >

  | < generation\_term >

  | < proximity\_term >

  }

   [ WEIGHT ( weight\_value ) ]

   } [ ,...n ]

        )

< AND > ::=

     { AND | & }

< AND NOT > ::=

     { AND NOT | & !}

< OR > ::=

     { OR | | }

http://i.msdn.microsoft.com/Global/Images/clear.gif **Arguments**

*column\_name*

Is the name of a full-text indexed column of the table specified in the FROM clause. The columns can be of type **char**, **varchar**, **nchar**, **nvarchar**, **text**, **ntext**, **image**, **xml**, **varbinary**, or **varbinary(max)**.

*column\_list*

Specifies two or more columns, separated by commas. *column\_list* must be enclosed in parentheses. Unless *language\_term* is specified, the language of all columns of *column\_list* must be the same.

\*

Specifies that the query will search all full-text indexed columns in the table specified in the FROM clause for the given search condition. The columns in the CONTAINS clause must come from a single table that has a full-text index. Unless *language\_term* is specified, the language of all columns of the table must be the same.

LANGUAGE *language\_term*

Is the language to use for word breaking, stemming, thesaurus expansions and replacements, and noise-word (or [stopword](http://msdn.microsoft.com/en-us/library/ms142551.aspx)) removal as part of the query. This parameter is optional.

If documents of different languages are stored together as binary large objects (BLOBs) in a single column, the locale identifier (LCID) of a given document determines what language to use to index its content. When querying such a column, specifying LANGUAGE *language\_term* can increase the probability of a good match.

*language\_term* can be specified as a string, integer, or hexadecimal value corresponding to the LCID of a language. If *language\_term* is specified, the language it represents will be applied to all elements of the search condition. If no value is specified, the column full-text language is used.

When specified as a string, *language\_term* corresponds to the **alias** column value in he [sys.syslanguages (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190303.aspx) compatibility view. The string must be enclosed in single quotation marks, as in '*language\_term*'. When specified as an integer, *language\_term* is the actual LCID that identifies the language. When specified as a hexadecimal value, *language\_term* is 0x followed by the hexadecimal value of the LCID. The hexadecimal value must not exceed eight digits, including leading zeros.

If the value is in double-byte character set (DBCS) format, SQL Server will convert it to Unicode.

If the language specified is not valid or there are no resources installed that correspond to that language, SQL Server returns an error. To use the neutral language resources, specify 0x0 as *language\_term*.

<*contains\_search\_condition*>

Specifies the text to search for in *column\_name* and the conditions for a match.

*<contains\_search\_condition>* is **nvarchar**. An implicit conversion occurs when another character data type is used as input. In the following example, the @SearchWord variable, which is defined as varchar(30), causes an implicit conversion in the CONTAINS predicate.

USE AdventureWorks;

GO

DECLARE @SearchWord varchar(30)

SET @SearchWord ='performance'

SELECT Description

FROM Production.ProductDescription

WHERE CONTAINS(Description, @SearchWord);

Because "parameter sniffing" does not work across conversion, use **nvarchar** for better performance. In the example, declare @SearchWord as nvarchar(30).

USE AdventureWorks;

GO

DECLARE @SearchWord nvarchar(30)

SET @SearchWord = N'performance'

SELECT Description

FROM Production.ProductDescription

WHERE CONTAINS(Description, @SearchWord);

You can also use the OPTIMIZE FOR query hint for cases in which a non optimal plan is generated.

*word*

Is a string of characters without spaces or punctuation.

*phrase*

Is one or more words with spaces between each word.

|  |
| --- |
| **Note:** |
| Some languages, such as those written in some parts of Asia, can have phrases that consist of one or more words without spaces between them. |

<simple\_term>

Specifies a match for an exact word or a phrase. Examples of valid simple terms are "blue berry", blueberry, and "Microsoft SQL Server". Phrases should be enclosed in double quotation marks (""). Words in a phrase must appear in the same order as specified in *<contains\_search\_condition>* as they appear in the database column. The search for characters in the word or phrase is not case-sensitive. Noise words (or [stopwords](http://msdn.microsoft.com/en-us/library/ms142551.aspx)) (such as a, and, or the) in full-text indexed columns are not stored in the full-text index. If a noise word is used in a single word search, SQL Server returns an error message indicating that the query contains only noise words. SQL Server includes a standard list of noise words in the directory \Mssql\Binn\FTERef of each instance of SQL Server.

Punctuation is ignored. Therefore, CONTAINS(testing, "computer failure") matches a row with the value, "Where is my computer? Failure to find it would be expensive." For more information on word-breaker behavior, see [Word Breakers and Stemmers](http://msdn.microsoft.com/en-us/library/ms142509.aspx).

<prefix\_term>

Specifies a match of words or phrases beginning with the specified text. Enclose a prefix term in double quotation marks ("") and add an asterisk (\*) before the ending quotation mark, so that all text starting with the simple term specified before the asterisk is matched. The clause should be specified this way: CONTAINS (column, '"text\*"'). The asterisk matches zero, one, or more characters (of the root word or words in the word or phrase). If the text and asterisk are not delimited by double quotation marks, so the predicate reads CONTAINS (column, 'text\*'), full-text search considers the asterisk as a character and searches for exact matches to text\*. The full-text engine will not find words with the asterisk (\*) character because word breakers typically ignore such characters.

When *<prefix\_term>* is a phrase, each word contained in the phrase is considered to be a separate prefix. Therefore, a query specifying a prefix term of "local wine\*" matches any rows with the text of "local winery", "locally wined and dined", and so on.

<generation\_term>

Specifies a match of words when the included simple terms include variants of the original word for which to search.

INFLECTIONAL

Specifies that the language-dependent stemmer is to be used on the specified simple term. Stemmer behavior is defined based on stemming rules of each specific language. The neutral language does not have an associated stemmer. The column language of the columns being queried is used to refer to the desired stemmer. If *language\_term* is specified, the stemmer corresponding to that language is used.

A given *<simple\_term>* within a *<generation\_term>* will not match both nouns and verbs.

THESAURUS

Specifies that the thesaurus corresponding to the column full-text language, or the language specified in the query is used. The longest pattern or patterns from the *<simple\_term>* are matched against the thesaurus and additional terms are generated to expand or replace the original pattern. If a match is not found for all or part of the *<simple\_term>*, the non-matching portion is treated as a *simple\_term*. For more information on the full-text search thesaurus, see [Thesaurus Configuration](http://msdn.microsoft.com/en-us/library/ms142491.aspx).

<proximity\_term>

Specifies a match of words or phrases that must be in the document that is being searched. Like the AND operator, *<proximity\_term>* requires both the search terms to exist in the document being searched.

NEAR | ~

Indicates that the word or phrase on each side of the NEAR or ~ operator must occur in a document for a match to be returned. Several proximity terms can be chained, as in **a NEAR b NEAR c** or a ~ b ~ c. Chained proximity terms must all be in the document for a match to be returned.

When used in the CONTAINSTABLE function, the proximity of the search terms affects the ranking of each document. The nearer the matched search terms are in a document, the higher the ranking of the document. If matched search terms are >50 terms apart, the rank returned on the document is 0.

For example, CONTAINS (*column\_name*, 'fox NEAR chicken') and CONTAINSTABLE (*table\_name*, *column\_name*, 'fox ~ chicken') would both return any documents in the specified column that contain both "fox" and "chicken". In addition, CONTAINSTABLE returns a rank for each document based on the proximity of "fox" and "chicken". For example, if a document contains the sentence, "The fox ate the chicken," its ranking would be high.

NEAR indicates the logical distance between terms, rather than the absolute distance between them. For example, terms within different phrases or sentences within a paragraph are treated as farther apart than terms in the same phrase or sentence, regardless of their actual proximity, on the assumption that they are less related. Likewise, terms in different paragraphs are treated as being even farther apart.

<weighted\_term>

Specifies that the matching rows (returned by the query) match a list of words and phrases, each optionally given a weighting value.

ISABOUT

Specifies the *<weighted\_term>* keyword.

WEIGHT(*weight\_value*)

Specifies a weight value, which is a number from 0.0 through 1.0. Each component in *<weighted\_term>* may include a *weight\_value*. *weight\_value* is a way to change how various portions of a query affect the rank value assigned to each row matching the query. WEIGHT does not affect the results of CONTAINS queries, but WEIGHT impacts rank in [CONTAINSTABLE](http://msdn.microsoft.com/en-us/library/ms189760.aspx) queries.

|  |
| --- |
| **Note:** |
| The decimal separator is always a period, regardless of the operating system locale. |

{ AND | & } | { AND NOT | &! } | { OR | | }

Specifies a logical operation between two contains search conditions.

AND | &

Indicates that the two contains search conditions must be met for a match. The ampersand symbol (&) may be used instead of the AND keyword to represent the AND operator.

AND NOT | &!

Indicates that the second search condition must not be present for a match. The ampersand followed by the exclamation mark symbol (&!) may be used instead of the AND NOT keyword to represent the AND NOT operator.

OR | |

Indicates that either of the two contains search conditions must be met for a match. The bar symbol (|) may be used instead of the OR keyword to represent the OR operator.

When *<contains\_search\_condition>* contains parenthesized groups, these parenthesized groups are evaluated first. After evaluating parenthesized groups, these rules apply when using these logical operators with contains search conditions:

* NOT is applied before AND.
* NOT can only occur after AND, as in AND NOT. The OR NOT operator is not allowed. NOT cannot be specified before the first term. For example, CONTAINS (mycolumn, 'NOT "phrase\_to\_search\_for" ' ) is not valid.
* AND is applied before OR.
* Boolean operators of the same type (AND, OR) are associative and can therefore be applied in any order.

*n*

Is a placeholder indicating that multiple CONTAINS search conditions and terms within them can be specified.

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

Full-text predicates and functions work on a single table, which is implied in the FROM predicate. To search on multiple tables, use a joined table in your FROM clause to search on a result set that is the product of two or more tables.

CONTAINS is not recognized as a keyword if the compatibility level is less than 70. For more information, see [sp\_dbcmptlevel (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms178653.aspx).

Full-text predicates are not allowed in the [OUTPUT clause](http://msdn.microsoft.com/en-us/library/ms177564.aspx) when the database compatibility level is set to 100.

**Comparison of LIKE to Full-Text Search**

In contrast to full-text search, the [LIKE](http://msdn.microsoft.com/en-us/library/ms179859.aspx) Transact-SQL predicate works on character patterns only. Also, you cannot use the LIKE predicate to query formatted binary data. Furthermore, a LIKE query against a large amount of unstructured text data is much slower than an equivalent full-text query against the same data. A LIKE query against millions of rows of text data can take minutes to return; whereas a full-text query can take only seconds or less against the same data, depending on the number of rows that are returned.

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

**A. Using CONTAINS with <simple\_term>**

The following example finds all products with a price of $80.99 that contain the word "Mountain".

USE AdventureWorks;

GO

SELECT Name, ListPrice

FROM Production.Product

WHERE ListPrice = 80.99

AND CONTAINS(Name, 'Mountain');

GO

**B. Using CONTAINS and phrase in <simple\_term>**

The following example returns all products that contain either the phrase "Mountain" or "Road".

USE AdventureWorks;

GO

SELECT Name

FROM Production.Product

WHERE CONTAINS(Name, ' "Mountain" OR "Road" ')

GO

**C. Using CONTAINS with <prefix\_term>**

The following example returns all product names with at least one word starting with the prefix chain in the Name column.

USE AdventureWorks;

GO

SELECT Name

FROM Production.Product

WHERE CONTAINS(Name, ' "Chain\*" ');

GO

**D. Using CONTAINS and OR with <prefix\_term>**

The following example returns all category descriptions containing strings with prefixes of either "chain" or "full".

USE AdventureWorks;

GO

SELECT Name

FROM Production.Product

WHERE CONTAINS(Name, '"chain\*" OR "full\*"');

GO

**E. Using CONTAINS with <proximity\_term>**

The following example returns all product names that have the word bike near the word performance.

USE AdventureWorks;

GO

SELECT Description

FROM Production.ProductDescription

WHERE CONTAINS(Description, 'bike NEAR performance');

GO

**F. Using CONTAINS with <generation\_term>**

The following example searches for all products with words of the form ride: riding, ridden, and so on.

USE AdventureWorks;

GO

SELECT Description

FROM Production.ProductDescription

WHERE CONTAINS(Description, ' FORMSOF (INFLECTIONAL, ride) ');

GO

**G. Using CONTAINS with <weighted\_term>**

The following example searches for all product names containing the words performance, comfortable, or smooth, and different weightings are given to each word.

USE AdventureWorks;

GO

SELECT Description

FROM Production.ProductDescription

WHERE CONTAINS(Description, 'ISABOUT (performance weight (.8),

comfortable weight (.4), smooth weight (.2) )' );

GO

**H. Using CONTAINS with variables**

The following example uses a variable instead of a specific search term.

USE AdventureWorks;

GO

DECLARE @SearchWord nvarchar(30)

SET @SearchWord = N'Performance'

SELECT Description

FROM Production.ProductDescription

WHERE CONTAINS(Description, @SearchWord);

GO

**I. Using CONTAINS with A Logical Operator (AND)**

The following example uses the **ProductDescription** table of the **AdventureWorks** database. The query uses the CONTAINS predicate to search for descriptions in which the description ID is not equal to 5 and the description contains both the word "Aluminum" and the word "spindle." The search condition uses the AND Boolean operator.

USE AdventureWorks;

GO

SELECT Description

FROM Production.ProductDescription

WHERE ProductDescriptionID <> 5 AND

CONTAINS(Description, ' Aluminum AND spindle');

GO

**J. Using CONTAINS to Verify a Row Insertion**

The following example uses CONTAINS within a SELECT subquery. Using the **AdventureWorks** database, the query obtains the comment value of all the comments in the **ProductReview** table for a particular cycle. The search condition uses the AND Boolean operator.

USE AdventureWorks;

GO

INSERT INTO Production.ProductReview

(ProductID, ReviewerName, EmailAddress, Rating, Comments)

VALUES

(780, 'John Smith', 'john@fourthcoffee.com', 5,

'The Mountain-200 Silver from AdventureWorks Cycles meets and exceeds expectations. I enjoyed the smooth ride down the roads of Redmond')

-- Given the full-text catalog for these tables is Adv\_ft\_ctlg,

-- with change\_tracking on so that the full-text indexes are updated automatically.

WAITFOR DELAY '00:00:30'

-- Wait 30 seconds to make sure that the full-text index gets updated.

SELECT r.Comments, p.Name

FROM Production.ProductReview r

JOIN Production.Product p

ON

 r.ProductID = p.ProductID

AND r.ProductID = (SELECT ProductID

                  FROM Production.ProductReview

                  WHERE CONTAINS (Comments,

                                 ' AdventureWorks AND

                                   Redmond AND

                                   "Mountain-200 Silver" '))

GO

**A. Using FREETEXT to search for words containing specified character values**

The following example searches for all documents containing the words related to vital, safety, components.

USE AdventureWorks;

GO

SELECT Title

FROM Production.Document

WHERE FREETEXT (Document, 'vital safety components' );

GO